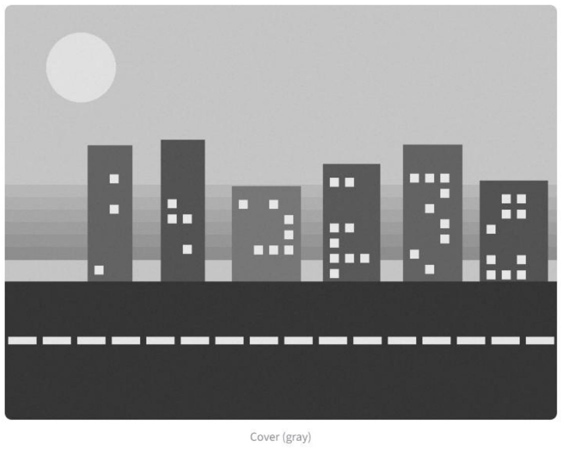
Wavelets: Meta information transport



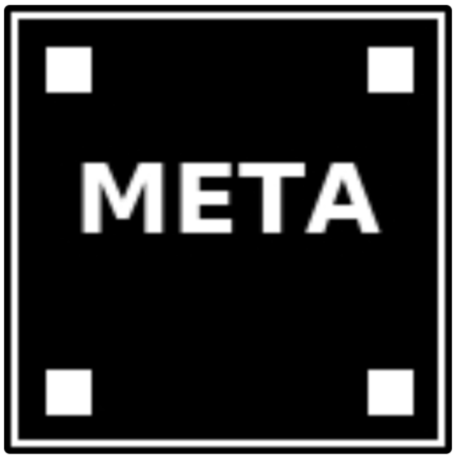
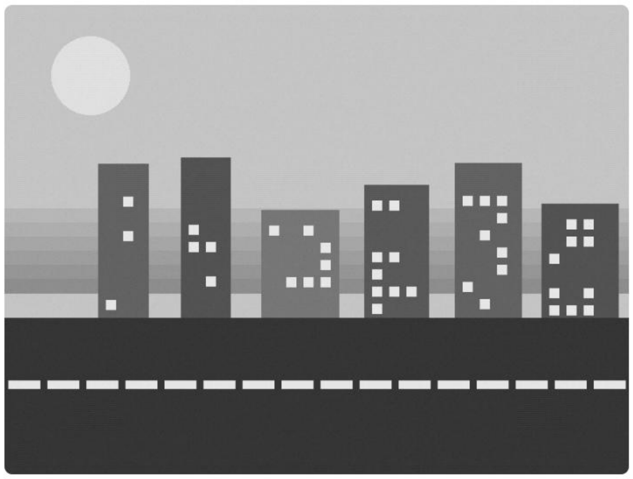


Fig.1 Cover image

Fig.2 Payload image



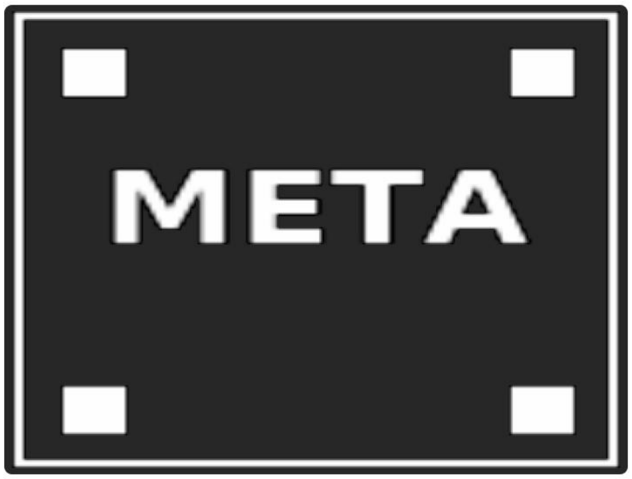


Fig.3 Marked image (after embedding)

Fig.4 Absolute difference (marked – cover image)

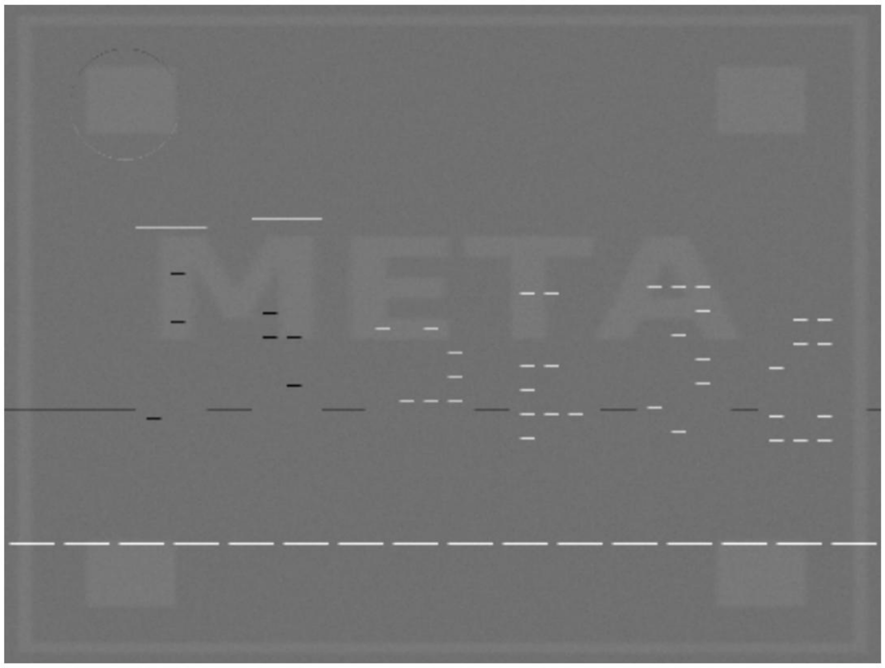
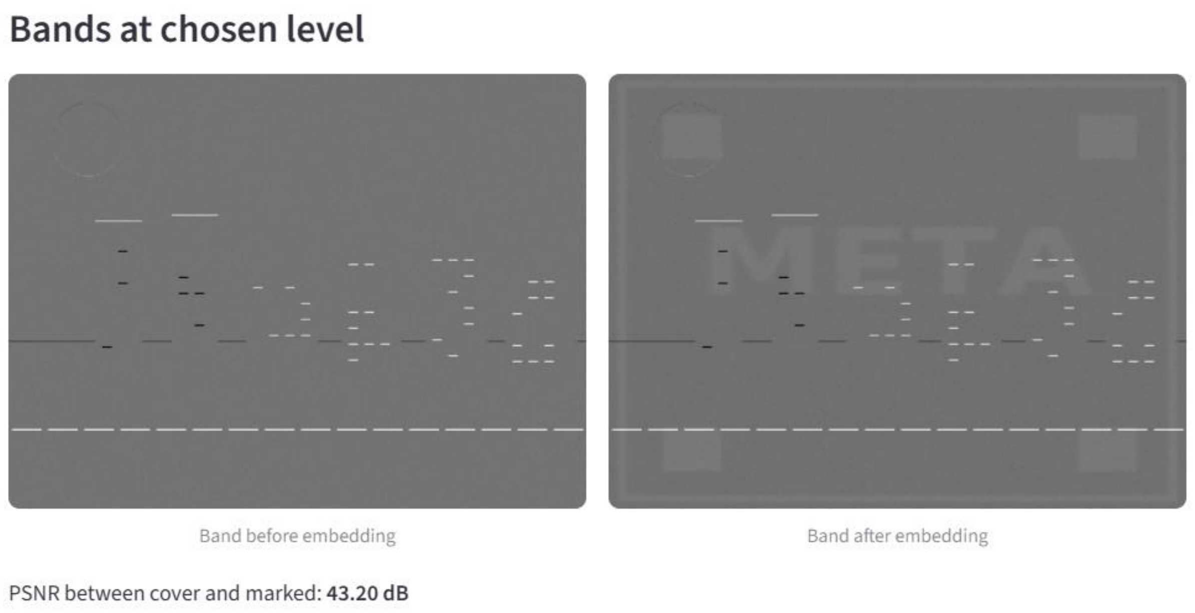


Fig.5 Extracted band (normalized)



Bands at chosen level

Band before embedding

Band after embedding

PSNR between cover and marked: **43.20 dB**

Fig.6 Wavelet detail band at the chosen level after embedding the payload (normalized for display).

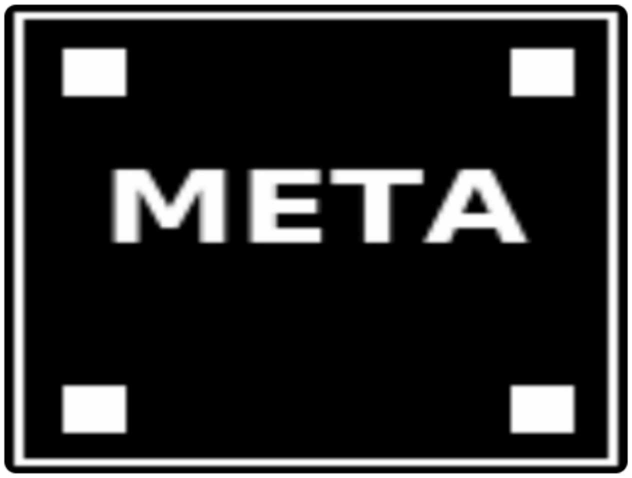
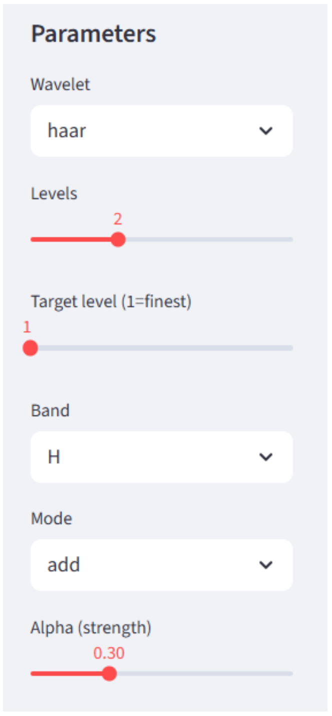


Fig7. Payload resized to band size


Parameters

Wavelet

haar

Levels

2

Target level (1=finest)

1

Band

H

Mode

add

Alpha (strength)

0.30

Fig.8 Parameter settings used for the experiment (haar wavelet, 2 levels, target level 1, horizontal band H, add mode, α = 0.3).

Discussion of results and code:

In this experiment I use a 2D wavelet transform to hide “meta information” (a payload image) inside one detail layer of a cover image. The cover image I use is a simple city scene, shown in **Fig.1**, and the payload is the small black-and-white “META” square in **Fig.2.** Both images are converted to grayscale before I do anything else, because it is easier to work with one channel when testing wavelets.

In all the screenshots and figures I use the same parameter settings as shown in **Fig.8**. I chose the haar wavelet and two decomposition levels, and I embed the payload in the finest horizontal detail band (target level 1, band H) using the “add” mode with $\alpha = 0.3$. When I tried smaller α values, the marked image looked even more similar to the cover, but the META pattern in the extracted band was harder to see. With larger α the payload became very clear in the band, but it also started to create visible artefacts in the image. I think $\alpha = 0.3$ is a reasonable compromise between hiding the payload and still being able to recognize it in the wavelet domain.

The marked image that comes out after embedding the payload is shown in **Fig.3**. When I compare **Fig.1** and **Fig.3** by eye, they look almost identical. This is what I want: the cover should still look normal, even though the payload information has been stored somewhere inside the wavelet coefficients. To see that there is a change, I show the absolute difference between the cover and the marked image in **Fig.4**. Here the dark areas mean “no change” and the brighter spots show where the wavelet band has been modified. The difference image is much weaker than the original, which tells me that the embedding only causes a small distortion of the cover image. The PSNR value printed by the app is also relatively high, which supports the visual impression that the marked image is close to the original.

To check that the meta information really has been transported into the wavelet domain, I do a second wavelet transform on the marked image and look at the chosen detail band again. **Fig.5** shows the extracted band after normalization to the range [0,1]. When I compare this to **Fig.7**, which is the payload resized to the same band size, I can clearly recognize the META pattern and the four white corners. **Fig.6** shows the same wavelet detail band after embedding the payload, again normalized for display. The fact that the structure of the payload appears inside this band, while the marked image in **Fig.3** still looks like the city scene, indicates that the method is working as intended. The meta information is transported in the transform coefficients instead of in the visible pixels.

The code for the app is written in Python with **Streamlit** as a simple GUI. First, I load the images with **load_image_to_gray**, which converts any uploaded image to a grayscale NumPy array with values between 0 and 1. If the user does not upload a payload image, I generate a small text image from the input string using **text_to_image**. After that I use **PyWavelets (pywt)** in the function dwt2_levels to compute a multi-level 2D DWT of the cover image. This gives me one approximation image and a list of detail bands (horizontal, vertical, diagonal) at different levels. The inverse transform is handled by idwt2_levels.

The main embedding step happens in the function **embed_payload**. Here I first pick the detail band that the user has selected in the GUI (level and H/V/D band). The payload image is resized to the same shape as this band in **resize_payload_to_band.** To avoid the payload being too strong or too weak, I normalize it to have roughly the same mean and standard deviation as the original band. Then I either add the payload to the band ("add" mode) or blend it with the original band ("replace" mode), scaled by the parameter alpha. A small alpha means that the payload is hidden more gently, and a larger alpha means it becomes easier to see in the band but risks creating more artefacts in the marked image. After modifying the band, I put all the coefficients back together and call the inverse DWT to get the marked image.

For extraction I use the function **extract_band**. It simply runs the DWT on the marked image with the same wavelet and number of levels, selects the same detail band and normalizes it for display. This is what I show as the extracted band in **Fig.5**. Finally, I measure the distortion using a small **psnr** function that computes the peak signal-to-noise ratio between the original cover and the marked image. All the images (cover, payload, marked, difference, and the different bands) are displayed in **Streamlit** so I can easily see how the embedding behaves when I change the parameters.

Overall, the results and the code together show that the app does what the assignment asks for. I can "transport" meta information by exchanging or modifying exactly one wavelet detail layer, while keeping the cover image visually almost unchanged.