Wavelet Transform for Image Processing

Zhang Jinrui*
jerryzhang40@gmail.com

20250728

Abstract

In this article, I tried to use Wavelet Transform to compress images.

1 Decomposition & Mother Wavelet

A mother wavelet ψ satisfied following property. $\langle \psi, \psi \rangle = 1$

And the family of the Hilbert Basis wavelets are as follow.

$$\psi_{j,k} = \sqrt{2^j}\psi(2^j(x - 2^{-j}k))\, j, k \in \mathbb{Z}$$

A function f can be decomposed use these basis as follow.

$$f = c_{j,k}\psi_{j,k}$$

There are 10 drones and fly on the sky obeys Newton's second law of motion. which is

$$\vec{F} = m\vec{a}$$
$$\vec{a} = \frac{d\vec{v}}{dt} = \frac{d^2\vec{x}}{dt^2}$$

And I mean the policy by, we need a function of force depending on some communication between drones to decide the $\vec{F}$

$$\vec{F} = f(thecurrentinformation)$$

And then we want the following dynamic system

$$\begin{bmatrix} \frac{d\vec{d}}{dt} \\ \frac{d\vec{v}}{dt} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \vec{a} = f/m \end{bmatrix}$$

has some Self-organized emergent phenomena, to automatically emergent a circle rounding pattern.

question: framelet multiresolution(sacle function and funciont)

*alternative email:zhangjr1022@mails.jlu.edu.cn

2 Haar 1D sequence Decomposition

For a 2^N points sequence, the mother wavelet is

$$\psi(n) = \begin{cases} 0, & \text{if } n < 0 \\ \frac{1}{\sqrt{2^N}}, & \text{if } 0 \leq n < 2^{N-1} \\ \frac{-1}{\sqrt{2^N}}, & \text{if } 2^{N-1} \leq n < 2^N \\ 0, & \text{if } n >= 2^N \end{cases}$$

We have

$$\langle \psi, \psi \rangle = \sum_{k=0}^{2^N-1} \frac{1}{2^N} = 1$$

For other derived basis we use this discretized formula

$$\psi_{j,k}(n) = \sqrt{2^j}\psi(2^j(n - 2^{N-j}k)), 0 \leq j \leq N-1, k <= 2^j - 1$$

$$\begin{bmatrix} \psi_{0,0} & 0 & \cdots & 0 \\ \psi_{1,0} & \psi_{1,1} & \cdots & 0 \\ \psi_{2,0} & \cdots & \psi_{2,3} & 0 \\ \vdots & \vdots & \ddots & 0 \\ \psi_{N-1,0} & \psi_{N-1,1} & \cdots & \psi_{N-1,2^{N-1}-1} \end{bmatrix}$$

And a average basis as following

$$\phi(n) = \begin{cases} 0, & \text{if } n < 0 \\ \frac{1}{\sqrt{2^N}}, & \text{if } 0 \leq n < 2^{N-1} \\ \frac{1}{\sqrt{2^N}}, & \text{if } 2^{N-1} \leq n < 2^N \\ 0, & \text{if } n >= 2^N \end{cases}$$

there are

$$1 + \sum_{k=0}^{N-1} 2^k = 2^N$$

basis. which will be 2^N coefficients then every 2^N sequence $f(n)$ can be written as

$$f(n) = a\phi(n) + \sum_{j=0}^{N-1} \sum_{k=0}^{2^j-1} c_{j,k}\psi_{j,k}$$

Then each coefficients can be obtain by inner product.

$$a = \langle f(n), \phi(n) \rangle$$

$$c_{j,k} = \langle f(n), \psi_{j,k}(n) \rangle$$

The inner product of two sequence are define as follow.

$$\langle f(n), g(n) \rangle = \sum_{k=0}^{2^N-1} f(k)g(k)$$

the transformed sequence $\hat{f}(n)$ satisfies

$$c_{j,k} = \hat{f}(2^j + k)$$

$$a = \hat{f}(0)$$

3 Iamge pre-processing

3.1 Overview

3.1.1 Background and Objectives

In the fields of digital image processing and computer vision, standardizing image dimensions is a fundamental requirement in the preprocessing stage. This program aims to achieve the following core objectives:

1. Convert any input image into a grayscale image of size $\mathbf{2^N} \times \mathbf{2^N}$ pixels (where N is an integer specified by the user). This meets the need for uniform dimensions in specific scenarios (e.g., inputs for deep learning models, academic experiment comparisons). It also lays a foundation for subsequent wavelet transform applications, as standardized dimensions enable more efficient and accurate wavelet-based processing.

2. Support the export and visualization of image matrices. This facilitates in-depth image data processing for numerical analysis and machine learning tasks. It also provides a structured format for wavelet transform operations, simplifying the extraction of frequency domain features.

3.1.2 Dependency Library Description

The program relies on the following Python libraries. Their functions and installation commands are as follows:

3.2 Program Design and Implementation

3.2.1 Core Functional Modules

The program implements the complete process via the `convert_image` function, consisting of 5 core steps. These steps not only enable basic image conversion but also prepare images for potential wavelet transform applications:

Library Name	**Function Description**	**Installation Command**
Pillow (PIL)	Image reading, format conversion, scaling, and saving. Provides basic image processing capabilities essential for pre-wavelet-transform image transformation.	`pip install pillow`
NumPy	Matrix-based conversion and numerical operations for image data. Critical for representing images as matrices (a prerequisite for wavelet transform calculations).	`pip install numpy`
math	Built-in Python mathematical operations (used for power calculations to determine dimensions). Assists in standardizing image size, which impacts wavelet transform performance.	No additional installation required

Table 1: Dependency Library Details

1. **Image Reading and Grayscale Conversion**: Use `Image.open` from the Pillow library to open an image, then convert it to a single-channel grayscale image via `convert('L')` (pixel values range from 0 to 255, with 0 as black and 255 as white). Grayscale images simplify wavelet transform calculations by reducing data complexity while preserving essential structural information.

2. **Target Size Calculation**: Calculate the target output image side length as `target_size = 2 ** N` based on the user-input N (e.g., $N = 3$ yields an 8×8 pixel image). Standardized dimensions align with wavelets' multi-resolution analysis characteristics, enabling consistent decomposition and reconstruction.

3. **Aspect-Ratio-Preserving Scaling**: Compare the original image's width and height, dynamically compute the scaling ratio to ensure at least one side reaches the target size, and scale using the LANCZOS interpolation algorithm (a high-fidelity method that preserves image details). Maintaining details is critical for accurate wavelet transform results, as wavelets are sensitive to fine-scale features.

4. **Centered Cropping**: Compute cropping coordinates (left, top, right, bottom) and perform centered cropping on the scaled image to ensure a strict $\mathbf{2^N \times 2^N}$ output size. This step ensures image dimensions suit wavelet transform requirements (wavelet operations typically need consistent input sizes).

5. **Result Saving and Output**: Save the cropped image file and convert the image to a 2D matrix using NumPy, supporting matrix data saving and

console output. The matrix format is compatible with wavelet transform processing, enabling seamless integration of subsequent wavelet-based algorithms.

3.2.2 Complete Code Implementation

The following Python code (runnable as-is) performs basic image conversion and prepares images for wavelet transform via standardized matrix representation:

Code Key Logic Supplementary Explanation:

- **LANCZOS Interpolation**: Invoked via `Image.Resampling.LANCZOS`, it ensures high-fidelity scaling—critical for preserving wavelet-transform-relevant features (e.g., edges, textures).
- **Matrix Saving**: The `np.save` function stores the image matrix as a `.npy` file, enabling reuse in wavelet transform workflows (e.g., import into MATLAB/Python analysis scripts).

3.3 User Guide

3.3.1 Parameter Configuration

When calling `convert_image`, configure the following parameters:

Parameter Name	Type	Description
`image_path`	String	Input image path (e.g., `"input.jpg"`).
`output_path`	String	Output image save path (e.g., `"output.png"`).
`N`	Integer	Controls output size as $\mathbf{2^N} \times \mathbf{2^N}$ (e.g., $N = 5$ for 32×32). Critical for defining wavelet transform resolution.
`save_matrix`	Boolean	Whether to save the image matrix (default: `False`). Saves time in wavelet transform preprocessing.

Table 2: Function Parameter Description

3.3.2 Example Call

The following example demonstrates `convert_image` usage, producing wavelet-transform-ready images:

Running Process:

1. Enter N in the terminal (e.g., `3` for an 8×8 image). This N influences wavelet transform granularity (different sizes enable multi-resolution analysis).

2. The program converts the image, outputting size info and the matrix. The matrix serves as direct input for wavelet transform functions.

3. If `save_matrix` is enabled, the matrix saves as a `.npy` file for wavelet transform tasks.

3.4 Application Scenarios and Expansion Suggestions

3.4.1 Typical Application Scenarios

1. **Academic Research**: Standardizes image sizes for uniform input in image recognition, style transfer, etc. Wavelet transform can further extract multi-scale features, aiding algorithm research.

2. **Teaching Demonstration**: Visualizes RGB-to-grayscale conversion and matrix representation. Can be extended to show wavelet transform's frequency-domain decomposition.

3. **Data Preprocessing**: Prepares standardized image data for machine learning (e.g., MNIST-like datasets). Wavelet-based feature extraction enhances model input quality.

3.4.2 Function Expansion Suggestions

1. **Multi-Channel Support**: Add RGB processing to retain color information, enabling wavelet transform on color images.

2. **Batch Processing**: Automate multi-image conversion via folder traversal, streamlining large-scale wavelet transform workflows.

3. **Visualization Enhancement**: Integrate Matplotlib to compare original and converted images. Extend to visualize wavelet decomposition/reconstruction for intuitive validation.

3.5 Summary

This program converts images to $\mathbf{2^N} \times \mathbf{2^N}$ grayscale format, balancing practicality and extensibility. Leveraging Pillow and NumPy, it meets basic image processing needs and provides standardized input for deep learning (e.g., TensorFlow, PyTorch). By outputting wavelet-transform-compatible matrices, it supports advanced image analysis. It serves as a foundation for entry-level image processing practice, academic experiments, and wavelet-integrated preprocessing pipelines.

4 Fractal Approach

For a mother wavelet $\psi(x)$ satisfied following property.

$$\langle \psi, \psi \rangle = 1$$

$$\int_{-\infty}^{\infty} \psi(x)dx = 0$$

And the family

$$\psi_{j,k} = \sqrt{2^j}\psi(2^j(x - 2^{-j}k))\, j, k \in \mathbb{Z}$$

For any function satisfied the following property

$$\int_{-\infty}^{\infty} f(x)dx = 0$$

We can decomposed it in to the form

$$f = c_{j,k}\psi_{j,k}$$

And the coefficients can obtained by the following inner product.

$$\langle f, \psi_{j,k} \rangle = c_{j,k}$$

For a function restricted on $[0, 1]$ we can only use

$$\psi_{j,k} = \sqrt{2^j}\psi(2^j(x - 2^{-j}k))\ k < 2^j\, j, k \in \mathbb{N}$$

To decomposed.

And even more for a discrete function restricted on $[0, 1]$, with 2^n sample points we can only use

$$\psi_{j,k} = \sqrt{2^j}\psi(2^j(x - 2^{-j}k))\ k < 2^j \leq 2^n\, j, k \in \mathbb{N}$$

To decomposed.

The inverse process to generate all $\psi_{j,k}$ for the discrete case I want to propose a different approach.

Say I can have a function

$$F : \mathbb{R}^2 \to \mathbb{R}^2$$

and a base vector

$$[v_0, v_1]$$

then the next basis would obtain by the following

$$F([v_0, v_1]) = [w_0, w_2]$$

the next basis is

$$[w_0, w_1, w_2, w_3]$$

To keep every iteration to be orthogonal We need to restrict F as following.

$$w_1 = -\frac{v_0 w_0}{v_1}$$

$$w_3 = -\frac{v_0 w_2}{v_1}$$

Then every basis need to be nomalized to fit the property

$$\langle \psi, \psi \rangle = 1$$

Then using

$$[v_0, v_1]$$

and F we can generate all the basis.

Do the nomal decomposition we will get coefficients $c_{j,k}$ We know $c_{j,k}c_{j,k} = C$ If we want all the information constrain in one coefficients we need to make $c_{j,k}c_{j,k}c_{j,k}c_{j,k}$ as large as possible.

This is a parameter optimization problem

$$\max_{F,[v_0,v_1]} c_{j,k}c_{j,k}c_{j,k}c_{j,k}$$

Then we can just transfer F and $[v_0, v_1]$

4.1 a simple prompt

To be more clear of the notations we use, we have $i \in \{1, 2, ..., 10\} = N$ And the drones are ignored of its flying height, which the position vector can be a 2d vector note it as $\vec{d_i}$ And so the velocity and acceleration we denote as $\vec{v_i} = \frac{d\vec{d_i}}{dt}$ and $\vec{a_i} = \frac{d\vec{v_i}}{dt}$ I want to prompt a f so that it can form a cirle.

$$\vec{f_i} = m_i(\sum_{\forall k \neq i, \|\vec{d_i} - \vec{d_k}\| \leq R} (\frac{\vec{d_i} - \vec{d_k}}{\|\vec{d_i} - \vec{d_k}\|^3}) + (\frac{\vec{d_{t(i)}} - \vec{d_i}}{\|\vec{d_{t(i)}} - \vec{d_i}\|} - v_i))$$

This model is easy to explain, the first term is just a inverse square propell force, the second term is make the velocity quickly approch a set direction the $t(i)$ is just a randomly choosed target drone other than i that is $t(i) \in N, t(i) \neq i$

This formula can be rewrite without physical term as follow.

$$\vec{a_i} = \sum_{\forall k \neq i, \|\vec{d_i} - \vec{d_k}\| \leq R} (\frac{\vec{d_i} - \vec{d_k}}{\|\vec{d_i} - \vec{d_k}\|^3}) + (\frac{\vec{d_{t(i)}} - \vec{d_i}}{\|\vec{d_{t(i)}} - \vec{d_i}\|} - v_i)$$

separately view this is combined by two independent force

$$(\vec{a_i})_{target} = (\frac{\vec{d_{t(i)}} - \vec{d_i}}{\|\vec{d_{t(i)}} - \vec{d_i}\|} - v_i)$$

$$(\vec{a_i})_{propell} = \sum_{\forall k \neq i, \|\vec{d_i} - \vec{d_k}\| \leq R} (\frac{\vec{d_i} - \vec{d_k}}{\|\vec{d_i} - \vec{d_k}\|^3})$$

4.2 a simple prompt:simulation

4.2.1 Four drone case:derivation

This case just choose $N = \{1, 2, 3, 4\}$ and don't allow $t(t(i)) = i$ which definitely form a three element loop and a dangling drone.

We have a (4,2)-tensor $\vec{d_i}$ and two other (4,2)-tensor $\vec{v_i}$ and $\vec{a_i}$ The initial points are randomly choosed in Uniformly $[0, 1] \times [0, 1]$

choose a time increment dt and the simulation update formula is simple to write

just as follow

$$\begin{bmatrix} \vec{d_{n+1}}_i \\ \vec{v_{n+1}}_i \end{bmatrix} = \begin{bmatrix} \vec{d_{n}}_i + \vec{v_{n}}_i dt \\ \vec{v_{n}}_i + (\sum_{\forall k \neq i, \|\vec{d_{n}}_i - \vec{d_{n}}_k\| \leq R} (\frac{\vec{d_{n}}_i - \vec{d_{n}}_k}{\|\vec{d_{n}}_i - \vec{d_{n}}_k\|^3}) + (\frac{\vec{d_{n}}_{t(i)} - \vec{d_{n}}_i}{\|\vec{d_{n}}_{t(i)} - \vec{d_{n}}_i\|} - v_{ni}))dt \end{bmatrix}$$

simple Euler method.

4.2.2 Four drone case:code & result

the computational code are [1, FourDroneCase] . the results are shown by the following pictrues which generated by the code. The video are [2, sample1-video] and [3, sample2-video] and more other in the same folder on github.

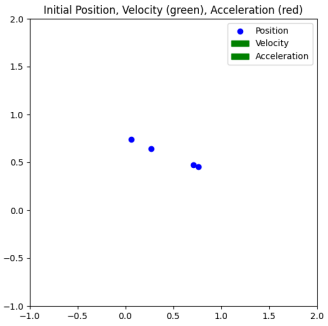

Figure 1: sample1 randomly initial position

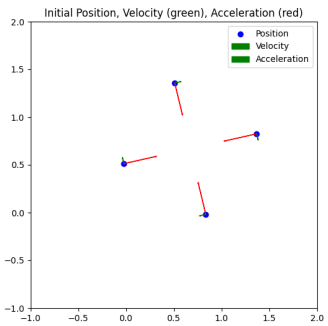

Figure 2: sample1 after a period of time

4.2.3 Ten drone case:derivation

undergoing

4.2.4 Ten drone case:result

undergoing

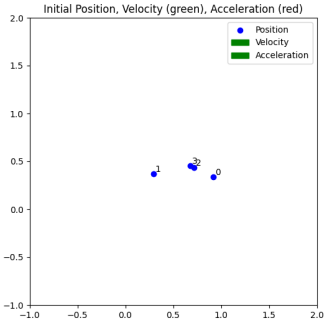

Figure 3: sample1 randomly initial position

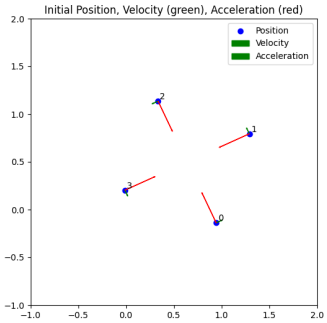

Figure 4: sample2 after a period of time

4.3 some analysis why it will have a stability property

4.3.1 the terminate radius R

undergoing

4.3.2 the terminate center O

undergoing

4.3.3 graph theory part

The $t(i)$ forms a graph which have n points and n oriented edges, this forms a tree with a extra edges, and this case It will obviously form a Unicyclic Graph.

Which is a tree if we treat all the point on the loop as the same point.

4.4 target distance method

undergoing

4.5 target distance method:simulation

undergoing

5 Zinan Su's approach

5.1 notations & equations

safe collide radius is d_s

$$\sigma = 2d_s$$

NUM is the total number of the drones. And then we want the following dynamic system

$$\begin{bmatrix} \frac{d\vec{p_i}}{dt} \\ \frac{d\vec{v_i}}{dt} \end{bmatrix} = \begin{bmatrix} \vec{v_i} \\ \vec{a_i} \end{bmatrix}$$

circle origin is a function

$$c = \frac{1}{NUM} \sum_{k=1}^{NUM} p_k$$

Four constants.

$$k_p =$$

$$k_d =$$

$$k_v =$$

$$k_r =$$

$$R^* = \frac{1}{NUM} \sum_{k=1}^{NUM} p_k(0) - c(0)$$

$$v_d = \frac{1}{NUM} \sum_{k=1}^{NUM} \|v_k(0)\|$$

and

$$r_i = p_i - c$$

$$d_i = \|r_i\|$$

$$\hat{r_i} = \frac{r_i}{d_i}$$

$$\hat{\theta_i} = \mathbb{M}_\theta r_i$$

$$\mathbb{M}_\theta = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$v_{i\parallel} = \hat{r_i} \cdot v_i$$

$$v_{i\perp} = \hat{\theta_i} \cdot v_i$$

$$U(r) = k_r e^{-\frac{r}{2\sigma^2}}$$

$$U_{ij} = U(\|p_i - p_j\|)$$

$$\vec{u_{i1}} = [-k_p(d_i - R^*) - k_d v_{i\parallel}]\hat{r_i}$$

$$\vec{u_{i2}} = [-k_v(v_{i\perp} - v_d)]\hat{\theta_i}$$

$$\vec{u_{i3}} = \sum_{\forall k \neq i} (-\nabla_{p_i} U_{ij})$$

$$\vec{u_i} = \vec{u_{i1}} + \vec{u_{i2}} + \vec{u_{i3}}$$

References

[1] Zhang Jinrui. Fourdronecase.py. https://github.com/jerzha40/2025_exchange_at_universityofalberta/blob/main/DroneInCircleEXP/FourDroneCase.py. Accessed: 2025-07-20.

[2] Zhang Jinrui. sample1-video. https://github.com/jerzha40/2025_exchange_at_universityofalberta/blob/main/DroneInCircle/fig/sample1/0001-18255.mp4. Accessed: 2025-07-20.

[3] Zhang Jinrui. sample2-video. https://github.com/jerzha40/2025_exchange_at_universityofalberta/blob/main/DroneInCircle/fig/sample2/0001-18994.mp4. Accessed: 2025-07-20.