People’s Democratic Republic of Algeria

The Ministry of Higher Education and Scientific Research

Saad Dahleb Blida University, Faculty of Sciences

Computer Science Department

End of studies project with a view to obtaining the
Master’s degree in computer systems and networks

Theme:

Distributed Image Compression for Image Sensor Networks

Presented by:

Mr.Merdj Anis

Mr.Khalfi Younes

Supervisor:

Dr.Midoun Khadidja

Year: 2023-2024

Acknowledgement

First, we would like to thank ALLAH for giving us the courage, strength, and willpower to succeed, and for guiding us throughout our lives.

We are very grateful to our supervisor, Mrs. Midoun Khadidja, for her guidance, support, and advice during this time.

We also want to thank the members of the jury for taking the time to review our thesis.

We extend our heartfelt thanks to our parents, to whom we owe everything, and may God keep them safe for us.

Abstract

As the demand for surveillance and monitoring continues to grow, the necessity for an efficient network to manage these tasks becomes increasingly critical. Wireless Multimedia Sensor Networks (WMSNs) effectively address these needs. However, the large volume of multimedia data poses several challenges for WMSNs, with energy consumption being the most significant issue. This work presents a solution to mitigate high energy consumption by implementing a hybrid image compression method distributed across sensors. This hybrid technique combines two lossy compression methods: discrete wavelet transform (DWT) and two-dimensional discrete cosine transform (DCT). Initially, the image is decomposed using DWT into low and high-frequency sub-bands. Each "n × n" block of the low-frequency sub-band undergoes transformation using two-dimensional DCT. A fuzzy logic system (FLS) selects optimal sensors for the distribution tasks within the network. This approach significantly reduces processing time and extends the network's lifespan while maintaining data quality.

Key work: WMSN, Energy Consumption, Image compression.

Résumé

Avec la croissance de la demande en surveillance et en monitoring, la nécessité d'un réseau efficace pour gérer ces tâches devient de plus en plus critique. Les réseaux de capteurs multimédias sans fil (RCMSF) répondent efficacement à ces besoins. Cependant, le volume important de données multimédia pose plusieurs défis pour les RCMSF, la consommation d'énergie étant le problème le plus significatif. Ce travail présente une solution pour atténuer la haute consommation d'énergie en mettant en œuvre une méthode hybride de compression d'image distribuée sur les capteurs. Cette technique hybride combine deux méthodes de compression avec pertes : la transformée en cosinus discrète bidimensionnelle (DCT) et la transformée en ondelettes discrète (DWT). Initialement, l'image est décomposée à l'aide de la DWT en sous-bandes de basses et hautes fréquences. Chaque bloc "n × n" de la sous-bande de basses fréquences est ensuite transformé à l'aide de la DCT bidimensionnelle. Un système logique des flous (FLS) sélectionne les capteurs optimaux pour les tâches de distribution au sein du réseau. Cette approche réduit considérablement le temps de traitement et prolonge la durée de vie du réseau tout en maintenant la qualité des données.

Les mots clés : RCMSF, Consommation d'énergie, la compression d’image.

ملخص

الطلب العالي و المستمر من اجل المراقبة الأمنية في مختلف المجالات ادى بالباحثين لضرورة تطوير شبكة الكترونية فعالة لادارة هذه المهام. تعالج شبكات أجهزة الاستشعار اللاسلكية هذه الاحتياجات بفعالية. و مع ذلك، فإن الحجم الكبير للبيانات التي تنتجها هذه المستشعرات يثير العديد من التحديات لهذه الشبكة، و من ابرز هذه التحديات: الاستهلاك الكبير للطاقة. يقدم هذا العمل حلاً للتخفيف من استهلاك العالي للطاقة من خلال تنفيذ طريقة ضغط للصور مطورة تجمع بين تقنيتين المعروفتين باسم: DWT و DCT ، و هذه التقنية المطورة يتم توزيع مهامها بين مستشعرات الشبكة . في البداية، يتم تفكيك الصورة باستخدام DWT إلى مجموعات فرعية منخفضة وعالية التردد و يخضع كل فرع "n × n" من المجموعة الفرعية للتردد المنخفض ، لتحول باستخدام DCT ثنائي الأبعاد. المهام الموزعة يتم اختيارها بواسطة نظام اسمه FLS ، وهذا النظام يختار المستشعرات الامثل من حيث الطاقة و الاداء للقيام بعمل الضغط . تساهم هذه التقنية المقترحة في التقليل من الوقت المستغرق في معالجة البيانات و ارسالها مع الحفاض على جودة البيانات .

الكلمات المفتاحية :شبكات أجهزة الاستشعار اللاسلكية ، استهلاك الطاقة ، ضغط الصور .

TABLE OF CONTENTS

List of Tables

List of Figures

Notation and Acronyms

complementary metal-oxide semiconductor (CMOS)

Wireless Multimedia Sensor Networks (WMSNs)

Distributed Hybrid Image Compression Architecture (DHICA)

Vector Quantization (VQ)

Scalar Quantization (SQ)

specific quality-of-service (QoS)

Sensor node (SN)

normal node (NN)

Camera node (CN)

Cluster Head (CH)

communication range (Rc)

low-energy adaptive clustering hierarchy (LEACH)

image compression cluster head (ICH)

analog-to-digital converters (ADCs)

Discrete Cosine Transform (DCT)

Discrete Wavelet Transform (DWT)

run-length encoding (RLE)

Fuzzy Logic System (FLS)

Distributed Image Compression Architecture (DICA)

General Introduction

General Introduction

In today's world, multimedia applications are everywhere, from surveillance cameras and traffic monitoring ensuring public safety to capturing memorable moments like sports events and various other domains. These tools, named sensors, make life easier and safer. As their usage increases, it is important to continue enhancing their capabilities. These sensors, often inexpensive CMOS cameras and microphones, are interconnected through specialized networks called Wireless Multimedia Sensor Networks (WMSNs) [1], interacting with each other and capturing multimedia data (audio/video) that describes their environment.

Multimedia data [2] generates a large amount of information that requires significant bandwidth and high-performance sensors for transmission, thus making it more complex to transmit while preserving its content throughout the network. Considering the energy limitations inherent in sensor nodes powered by batteries, the primary limiting constraint for WMSNs is the energy consumption that affects the network lifetime. Therefore, it is essential for the network to remain operational for a duration of time that depends on the application of the deployed network, which may last from several weeks to a few years. Consequently, minimizing network energy consumption becomes crucial, and this is the primary goal of our thesis.

The transmission of data and communication between sensors consumes the majority of the system energy in a WMSN. For an efficient WMSN, it is imperative to focus on minimizing energy consumption [3]. To address this challenge, we propose a technique aimed at minimizing energy usage and enhancing network lifetime. This technique focuses on two key elements: data compression and distributed compression tasks across the network. By compressing data and distributing compression tasks strategically, significant energy savings can be achieved, contributing to prolonged network operation and improved performance.

The domain of compression is crucial within WMSNs. When a sensor captures images or video and transmits them, a significant amount of data is generated. Transmitting this data without compression is impractical due to its large size. Compression reduces the size of the data, making it quicker to transmit to its destination. However, compression tasks also require energy. Distributing the compression tasks among sensors further improves the network's lifetime. By combining these two approaches, we achieve a significant enhancement in network lifetime.

This thesis is organized into three chapters as follows:

- In the first chapter, we discuss the various concepts of Wireless Multimedia Sensor Networks (WMSNs), the different challenges in this domain, and how we address them with image compression. We also explore different compression techniques and the steps involved in JPEG and JPEG2000 compression.
- In the second chapter, we delve into our proposed technique and architecture to address the problem of network lifetime. We detail how we implemented our network and the step-by-step process of building our architecture.
- In the third chapter, we present the various practical tests conducted and evaluate the results obtained.
- Finally, we conclude our work with a general conclusion and discuss the future perspectives of our research.

Chapter Ⅰ

Background and State-of the Art

I. Introduction

WMSNs extend the capabilities of conventional WSNs by enabling the integration of multimedia content into sensor network applications, offering richer and more diverse information for monitoring and analysis. In this chapter, we first present an overview of Wireless Multimedia Sensor Network and its architecture. Then, we present the main image compression techniques in Section **III.** After then, in Section **IV**, we present an overview of some of the proposed algorithms to improve the image compression in WMSN.

II. Overview of Wireless Multimedia Sensor Network

Wireless Multimedia Sensor Network (WMSN) is a specialized type [5] of wireless sensor network (WSN) designed to capture, transmit, and process multimedia data such as images, videos, and audio streams in addition to traditional scalar sensor data.

In WMSNs, dealing with multimedia data brings unique challenges not seen in traditional Wireless Sensor Networks. Because multimedia content requires more bandwidth and energy [6] , it strains network resources and drains battery life faster. Overcoming these hurdles requires smart solutions in areas like compression, energy management, and communication protocols, paving the way for exciting applications across different fields.

1. Application domain of WMSNs

WMSNs are highly used in numerous domains due to their capability to capture, process, and transmit multimedia data. When it comes to gathering and analyzing multimedia content from the environment, WMSNs play a crucial role in providing real-time information and establishing a comprehensive database for analysis in a timely manner. Below are some applications [7] :

- **Surveillance:** WMSNs enhance surveillance systems with video and audio sensors, helping in crime prevention and investigation by recording relevant activities.
- **Traffic Monitoring:** WMSNs enable real-time traffic monitoring, offering routing advice to alleviate congestion and facilitating law enforcement in detecting violations.
- **Personal and Health Care:** WMSNs monitor elderly behavior and vital signs, enabling remote healthcare services and emergency.

- **Gaming:** WMSNs enrich networked gaming experiences by integrating multimedia data, enhancing player interactions with the game environment.
- **Environmental and Industrial Monitoring:** WMSNs facilitate habitat monitoring and industrial process control through time-critical data conveyance and automated inspections, improving efficiency and accuracy in various industries.

2. WMSNs Architecture

WMSN is an advanced data acquisition and monitoring system. It consists of two types of nodes: Typical sensor nodes, which are called normal nodes (NNs) and are traditionally used to aid the transmission or perform other scalar-based sensing tasks, and camera nodes (CNs): Captures the multimedia content and sends it to an NN for processing or transmission to its base station (BS). These nodes, operating autonomously, are deployed in a distributed manner within an environment of interest, without requiring prior configuration.

Following principles similar to mobile ad hoc networks, sensor nodes in a WMSN discover each other and self-organize to form a wireless network. This network is designed to collect, process, and transmit multimedia and scalar data through successive hops, before routing them to a base station or end node via a multi-hop communication infrastructure.

A multimedia sensor device may be composed of several basic components [7], as shown in (Figure 1):

- **Sensing unit:** Composed of sensors (cameras, microphones, scalar sensors) and analog-to-digital converters (ADCs).
- **Processing unit (CPU):** Executes system software for coordinating sensing and communication tasks, interfaced with a storage unit.
- **Communication subsystem:** Facilitates the connection between the device and the network, including a transceiver unit and communication software such as protocol stacks, middleware, and operating systems.
- **Coordination subsystem:** Coordinates the operation of different network devices, performing tasks like network synchronization and location management.
- **Storage unit (memory):** Stores digital signals and system software.
- **Optional mobility/actuation unit:** Enables movement or manipulation of objects.

- **Power unit:** Powers the entire system, potentially supported by an energy scavenging unit like solar cells**.**

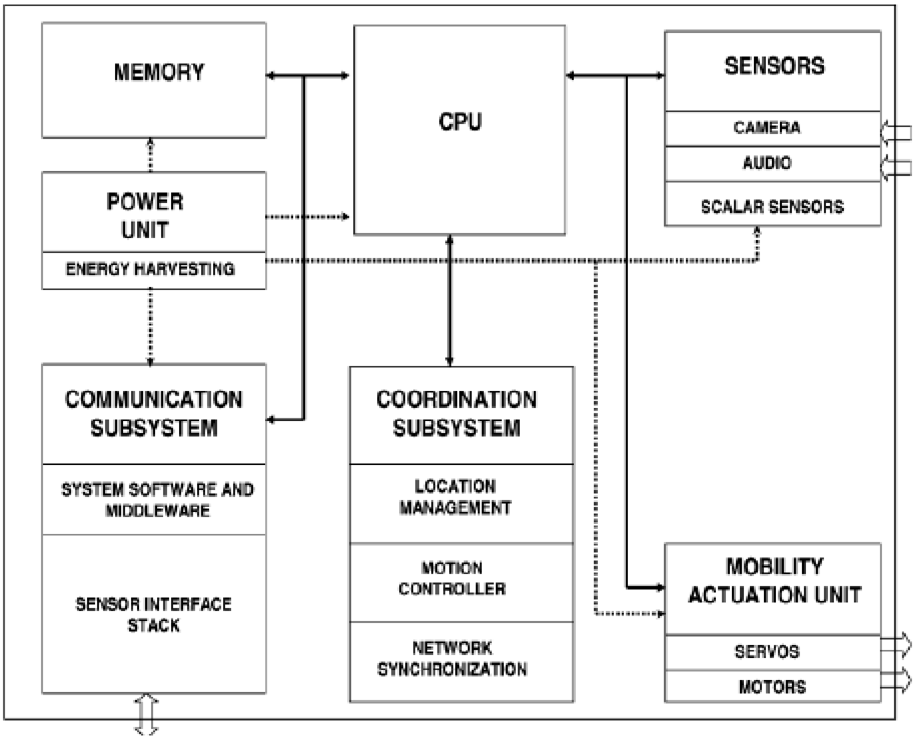


Figure 1:Internal organization of a multimedia sensor [7].

The design and implementation of WMSNs are influenced by various parameters, including:

- **Network lifetime:** The duration between network deployment and the depletion of the first node's energy, which can vary from hours to years depending on the application.
- **Limited resources:** Sensor nodes have limited processing and memory capacity, aiming for simple, small, and cost-effective sensor implementation.
- **Limited bandwidth:** To reduce energy consumption during data transfer between nodes, the data rate is typically a few tens of Kb/s. Low transmission rates are not a hindrance because transmission frequencies are not critical.
- **Scale factor:** Applications may require thousands of nodes, resulting in high sensor density, extensive internal transfers, and necessitating sufficient memory in the base station to store received information**.**

- **Dynamic topology:** Network topology may change over time due to energy depletion, node failure, or deployment in hostile environments.
- **Data aggregation:** Data produced by neighboring sensor nodes are often highly spatially and temporally correlated, leading to redundant information. Reducing redundant information transmission can decrease energy consumption and increase network lifetime. Data aggregation, where intermediate nodes collect information from multiple sources, is one technique used to reduce redundant information transmission.

Most of scalar WSNs have a flat architecture [8] of distributed homogeneous nodes. These networks measure physical phenomenon like temperature and provide as an output a scalar value for each measurement. WMSNs have been introduced in emerging application fields that raise the need to have alternative network architectures. Scalability, efficiency, and QoS are the main requirements for new network architectures. Generally, WSNs architecture [8] can be either single-tier flat, single-tier clustered, or multi-tier, as shown in (Figure 2).

- **Single-tier flat architecture:** The network is set up with homogeneous sensor nodes with the same capabilities and functionality. All the nodes perform functions like image capturing, multimedia processing, or data delivery to the sink. This flat architecture is easy to manage and the multimedia processing is distributed among the nodes, contributing to the extension of the network life time.
- **Single-tier clustered architecture:** The network is deployed with heterogeneous sensors like video, audio, and scalar ones. All different sensors within each cluster relay data to a Cluster Head (CH), which is connected with the sink directly or through other CHs.
- **Multi-tier architecture:** This network is deployed using heterogeneous sensors in a multi-tier architecture.

- The first tier performs simple tasks via scalar sensors like motion detection.
- The second tier performs more complex tasks like object detection or recognition.
- The third tier may achieve an object tracking via powerful high resolution camera sensors, which is a complex task to do.

Each tier can have a central hub to enhance data processing and communication with the higher tier. This architecture meets different needs while balancing between costs, coverage functionality, and reliability requirements.

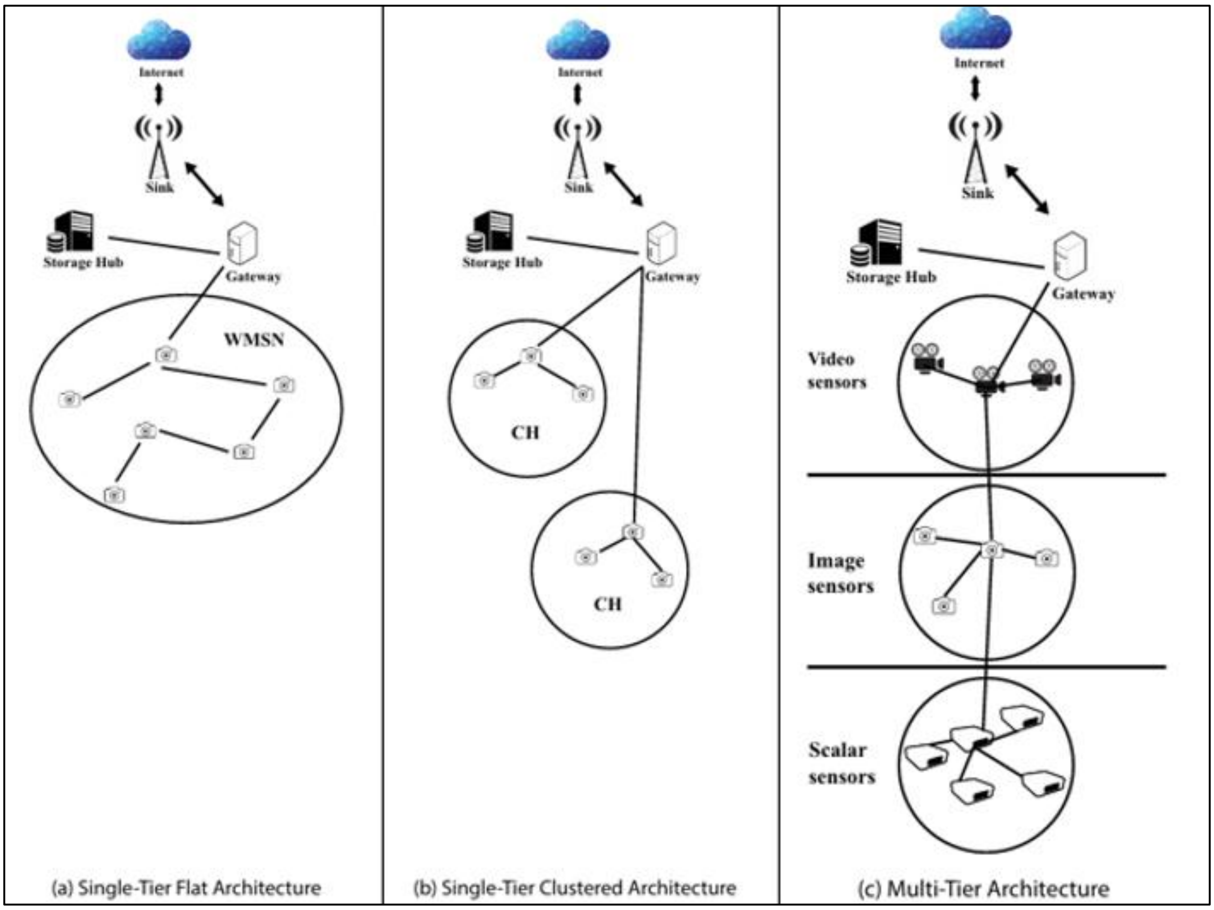

Internet
Sink
Storage Hub
Gateway
WMSN
CH
CH
Video sensors
Image sensors
Scalar sensors
(a) Single-Tier Flat Architecture
(b) Single-Tier Clustered Architecture
(c) Multi-Tier Architecture

Figure 2: WMSN architectures [8].

3. Challenges in WMSNs

The recent attention towards WMSNs shows a big step forward in sensor technology. WMSNs use communication, computation, and signal processing to make sensors do many things. However, making WMSNs is hard because they need to meet specific quality standards, handle lots of data, use energy wisely, and work reliably.

In this part, we explore the emerging trends and challenges [3] in WMSNs, aiming to gain a deeper understanding of this evolving field.

- **QoS Requirement**: In WMSNs, different applications require the network to meet specific quality-of-service (QoS) needs. These include:
  - Real-time constraints: Physical event must be reported within a certain period of time.
  - Robustness: The network should remain operational even if certain well-defined failures occur.
  - Tamper-resistance: The network should remain operational even when subjected to deliberate attacks.
  - Eavesdropping resistance: External entities shouldn't be able to listen in on data traffic.
  - Unobtrusiveness or stealth: The network's presence should be hard to detect.
  - High bandwidth demand **:** Real-time multimedia applications are well known with their high bandwidth requirements and stringent delay constraints, which may be hard to satisfy even on wired links.
- **Scalable and flexible architectures :** WMSNs must also be capable of expanding in size and supporting diverse applications. To address this, adaptable architectures are crucial, capable of accommodating different application needs within the same network. Common approaches for scalability and flexibility in WMSNs include clustering, multihop delivery, and efficient organization of computation and protocols.
- **Localized processing and data fusion**: WMSNs usually generate a great volume of multimedia-stream redundancy. To eliminate data redundancy, collaborative efforts should be made among different microsensors performing a variety of localized processing. Instead of sending the raw data to the destination directly, microsensors might locally filter the data according to the requirements, carry out related video-stream or audio-stream compression, process the data, and transmit only the processed data.
- **Reliability and fault tolerance:** Reliability and fault tolerance are critical aspects in wireless sensor networks. In traditional sensor networks, faults may occur frequently or stop occurring after a certain time. However, in multimedia sensor networks, faults have a more serious impact due to the volume of video or audio streams. Resource and energy constraints limit testing and fault tolerance capabilities. Additionally, complex architectures and hostile environments further complicate fault management. Fault tolerance techniques must be tailored to address these challenges.
- **Multimedia coverage:** Multimedia coverage in wireless networks refers to how well sensor nodes monitor an area. It depends on sensor range and deployment density. Sparse coverage

means only some areas are monitored, while dense coverage means the entire area is covered. Redundant coverage occurs when multiple sensors cover the same spot. The coverage level depends on observation accuracy and redundancy needs. Nodes may be closer together in some areas, and video sensors have wider ranges and are sensitive to direction. It is crucial to consider coverage, not just connectivity, when setting up these networks.

- **Models for Programmability:** Microsensors equipped with cameras and microphones must perform various tasks such as capturing images, recording audio, and processing multimedia data. However, storing all the necessary programs for these diverse tasks in a single sensor is impractical due to memory constraints. Therefore, dynamic programmability is essential WMSNs instead of relying on fixed coding methods. With dynamic programmability, sensors can adjust their functionality in real-time based on the specific requirements of the application or environment. For example, a sensor deployed in a surveillance scenario may prioritize image capture during the day and switch to audio recording at night, optimizing resource usage and adapting to changing conditions.
- **Energy Efficiency Design**: In contrast to traditional sensor networks, where communication functions dominate energy consumption, in WMSNs, both communication and processing tasks significantly contribute to energy usage. So, optimizing protocols, algorithms, and architectures to extend the system's battery life is crucial for the efficient operation of WMSNs.

While WMSNs face various hurdles such as quality-of-service requirements, scalability, high bandwidth demands, and programmability, among others, one of the most crucial problems is energy consumption. Effectively managing energy usage is paramount in WMSNs to ensure prolonged network operation and optimal performance. Fortunately, image compression technology offers a promising solution to this challenge. By compressing multimedia data, WMSNs can reduce bandwidth requirements, optimize energy usage, and enhance overall network performance.

III. The Image Compression

A digital image consists of pixels that are highly correlated to each other [9]. When a two-dimensional light intensity signal is sampled and quantized to create a digital image, a huge amount of data is produced. The size of the digitized picture could be so great that results in impractical storage or transmission requirements. Image compression deals with this problem such that the information required to represent the image is reduced thus making the transmission or storage requirements of images more practical.

Image compression in WMSNs presents significant challenges. Nodes in these networks must perform real-time compression, and the compression ratio required per node is exceptionally high. Traditional image compression algorithms, while effective, often come with high computational complexity and overhead, making them unsuitable for resource-constrained WMSNs. As such, there is a need for specialized compression techniques tailored to the requirements and constraints of WMSNs, and this will be discussed in detail in the next chapter .

1. Image Compression steps

The diagram of a typical compression system is show in (Figure 3), it consists of three steps:

a) **Transform**: The objective of transformation is to reduce data redundancy by transforming the image to make the pixels less correlated with each other. A common method is the Discrete Cosine Transform (DCT) used in JPEG format, or the Discrete Wavelet Transform (DWT) used in JPEG2000 format. The process involves applying a mathematical transformation to the image, converting pixel values into coefficients representing frequency or other characteristics. This concentrates the image's energy into a few coefficients, facilitating compression.

b) **Quantization:** A quantizer reduces the number of bits required to store transformed coefficients by lowering their precision [8]. This many-to-one mapping process is lossy and is the primary source of compression in an encoder. Quantization can be applied to each coefficient individually, known as Scalar Quantization (SQ), or to a group of coefficients together, known as Vector Quantization (VQ). Depending on the specific application, either uniform or non-uniform quantizers may be used.

c) **Entropy coding**: An entropy encoder further compresses the quantized values lossless to give better overall compression. It uses a model to accurately determine the probabilities for each quantized value and produces an appropriate code based on these probabilities so that the resultant output code stream will be smaller than the input stream. The most commonly used entropy encoders are the Huffman encoder and the arithmetic encoder, although for applications requiring fast execution, simple run-length encoding (RLE) has proven very effective.

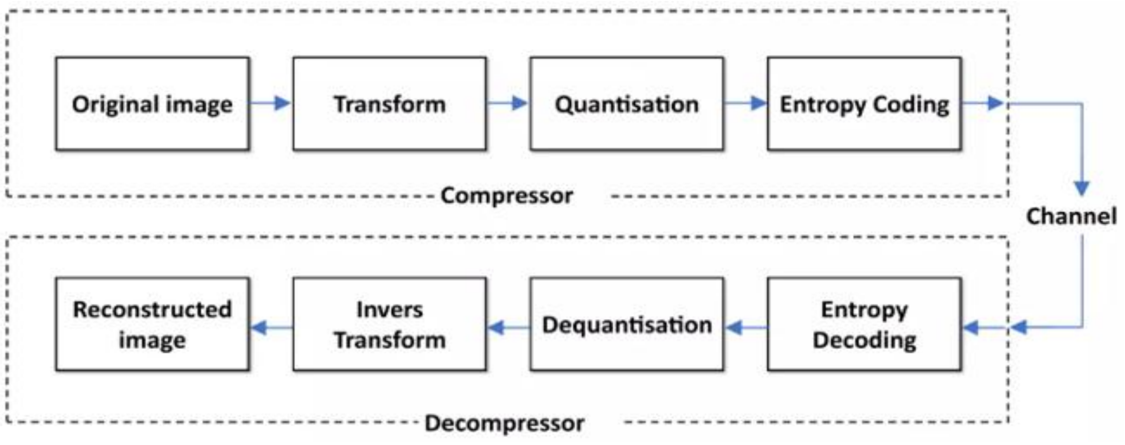


Figure 3: General compression scheme [10].

2. Image compression performance metrics

Performance of compression techniques depends on the following criteria [10]:

- **Quality of image:** To ensure image quality after compression, methods for judging image quality and measuring distortion are essential. There are two types of measures: subjective and objective. Subjective measures rely on human observers to assess image quality based on their experience, either relatively or absolutely. Objective measures, on the other hand, use mathematical calculations to quantify image distortion and quality.

- **Compression ratio (CR) :** CR can be used to judge how compression efficiency is, as higher CR means better compression**.**

$$CR = \frac{\textbf{uncompressed image size}}{\textbf{Compressed image size}}$$

- **Speed of compression:** is influenced by**:**

  – **Computational complexity:** Complex algorithms require more processing power and time to execute, which can slow down the compression process.

  – **Memory resources:** Efficient memory management can enhance compression speed by reducing the time needed for data access and manipulation.

- **Power consumption:** Power consumption is a critical factor in the performance of compression techniques, especially WMSNs where energy resources are limited. Efficient compression algorithms should minimize power usage while maintaining acceptable levels of image quality, compression ratio, and speed.

3. Image Compression Techniques for Efficient Data Transfer in WMSNs

Understanding the chosen compression technique is pivotal for effective data transfer in WMSNs, where efficiency is paramount. There are two types of compression methods, lossless and lossy image compression:

The Lossless compressed image technique has a very large size compared with lossy one which consumes more power of sensor nodes and bandwidth in case of power constrained applications. Therefore, lossless compression is not preferred for image transfer over WMSN which leads us to focus on lossy compression techniques that highly encouraged in WMSN.

Lossy compression techniques provide high compression ratio compared with lossless compression ratio. In lossy compression, the compressed image is not usually the same as the original one, but forms a close approximation to the original image.

The most important lossy techniques used for restricted resources applications that are classified into 2 categories: (1) Transform-based transmission techniques and (2) Non transform based transmission techniques.

1) Transform-based transmission techniques

Transform-based techniques involve converting an input vector X (which could be an image) through a transformation T into another form Y that is less correlated than X. The transformation T itself does not compress any data; rather, compression occurs through the processing and quantization of the components of Y [11]. In this context, two common transformations are described: the Discrete Cosine Transform (DCT) and the Discrete Wavelet Transform (DWT).

A- Discrete Cosine Transform (DCT):

The DCT process transforms image blocks into frequency domain coefficients, quantizes them to reduce precision, encodes them for compression, and reconstructs the image during decoding. It efficiently compresses images by converting spatial data into frequency components, allowing for reduced storage or transmission while preserving image quality.

One of the most commonly used techniques that utilize DCT is JPEG. In order to get a deeper understanding of JPEG process, it is necessary to read the Annex A (page 77).

B- Discrete Wavlete Transform (DWT):

Wavelet-based transforms use a set of basis functions, known as wavelets, to represent a signal with high resolution in both time and frequency domains. In image compression, 2D wavelets are employed, which are separable functions. The Implementation involves applying a **low-pass filter** on rows to produce L and H subbands, followed by applying a **high-pass filter** on columns to generate four subbands: LL, LH, HL, and HH.

In subsequent levels, each of these four subbands is further decomposed into four more subbands: LL2, LH2, HL2, and HH2, and so on. This decomposition process can continue for multiple levels, providing increasingly detailed representations. (Figure 4) illustrates the decomposition of the subbands.

One of the most commonly used techniques that utilize DWT is JPEG2000. In order to get a deeper understanding of JPEG2000 process, it is necessary to read the Annex A (page 87).

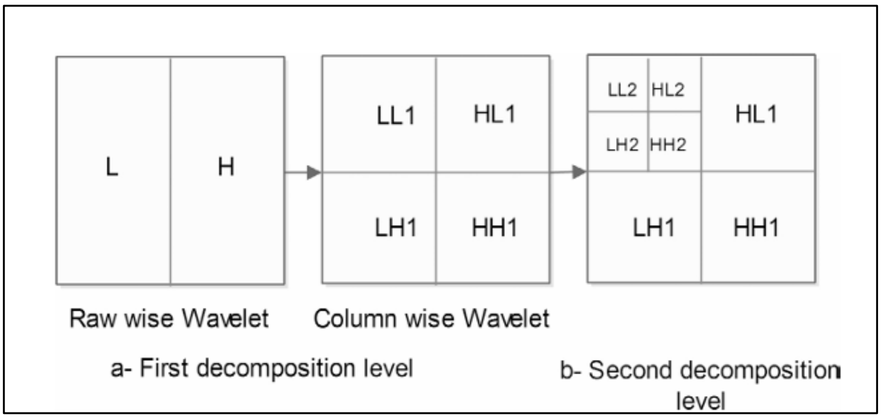


Figure 4:Two level of decomposition of 2D-DWT.[11]

C- Comparison between DCT and DWT over WMSN:

While both DCT and DWT are utilized in WMSNs, the choice depends on specific application requirements. These include balancing factors. Utilizing a hybrid technology that combines DCT and DWT can leverage the strengths of both transforms to optimize performance, achieving better compression efficiency and image quality while managing computational load and energy consumption effectively. Table 1 presents the comparison between DCT and DWT over WMSN

Table 1:Comparison between DCT and DWT over WMSN

	DCT	**DWT**
Applicability in WMSNs	Widely applied in WMSNs due to its simplicity and energy efficiency.	Also used in WMSNs, especially for its better compression ratio and image quality.
Compression Algorithms	Utilizes a fast and energy-efficient algorithm, such as JPEG.	Employs more complex algorithms like JPEG2000, offering better compression ratio and image quality.
Image Compression Process	Applies discrete cosine transform to separate the image into pixels and quantizes them for compression	Divides the image into LL, LH, HL, and HH subbands using wavelet transform, offering more detailed compression.
Decentralized Processing	Involves distributing compression tasks among sensor nodes, but may face challenges in load balancing.	Also employs distributed processing, with tasks distributed among nodes, leading to improved network lifetime and load balancing.
Energy Consumption	Requires less computational complexity and energy consumption but may result in lower image quality.	More computationally intensive but offers better image quality and compression ratio, suitable for high-quality image transmission.
Image Quality and Compression Ratio	Provides moderate compression ratio but may sacrifice image quality, especially in lossy compression.	Offers better compression ratio and image quality, making it suitable for applications where image fidelity is crucial.

Adaptability and Scalability	Relatively simpler and easier to implement, suitable for small-scale WMSNs.	More complex but offers scalability and adaptability for larger and denser WMSNs, especially with advanced compression algorithms like JPEG2000.

2) Non-Transform-based transmission techniques

Non-transform-based techniques in image compression, such as Vector Quantization (VQ), rely on vector quantizers to achieve compression. These methods, as illustrated in the VQ encoding block diagram (Figure 5), involve partitioning the image into blocks of pixels, with each block represented by a vector, x. These vectors are then compared against codewords stored in a codebook at the encoder's side. The goal is to find the best match, obtaining the index of the code word that corresponds most closely to the vector. Consequently, instead of transmitting the entire codeword, only the index, stored in fewer bits, is transmitted. This approach significantly enhances compression ratios by reducing the amount of data transmitted.

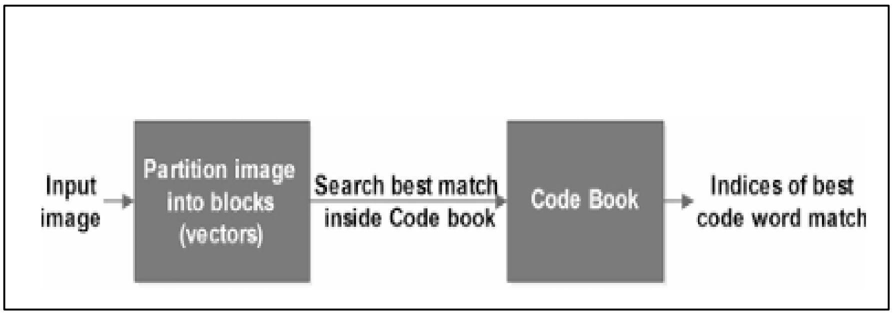

Input image
Partition image into blocks (vectors)
Search best match inside Code book
Code Book
Indices of best code word match

Figure 5:Block diagram of vector quantization encoder[11]

IV. Related Work

In this section, we provide an overview of some of the proposed algorithms to improve image compression in WMSNs. This review covers various techniques that have been suggested in different related works and have been developed to address the challenges associated with the large volume of multimedia data, focusing on methods that enhance compression efficiency while preserving data quality and reducing energy consumption.

Table 2:Overview of some of the proposed algorithms to improve the image compression in WMSN.

Title	System abstract	Technique	Results obtained
[01]	This article introduces a parallel APBT-JPEG system using CUDA on GPUs, which addresses JPEG's blocking artifacts, enhancing speed and efficiency for real-time image compression in WMSNs.	JPEG	The parallel APBT-JPEG algorithm achieves over 100 times acceleration compared to the serial version,PSNR values indicating superior image quality over DCT-JPEG. Visual assessments show that APBT-JPEG significantly reduces blocking artifacts, and leveraging GPU parallel computing enables efficient processing of multiple images simultaneously.
[02]	When a high temperature is detected by scalar sensors, an alarm is sent to the base station, which then activates nearby camera sensors to capture and process images for fire detection and compression using the Downsample Method .	JPEG (down-sampling)	The Downsample Method effectively reduces the size of images while maintaining necessary details for fire detection. By downsampling an image by a factor of k=2, the system achieves significant data reduction, facilitating quicker transmission and lower energy consumption in the network.
[03]	By sharing the image compression work among multiple sensors, the system saves energy compared to centralized methods. Two schemes were tested: dividing images by rows and by blocks, with the block method using less energy. However, if the source and base station are close, the nodes near the base station may use up their energy quickly.	JPEG2000 (DWT)	JPEG2000, DWT, demonstrates high effectiveness in image compression for WMSNs. It achieves a high compression ratio and maintains high image quality with a high PSNR. However, it requires significant memory and processing power, making it computationally intensive. JPEG2000 avoids blocking artifacts commonly seen in other compression techniques but is less efficient in energy compaction.
[04]	This article focuses on designing a cost-effective WMSN node using a low-end FPGA to achieve efficient image compression. The node offers high processing ability, low power consumption, flexibility, and reliability.	JPEG	The system utilizes a modular implementation of JPEG image compression on reconfigurable hardware, resulting in higher resource utilization, reduced power consumption, and faster processing times.

Title	System abstract	Technique	Results obtained
[15]	This article introduces a parallel APBT-JPEG system using CUDA on GPUs, which addresses JPEG's blocking artifacts, enhancing speed and efficiency for real-time image compression in WMSNs.	JPEG	The parallel APBT-JPEG algorithm achieves over 100 times acceleration compared to the serial version,PSNR values indicating superior image quality over DCT-JPEG. Visual assessments show that APBT-JPEG significantly reduces blocking artifacts, and leveraging GPU parallel computing enables efficient processing of multiple images simultaneously.
[16]	This article introduces an innovative architecture and protocol for energy-efficient image processing and communication in WMSN.	JPEG2000	The results show that the proposed architecture provides high processing speed and significantly reduces energy consumption for both image processing and communication compared to Commercial Off-The-Shelf (COTS) based WMSN motes.

V. Conclusion

In conclusion, we now understand the essential details about how WMSNs function, including the architecture of node deployment and the various applications that benefit from these networks. We have also explored the challenges faced when deploying WMSNs, such as network lifetime and energy requirements. To address these challenges, we identified image compression as a key solution and discussed various compression techniques in detail. With this comprehensive understanding of WMSNs we are now prepared to implement these networks effectively. This foundation allows us to optimize their performance using advanced techniques. In the next chapter, we will focus on the best practices and methodologies to further enhance the efficiency and functionality of WMSNs.

Chapter II

The proposed Distributed Image Compression Algorithm for Image Sensor Networks

I. Introduction

In this chapter, we'll focus on putting WMSNs into action, aiming to make them more energy-efficient and last longer by using image compression. Moving from theory to practice, we'll dig into how WMSNs actually work, covering things like where sensors go, what nodes do, and how they talk to each other. This groundwork sets the stage for our efforts to tweak WMSNs for multimedia tasks while saving energy.

Choosing the right routing protocols for image compression is the key to our plan. It is vital for cutting down on energy use while keeping data flowing smoothly across the network.

Throughout this chapter, our main goal stays the same: to turn theory into real-world improvements. By combining ideas with hands-on work, we're aiming to build WMSNs that are great at saving energy and handling multimedia tasks. With careful planning and execution, we hope to unleash the full potential of WMSNs and push forward innovation in wireless sensor networks.

II. System Model

1. Network assumption

In this section, we'll outline the foundational assumptions of our network setup. These assumptions principles serve as the basis for understanding how sensors network operates and communicates effectively within its designated area.

- Sensor nodes (SN) are distributed randomly and uniformly within a designated area ($M \times M$).
- The network comprises two types of SNs: camera sensor nodes (CNs) and typical sensor nodes (NNs).
- The number of CNs is significantly lower than the number of NNs.
- All NNs are homogeneous in each type, possessing identical initial energy, computational power, memory, etc.
- SNs are unaware of their locations and remain static post-deployment.
- All SNs share the same communication range (radius = Rc).

- Each SN can estimate its distance to other SNs based on received signal strength index (RSSI).
- A single base station (BS) is positioned at a specific location outside the sensor field, equipped with adequate hardware, software, and constant power supply.
- The operation is divided into rounds, with the transmission of a full image from a CN to the BS considered as one round.

2. Routing protocol

Several issues are encountered in WMSNs, including energy consumption, sensor node deployment, routing algorithms, energy efficiency, Cluster-Head (CH) selection, and robustness. Researchers have developed numerous routing protocols and proposed optimization techniques to address these constraints and define the optimal path between the transmitter and the receiving node. Routing protocols serve as maps for WMSNs, guiding sensor nodes on effective communication. Without these protocols, nodes risk being lost in a random setup, leading to network disorder. In our network, where sensors are randomly scattered, routing protocols are crucial for efficient communication and energy conservation. These protocols enable nodes to communicate effectively, ensuring smooth operation and prolonging the network's lifetime.

Routing protocols are categorized into flat, geographic, and hierarchical protocols, each with distinct approaches to node coordination and data transmission [17] :

- Flat protocols treat all nodes equally, while geographic protocols leverage node location data for decision-making.
- Hierarchical protocols organize nodes into clusters, where cluster heads are responsible for transmitting data to the base station. We opt for this protocol over others due to its numerous advantages [18], including reduced energy consumption, enhanced functionality, scalability, and reliability. These benefits arise from several factors: nodes taking on various roles or functions, cluster heads aggregating data to minimize unnecessary transmission, and non-cluster heads conserving energy by disabling their radios after transmitting packets.

One commonly used hierarchical routing protocol in WMSNs is: the low-energy adaptive clustering hierarchy (LEACH).

3. The low-energy adaptive clustering hierarchy LEACH

LEACH is an adaptive and self-organizing clustering protocol designed initially as a single-hop routing protocol. It operates by randomly selecting a cluster head (CH) from among sensor nodes (SNs) in rotation [18]. SNs transmit data to their designated CH, which then aggregates and forwards the data toward the base station (BS). This approach enables LEACH to efficiently distribute energy among SNs, resulting in reduced energy consumption and prolonged network lifetime. The (Figure 6) shows this process.

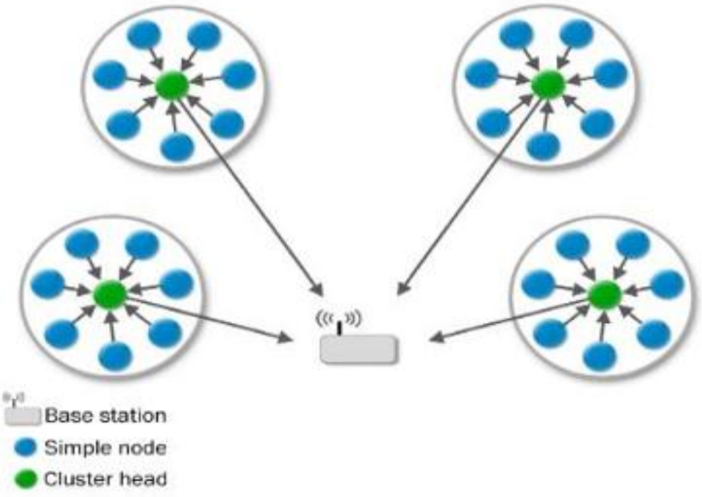


Figure 6:LEACH Process.

- **Limitation:** its inefficiency in large-scale networks [22], because each cluster head is directly communicating with BS no matter the distance between cluster head and BS is far or near. It will consume lot of energy if the distance is far. Additionally, LEACH was originally designed for WSN applications with low Quality of Service (QoS) requirements, while WMSN applications typically have larger QoS requirements that means more energy consumption, so making LEACH less suitable for such scenarios.

LEACH multi-hop can be considered as a solution to some of the limitations of traditional LEACH in WMSNs.

- **Multi-hop LEACH**

In Multi-hop LEACH, CHs utilize multi-hop communication to transmit data towards BS. Initially, CHs establish communication links with neighboring CHs to form an optimal data transmission path. Once this path is determined, CHs relay the aggregated data to the nearest cluster head in the direction of the sink. Eventually, the final cluster head in the transmission path forwards the data directly to the BS, thereby optimizing energy usage and improving the efficiency of data transmission in the network.

LEACH multi-hop can reduce energy consumption in several ways. Firstly, by employing multi-hop communication among cluster heads, it minimizes the direct transmission distance between cluster heads and the base station. This reduces the energy expended by cluster heads during data transmission, as shorter distances require less energy. Secondly, by selecting optimal paths for data transmission, LEACH multi-hop ensures that data is routed through intermediate cluster heads that are closer to the base station. This further reduces energy consumption compared to direct transmission to the base station, as data travels shorter distances and encounters fewer obstacles. Overall, LEACH multihop optimizes energy usage by efficiently routing data through the network, thereby extending the lifetime of the WMSN.

4. Energy Consumption Model

During the transmission and reception of data packets following the IEEE 802.15c framework in WSNs or WMSNs, SN consumes energy. This energy consumption depends on factors such as the size of the packet and the distance between the sender and receiver.

A. Energy Consumption for Transmitting and Receiving Data:

The energy consumed for transmitting 1 bits of data are given in (1) and (2) respectively

$$.\,E_{tx}(l,d) = \begin{cases} l \times E_{elec} + l \times \varepsilon_{fs} \times d^2 \,, if\ d < d_0 \\ l \times E_{elec} + l \times \varepsilon_{mp} \times d^4 \,, if\ d < d_0 \\ \end{cases} \quad (1)$$

$$.\,E_{tx}(l,d) = E_{elec} \times l \qquad (2)$$

. E_{elec} : is the energy that is consumed by the circuit per bit.

.d : is the distance between sender and receiver.

. ε_{fs} : corresponds to free space.

. ε_{mp} *: corresponds to multipath fading .*

B. Energy Consumption for Receiving Data:

When a sensor node receives a packet of k bits, the consumed energy is given by the equation (3):

$$E_R = K \times E_{elec} \quad (3)$$

C. Energy Consumption for Data Aggregation:

After receiving data, the sensor node requires energy for data aggregation, computed by equation (4) where E_{DA} is the energy used for aggregating certain data.

.
$$E_{agg} = E_{DA} \times n \times k \quad (4)$$

D. Image Compression Energy Consumption:

For calculating the energy per bit of image compression, we utilize equation (5). This equation factors in the energy costs of a two-level DWT compression, as well as the energy consumed during encoding. Furthermore, it considers the energy required for DCT processing and adjusts for the desired image quality level.

$$E_{comp} = E_{DWT} \times 2 + Q \times E_{DCT} + E_{encode} \quad (5)$$

E. Measured Values:

The measured values for E_{elec} , ε_{fs}, and ε_{mp} are approximately 50nj/bit, 10pj/bit, and 0.0013pj/bit/m4 respectively. For DWT energy (E_{DWT}), it is about 220nj/bit ,and for E_{DCT} it is about 200nj/bit, while the encoding E_{encode} energy is about 20nj/bit.

III. Our Distributed Hybrid Image Compression Architecture over Multi-hop WMSN

It is widely acknowledged that image compression addresses certain challenges within WMSN, yet the proliferation of sensors capable of capturing images introduces complexities. The compression process demands significant energy consumption and computational resources, potentially hampering network performance. Our primary aim is to optimize network efficiency while ensuring optimal performance. To mitigate these challenges, we advocate for the distribution of the image compression process. This approach offers several advantages, including reduced energy consumption, improved computational efficiency, and enhanced network reliability.

In this section, we describe in detail our proposed distributed hybrid image compression architecture for multi-hop WMSNs, namely, DHICA. This architecture was inspired by an architecture named DICA[22]. Our proposed architecture is divided into five phases per round: communication cluster setup, multi-hop hierarchical routing setup, camera cluster setup, image compression cluster setup, and image transmission see (Figure 7). Each phase plays a crucial role in ensuring efficient and effective image compression and transmission within the network.

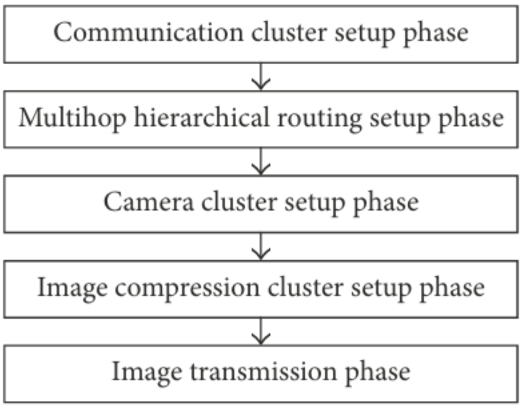

Communication cluster setup phase
Multihop hierarchical routing setup phase
Camera cluster setup phase
Image compression cluster setup phase
Image transmission phase

Figure 7:Phases of the proposed architecture .

1. Definition and notation

in our algorithm, we use nine different control messages:

1. CH-Msg (ID): Sent by the communication CH to its neighbors to announce itself as a CH. It contains only the CH's ID.

2. CL-Join-Request-Msg (CH-ID, ID): A request to join the communication cluster. This message includes the ID of the CH and the ID of the SN that wants to join the cluster.

3. Level-Msg (ID, routing level): Broadcast by the BS and communication CHs, containing their IDs and routing level information, to their neighbors to build a hierarchical routing structure.

4. Level-Request-Msg (ID): Sent by communication CHs that have not received any Level-Msg, during a specific time, to neighboring SNs to request routing level information. The message contains the requesting CH's ID.

5. Level-Reply-Msg (ID, routing level): Sent by the SN after receiving a Level-Request-Msg from the requesting communication CH. It includes the SN’s ID and routing level information.

6. Cam-Msg (ID, cam energy, node degree): Sent by the CN to its neighboring nodes, containing its ID, remaining energy, and node degree, to form a camera cluster.

7. Cam-Join-Request-Msg (ID, energy): Sent by a neighboring node of the CN to join the camera cluster. It contains the ID and remaining energy of the neighboring node.

8. Relay-Msg (ID): Sent by the source SN to its neighbors to find the relay node. It includes the ID of the source node.

9. Relay-Reply-Msg (ID, routing level, energy): Sent by neighboring nodes of the source node after receiving a Relay-Msg. It contains their IDs, routing level information, and remaining energies.

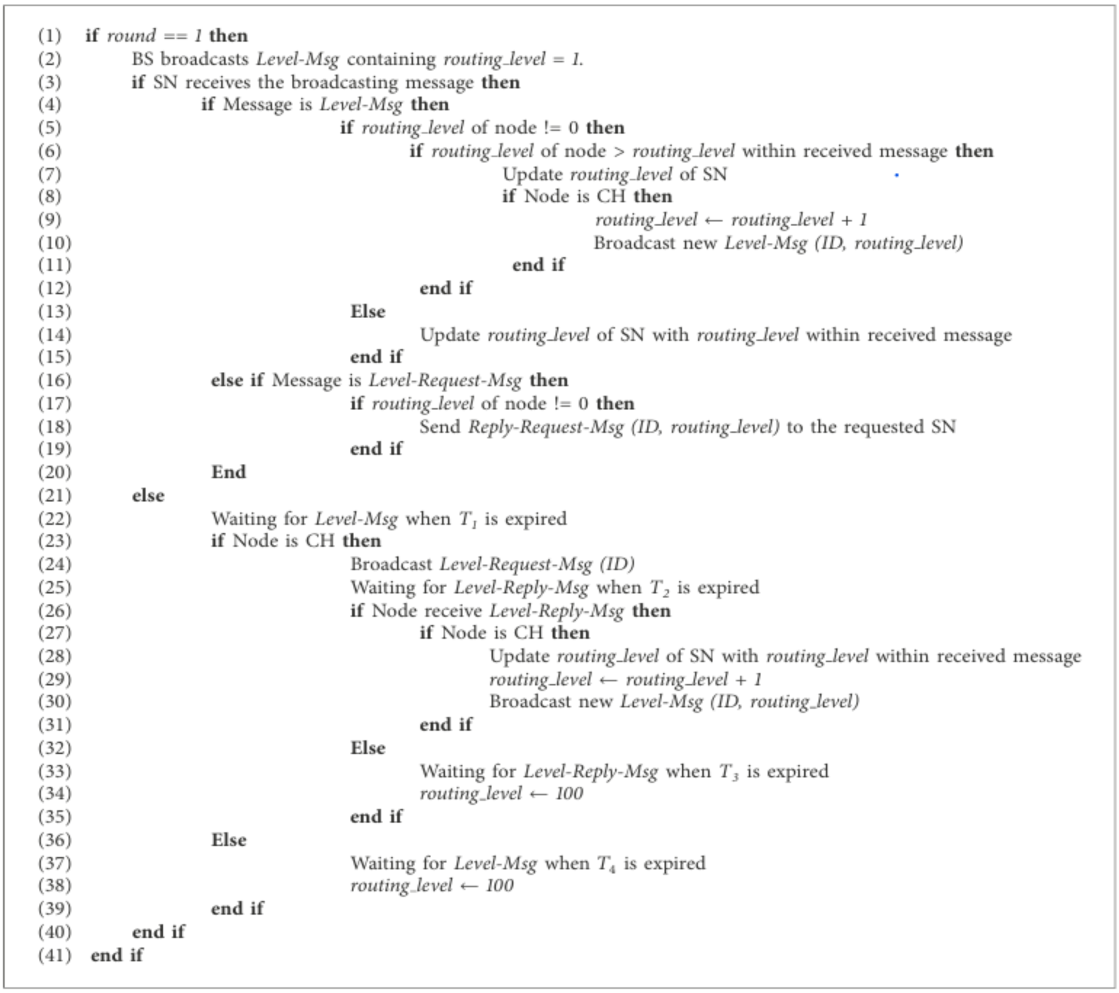
2. Communication cluster setup

The first step is establishing communication clusters. These clusters are essential for efficiently forwarding images to the Base Station (BS). The WMSN is divided into several communication clusters, each with its own Cluster Head (CH) and member nodes. Member nodes send their data to their respective CHs, which then forward the aggregated data to the destination.

We utilize LEACH, which is a pioneering cluster-based routing algorithm known for its effective energy consumption management in WMSNs, for our communication cluster setup. To conserve energy, Camera Nodes (CNs) are not assigned the role of CHs, as the CH role requires more energy for data transmission and aggregation.

3. Multi-hop Hierarchical Routing Setup

In this phase, a hierarchical routing information structure is created to provide better routing information for multi-hop communications. Each Sensor Node (SN) is assigned a routing level, which indicates its proximity to the BS. The routing level starts from 1, with lower numbers signifying closer proximity to the BS. This setup phase occurs only during the first round of the network, following the completion of the communication cluster setup. The algorithm uses CHs to broadcast routing level information to their member nodes, making the process more efficient than broadcasting to the entire network. The details of the multihop:



```
if round == 1 then
    BS broadcasts Level-Msg containing routing_level = 1.
    if SN receives the broadcasting message then
        if Message is Level-Msg then
            if routing_level of node != 0 then
                if routing_level of node > routing_level within received message then
                    Update routing_level of SN
                    if Node is CH then
                        routing_level ← routing_level + 1
                        Broadcast new Level-Msg (ID, routing_level)
                    end if
                end if
            Else
                Update routing_level of SN with routing_level within received message
            end if
        else if Message is Level-Request-Msg then
            if routing_level of node != 0 then
                Send Reply-Request-Msg (ID, routing_level) to the requested SN
            end if
        End
    else
        Waiting for Level-Msg when $T_1$ is expired
        if Node is CH then
            Broadcast Level-Request-Msg (ID)
            Waiting for Level-Reply-Msg when $T_2$ is expired
            if Node receive Level-Reply-Msg then
                if Node is CH then
                    Update routing_level of SN with routing_level within received message
                    routing_level ← routing_level + 1
                    Broadcast new Level-Msg (ID, routing_level)
                end if
            Else
                Waiting for Level-Reply-Msg when $T_3$ is expired
                routing_level ← 100
            end if
        Else
            Waiting for Level-Msg when $T_4$ is expired
            routing_level ← 100
        end if
    end if
end if
```

ALGORITHM 1: Multi-hop hierarchical routing level setup [22].

Detail of the algorithm :

In the first, the routing levels of all SNs in the field are initially set to 0. The BS broadcasts a Level-Msg containing its ID and a routing level of 1 to all SNs within its communication radius. Upon receiving this message, all SNs save the routing level information. If the message is received by a CH, the CH saves the routing level information, increments the routing level by 1, and rebroadcasts it to its neighboring SNs. This process continues until no more CHs receive the broadcasting message. If an SN receives multiple messages, it compares the saved routing information with the new information, keeps the lower routing level, and discards the higher one, ensuring the node's routing level is always set to the lowest value received.

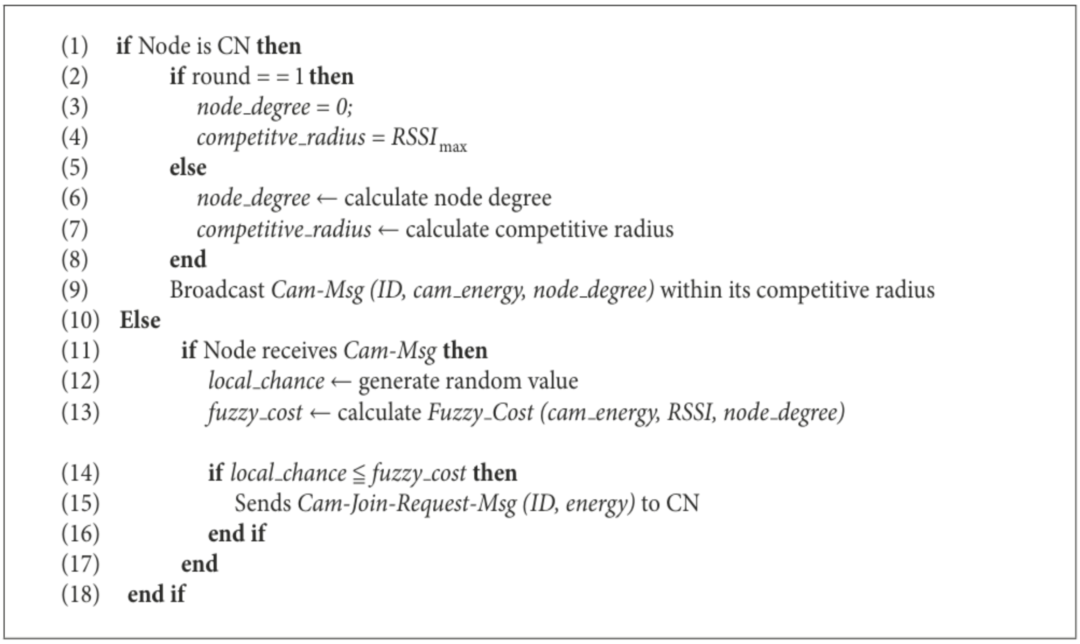
CHs that do not receive any broadcasting messages for a specific time $T1$ are considered far away from other CHs; however, it is possible that their neighbors have the routing level information. Therefore, these CHs broadcast Level Request-Msg to their neighbors to request the routing level information. If there is a neighboring node that receives Level-Request-Msg and its routing level is not equal to 0, it replies with Level-Request-Reply-Msg, which contains its routing level information to the source CH. After receiving the replies from its neighbors, the CH saves the received routing level information for future use.

If more than one reply message is received, the CH chooses the lowest routing level. Then, this CH increases the routing level by 1 and broadcasts it to its neighbors. If the source CH does not receive any replies from its neighbors during a specific time $T2$, it waits for another specific time $T3$. If, in the worst case, there is still no any reply during time $T3$ to the source CH, the source CH sets its routing level to the maximum value(here,100). For the NNs, after waiting for a specific time $T4$ and not receiving any routing level information, these nodes also set their routing levels to the maximum value.

4. Camera cluster setup

The camera cluster consists of a single CN that acts as the CH for the cluster and its nearby member nodes. After setting up the communication clusters and multi-hop hierarchical routing, it seems like we're ready to distribute multimedia data compression tasks. But there's a big problem: we don't know how much energy each SN has left because its changing all the time and how much of SNs we need in the camera cluster. So this phase must be implemented in every round in the network to ensure that the camera cluster has enough member nodes for image processing and transmission.

The detail algorithm of this phase is described in algorithm 2:



```
if Node is CN then
    if round = = 1 then
        node_degree = 0;
        competitve_radius = $RSSI_{max}$
    else
        node_degree ← calculate node degree
        competitive_radius ← calculate competitive radius
    end
    Broadcast Cam-Msg (ID, cam_energy, node_degree) within its competitive radius
Else
    if Node receives Cam-Msg then
        local_chance ← generate random value
        fuzzy_cost ← calculate Fuzzy_Cost (cam_energy, RSSI, node_degree)

        if local_chance ≦ fuzzy_cost then
            Sends Cam-Join-Request-Msg (ID, energy) to CN
        end if
    end
end if
```

ALGORITHM 2: Fuzzy logic-based camera cluster setup [22].

Detail of algorithm:

In Algorithm 2, the Camera Node (CN) broadcasts a message (Cam-Msg) to its neighbors within its competitive radius to form the camera cluster. The competitive radius will be discussed in the next subsection(subsection B). The Cam-Msg includes the CN's ID, remaining energy, and node degree from the previous round. Upon receiving this message, the neighboring nodes calculate their potential membership in the camera cluster. This calculation involves determining an FLS (Fuzzy Logic System) cost, which ranges from 0 to 1. Each neighboring node then generates a random number between 0 and 1, known as the local chance. If the local chance is lower than the FLS cost, the neighboring node sends a Cam-Join-Request-Msg, which includes its ID and remaining energy, to the CN. This process results in the node becoming a member of the camera cluster.

A. Fuzzy Logic System:

The Fuzzy Logic System (FLS) was developed by Zadeh [19] and is a powerful technique for enhancing decision-making in resource-constrained networks like WSNs. FLS can reduce resource consumption while maintaining effective performance and providing good solutions for many control problems by making the human thought process. FLS consists of four key components: the fuzzifier, the inference engine, the rule base, and the defuzzifier. As shown in (Figure 8).

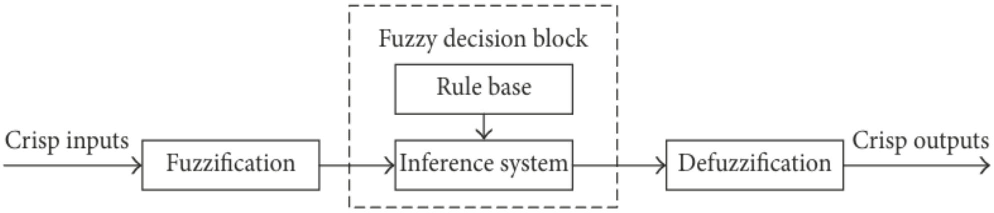


Figure 8:Basic block diagram of a fuzzy system.[20]

- **Fuzzifier**: This component takes crisp inputs (precise values) and converts them into a fuzzy set, which is represented by linguistic terms such as "near," "medium," or "far." This process is known as fuzzification. A membership function is used to quantify these linguistic terms, mapping non-fuzzy input values to fuzzy linguistic terms, and vice versa. Common types of membership functions include triangular, trapezoidal, and Gaussian functions. The choice of membership function depends on the experience and assessment of the researcher.
- **Inference Engine**: After fuzzification, the inference engine processes the fuzzy inputs based on a set of rules stored in the rule base. These rules typically follow an IF-THEN structure with conditions and conclusions.
- **Rule Base**: This component contains the set of rules used by the inference engine to make decisions. Each rule specifies what output should be generated based on given input conditions.
- **Defuzzifier**: The final step in FLS is defuzzification, where the fuzzy output generated by the inference engine is converted back into a crisp value using the membership function. This step translates the fuzzy conclusions into a specific action or decision.

Example :

1. **Input (Crisp Input) :**
    - The current remaining energy of the SN is 70%.
    - The distance to the CN is 30 meters.
2. **Fuzzification :**
    - The fuzzifier converts 70% remaining energy and 30 meters distance into fuzzy values.
    - Assume the membership functions are defined as follows:
        - Energy : "Low" (0-30%), "Medium" (20-80%), "High" (70-100%)
        - Distance: "Near" (0-20 meters), "Medium" (10-50 meters), "Far" (40-100 meters)
    - For 70% Energy :
        - Membership in "Low": 0 (completely outside the range)
        - Membership in "Medium": 0.5 (partially within the range)
        - Membership in "High": 0.8 (mostly within the range)
    - For 30 meters distance :
        - Membership in "Near": 0 (completely outside the range)
        - Membership in "Medium": 0.7 (partially within the range)
        - Membership in "Far": 0.2 (barely within the range)
3. **Inference :**
    - Apply the fuzzy rules to the fuzzy input values.
    - Sample rules might include :
        - IF "Energy" is "High" AND "Distance" is "Near" THEN "Join Probability" is "High."

  - IF "Energy" is "Medium" AND "Distance" is "Medium" THEN "Join Probability" is "Medium."
  - IF "Energy" is "Low" AND "Distance" is "Far" THEN "Join Probability" is "Low."
- With the input memberships :
  - "High" energy (0.8) AND "Medium" distance (0.7) → Medium Join Probability
  - "Medium" energy (0.5) AND "Medium" distance (0.7) → Medium Join Probability
  - "Medium" energy (0.5) AND "Far" distance (0.2) →Low Join Probability

4. Aggregation :

- Combine the results of the fuzzy rules to form a fuzzy output.
- For simplicity, assume we average the results:
  - Join Probability = (Medium * 0.8 * 0.7 + Medium * 0.5 * 0.7 + Low * 0.5 * 0.2) / (0.8 * 0.7 + 0.5 * 0.7 + 0.5 * 0.2)
  - This simplifies to a combined fuzzy output for "Join Probability."

5. Defuzzification :

- Convert the fuzzy output back into a crisp value.
- Assume "Medium" Join Probability corresponds to a value of 0.5 and "Low" to 0.3:
  - Join Probability = (0.5 * 0.56 + 0.3 * 0.1) / (0.56 + 0.1) = 0.43

6. Output (Crisp Output):

- The sensor node has a join probability of 0.43, meaning it has a moderate chance of joining the camera cluster based on its energy and distance.

B. Competitive radius:

The competitive radius is crucial for the camera cluster because it determines which SNs can become members of the camera cluster. It defines the maximum distance within which a CN considers other SNs as potential members of its cluster. If the competitive radius is too large, the CN will need more energy to communicate with its node members. On the other hand, if the radius is too small, there might not be enough SNs to join the camera cluster, which would affect image processing and transmission to the BS. The competitive camera radius (R_{CN}) is calculated based on the Received Signal Strength Indicator (RSSI) using the following formula :

$$. R_{CN} = \propto RSSI_{max} \quad ,$$

Where $RSSI_{max}$ denotes the maximum RSSI of the SNs, which represents the maximum distance at which SNs can communicate effectively. and $\propto$ is the weight of the camera radius, which is used to adjust the camera radius. This weight of the camera radius $\propto$ ranges between 0 and 1. In the first round $\propto$ is equal to 1.

In subsequent rounds, α is adjusted based on various factors using a FLS, which takes into account:

- **CN Energy**: Remaining energy of the CN.

$$. CN_{energy} = \frac{CN_{remaining\_energy}}{CN_{init\_energy}}$$

Example: If a CN starts with 100 units of energy and has 80 units remaining, the CN Energy is 80/100=0.8 or 80%

- **Camera Node Degree (proximity to other CN)** : Number of neighboring CNs within the competitive radius.

$$. CAM_{deg} = \frac{CAM_{NB}}{\#CNs}$$

This is the number of other CNs within the competitive radius of a given CN, divided by the total number of CNs in the network.

Example: If a CN has 2 other CNs within its radius and there are 10 CNs in total, the Camera Node Degree is 2/10 = 0.2

- **Node Degree**: Number of SNs in the cluster from the previous round.

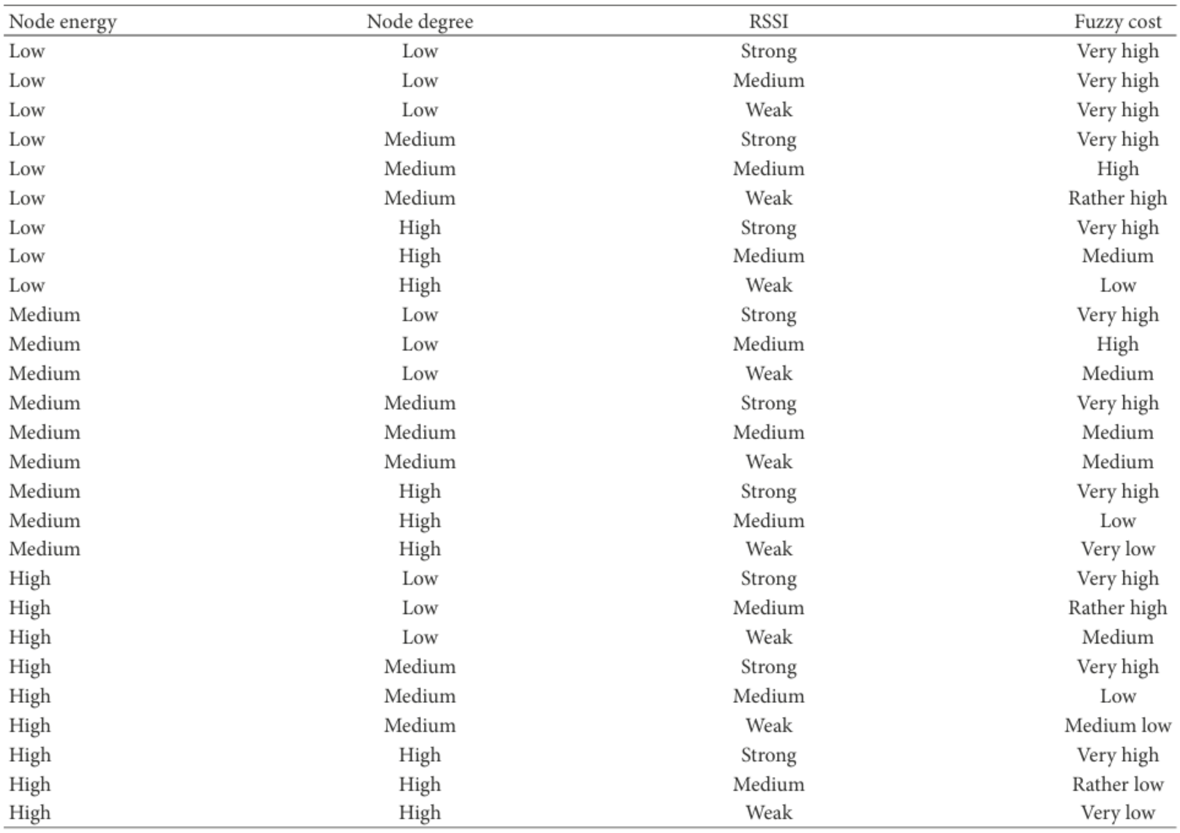
  . $node_{deg} = \frac{deg(CN)}{\#SNs}$

  This is the number of SNs in the previous round devided by the total number of SNs in the network.

C. Camera cluster membership:

The camera cluster membership algorithm employs FLS to determine the size of the camera cluster. Not all SNs within the competitive radius can be members of the camera cluster, only the qualified ones are selected to join this cluster. As shown in table 3 Three fuzzy logic input variables are utilized for camera member node selection, each with its own membership function. The first and second variables, camera node energy and node degree. Thus, we focus on the third variable, RSSI, which measures the power of the radio signal from the where a higher RSSI indicates a stronger degree of membership possibility. Weak, medium, and strong represent the linguistic input variables for the fuzzy set. A triangular membership function is applied to weak and strong, while a trapezoidal membership function is employed for medium [21].

Table 3:fuzzy rules for camera cluster membership [22]



Node energy	Node degree	RSSI	Fuzzy cost
Low	Low	Strong	Very high
Low	Low	Medium	Very high
Low	Low	Weak	Very high
Low	Medium	Strong	Very high
Low	Medium	Medium	High
Low	Medium	Weak	Rather high
Low	High	Strong	Very high
Low	High	Medium	Medium
Low	High	Weak	Low
Medium	Low	Strong	Very high
Medium	Low	Medium	High
Medium	Low	Weak	Medium
Medium	Medium	Strong	Very high
Medium	Medium	Medium	Medium
Medium	Medium	Weak	Medium
Medium	High	Strong	Very high
Medium	High	Medium	Low
Medium	High	Weak	Very low
High	Low	Strong	Very high
High	Low	Medium	Rather high
High	Low	Weak	Medium
High	Medium	Strong	Very high
High	Medium	Medium	Low
High	Medium	Weak	Medium low
High	High	Strong	Very high
High	High	Medium	Rather low
High	High	Weak	Very low

Based on three fuzzy input variables, outlined in Table 1, the output, illustrated in (Figure 9), represents the fuzzy cost. This fuzzy cost determines the possibility that nodes can be a member of camera cluster.

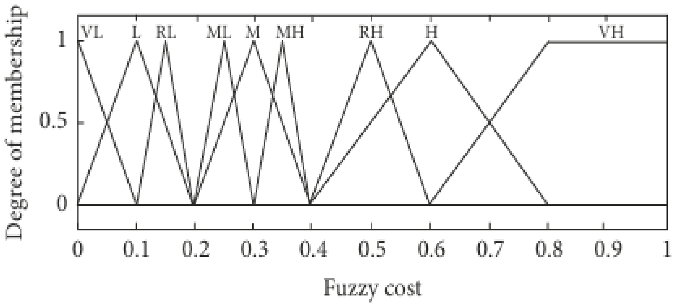


Figure 9:Fuzzy output variable.

5. Image compression cluster setup

It is important to choose the best cluster who does the image compression tasks because any errors in compression will affect the whole network, so the image compression cluster is formed to prepare SNs for distributed image compression. So this cluster should have the must performed SNs for best compression.

In this setup phase, the CN initially selects the highest-energy SN in the camera cluster. This selected SN becomes the image compression cluster head (ICH). One ICH is programmed for DWT tasks (DWT-ICH) and the other for DCT tasks (DCT-ICH). As previously mentioned, we should choose the best SN, so not all the neighbors of the ICH become members of the cluster. Instead, the ICH selects a set of SNs that have the highest energies to form its image compression cluster and participate in image compression. The number of SNs depends on the compression target. However, the ICH typically chooses the two highest-energy SNs within its communication radius to be members of the image compression cluster.

6. image transmission

After we are created and prepared the best clusters, now we are ready to transmit the image, but in this phase we should ensure that the image has been captured, compressed, and sent hop by hop to the BS successfully. So in this phase there are two main subphases:

Distributed image compression and relay node selection.

A. Distributed Image Compression:

As we discussed in Chapter 1, for WMSN, it is crucial to use a compression technique that requires minimal energy and provides excellent compression quality. Additionally, we need a method that can efficiently distribute the compression tasks among the sensor nodes (SNs). After careful consideration, we have selected a hybrid compression technique that first applies JPEG2000 and then uses JPEG for the rest of the process.

Traditional image compression techniques, such as DCT and DWT, each have their strengths and weaknesses. The DCT is renowned for its energy compaction capabilities and lower computational complexity but can introduce blocking artifacts and limited bit depth at high compression ratios. On the other hand, DWT excels as a multi-resolution compression method, capable of representing an image at various resolutions by discarding detail coefficients and retaining only the approximation coefficients. Despite this advantage, DWT's energy compaction is less efficient than DCT and requires more computational power.

To address these limitations, a hybrid DCT-DWT compression algorithm has been proposed, combining the strengths of both techniques. Given that DCT is applied to small 8x8 pixel blocks, it can become inefficient for large images with substantial data. In contrast, DWT can handle larger bit depths, such as 14 or 16 bits, making it more suitable for compressing large images efficiently. Therefore, the hybrid method first applies a multi-level DWT to the image, decomposing it into sub-bands. This process effectively reduces the number of pixels in the blocks of the image, making it suitable for the next compression technique. Subsequently, a 2D DCT is applied to the low-frequency sub-bands, which contain most of the important information. This combination ensures that high-frequency domains are effectively managed, providing a robust solution for image compression in WMSNs. Following these transformations, quantization and encoding are performed to complete the compression process, ensuring efficient data transmission and storage while preserving critical image details.

This method is highly adaptable for distributing compression tasks. By assigning each subband to a separate SN for processing, we can significantly reduce the computational complexity for each node. Processing a single subband is quicker and requires less computational power, thereby minimizing the energy consumption. Additionally, this approach balances the workload across the network, preventing any single node from becoming a bottleneck and enhancing the overall efficiency and longevity of the sensor network. The decentralized nature of this method also improves fault tolerance, as the failure of one node has minimal impact on the overall compression process.

- **Distributed DCT-DWT Compression:**

As shown in (Figure 10), before transmitting each small image, the CN forms an image compression cluster that performs all the processes of our distributed image compression.

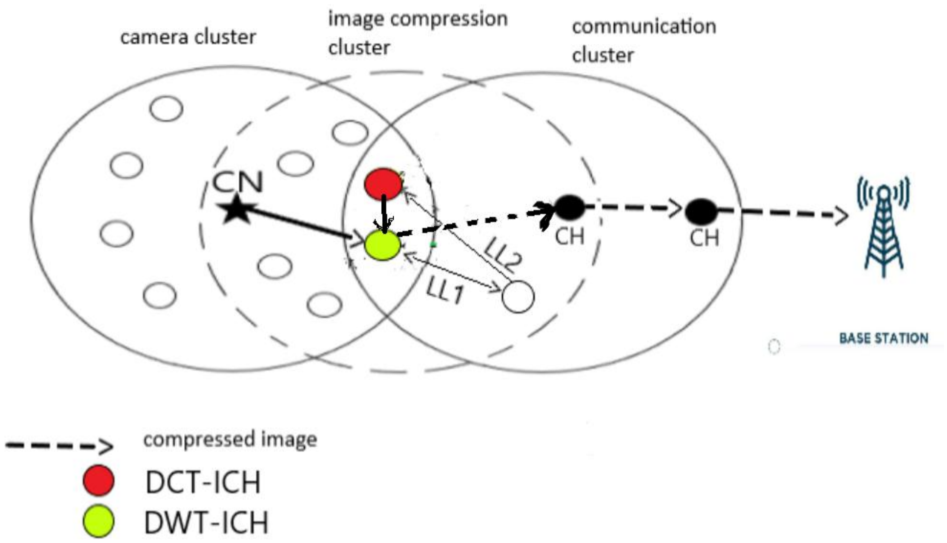


Figure 10:Distributed hybrid image compression.

 Detailed process:

- **DWT Transform:**
  - The cluster setup is completed, and the CN is ready to send data.
  - The CN sends each small image to the DWT-ICH.
  - The DWT-ICH applies the first level of DWT to the received image, and this results in four subbands : LL1,HL1,LH1 ,LL1
  - The DWT-ICH encodes the subbands HL1,LH1,LL1 , and then send the compressed data to its communication CH, whereas the remaining LL1 is sent to the nearest SN in the image compression cluster for further process.
  - The nearest SN received the LL1 subband and then applied another level of DWT , and encodes the new HL2,LH2,LL2. Then sends these encoded subbands back to its DWT-ICH, and the new LL is sents to the DCT-ICH.
- **DCT Transform:**

DCT is the second technique used in our architecture. After the LL2 subband is obtained, the next step is to apply the DCT to this LL2 subband, as shown in (Figure 11). In this step, the LL2 subband is sent to the DCT-ICH, which specializes in DCT compression. The DCT-ICH then starts the compression process:

- The DCT-ICH receives the LL2 subband from the nearest SN.
- The DCT-ICH applies the 2D DCT to the LL2 subband, transforming it into frequency coefficients.
- The DCT-ICH quantizes the DCT coefficients, reducing the number of bits needed to represent the data while maintaining a balance between compression efficiency and image quality.
- The quantized DCT coefficients are then sent to the nearest SN.
- The nearest SN encodes the quantized coefficients.
- The encoded data is then sent back to the DCT-ICH to complete all the compression steps.
- The DCT-ICH sends the compressed data back to the DWT-ICH, where all components of the image compression are aggregated.
- The compressed data is finally sent back to the communication CH for transmission or storage.

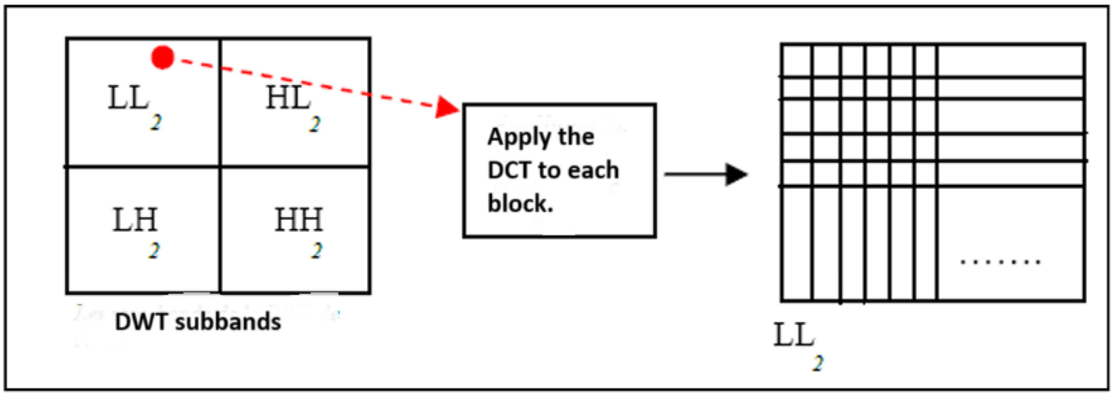


Figure 11:DCT applied to each n x n block of the LL2 subband

Given the trade-off between compression and transmission complexity, to manage the energy consumption of the SNs involved in compression tasks before they proceed with their regular tasks, each SN needs to perform the following preprocessing calculations:

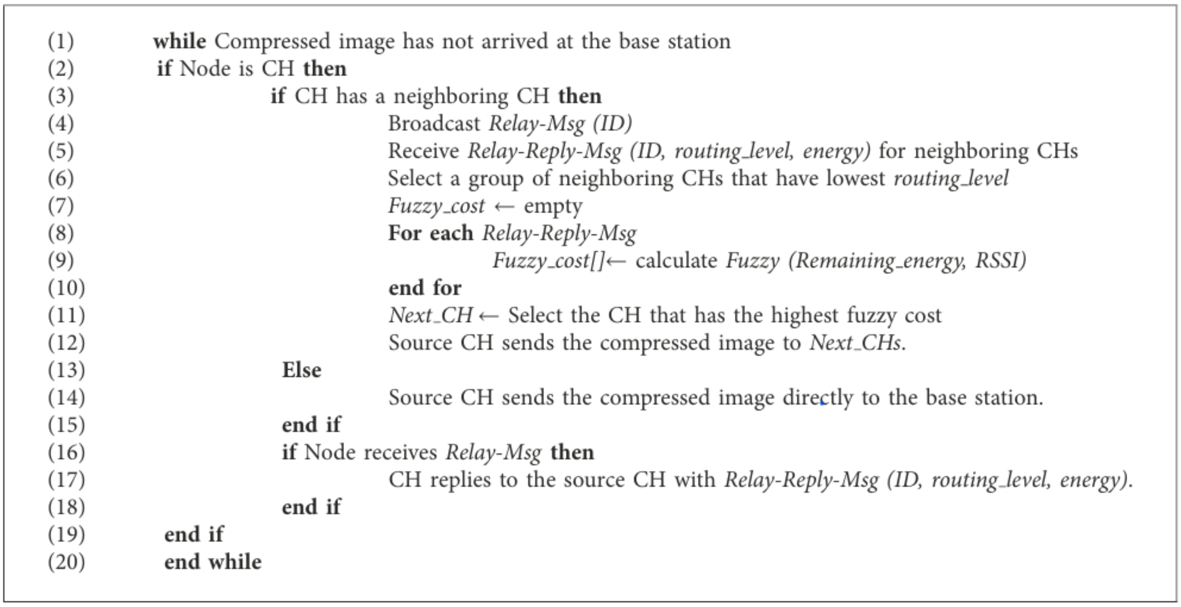
- If the node is the ICH (DCT-ICH or DWT-ICH) , it is required to calculate the energy consumption of implementing an entire compression task and forwarding the compressed image to its CH.
- If the node is a member node in the image compression cluster and is selected to participate in image compression tasks, it needs to calculate the energy consumption of processing all remaining image compression tasks and forwarding the compressed data to the ICH.

The SNs perform a low-energy calculation to estimate the energy needed for image compression and forwarding. They then compare this with the energy required for their regular tasks (like continuous sensing and communication). They choose the task that consumes less energy overall.

B. Relay node selection:

After achieving the desired compressed image, the DCT-ICH sends all details to its communication Cluster Head CH . The data is then transmitted hop by hop to the next CHs until it arrives at the Base Station (BS). This phase explains how data is transmitted using a relay node selection algorithm. In this study, the relay node selection algorithm is used to choose the next CHs to forward the compressed image step by step to the BS.

In algorithm 4 we describe the relay node selection phase:



```
while Compressed image has not arrived at the base station
if Node is CH then
        if CH has a neighboring CH then
                Broadcast Relay-Msg (ID)
                Receive Relay-Reply-Msg (ID, routing_level, energy) for neighboring CHs
                Select a group of neighboring CHs that have lowest routing_level
                Fuzzy_cost ← empty
                For each Relay-Reply-Msg
                        Fuzzy_cost[]← calculate Fuzzy (Remaining_energy, RSSI)
                end for
                Next_CH ← Select the CH that has the highest fuzzy cost
                Source CH sends the compressed image to Next_CHs.
        Else
                Source CH sends the compressed image directly to the base station.
        end if
        if Node receives Relay-Msg then
                CH replies to the source CH with Relay-Reply-Msg (ID, routing_level, energy).
        end if
end if
end while
```

Algorithm 4: Relay Node selection [22].

a) **Broadcasting Relay Message**: The source Cluster Head (CH) broadcasts a Relay-Msg, which contains its ID, to its neighboring CHs within its communication radius to find a relay CH.
b) **Receiving Relay Messages**: The neighboring CHs receive the broadcasted Relay-Msg.
c) **Replying to Source CH**: Each neighboring CH replies to the source CH by sending a Relay-Reply Msg directly, which contains their IDs, remaining energies, and routing levels.
d) **Selecting Suitable Neighboring CHs**: The source CH reviews the reply messages from the neighboring CHs. Then, It selects the group of neighboring CHs that have a lower routing level than itself. If no neighboring CHs have a lower routing level, the source CH selects those with the same routing level.
e) **Calculating Fuzzy Cost**: The source CH calculates the fuzzy cost for the selected neighboring CHs using FLS.
f) **Choosing the Relay CH**: The neighboring CH with the highest fuzzy cost is chosen as the relay CH to forward the compressed image to the Base Station (BS).
g) This process is repeated until the compressed image reaches the BS.

Ⅳ. Conclusion

In this chapter, we've delved into the practical aspects of WMSNs, focusing on energy efficiency and multimedia task optimization through image compression. We've explored the fundamental network setup assumptions, routing protocols, and energy consumption models. Specifically, we've discussed the significance of routing protocols LEACH in guiding effective communication while conserving energy.

Moving on, we introduced our proposed distributed image compression architecture for multi-hop WMSNs, outlining its phases from communication cluster setup to image transmission. By employing techniques like fuzzy logic-based camera cluster setup and distributed JPEG-JPEG2000 compression, we aim to reduce energy consumption, enhance computational efficiency, and ensure reliable image transmission.

Finally, we addressed relay node selection for transmitting compressed images to the Base Station, underscoring the importance of efficient data relay to optimize network performance. Through these strategies, we strive to bridge the gap between theoretical concepts and real-world applications, paving the way for more robust and energy-efficient WMSNs.

Chaptre Ⅲ

Results of simulation and Interpitation

I. Introduction

In the previous chapters, we explored the workings of WMSNs, highlighting their capabilities and the significant challenges they face, particularly regarding energy consumption and the overall lifetime of the network. We proposed a novel architecture designed to mitigate these issues by optimizing energy usage and extending the network's lifetime. A key aspect of our proposed architecture involves the strategic distribution of compression tasks among SNs to enhance efficiency.

In this chapter, we will delve into the implementation and experimentation of our proposed architecture. We will simulate the network to demonstrate how it can be practically implemented and evaluate its performance against traditional WMSN setups.

II. Implementation Tools Used

In this section, we will detail the tools used in our work. These tools were crucial for the successful implementation and evaluation of our proposed techniques and methodologies. For the realization of a wireless network, we have two main options:

- **Use a programming language with rich graphical capabilities:** This option allows for the development of visually intuitive interfaces and comprehensive visual representations of network behaviors and performance metrics.
- **Use a language dedicated to scientific calculation and mathematical equations**: These languages offer advanced interpreters and are capable of handling complex mathematical computations efficiently. They support the addition of custom functions, organized into toolboxes, which extend their application to various domains such as control systems, optimization, image processing, and signal processing.

Given these choices, we opted to use both options. MATLAB was chosen for realizing our network architecture and simulating the behavior and communication protocols of sensors. MATLAB's robust capabilities in scientific computing were also utilized for image compression, playing a crucial role in analyzing our proposed system architecture. Concurrently, we employed OpenCV, an open-source computer vision and machine learning library primarily utilized with Python. OpenCV proved invaluable for simulating camera node operations, enabling tasks such as object detection and tracking within our network. This dual approach allowed us to enhance data compression efficiency, refine object tracking algorithms, and conduct real-time video analysis, thereby optimizing network performance and prolonging its operational longevity. The cross-platform and multi-language support of OpenCV further enhanced its integration into our WMSN framework.

1. Hardware

For the implementation of our tests and the development of our network architecture, we utilized a DELL laptop with the following specifications:

- Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
- Windows 11 Professional
- 8,00 GB of RAM
- 64-bits operating system

2. Software :

A. MATLAB :

Today, many programming software options facilitate easy development under Windows. We chose the programming language "MATLAB R2023b 23.2 (Windows 64 bits)." MATLAB provides an interactive environment for algorithm development and data analysis. Compared to traditional programming languages such as C/C++, Java, Pascal, or Fortran, MATLAB significantly reduces computation time for typical tasks and greatly simplifies the development of complex algorithms.

MATLAB is the cornerstone of the MathWorks product family. It can solve a wide range of scientific problems, including control system development, communication system design, signal and image processing, financial modeling, and computational biology. It is a highly effective tool widely used for numerical computation and graphical visualization.

In our case, MATLAB was used to simulate our WMSN. With its advanced features, we were able to model and analyze the performance of our network architecture, focusing on data compression and the distribution of compression tasks. MATLAB allowed us to test different configurations and visualize the results interactively, facilitating the optimization of energy consumption and the improvement of network lifetime.

- **MATLAB Environement:**

In this environment, variables and scalars are manipulated as matrices with "n" columns by "m" rows. For instance, a scalar would be represented as a 1 x 1 matrix. During execution, MATLAB displays several windows on the screen. The three most important types of windows are:

- **Command Window:** This is where commands and scripts are executed and where MATLAB provides textual output (Figure 12).

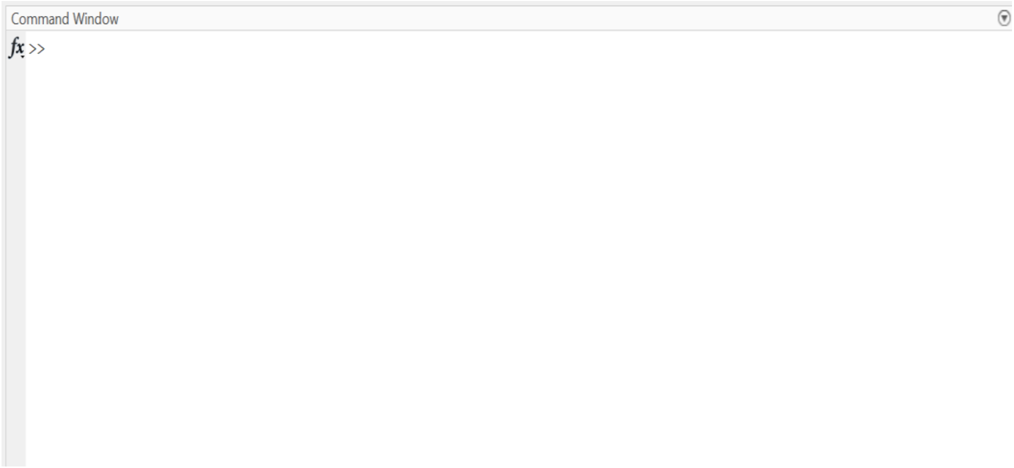


Figure 12 :command window.

A window called "Command Window" appears on the screen. The user can enter multiple commands or mathematical equations after the ">>" prompt that appears on the left side of the window. To execute an operation, the "Enter" key on the keyboard must be pressed. Additionally, each operation should be terminated with a semicolon ";" otherwise, all steps of the calculation will be displayed on the screen.

- **Edit Windows:** This is where the user can modify or create MATLAB programs ("M-files") (Figure 13).

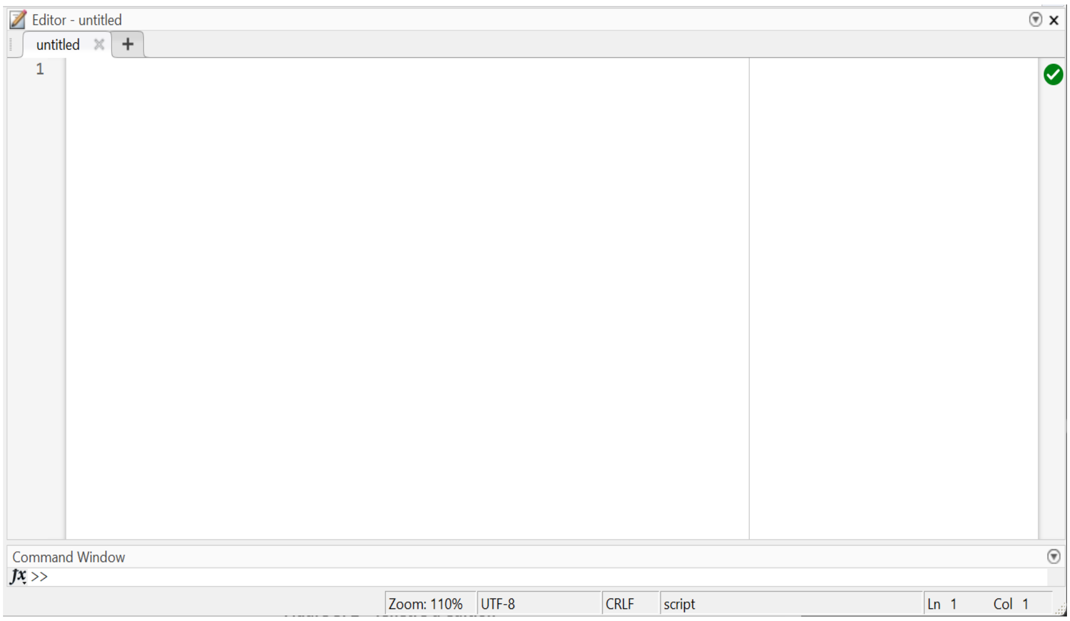
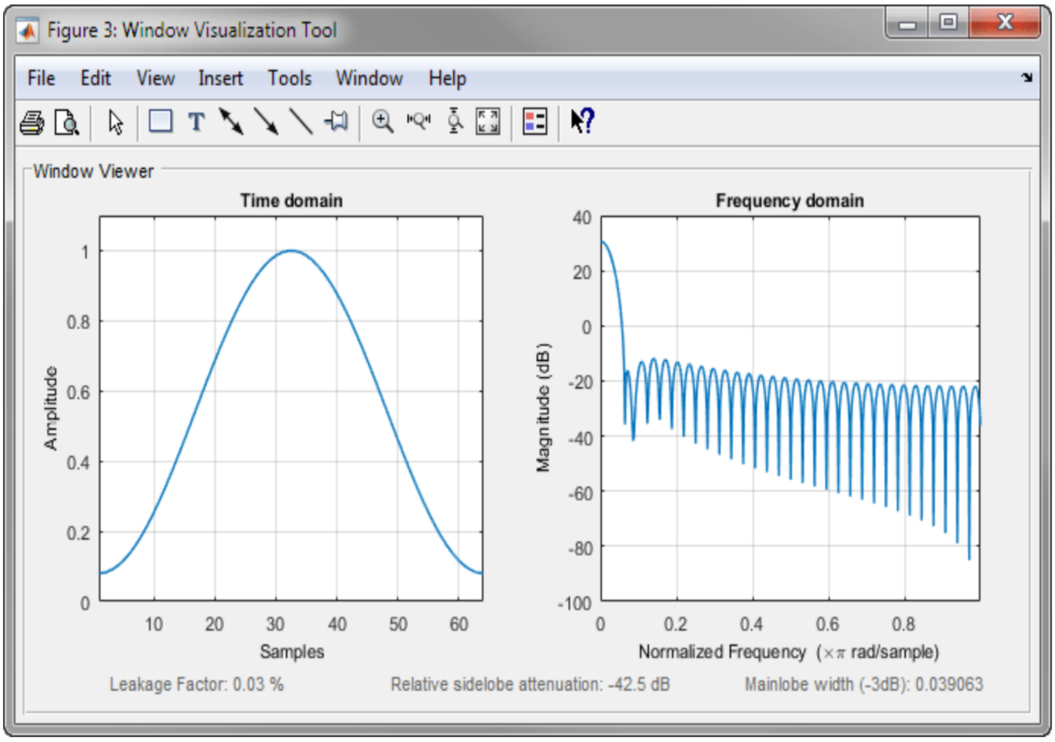


Figure 13 :edit window

Instead of typing commands individually and directly into the Command Window, it is possible to create a file called an "m-file" that contains all the necessary functions and commands. This file can be quickly executed by typing its name into the Command Window. These files are called "script files" and end with the ".m" extension. The Edit Window is used to create or modify m-files. To create a new file, one must go to the selection menu (note that m-files are executed by typing the file name in the Command Window).

The power of the software can be realized by running the demo command. Additionally, toolboxes are available in many domains (signal processing, image processing, optimization, control, etc.). MATLAB expands on the library of mathematical functions, the graphical environment, and the development interface.

- **Figure Windows:** These windows display graphical output such as plots, charts, and images generated by MATLAB commands (Figure 14).



Leakage Factor: 0.03 %

Relative sidelobe attenuation: -42.5 dB

Mainlobe width (-3dB): 0.039063

Figure 14:figure window.

- **Simulink :**

Simulink is a versatile block diagram environment within MATLAB that enables the design, simulation, and deployment of multidomain systems. Users can create complex models visually by connecting blocks, representing various components and functions. Simulink allows for thorough system testing and validation in a virtual environment before moving to hardware, reducing the risk of errors and iterations. It also supports automatic code generation for seamless deployment to hardware platforms, eliminating the need for manual coding. Additionally, Simulink facilitates real-time simulation and hardware-in-the-loop testing, ensuring reliable performance in real-world conditions, thus streamlining the entire development process from design to implementation see (Figure 15).

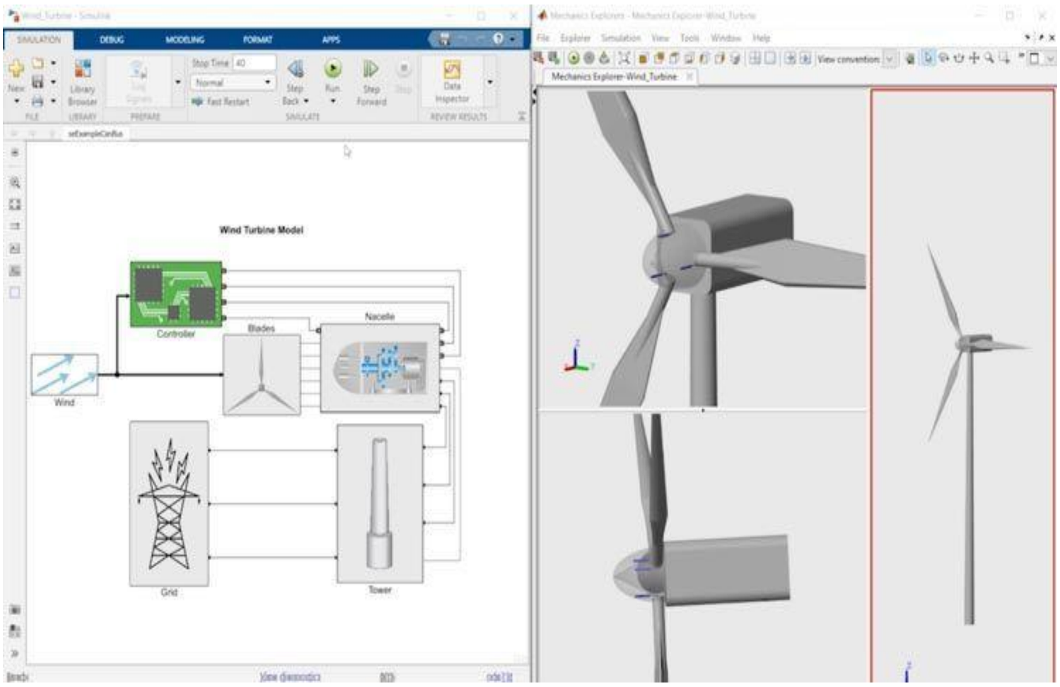


Figure 15 :Simulink

B. OpenCV :

OpenCV (Open Source Computer Vision Library) is a powerful open-source software library designed for computer vision and machine learning applications. Initially developed by Intel, OpenCV has grown into a widely adopted framework with a vast community and extensive industry usage. It provides over 2500 optimized algorithms that encompass a wide range of functionalities, from basic image processing to advanced machine learning techniques. These algorithms enable tasks such as object detection, recognition, tracking, and image enhancement. OpenCV supports multiple programming languages including C++, Python, Java, and MATLAB, making it accessible across various platforms including Windows, Linux, Android, and macOS. Its flexibility and efficiency make it suitable for applications in diverse fields such as robotics, augmented reality, medical imaging, surveillance, and automotive safety systems. OpenCV's continuous development and support for cutting-edge technologies like CUDA and OpenCL ensure its relevance in modern computer vision research and industrial applications.

In our case, OpenCV plays a crucial role in simulating and analyzing the behavior of camera nodes within our WMSN. By leveraging OpenCV's rich set of algorithms, we can simulate tasks such as object detection, tracking, and real-time video analysis. This capability allows us to enhance data processing efficiency, refine algorithms for object recognition and tracking, and optimize the performance of our network.

- **OpenCV setup:**

Setting up OpenCV can vary slightly depending on the operating system and the programming language you intend to use. Below are the steps to set up OpenCV for Python on Windows:

- Install Python :

1. Download and install Python from the official Python website (https://www.python.org/downloads/).
2. Ensure Python is added to your system PATH during installation.

- Install OpenCV :

3. Open Command Prompt (cmd).
4. Run the following command to install OpenCV using pip:

“ Pip install opencv-python “

“ pip install opencv-python-headless “ (if you don't need GUI features)

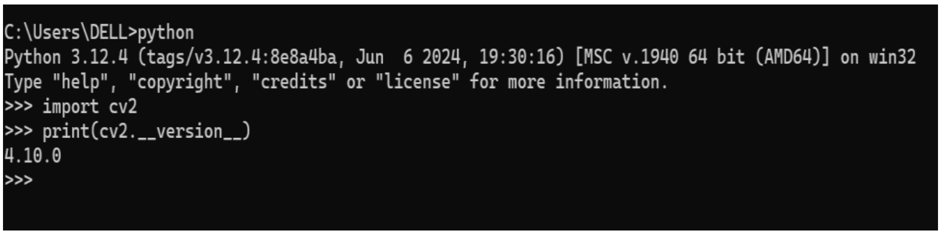
- Verify Installation :

5. Open Python from the Command Prompt by typing python.

Import OpenCV to check the installation :

“ import cv2 “

“ print(cv2.__version__) “



```
C:\Users\DELL>python
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print(cv2.__version__)
4.10.0
>>>
```

Ⅲ. Simulation

1. Simulate camera node operation with OpenCV

The simulation of camera node operations is a crucial aspect of our project, utilizing OpenCV to replicate and analyze real-world scenarios. OpenCV's powerful computer vision and machine learning capabilities allow us to accurately simulate various applications of camera nodes. For example, we can count how many people are in a certain area, check how many objects pass on a conveyor belt, or count vehicles on a highway.

We take the example of tracking vehicles on a highway and accurately counting their numbers :

in this example we use three file :

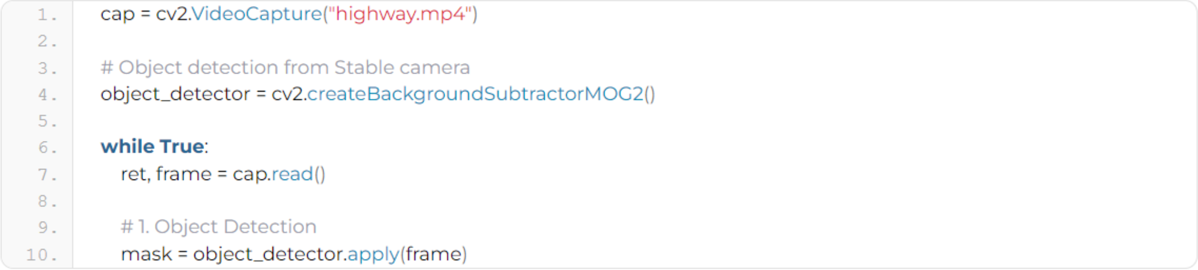
1 The video of the highway we will use to count the vehicles tracker files
2 This has already been written and you can simply download it main file.
3 Write me in real-time and we will proceed step by step with the integration of the libraries.

First of all it must be clear that what is the difference between object detection and object tracking:

- **Object detection :**

Is the detection on every single frame and frame after frame.

First we need to call the **highway.mp4** file and create a mask.



```
cap = cv2.VideoCapture("highway.mp4")

# Object detection from Stable camera
object_detector = cv2.createBackgroundSubtractorMOG2()

while True:
    ret, frame = cap.read()

    # 1. Object Detection
    mask = object_detector.apply(frame)
```

As you can see in the example code we also used the “createBackgroundSubtractorMO2” function which Returns the “background ratio” parameter of the algorithm and then create the mask (Figure 16).

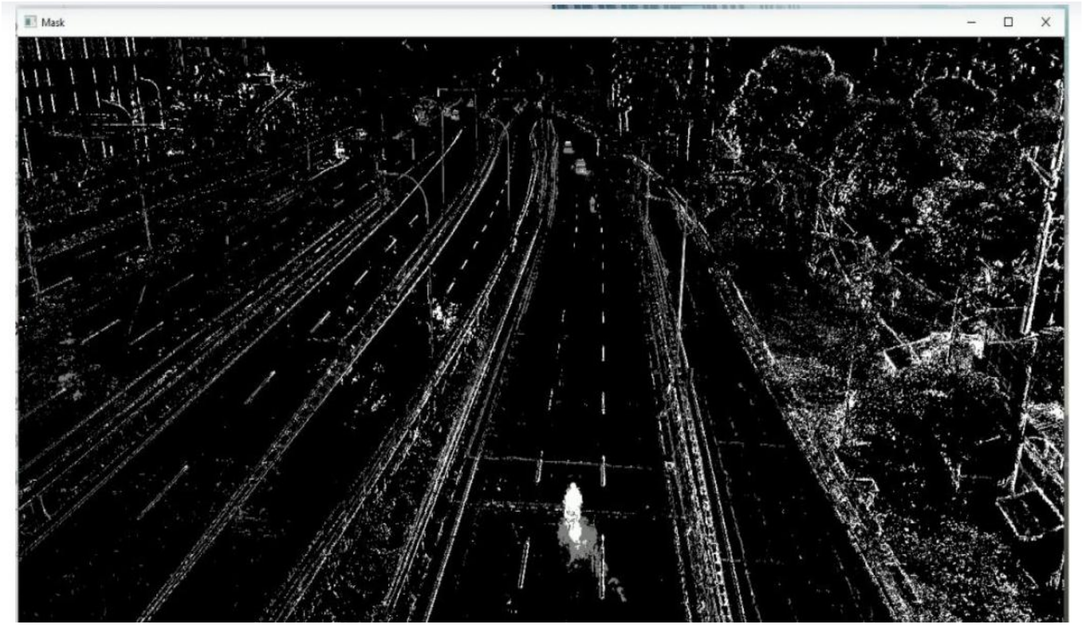
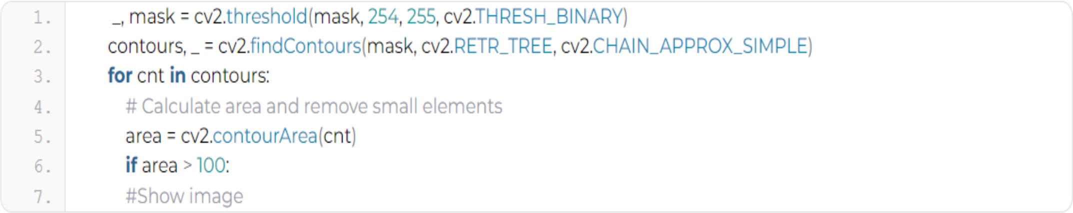


Figure 16:mask.

As you can see, however, there is a lot of noise in the image. So let’s improve the extraction by removing all the smaller elements and focus our attention on objects that are larger than a certain area.



```
 _, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
   # Calculate area and remove small elements
   area = cv2.contourArea(cnt)
   if area > 100:
   #Show image
```

Drawing the contours with OpenCV’s **cv2.drawContours** function we obtain this result. You won’t need to use this function (Figure 17), consider it as a debug of a first result .

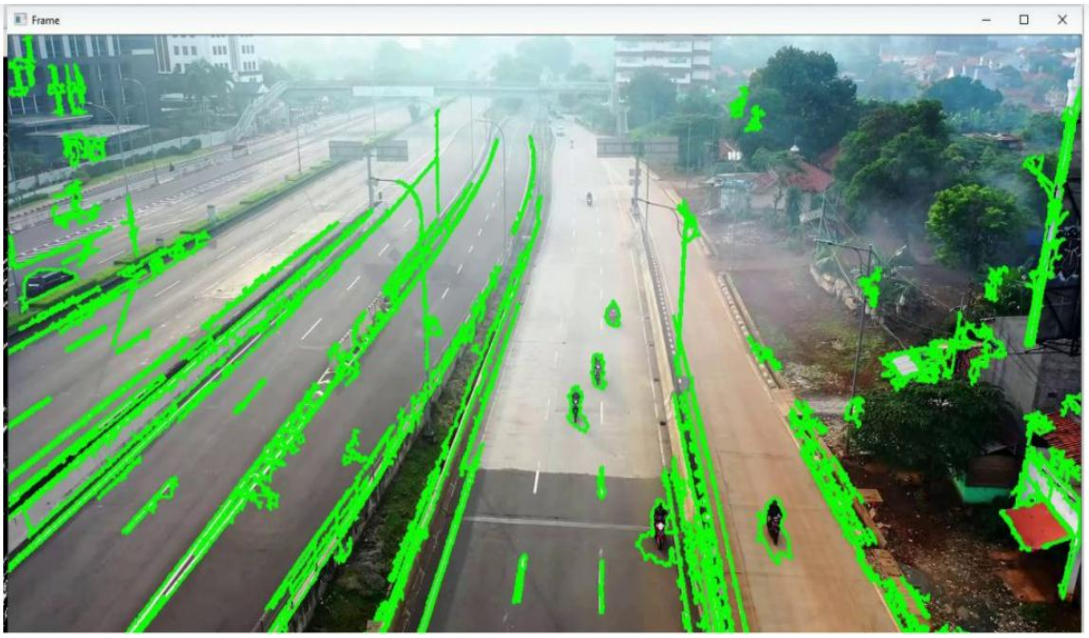
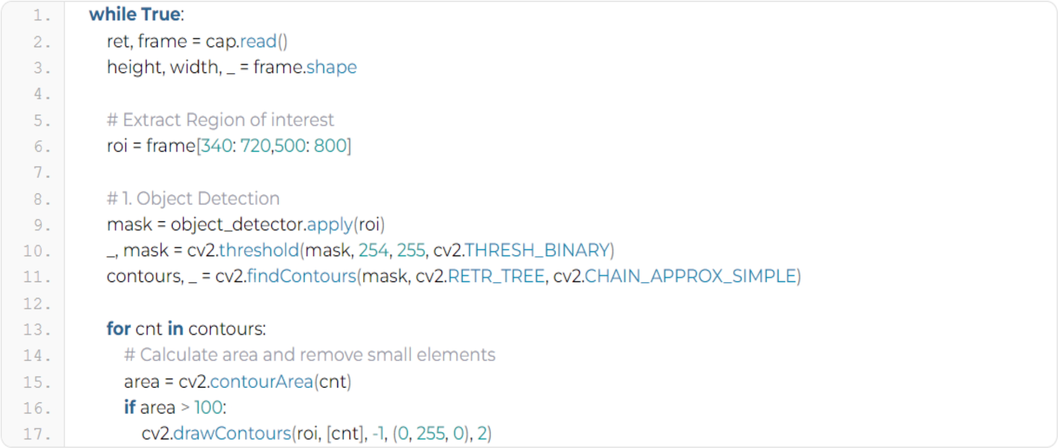


Figure 17:Drawing the contours

- **We Define A Region Of Interest:**

it is not important to analyze the entire window. We are only interested in counting all the vehicles that pass at a certain point, for this reason, we must define a region of interest ROI and apply the mask only in this area.



```
while True:
    ret, frame = cap.read()
    height, width, _ = frame.shape

    # Extract Region of interest
    roi = frame[340: 720,500: 800]

    # 1. Object Detection
    mask = object_detector.apply(roi)
    _, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        # Calculate area and remove small elements
        area = cv2.contourArea(cnt)
        if area > 100:
            cv2.drawContours(roi, [cnt], -1, (0, 255, 0), 2)
```

Already in the image you can see a good first result see (Figure 18).

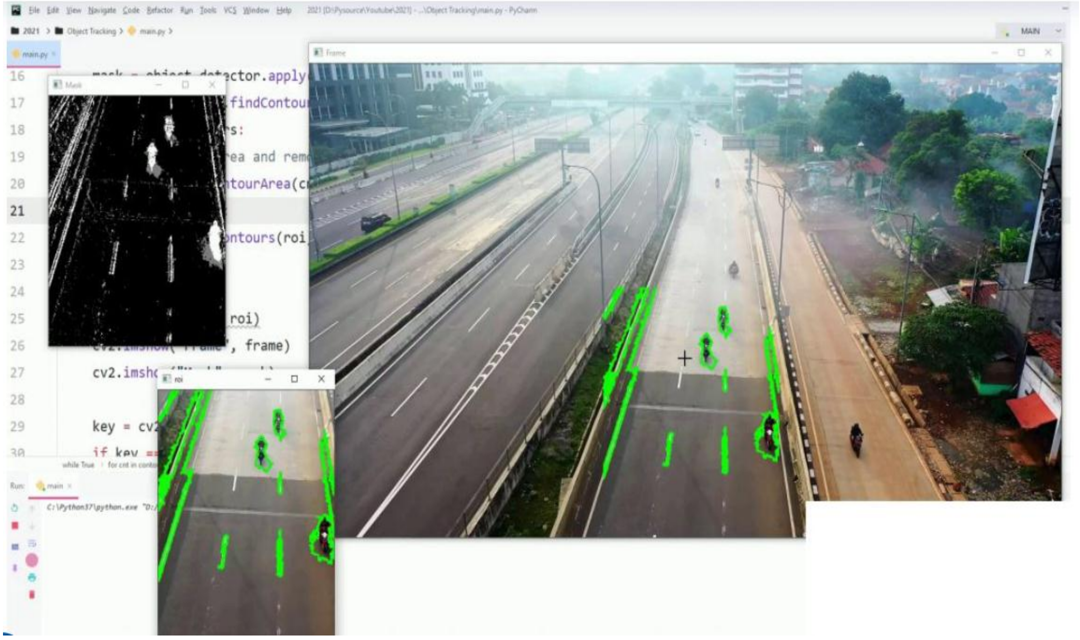
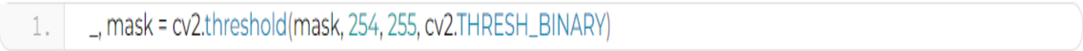


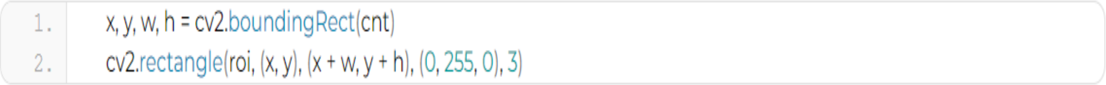
Figure 18;first result.

- **Draw The Box Around The Object:**



```
_, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)
```

We then insert the coordinates of the found object into the if condition and draw the rectangle see (Figure 19):



```
x, y, w, h = cv2.boundingRect(cnt)
cv2.rectangle(roi, (x, y), (x + w, y + h), (0, 255, 0), 3)
```

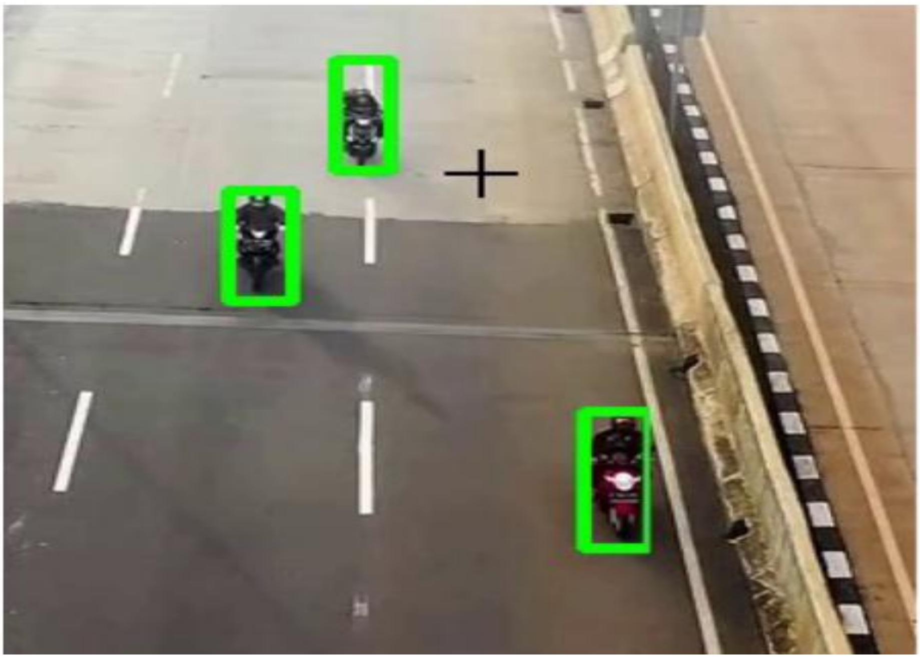
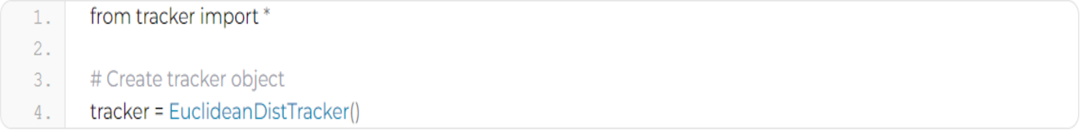


Figure 19:Box Around The Object.

- **Object tracking :**

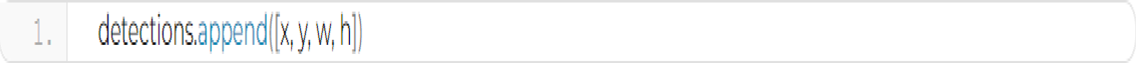
Is to does frame-by-frame tracking but keeps the history of where the object is at a time after time.

We now simply have to import and integrate the tracking functions :



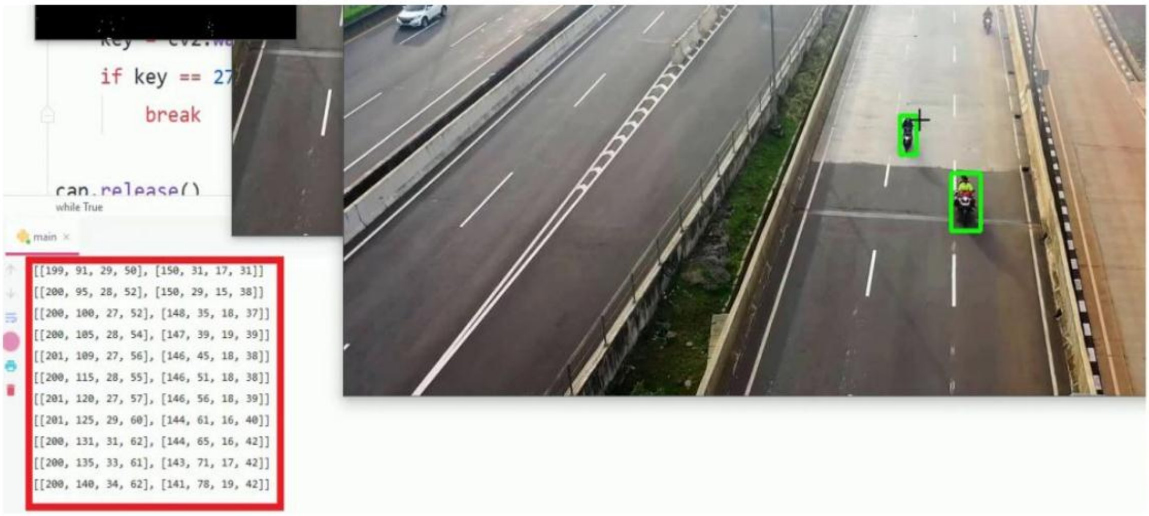
```
from tracker import *

# Create tracker object
tracker = EuclideanDistTracker()
```

Once the object has been created, we must therefore take each position of the bounding box and insert them in a single array.



```
detections.append([x, y, w, h])
```

By showing the result on the screen you can see how all the lanes that pass through our ROI are identified and their positions inserted in a specific array. Obviously, the more motorcycles identified the larger our array will be.



[redacted]

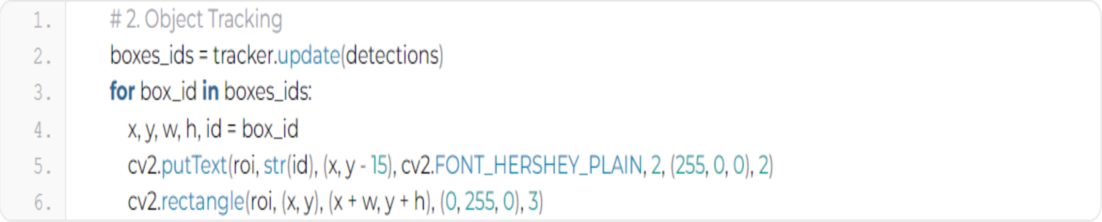
```
    if key == 27
        break

can.release()
```

```
[[199, 91, 29, 50], [150, 31, 17, 31]]
[[200, 95, 28, 52], [150, 29, 15, 38]]
[[200, 100, 27, 52], [148, 35, 18, 37]]
[[200, 105, 28, 54], [147, 39, 19, 39]]
[[201, 109, 27, 56], [146, 45, 18, 38]]
[[200, 115, 28, 55], [146, 51, 18, 38]]
[[201, 120, 27, 57], [146, 56, 18, 39]]
[[201, 125, 29, 60], [144, 61, 16, 40]]
[[200, 131, 31, 62], [144, 65, 16, 42]]
[[200, 135, 33, 61], [143, 71, 17, 42]]
[[200, 140, 34, 62], [141, 78, 19, 42]]
```

Figure 20:second result.

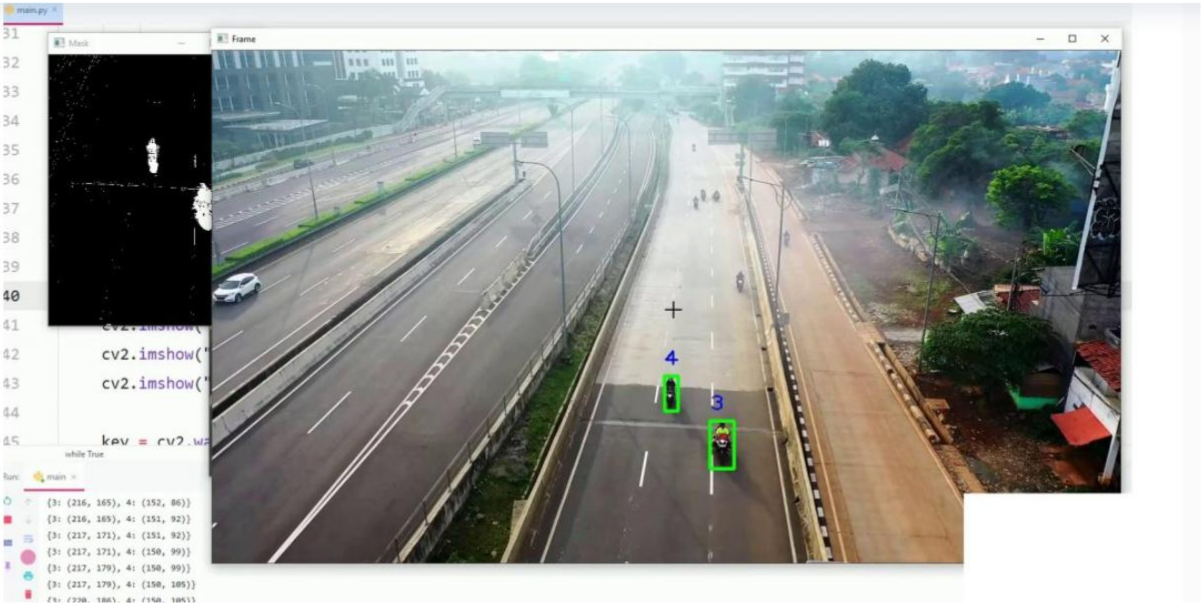
- **Associate Unique ID To The Object:**

we can analyze everything with a for a loop. At this point we just have to draw the rectangle and show the vehicle ID.



```
# 2. Object Tracking
boxes_ids = tracker.update(detections)
for box_id in boxes_ids:
    x, y, w, h, id = box_id
    cv2.putText(roi, str(id), (x, y - 15), cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 0), 2)
    cv2.rectangle(roi, (x, y), (x + w, y + h), (0, 255, 0), 3)
```

In the image you can see the result (Figure 21):



```
cv2.imshow('
cv2.imshow('

key = cv2.wa
```

```
{3: (216, 165), 4: (152, 86)}
{3: (216, 165), 4: (151, 92)}
{3: (217, 171), 4: (151, 92)}
{3: (217, 171), 4: (150, 99)}
{3: (217, 179), 4: (150, 99)}
{3: (217, 179), 4: (150, 105)}
{3: (220, 186), 4: (150, 105)}
```

Figure 21:final result.

2. simulate our proposed architecture for WMSN

A. Comparison between JPEG and JPEG2000 compression:

To evaluate the effectiveness of the two compression techniques, we use a group of images with varying sizes and dimensions to compare the quality (PSNR), compression ratio, and compression time between JPEG and JPEG 2000 .

The test is conducted using a MATLAB script that provides two wavelet options (HAAR and WALSH) and a DCT transform option. This codec calculates the PSNR, compression ratio, and operation time. Here's how it works:

The provided MATLAB script performs image compression using the Haar wavelet transform For DWT and dctmtx for DCT transform. It begins by reading an image and resizing it to specified dimensions. The image is then converted from RGB to YCbCr color space, and the chrominance components (Cb and Cr) are downsampled to reduce data size. The script applies the Walsh wavelet transform to 8x8 blocks of the luminance (Y) and downsampled chrominance components. After transformation, the coefficients are quantized using predefined quantization matrices. Differential encoding is applied to the DC coefficients, and the AC coefficients undergo ZigZag scanning, followed by Huffman encoding for efficient compression. The script calculates the compression ratio by comparing the compressed data size to the original image size. It also includes a decoding process to reconstruct the image from the compressed data, converting it back to RGB color space and calculating the Peak Signal-to-Noise Ratio (PSNR) to evaluate the quality of the compressed image. Finally, the original and compressed images are displayed, and the PSNR and compression ratio are outputted to assess the compression performance.

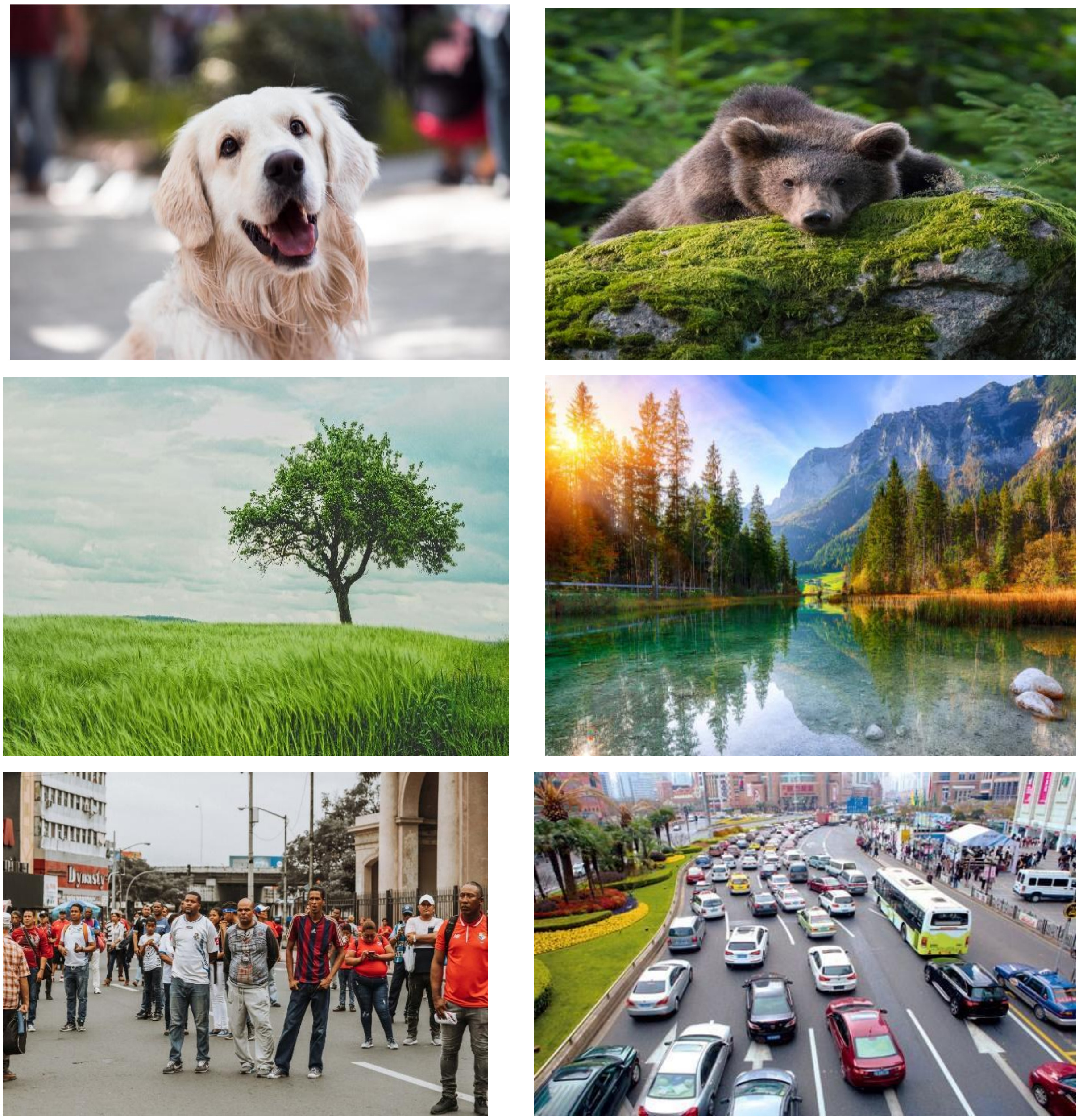


Figure 22:images before compression.

- **JPEG compression :**

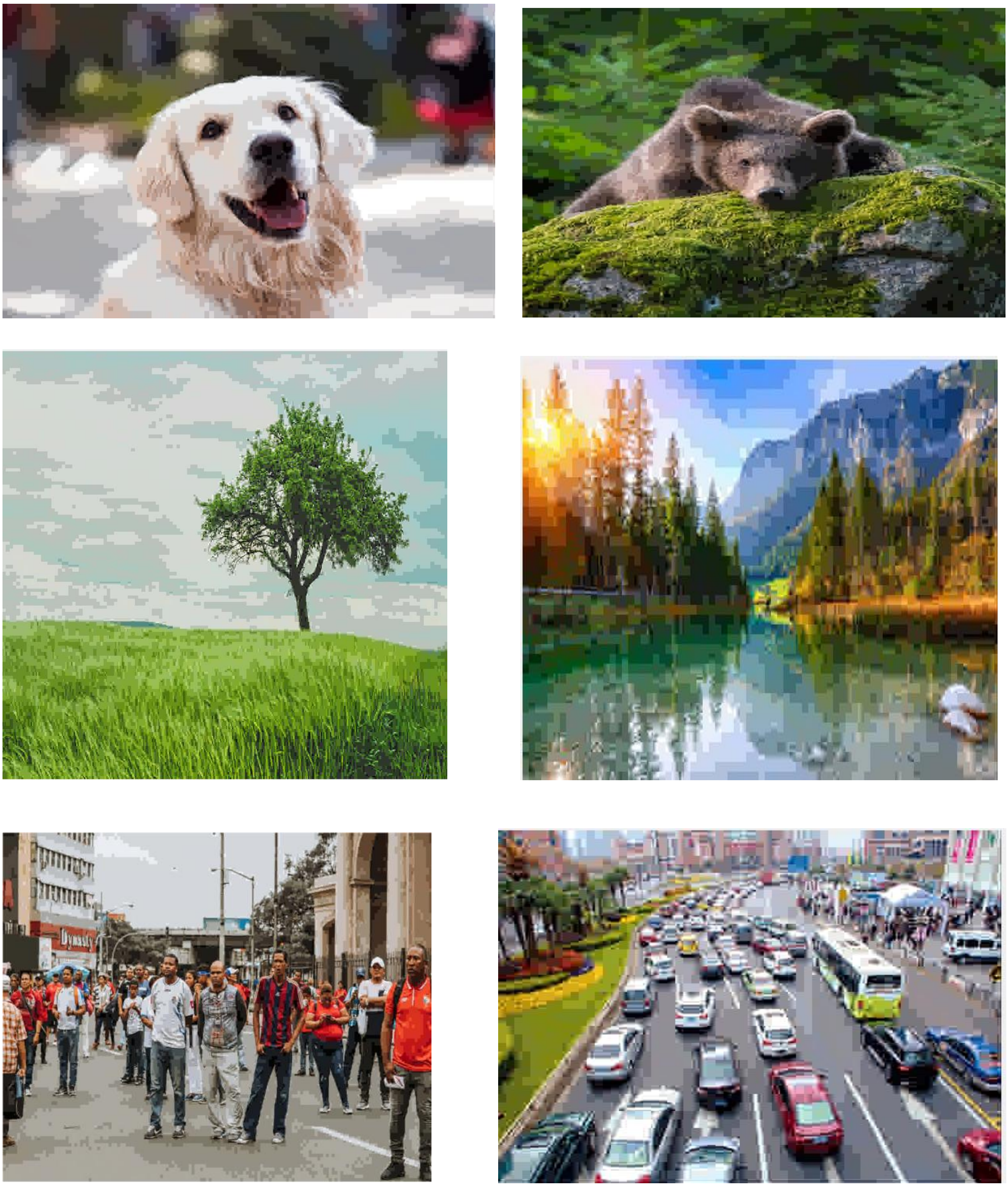


Figure 23:Images compressed with JPEG

- **JPEG2000 compression :**

Figure 24:Images compressed with JPEG2000

Table 4:comparison between JPEG and JPEG 2000

Image	JPEG					JPEG 2000				
	Image size	Compression ratio	Image cimpressed size	PSNR	Time (s)	Image size	Compression ratio	Image cimpressed size	PSNR	Time
1	588624	27.40	21479	28.6	1.19	588624	13.9290	42259	29.9	13.28
2	110262	5.13	21479	24.3	1.01	110262	11.7778	9361.9	25.8	13.43
3	3230265	1.03	3145728	24.6	1.49	3230265	10.1141	319380	24.83	15.11
4	58060	2.70	21479	23.8	0.53	58060	9.8062	5920.7	26.6	13.45
5	266835	12.42	21479	24.5	1.65	266835	11.2126	23798	24.85	14.90
6	72697	3.38	21479	21.5	0.30	72697	10.0611	7225.6	25.4	13.24

- **Analyse the results :**

After conducting tests on a group of images, it was observed that JPEG generally achieves higher compression ratios compared to JPEG 2000, with the exception of the first image in the dataset. For the first image, JPEG achieved a compression ratio of approximately 27.40, while JPEG 2000 achieved a ratio of about 13.93. This anomaly suggests that for smaller or less complex images, JPEG can sometimes achieve higher compression ratios due to its simpler compression approach.

Across the other images tested (Images 2 to 6), JPEG consistently demonstrated lower compression ratios ranging from approximately 2.70 to 12.42, whereas JPEG 2000 showed ratios ranging from about 9.81 to 11.77. This indicates that JPEG 2000 generally performs better in compressing images efficiently, especially for larger or more complex images where its advanced compression techniques can leverage redundancies more effectively.

Regarding Peak Signal-to-Noise Ratio (PSNR), both JPEG and JPEG 2000 showed varying performance across different images. JPEG 2000 generally exhibited PSNR values ranging from approximately 24.38 dB to 29.93 dB, reflecting good image quality preservation even at higher compression ratios. JPEG showed PSNR values between approximately 21.57 dB and 28.67 dB, with more variability depending on the specific image and compression settings used.

Additionally, JPEG 2000 consistently required longer compression times compared to JPEG, averaging between 13 to 15 seconds across all tested images, while JPEG compression times ranged from a few seconds up to approximately 12 seconds. This longer processing time is due to the more complex compression algorithm employed by JPEG 2000, which aims to achieve superior compression efficiency and image quality preservation.

Furthermore, JPEG is notably faster and more suitable for real-time compression applications compared to JPEG 2000. This advantage makes JPEG preferred in scenarios where rapid image compression and transmission are critical, such as in live video streaming, web applications, and real-time communication systems.

B. Our proposed architecture using a hybrid technique combine JPEG AND JPEG2000 compression:

Our algorithm is simulated using MATLAB. The Sensor Nodes (SNs) are randomly deployed in a topographical area *A* with dimensions of 300m × 300m. The number of sensor nodes SNs (including Normal Nodes (NNs) and a Camera Node (CN)) is varied from 100 to 200. For comparative purposes, the number of CNs is set to 1, placed randomly. The CN and NNs have initial energies of 15 J and 10 J, respectively. The communication radius CR of all sensor nodes is 75m. The CN captures images of 512 × 512 pixels,and then are sents to the compression cluster.

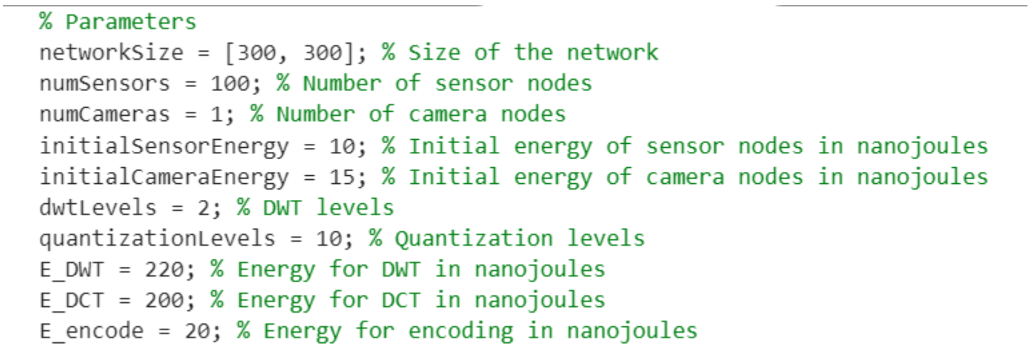
Table 5:Simulation parameters.

Parameters	**Symbol**	**value**
Network size	A	300 m × 300m
Number of sensor node	NN	100
Number of camera nodes	CN	1
Initial normal node energy	Einit	10
Initial camera node energy	Ecam_init	15
Energy of DWT transfer	E_DWT	220 nj/bit
Energy of DCT transfer	E_DCT	200 nj/bit
DWT level	-	2
Quantization level	-	10
Number of SN In compression cluster	-	3

- **Code Steps :**

1. Network Setup :

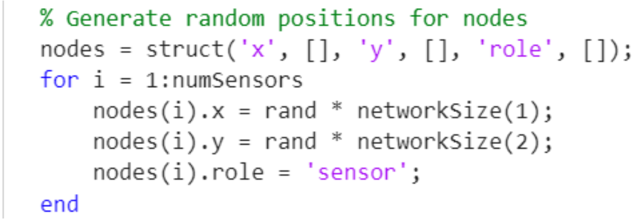
Define network size, number of nodes, and energy parameters.



```
% Parameters
networkSize = [300, 300]; % Size of the network
numSensors = 100; % Number of sensor nodes
numCameras = 1; % Number of camera nodes
initialSensorEnergy = 10; % Initial energy of sensor nodes in nanojoules
initialCameraEnergy = 15; % Initial energy of camera nodes in nanojoules
dwtLevels = 2; % DWT levels
quantizationLevels = 10; % Quantization levels
E_DWT = 220; % Energy for DWT in nanojoules
E_DCT = 200; % Energy for DCT in nanojoules
E_encode = 20; % Energy for encoding in nanojoules
```

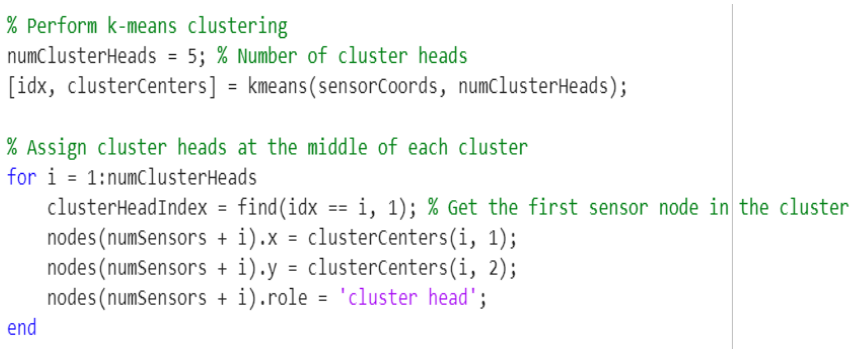
2. Node Positioning :

- Generate random positions for sensor nodes.:



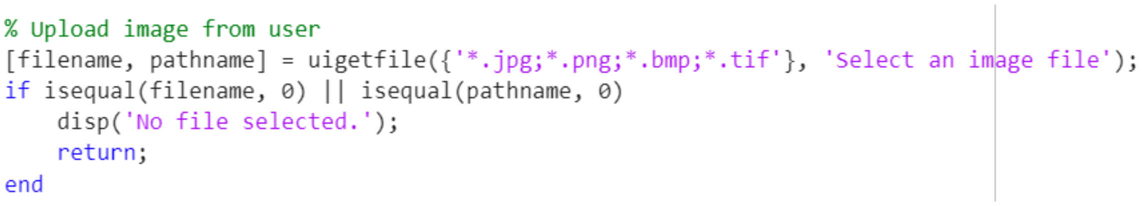
```
% Generate random positions for nodes
nodes = struct('x', [], 'y', [], 'role', []);
for i = 1:numSensors
    nodes(i).x = rand * networkSize(1);
    nodes(i).y = rand * networkSize(2);
    nodes(i).role = 'sensor';
end
```

- Perform k-means clustering on the coordinates of sensor nodes to group them into clusters with cluster heads:



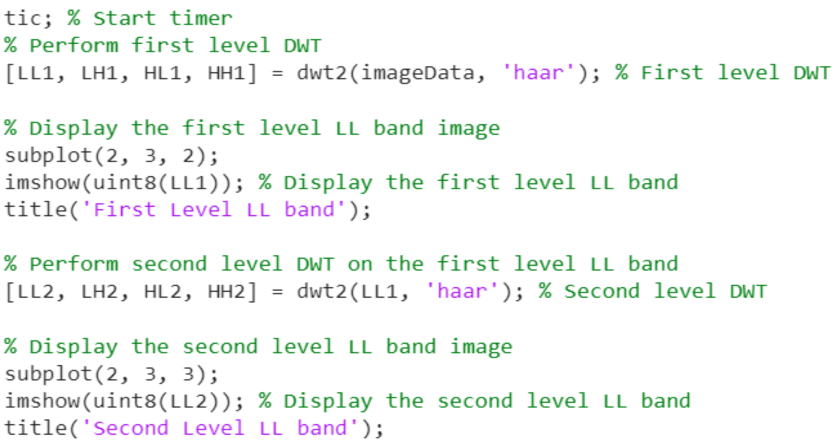
```
% Perform k-means clustering
numClusterHeads = 5; % Number of cluster heads
[idx, clusterCenters] = kmeans(sensorCoords, numClusterHeads);

% Assign cluster heads at the middle of each cluster
for i = 1:numClusterHeads
    clusterHeadIndex = find(idx == i, 1); % Get the first sensor node in the cluster
    nodes(numSensors + i).x = clusterCenters(i, 1);
    nodes(numSensors + i).y = clusterCenters(i, 2);
    nodes(numSensors + i).role = 'cluster head';
end
```

3. Prompt user to select an image file and load it :



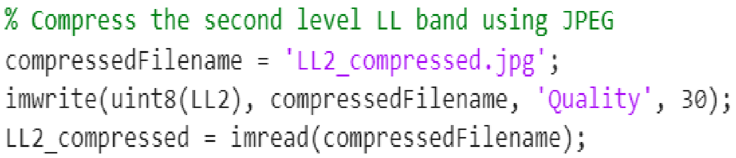
```
% Upload image from user
[filename, pathname] = uigetfile({'*.jpg;*.png;*.bmp;*.tif'}, 'Select an image file');
if isequal(filename, 0) || isequal(pathname, 0)
    disp('No file selected.');
    return;
end
```

4. Perform the two Levels DWT and start the timer of compression:



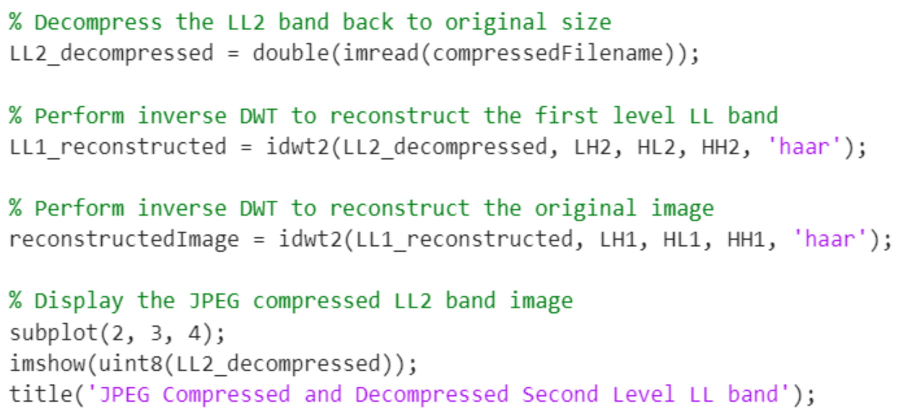
```
tic; % Start timer
% Perform first level DWT
[LL1, LH1, HL1, HH1] = dwt2(imageData, 'haar'); % First level DWT

% Display the first level LL band image
subplot(2, 3, 2);
imshow(uint8(LL1)); % Display the first level LL band
title('First Level LL band');

% Perform second level DWT on the first level LL band
[LL2, LH2, HL2, HH2] = dwt2(LL1, 'haar'); % Second level DWT

% Display the second level LL band image
subplot(2, 3, 3);
imshow(uint8(LL2)); % Display the second level LL band
title('Second Level LL band');
```

5. Compress LL2 using JPEG :



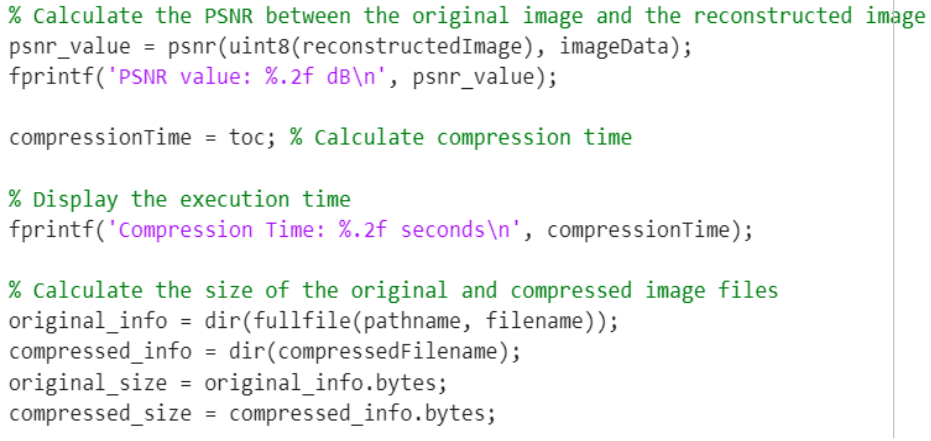
```
% Compress the second level LL band using JPEG
compressedFilename = 'LL2_compressed.jpg';
imwrite(uint8(LL2), compressedFilename, 'Quality', 30);
LL2_compressed = imread(compressedFilename);
```

6. Decompress LL2 band back to original size using inverse DWT :

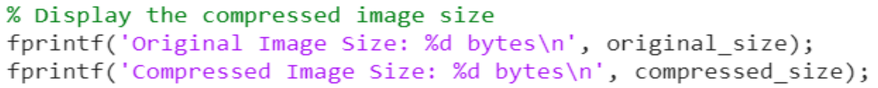


```
% Decompress the LL2 band back to original size
LL2_decompressed = double(imread(compressedFilename));

% Perform inverse DWT to reconstruct the first level LL band
LL1_reconstructed = idwt2(LL2_decompressed, LH2, HL2, HH2, 'haar');

% Perform inverse DWT to reconstruct the original image
reconstructedImage = idwt2(LL1_reconstructed, LH1, HL1, HH1, 'haar');

% Display the JPEG compressed LL2 band image
subplot(2, 3, 4);
imshow(uint8(LL2_decompressed));
title('JPEG Compressed and Decompressed Second Level LL band');
```

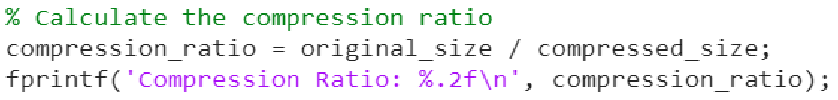
7. **Calculate and Display PSNR, compression ratio , compressed image size :**



```
% Calculate the PSNR between the original image and the reconstructed image
psnr_value = psnr(uint8(reconstructedImage), imageData);
fprintf('PSNR value: %.2f dB\n', psnr_value);

compressionTime = toc; % Calculate compression time

% Display the execution time
fprintf('Compression Time: %.2f seconds\n', compressionTime);

% Calculate the size of the original and compressed image files
original_info = dir(fullfile(pathname, filename));
compressed_info = dir(compressedFilename);
original_size = original_info.bytes;
compressed_size = compressed_info.bytes;
```

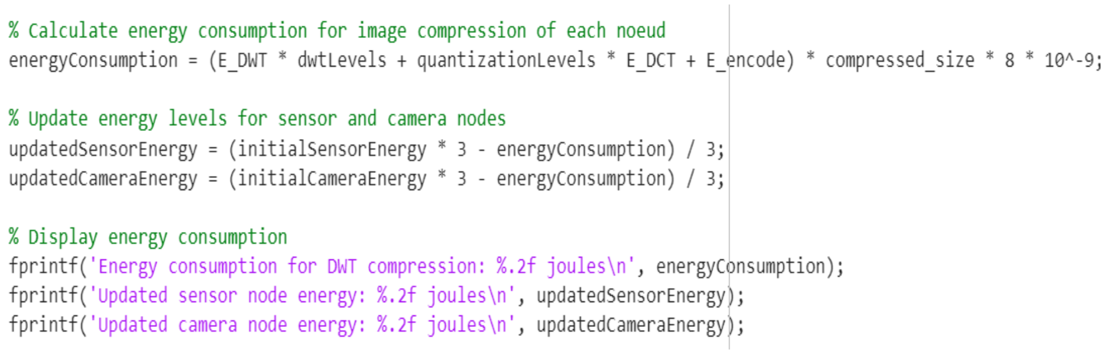


```
% Display the compressed image size
fprintf('Original Image Size: %d bytes\n', original_size);
fprintf('Compressed Image Size: %d bytes\n', compressed_size);
```



```
% Calculate the compression ratio
compression_ratio = original_size / compressed_size;
fprintf('Compression Ratio: %.2f\n', compression_ratio);
```

8. **Calculate Energy Consumption of each nœud on distributed image compression :**



```
% Calculate energy consumption for image compression of each noeud
energyConsumption = (E_DWT * dwtLevels + quantizationLevels * E_DCT + E_encode) * compressed_size * 8 * 10^-9;

% Update energy levels for sensor and camera nodes
updatedSensorEnergy = (initialSensorEnergy * 3 - energyConsumption) / 3;
updatedCameraEnergy = (initialCameraEnergy * 3 - energyConsumption) / 3;

% Display energy consumption
fprintf('Energy consumption for DWT compression: %.2f joules\n', energyConsumption);
fprintf('Updated sensor node energy: %.2f joules\n', updatedSensorEnergy);
fprintf('Updated camera node energy: %.2f joules\n', updatedCameraEnergy);
```

C. The results of simulation :

Our network is illustrated in (Figure 25), where 100 sensor nodes are organized into clusters, each directly connected to a cluster head positioned at the center of each group. Additionally, a particular cluster, marked with a red circle, is designated for image compression tasks because it contains the camera node (CN). The camera node is responsible for capturing images and transmitting the compressed data to the base station. The network demonstrates efficient communication by utilizing hierarchical clustering.

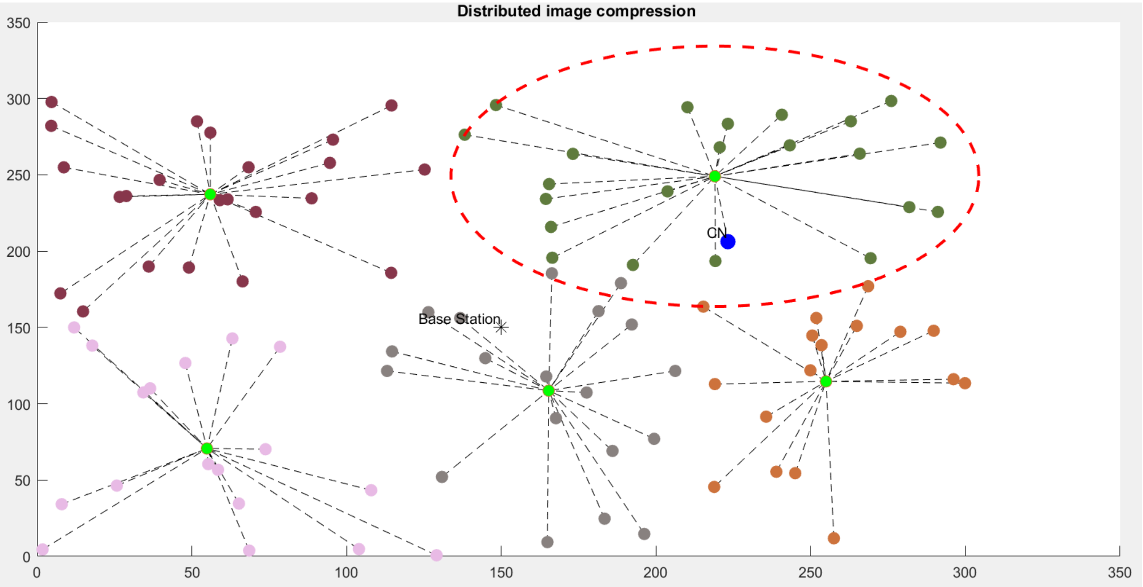


Figure 25:distributed image compression network..

In another plot of our code, the results of the image compression are illustrated in (Figure 26), showcasing 5 images: the uploaded image, the first level of DWT, the second level of DWT, the JPEG-compressed second level, and the final reconstructed image. These images demonstrate the effectiveness of our hybrid technique, which combines DWT and JPEG compression.

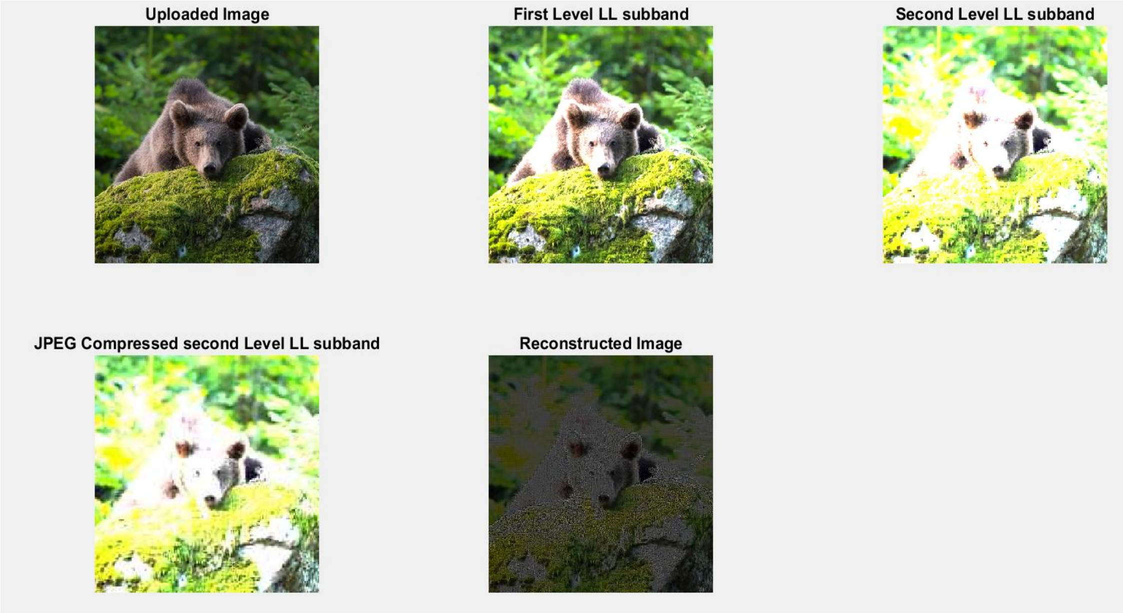
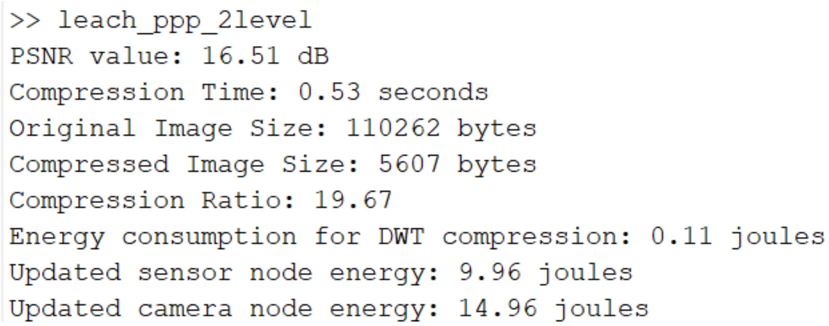


Figure 26:image compression steps.

We also have the calculation results of our code, which are illustrated in Figure 27 and displayed in the command window.



```
>> leach_ppp_2level
PSNR value: 16.51 dB
Compression Time: 0.53 seconds
Original Image Size: 110262 bytes
Compressed Image Size: 5607 bytes
Compression Ratio: 19.67
Energy consumption for DWT compression: 0.11 joules
Updated sensor node energy: 9.96 joules
Updated camera node energy: 14.96 joules
```

Figure 27:calculation results.

We can also achieve better image quality using our code by utilizing 1-level DWT for compression, as shown in Figure 28.

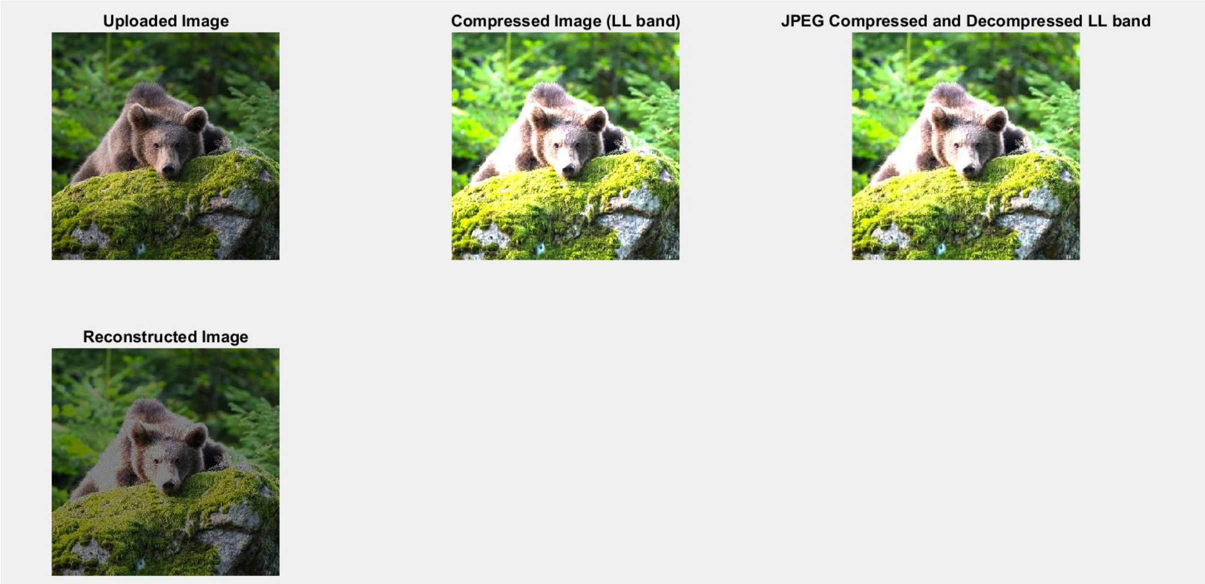

Uploaded Image
Compressed Image (LL band)
JPEG Compressed and Decompressed LL band
Reconstructed Image

Figure 28:results of compression using 1-leving DWT

- **Tests using our code :**

Here, we used the same set of images previously utilized in JPEG and JPEG2000 compression techniques, with the results presented in Table

Table 6:Hybrid DWT-DCT technique result

Image	**Image size**	**Compression ratio**	**Compressed image size**	**PSNR (db)**	**Time (second)**	**Energy consumption (joules)**
1	588624	37.4	15738	9.32	1.24	0.78
2	110262	19.67	5607	16.51	0.44	0.11
3	3230265	68.76	46978	7.60	10.64	0.92
4	58060	31.45	1846	9.56	0.40	0.04
5	266835	36.17	7378	9.15	0.72	0.15

6	72697	34.19	2126	10.2	0.6	0.09

- **Analyse the results :**

After conducting tests using the Hybrid DWT-DCT technique on a set of images, it was observed that this method achieves varying compression ratios, PSNR values, and energy consumption across different images. For example, the first image, with an original size of 588,624 bytes, achieved a compression ratio of 37.4, reducing its size to 15,738 bytes, with a PSNR of 9.32 dB and a processing time of 1.24 seconds. In contrast, the second image, initially 110,262 bytes, showed a lower compression ratio of 19.67 but a higher PSNR of 16.51 dB, indicating better image quality retention. This image also had a shorter compression time of 0.44 seconds.

The third image, being the largest at 3,230,265 bytes, achieved the highest compression ratio of 68.76, reducing its size to 46,978 bytes. However, this resulted in a lower PSNR of 7.60 dB, reflecting significant quality loss, and required the longest compression time of 10.64 seconds. The remaining images demonstrated compression ratios between 31.45 and 36.17, with PSNR values ranging from 9.15 dB to 10.2 dB and compression times from 0.40 to 0.72 seconds. Energy consumption for the compression process varied, with the smallest image consuming as little as 0.04 joules and the largest consuming up to 0.92 joules.

Compared to traditional JPEG and JPEG 2000 methods, the Hybrid DWT-DCT technique shows significant efficiency in WMSNs. JPEG, while faster and more suitable for real-time applications, generally achieves lower compression ratios and exhibits more variability in PSNR, often resulting in lower image quality retention. JPEG 2000, on the other hand, provides better compression efficiency and image quality but requires longer processing times, which can be a drawback in time-sensitive WMSN applications.

The Hybrid DWT-DCT technique strikes a balance by offering high compression ratios with moderate PSNR values, making it more efficient in terms of storage and transmission while consuming less energy. This is particularly advantageous in WMSN, where energy efficiency and quick processing are critical due to the limited power and bandwidth resources. Therefore, the Hybrid DWT-DCT method proves to be more effective for WMSN applications compared to JPEG and JPEG 2000, offering a compelling compromise between compression efficiency, image quality, processing time, and energy consumption.

Ⅳ Conclusion

In this chapter we focused on implementing our new architecture for wireless multimedia WMSNs . We explained the tools we used, like MATLAB and OpenCV, to simulate how the network works and improve how it uses energy. We compared JPEG and JPEG 2000 compression methods to see which is better for saving space and keeping image quality. Our new method, combining DWT and DCT, showed good results by balancing how well it compresses, keeps image quality, and uses energy. This chapter shows how we're working to make compression methods fit better with WMSNs, aiming to improve how these networks work in real-life situations.

GENERAL Conclusion

GENERAL Conclusion

WMSNs are becoming more important in our daily lives. However, they still have some problems that need to be solved. These networks have many sensors connected to each other, and these sensors need to work properly all the time. If one sensor fails, it can affect the whole network, and we might lose important information. To keep WMSNs effective, it is important to install and maintain them properly and keep the sensors working well.

A big part of keeping these sensors working is making sure their batteries last. Researchers are working on ways to save energy without just using bigger or more expensive batteries. They are looking at how to communicate and process data in ways that use less energy.

One way to save energy is to reduce the work each sensor does when processing multimedia data. By compressing the data and sharing tasks among many sensors, we can use less energy. Instead of one sensor doing all the work, the tasks are divided, which uses less energy. This is what we discussed in our thesis.

In our thesis, we suggested a way to save energy in WMSNs. First, we talked about WMSNs, their challenges, and what needs to be improved. We also explained different image compression techniques in detail. Then, we introduced our method for compressing images in a way that saves energy. Our approach was inspired by advancements in techniques used by researchers Sovannarith Heng, Chakchai So-In, and Tri Gia Nguyen, as referenced in article [99]. Finally, we conducted simulations and tests to assess the effectiveness of our proposed method.

Our future perspective, we plan to test our method in real-world situations. We will set up a simple network with camera nodes and regular nodes. Then, we will apply our image compression method and show the actual results. This will help us understand how well our solution works in real life.

Annexe

Annexe A : JPEG and JPEG2000 process

A. JPEG

A-1) JPEG Process

JPEG analyzes the image to detect areas where human eyes are less sensitive, taking advantage of the fact that human vision is not perfect. By identifying these parts, JPEG optimizes compression while maintaining perceived image quality. In JPEG, the lossy compression is based on two psychovisual principles:

- Changes in brightness are more important than changes in color: the human retina contains about 120 million brightness-sensitive rod cells, but only about 6 million color-sensitive cone cells. As a result, the human eye can perceive details in a luminance image more clearly than in a chrominance image.(Figure 29) supports this point:

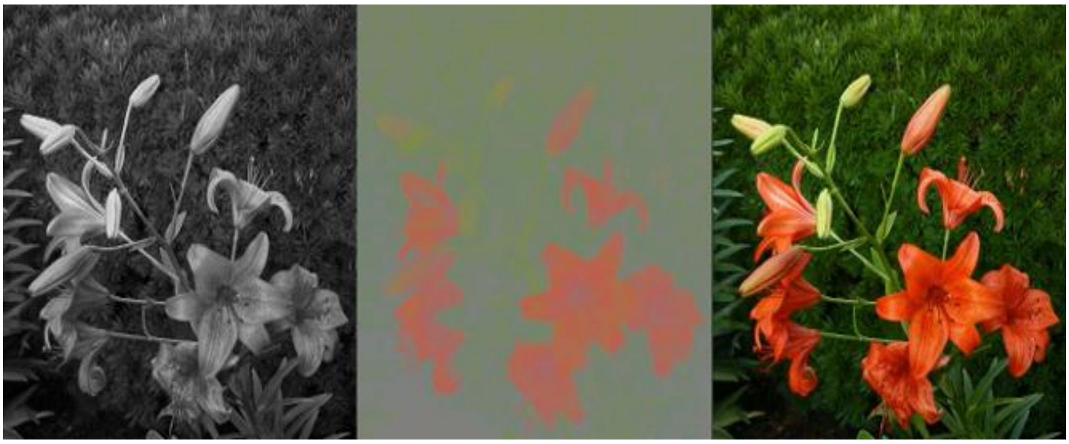


Figure 29 :Luminance , chrominance, original image

- Low-frequency changes are more important than high-frequency changes: The human eye is good at judging low-frequency light changes, like the edges of objects (we can easily notice where one object ends and another begins). It is less accurate at judging high-frequency light changes, like the fine detail in a busy pattern or texture. Camouflage works in part because higher-frequency patterns disrupt the lower-frequency edges of the thing camouflaged.

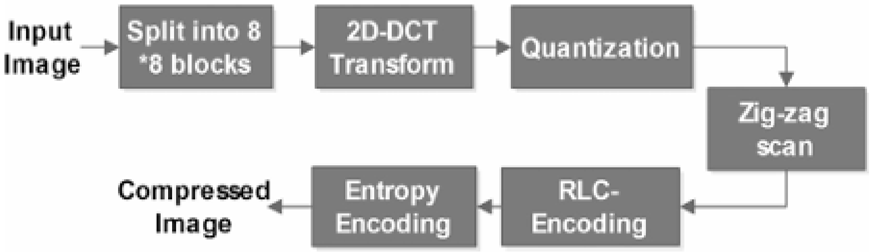


Figure 30:Lossy JPEG compression scheme.

JPEG compression applies each of these ideas in turn. In each case, the image data is transformed to give easier access to the kind of information needed (either brightness or frequency information). Then some of the less important information is discarded. As a final step, the information that is left is compressed with traditional lossless compression to pack the end result into the smallest space possible. (Figure 30) shows lossy JPEG compression scheme steps. In this part, we explore the process of JPEG technique step-by-step :

a) Step 1: Isolate the color information:

Images are grids of colored pixels ranging from 0 to 255, each stored as three numbers representing red, green, and blue light, forming an RGB image (Figure 31). However, JPEG needs to isolate brightness from color, as brightness is more crucial. Thus, JPEG converts RGB to YCbCr, where Y contains brightness (luminance) and Cb and Cr contain color information (chrominance). This separation allows better compression by focusing on brightness.(Figure 32) shows the conversion from RGB to YCbCr.

The conversion formulas from RGB to YCbCr are as follows:

Y=0.299 R + 0.587 G + 0.114B

Cb=128 − 0.168736 R − 0.331264 G + 0.5 B

Cr=128 + 0.5 R − 0.418688 G − 0.081312 B

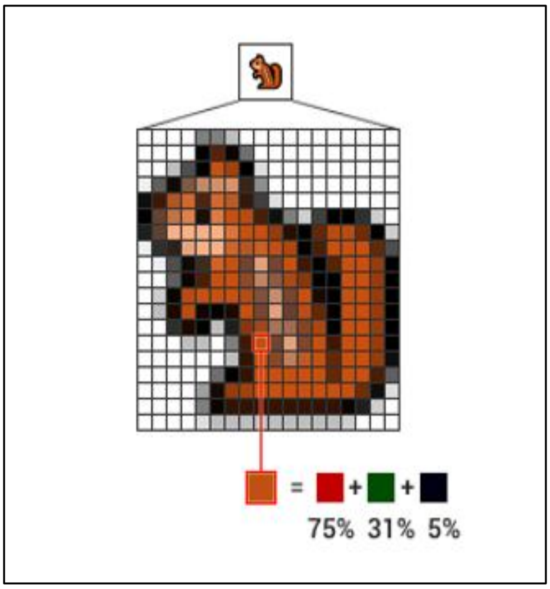


Figure 31:RBG image

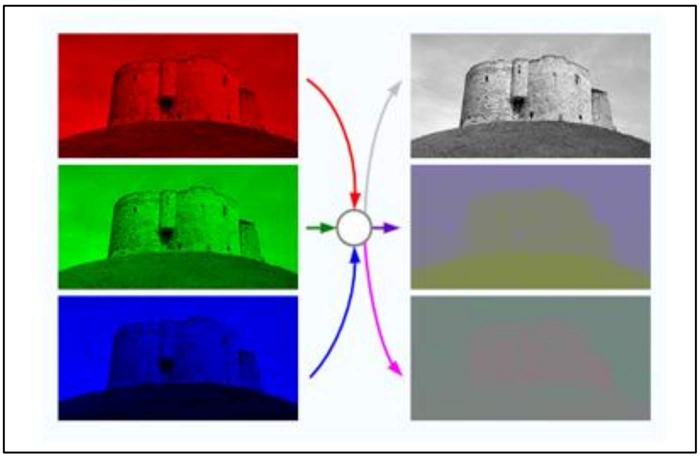


Figure 32:conversion from RGB to YCbCr

b) Step 2: Downsampling chrominance:

As mentioned earlier, human eyes have limited ability to perceive color under bright illumination conditions. Therefore, during this step, a significant portion of data will be discarded.

So for example we have this image (Figure 33):

Figure 33:input image

After separating the color information (chrominance) from the brightness information (luminance) in an image, JPEG compression reduces the amount of chrominance data by downsampling it. This means that instead of storing color information for each individual pixel, the chrominance channels are divided into small blocks of 4 pixels (2×2 block). For each block, the color values of the 4 pixels are averaged, and this average value is then used to represent the color for all 4 pixels in that block (Figure 34). So instead of having 4 pixels in a block, we will effectively have only 1 pixel of chrominance data for that block.

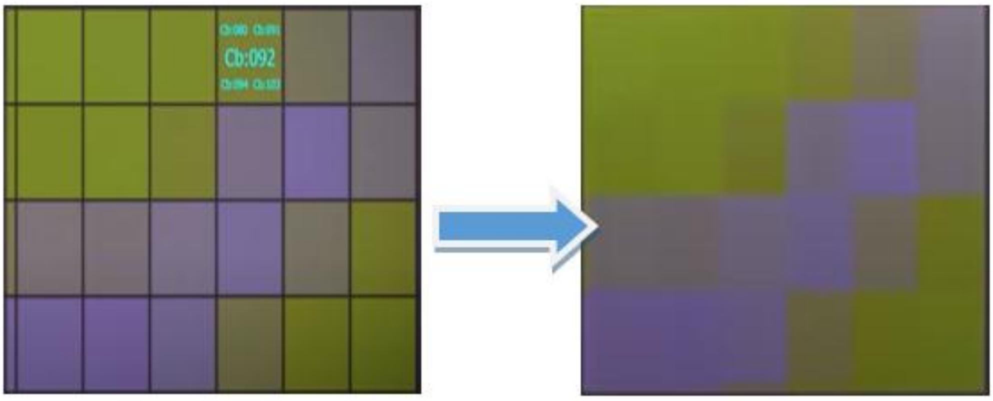


Figure 34:downsampling chrominance.

For this block the average is become 142

As a result, some information is lost, and the size of the picture is halved, but since the human eye is not very sensitive to color, the changes are not easily distinguishable (Figure 35).

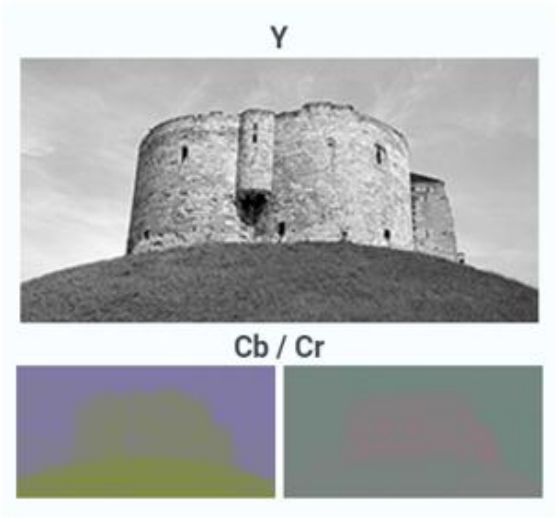


Figure 35:Result of chroma

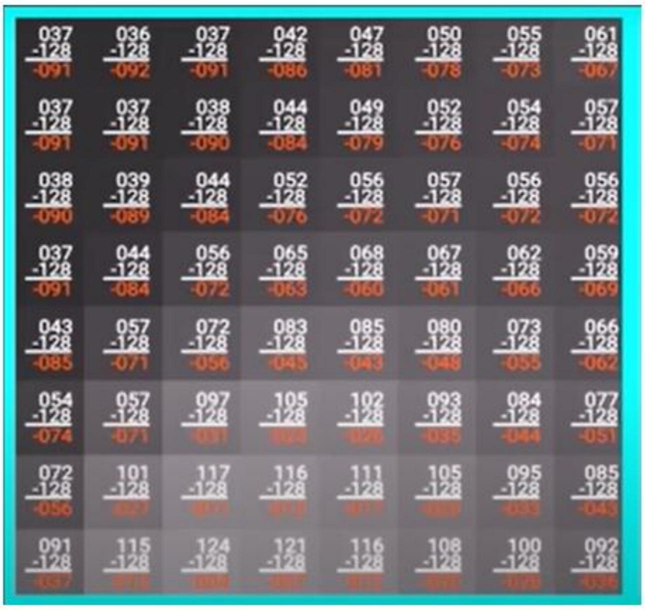
Notice that we started with 3 full channels and now we have 1 full channel and 2 × ¼ channels, for a total of 1 ½. We're just getting started and we are already down to half of the information we started with.

c) Step 3: Convert to the frequency domain (DCT):

Human eyes are less sensitive to areas in an image with high-frequency details, such as edges and fine textures. To exploit this, the Discrete Cosine Transform (DCT) is applied to convert the image from the spatial domain to the frequency domain. During this transformation, the DCT identifies areas with high-frequency changes in chrominance or luminance. After downsampling, the pixel data of each channel is divided into 8×8 blocks of 64 pixels. From now on, the algorithm processes each block of pixels independently.

Forward DCT:

First, each pixel value for each channel (chrominance and luminance) is subtracted by 128 to make the value range from -128 to +127. (-128 is black color +127 is white), (Figure 36) represents one block of 8×8 pixels:



037 -128 -091	036 -128 -092	037 -128 -091	042 -128 -086	047 -128 -081	050 -128 -078	055 -128 -073	061 -128 -067
037 -128 -091	037 -128 -091	038 -128 -090	044 -128 -084	049 -128 -079	052 -128 -076	054 -128 -074	057 -128 -071
038 -128 -090	039 -128 -089	044 -128 -084	052 -128 -076	056 -128 -072	057 -128 -071	056 -128 -072	056 -128 -072
037 -128 -091	044 -128 -084	056 -128 -072	065 -128 -063	068 -128 -060	067 -128 -061	062 -128 -066	059 -128 -069
043 -128 -085	057 -128 -071	072 -128 -056	083 -128 -045	085 -128 -043	080 -128 -048	073 -128 -055	066 -128 -062
054 -128 -074	057 -128 -071	097 -128 -031	105 -128 -023	102 -128 -026	093 -128 -035	084 -128 -044	077 -128 -051
072 -128 -056	101 -128 -027	117 -128 -011	116 -128 -012	111 -128 -017	105 -128 -023	095 -128 -033	085 -128 -043
091 -128 -037	115 -128 -013	124 -128 -004	121 -128 -007	116 -128 -012	108 -128 -020	100 -128 -028	092 -128 -036

Figure 36:block of of 8×8 pixels

Now after this we Use DCT, for each channel, each block of 64 pixels can be reconstructed by multiplying a constant set of base images by their corresponding weight values and then summing them up together. Here is what the base images look like (Figure 37):

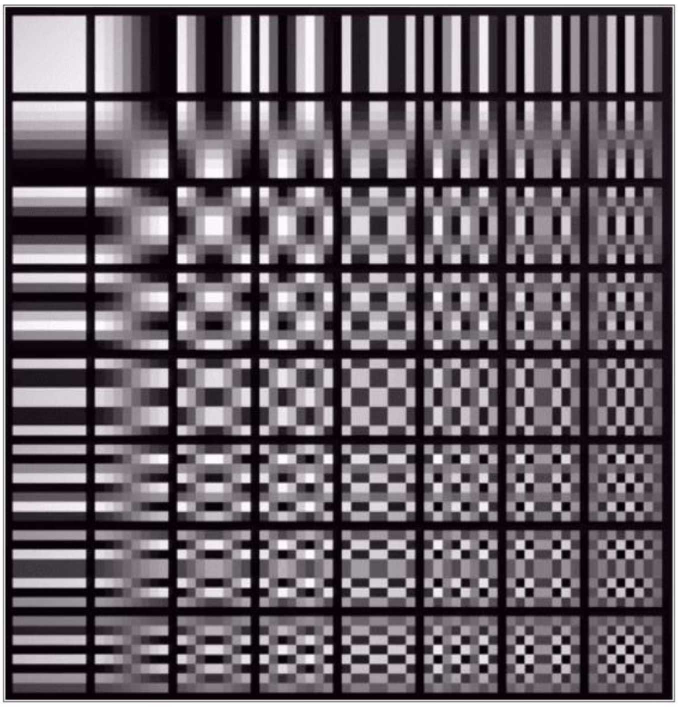


Figure 37:Base image.

Applying forward DCT on each block of 64 pixels gives us a matrix containing the corresponding weight values for each base image. These values show how much of each base image is used to reconstruct the original 8×8 pixel block, and it looks something like this:

$$\begin{bmatrix} 463 & -170 & -15 & 33 & -33 & 8 & 4 & -5 \\ -197 & 1 & 40 & 0 & 10 & 5 & 1 & -1 \\ 36 & 41 & 9 & -17 & 12 & -11 & 0 & 0 \\ -5 & -25 & 0 & -4 & 0 & 8 & 0 & 0 \\ 4 & 9 & 0 & 5 & 2 & -5 & 4 & -5 \\ -2 & -1 & 3 & 0 & 2 & -1 & -2 & 4 \\ -8 & 5 & 4 & -6 & 6 & -1 & -4 & 5 \\ 7 & -1 & -2 & 0 & 3 & 4 & -2 & 0 \end{bmatrix}$$

For every block of 64 pixels, we'll have three weight matrices, one for luminance and two for chrominance. In addition, no information is lost in this phase.

d) Step 4: Quantization:

In this step, JPEG removes some of this high-frequency information without affecting the perceived quality of the image. To do this, the weight matrix is divided by a precalculated quantization table, and the results are rounded to the closest integer. Here is what a quantization table for chrominance channels looks like:

$$\begin{bmatrix} 10 & 8 & 9 & 9 & 9 & 8 & 10 & 9 \\ 9 & 9 & 10 & 10 & 10 & 11 & 12 & 17 \\ 13 & 12 & 12 & 12 & 12 & 20 & 16 & 16 \\ 14 & 17 & 18 & 20 & 23 & 23 & 22 & 20 \\ 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 \\ 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 \\ 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 \\ 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 \end{bmatrix}$$

And for the luminance channel:

$$\begin{bmatrix} 6 & 4 & 4 & 6 & 10 & 16 & 20 & 24 \\ 5 & 5 & 6 & 8 & 10 & 23 & 24 & 22 \\ 6 & 5 & 6 & 10 & 16 & 23 & 28 & 22 \\ 6 & 7 & 9 & 12 & 20 & 35 & 32 & 25 \\ 7 & 9 & 15 & 22 & 27 & 44 & 41 & 31 \\ 10 & 14 & 22 & 26 & 32 & 42 & 45 & 37 \\ 20 & 26 & 31 & 35 & 41 & 48 & 48 & 40 \\ 29 & 37 & 38 & 39 & 45 & 40 & 41 & 40 \end{bmatrix}$$

We can see that the quantization tables have higher numbers at the bottom right, where high-frequency data is located, and have lower numbers at the top left, where low-frequency data is located. As a result, after dividing each weight value, high-frequency data is rounded to 0.

$$\begin{bmatrix} -78 & -61 & -36 & -13 & -16 & -10 & -2 & 1 \\ -45 & -33 & -25 & -13 & -15 & -7 & -3 & -2 \\ 9 & 7 & 2 & -2 & -1 & 0 & 0 & 0 \\ -10 & -7 & -5 & -4 & 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

e) Step 5 : Entropy encoding :

Now all that’s left to do is to list the values that are inside each matrix, run RLE (Run Length Encoding) since we have a lot of zeroes, and then run the Huffman Coding algorithm before storing the data.

RLE and Huffman Coding are lossless compression algorithms that reduce the space needed to store information without removing any data.

As we can see, the result of quantization has many 0s toward the bottom left corner. To keep the 0 redundancy, the JPEG algorithm does a zig-zag scanning pattern that keeps all zeroes together (Figure 38):

After zig-zag scanning, we get a list of integer values, and after running RLE on the list, we get each number and how many times they’re repeated.

Finally, JPEG runs Huffman coding on the result, and the values that have more frequency (like the zeroes) are represented with fewer bits than less frequent values. No data is lost during this phase.

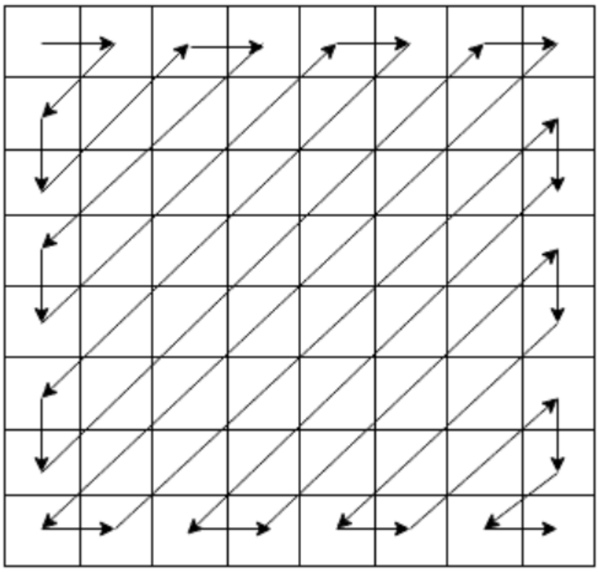


Figure 38:zig zag scan.

A-2) JPEG Decoding:

Whenever we open a .jpg file using an image viewer, the file needs to be decoded before it can be displayed. To decode a *.jpg* image, the image viewer does all the above steps in reverse order:

- Run Huffman Decoding on the file data

- Deconstruct RLE
- Put the list of numbers into an 8×8 matrix
- Multiply the integer values by their corresponding quantization table
- Multiply the weight values by their corresponding base images, then sum them up to get the pixel values
- Upscale the chrominance channels
- Change the color space from YCbCr to RGB
- Display the image

A-3) Limitation of JPEG:

- **Reduced Image Quality :**
  - **Lossy Compression:** JPEG's lossy compression results in the loss of image details and introduces artifacts, which can be problematic for applications in WMSNs that require high-fidelity images, such as medical imaging or surveillance.
  - **Artifacts:** The blockiness and ringing artifacts introduced by JPEG compression can degrade the quality of images, making it harder to interpret critical details in WMSN applications.
- **Inefficiency with Sharp Edges and Lines:**
  - **Poor Handling of High-Frequency Details:** In WMSNs, images often contain important details like edges and lines that are crucial for accurate analysis and decision-making. JPEG's inefficiency in handling these high-frequency details can lead to a significant loss of vital information.
  - **Edge Artifacts:** The presence of edge artifacts can interfere with the accuracy of image processing algorithms used in WMSNs, such as object detection and recognition.
- **No Support for Animated Graphics:**

  - **Static Format:** WMSNs may require the transmission of animated graphics or video streams for monitoring and real-time analysis. JPEG's inability to support animation limits its applicability in scenarios where dynamic content is essential.

- **Lack of Support for Layered Images:**
  - **Single Layer Only**: JPEG's lack of support for layered images restricts its use in WMSNs where layered image processing might be necessary for tasks such as multi-spectral imaging or depth map generation.
  - **Flattening:** The need to flatten images into a single layer before compression can result in the loss of important contextual information, hindering the effectiveness of image analysis in WMSNs.

- **Limited Bit Depth :**
  - **8-Bit Restriction:** JPEG's restriction to 8-bit images per color channel limits its ability to handle high dynamic range (HDR) images, which are often required in WMSNs for capturing scenes with a wide range of luminance levels.
  - **Loss of Detail**: The reduction in bit depth results in a loss of detail and color information, affecting the quality and accuracy of images transmitted over WMSNs, which is critical for applications like environmental monitoring and disaster response.

B-1) JPEG 2000:

JPEG2000 is a modern image compression standard designed to address the limitations of the original JPEG standard. Similar to JPEG, the JPEG2000 compression process involves four fundamental steps: preprocessing, transformation, quantization, and encoding. However, unlike JPEG, JPEG2000 offers the option to skip the quantization step for lossless compression. Additionally, while JPEG relies on an advanced form of Huffman coding, JPEG2000 employs a sophisticated coding method known as Embedded Block Coding with Optimized Truncation (EBCOT).

a-Features of JPEG2000:

- **Lossless and Lossy Compression:** JPEG2000 offers superior lossy compression at low bit rates and supports lossless compression with progressive decoding, benefiting applications like digital libraries and medical imaging.
- **Image Security:** Its open architecture facilitates the use of image protection techniques such as watermarking, labeling, stamping, and encryption.
- **Region-of-Interest Coding:** JPEG2000 allows for defining regions of interest (ROIs) which can be encoded and transmitted with higher quality than the rest of the image (Figure 39).
- **Robustness to Bit Errors:** The standard includes error-resilient tools to enhance bit-stream robustness against transmission errors.

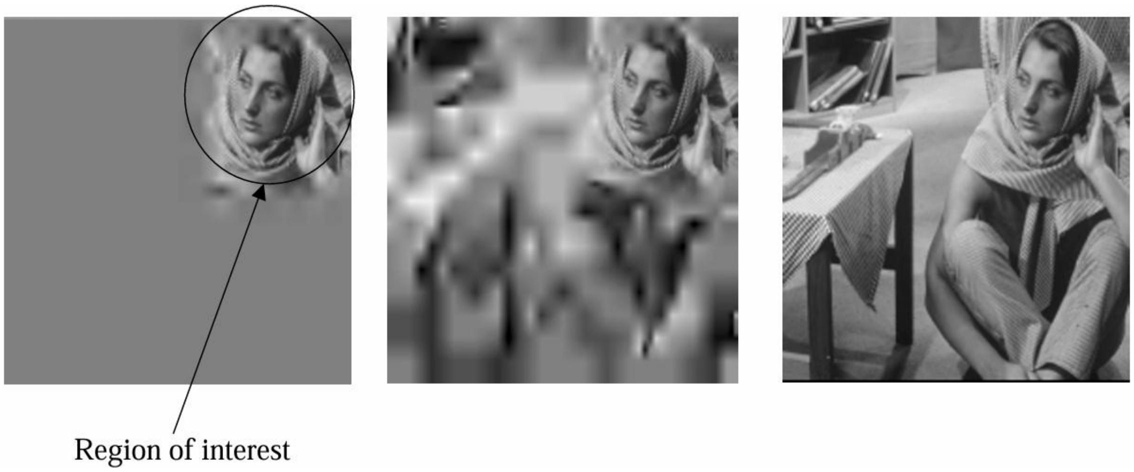


Figure 39:example of ROIs.

b*- JPEG 2000 Algorithm*:

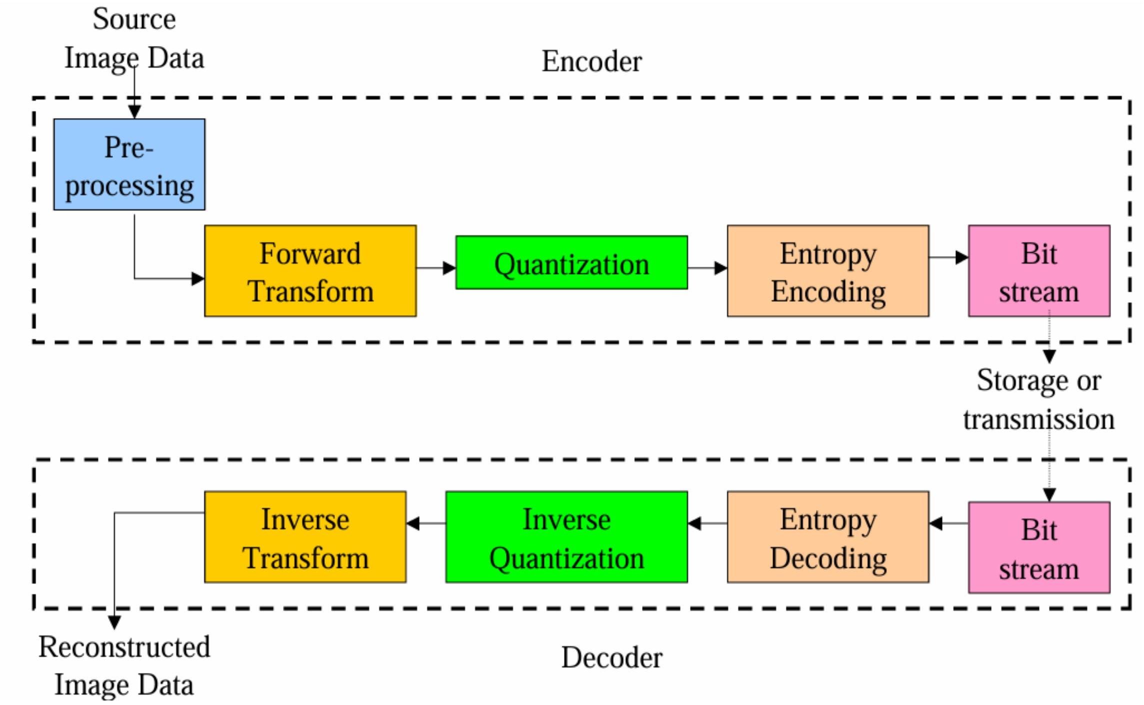


Figure 40:JPEG 2000 Algorithme Overview

b-1- Pre-processing:

Three steps:

Color component transformation: from RGB to Y Cb Cr.

Tiling: After color transformation, the image is split into so-called *tiles*, rectangular regions of the image that are transformed and encoded separately. Tiles can be any size, and it is also possible to consider the whole image as one single tile. Once the size is chosen, all the tiles will have the same size as shown on (Figure 41). Dividing the image into tiles is advantageous in that the decoder will need less memory to decode the image and it can opt to decode only selected tiles to achieve a partial decoding of the image.

DC level shifting: samples of each tile are subtracted the same quantity. (substract 128 from each tiles pixels)

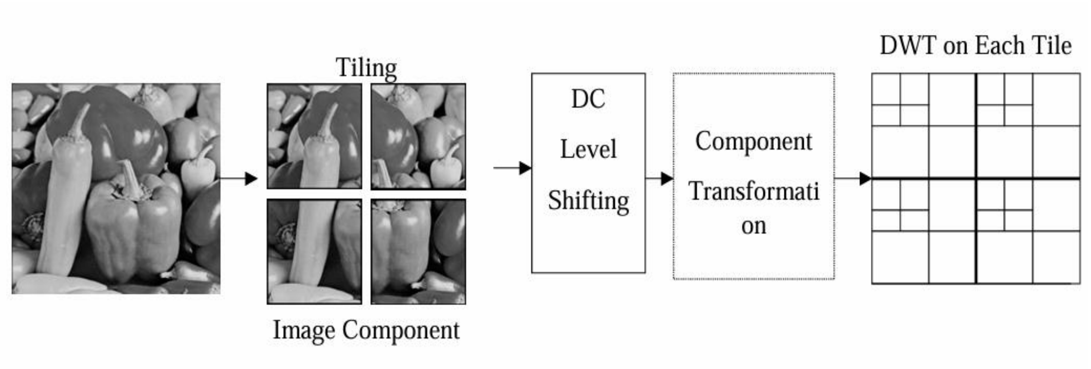


Figure 41:pre-processing

b-2- Forward transform :

These tiles are then wavelet-transformed to an arbitrary depth, unlike traditional JPEG which uses an 8×8 block-size discrete cosine transform. JPEG 2000 employs different wavelet transforms:

Irreversible: The CDF 9/7 wavelet transform, developed by Ingrid Daubechies. This transform is termed "irreversible" because it introduces quantization noise that depends on the precision of the decoder.

Reversible: A rounded version of the biorthogonal Le Gall–Tabatabai (LGT) 5/3 wavelet transform, developed by Didier Le Gall and Ali J. Tabatabai. It uses only integer coefficients, so the output does not require rounding (quantization), avoiding any quantization noise. This transform is used in lossless coding.

Haar disctete wavelet transfer :

One commonly used wavelet transform in JPEG 2000 is the Haar Discrete Wavelet Transform (Haar DWT). Although it does not provide compression itself, it enhances the Huffman coding stage by partitioning the image into approximation (average) and detail (difference) parts. Applying Haar DWT twice reduces the approximation part and increases the detail part, enhancing compression efficiency. This approach, known as HW-JPEG2000, aims to improve image quality and reduce energy consumption, compared to traditional image compression algorithms [10].

b-3- Quantization :

After the wavelet transform, if we are performing lossless compression, the DWT pictured is encoded and the algorithm is concluded. For lossy compression the coefficients are scalar-quantized to reduce the number of bits to represent them, at the expense of quality. The output is a set of integer numbers which have to be encoded bit-by-bit.

b-4- Entropy coding:

Bit plane: The bit plane encoding process begins by partitioning each subband into non-overlapping rectangular code blocks, which are encoded independently. The encoding starts from the most significant non-zero bit-plane and proceeds to the least significant bit-plane. Within each bit-plane, three coding passes are performed: significance propagation, magnitude refinement, and cleanup. Each pass scans the block according to a specific pattern, generating coding symbols based on predefined rules shown in (Figure 42). During the significance propagation pass, significant coefficients with non-zero neighbors are identified and encoded. The magnitude refinement pass processes previously identified significant coefficients, encoding their magnitude bits. The cleanup pass handles all remaining insignificant coefficients, using run-length encoding to optimize symbol generation. The generated symbols are then passed through a binary adaptive arithmetic coder with context modeling to further compress the data, using specific contexts for each coding pass and the MQ coder to reduce computational complexity.

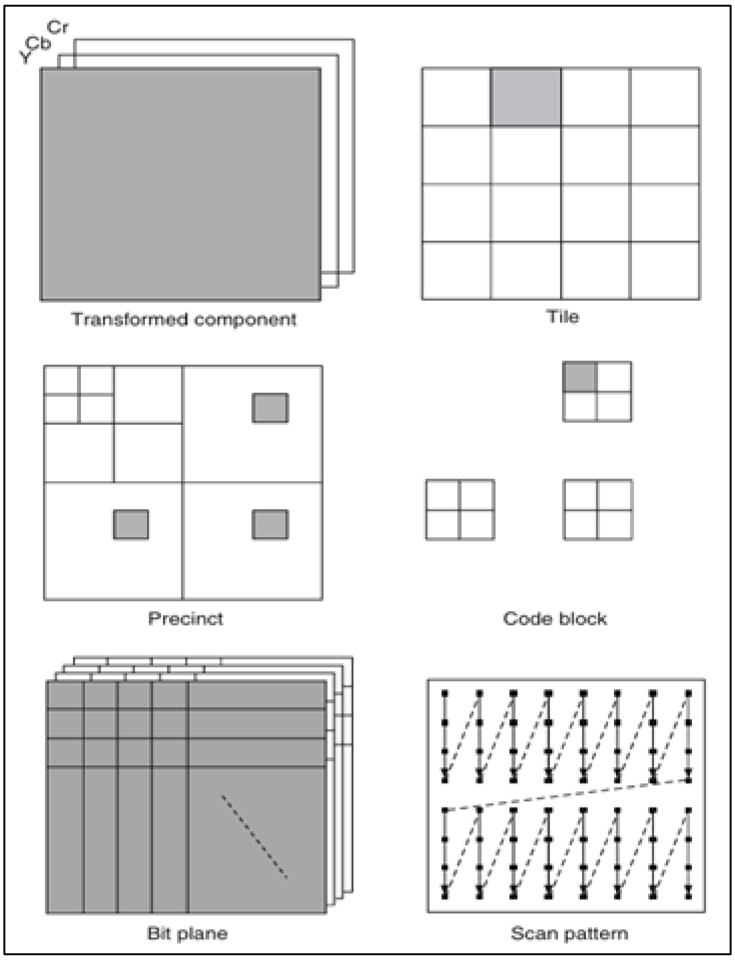


Figure 42:Components of the JPEG 2000 image data structure.

b-5- Bitstream :

The final step involves organizing the encoded coefficients into a compressed bitstream. This bitstream includes crucial information such as the quantization step sizes, the positions of the significant coefficients, and the actual values of these significant coefficients. This organized bitstream is essential for efficient storage and transmission, ensuring that all necessary data is compactly and accurately represented for reconstruction during decompression.

B-2) Limitation of JPEG 2000 :

Complexity and Computational Requirements: Higher computational power and memory are needed [80], leading to longer encoding and decoding times.

Adoption and Compatibility: Limited support in software and devices compared to traditional JPEG, causing compatibility issues.

File Size for Simple Images: May produce larger files for simple images due to the overhead of advanced compression techniques.

Encoding Time: Slower encoding process, especially at higher compression ratios.

File Size Consistency: Less predictable file sizes compared to traditional JPEG.

Limited Hardware Support: Lacks widespread hardware acceleration, relying on slower software processing.

Less Effective for Small Images: Overhead can result in larger file sizes for very small images.

Bibliography

[1] Akyildiz, I. F., & Vuran, M. C. (2010). *Wireless sensor networks*. John Wiley & Sons.

[2] Azim, M. M. A., & Jiang, X. (Eds.). (2015). *Wireless Sensor Multimedia Networks: Architectures, Protocols, and Applications*. CRC Press.

[3] Del-Valle-Soto, C., Rodríguez, A., & Ascencio-Piña, C. R. (2023). A survey of energy-efficient clustering routing protocols for wireless sensor networks based on metaheuristic approaches. *Artificial Intelligence Review*, *56*(9), 9699-9770.

[4] Akyildiz, I. F., Melodia, T., & Chowdhury, K. R. (2007). A survey on wireless multimedia sensor networks. *Computer networks*, *51*(4), 921-960..

[5] Shakshuki, E., Xing, X., & Malik, H. (2009). An introduction to wireless multimedia sensor networks. In *Handbook of Research on Mobile Multimedia, Second Edition* (pp. 1-16). IGI Global.

[6] Shen, H., & Bai, G. (2016). Routing in wireless multimedia sensor networks: A survey and challenges ahead. *Journal of Network and Computer Applications*, *71*, 30-49.

[7] Akyildiz, I. F., Melodia, T., & Chowdhury, K. R. (2008). Wireless multimedia sensor networks: Applications and testbeds. *Proceedings of the IEEE*, *96*(10), 1588-1605.

[8] Almalkawi, I. T., Zapata, M. G., Al-Karaki, J. N., & Morillo-Pozo, J. (2010). Wireless multimedia sensor networks: current trends and future directions. *Sensors*, *10*(7), 6662-6717.

[9] Khobragade, P. B., & Thakare, S. S. (2014). Image compression techniques-a review. *Int. J. Comput. Sci. Inf. Technol*, *5*(1), 272-275.

[10] Hussain, A. J., Al-Fayadh, A., & Radi, N. (2018). Image compression techniques: A survey in lossless and lossy algorithms. *Neurocomputing*, *300*, 44-69.

[11] ZainEldin, H., Elhosseini, M. A., & Ali, H. A. (2015). Image compression algorithms in wireless multimedia sensor networks: A survey. *Ain Shams engineering journal*, *6*(2), 481-490.

[12] Bouakkaz, F., Ali, W., & Derdour, M. (2021). Forest fire detection using wireless multimedia sensor networks and image compression. *Instrum. Mes. Métrologie*, *20*, 57-63.

[13] Patel, N., & Chaudhary, J. (2017, February). Energy efficient WMSN using image compression: A survey. In *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)* (pp. 124-128). IEEE.

[14] Patel, N., & Chaudhary, J. (2017, February). Energy efficient WMSN using image compression: A survey. In *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)* (pp. 124-128). IEEE.

[15] Wang, C., Shan, R., & Zhou, X. (2015). APBT-JPEG image coding based on GPU. *KSII Transactions on Internet and Information Systems (TIIS)*, *9*(4), 1457-1470.

[16] Pham, D. M., & Aziz, S. M. (2013, April). An energy efficient image compression scheme for Wireless Sensor Networks. In *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing* (pp. 260-264). IEEE.

[17] Chaima Bejaoui , Mohamed Alamine Ben Farah , Abdennaceur Kachouri, "USABILITY OF LEACH AND LEACH-C IN WIRELESS MULTIMEDIA SENSOR NETWORKS," 2011.

[18]] Rosário, D., Costa, R., Paraense, H., Machado, K., Cerqueira, E., Braun, T., & Zhao, Z. (2012). A hierarchical multi-hop multimedia routing protocol for wireless multimedia sensor networks. *Network protocols and algorithms*, *4*(2).

[19] I.S.AlShawi, L.Yan, W.Pan, and B.Luo, “Lifetime enhancement in wireless sensor networks using fuzzy approach and A-star algorithm,” in Proceedings of the IET Conference on Wireless Sensor Systems, WSS 2012,UK,June2012.

[20] B. Balakrishnan and S. Balachandran, “FLECH: fuzzy logic based energy efficient clustering hierarchy for nonuniform wireless sensor networks,” Wireless Communications and Mobile Computing,vol.2017,13pages,2017.

[21] A. Alomari et al., “Dynamic fuzzy-logic based path planning for mobility-assisted localization in wireless sensor networks,” Sensors,vol.17,no.8,p.1904,2017.

[22] Heng, S., So-In, C., & Nguyen, T. G. (2017). Distributed image compression architecture over wireless multimedia sensor networks. *Wireless Communications and Mobile Computing*, *2017*(1), 5471721.