DWT_QR_STEGO Project

Group #11

CS4463

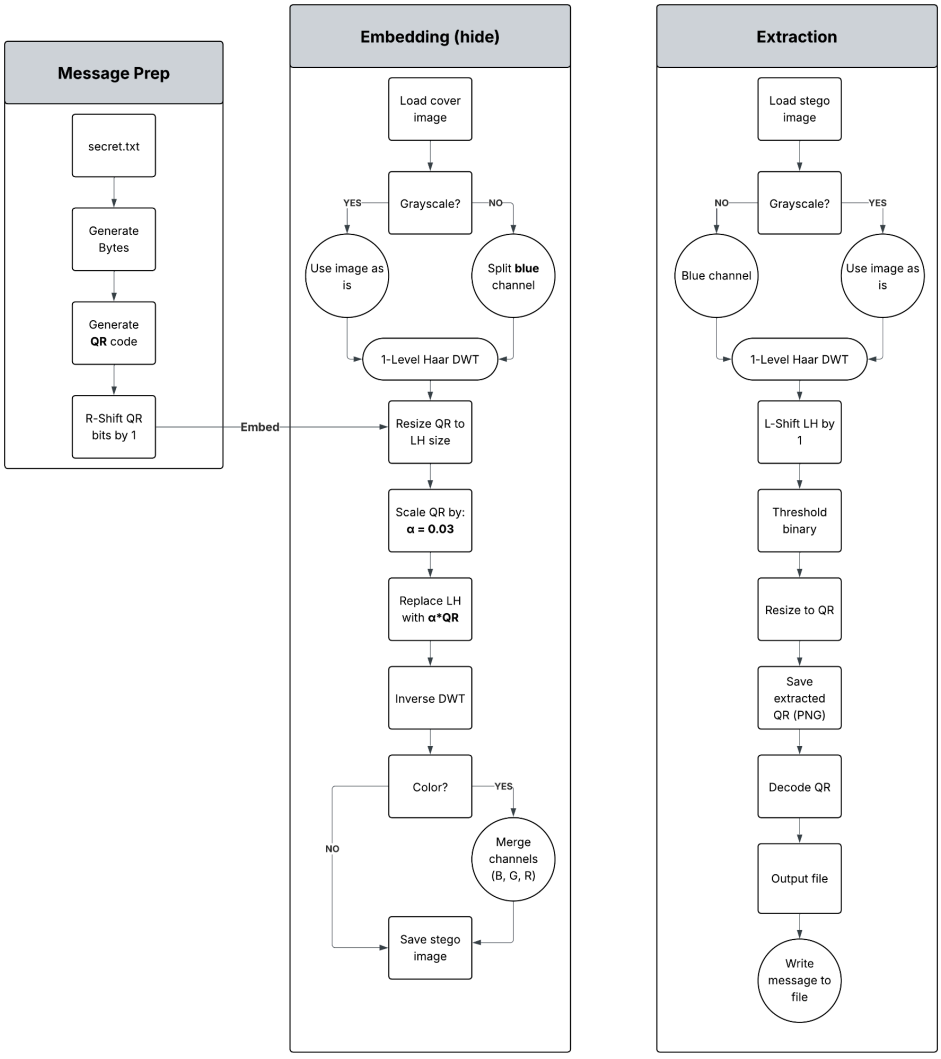
Luke Alvarado, Preston Mumphrey, Carlos Osuna, Aradhnna Reddy.

8/7/2025

Project description:

DWT_QR_STEGO is a steganographic program that hides text as a QR image into a chosen cover and allows extraction of the same image to reveal the text. This project is based on the academic research paper titled “DWT Based QR Steganography” where we build a model that performs Discrete Wavelet Transform (DWT) based steganography technique. This project securely hides a secret message by first converting it from a .txt file into a QR code, then embedding that QR code into the LH subband of a cover image using the Haar DWT. The program supports two modes: "hide" mode creates a stego image with the provided message as an embedded QR, while "extract" mode extracts, saves, and decodes the QR to recover the original message.

Functional block diagram:



Step by Step:

Message prep:

Step 1: Message is chosen

```
data = "my secret messages"
```

Step 2: Generate bytes/QR setup

```
qr = qrcode.QRCode(
    version=1,  # size of the QR code: 1 (21x21) to 40 (177x177)
    error_correction=qrcode.constants.ERROR_CORRECT_H,  # higher = more robust
    box_size=10,  # size of each box in pixels
    border=4,  # border in boxes (minimum is 4)
)
```

Step 3: Generate QR code

```
qr.add_data(data)
qr.make(fit=True)

# Create image
myqr = qr.make_image(fill_color="black", back_color="white")
```

Step 4: Right shift QR

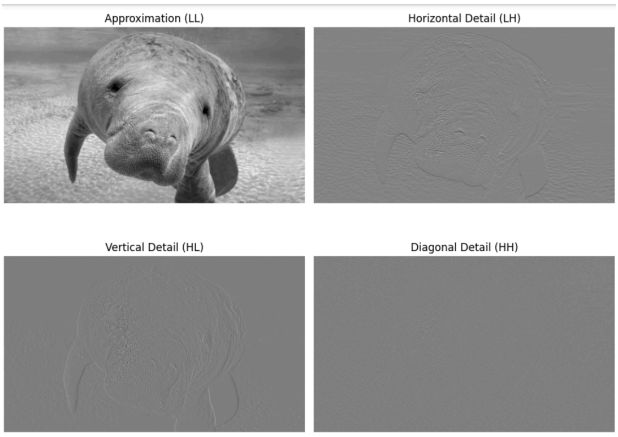
```
myqr_array = np.array(myqr.convert('L'))
shifted_qr = np.right_shift(myqr_array, 1)
```

Embedding:

Step1: Load cover image

Step 2: Read as grayscale or color

Step 3: 1-level Haar DWT



Step 4: Resize QR and place in LH band

Step 5: Inverse DWT

As can be seen in the top right of when the alpha is .75 and 1, the QR is visible

Extraction:
Step 1: Load and read image
Step 2: Extract and bit shift LH subband

Step 3: Threshold Binary

Step 4: Resize QR
Step 5: Read and Save

Python packages:

sys: sys is a python package that gives us access to system-specific parameters and functions to help navigate the python interpreter.
Related code: subprocess.check_call([sys.executable, "-m", "pip", "install", package])

subprocess: subprocess allows us to implement new processes, connect to the input/output/error pipes of new processes, and get their return codes.
Related code: subprocess.check_call([sys.executable, "-m", "pip", "install", package])

argparse: This is a parser for command-line options, argument, and subcommands. Allows implementation of basic command line applications.
Related code: parser = argparse.ArgumentParser(

os: os accounts for access to various operating system functionalities that are usually built into the interface.
Related code: _, ext = os.path.splitext(input_path)

secrets: secrets generates random numbers that are cryptographically strong, especially useful when needed to generate random passwords, tokens, and authentication codes.
Related code: data_hex = secrets.token_bytes(128).hex()

numpy: NumPy is a computing library for mathematical and scientific programs, usually used for arrays, matrices, or linear algebra.
Related code: return np.right_shift(arr, 1)

cv2: cv2 is an interface that allows for working with image and video processing functions.
Related code: cover = cv2.imread(cover_path, cv2.IMREAD_GRAYSCALE)

pywt: pywt is a wavelet transformation software typically used in signal and image processing tasks.
Related code: LL, (LH, HL, HH) = pywt.dwt2(cover, 'haar')

qrcode: qrcode generates QR codes which are then able to store various forms of data.
Related code: qr.add_data(data)

PIL: Python Imaging Library is an image processing package that allows for opening, editing, and saving images in various file formats.
Related code: pil_qr = Image.fromarray(qr_img)

pyzbar: pyzbar enables the use of zbar to read and decode barcodes and QR codes from images.
Related code: decoded = decode(pil_qr)

Discussion of algorithm modifications:

During the coding process, some changes were made from the paper's outline of the method. Most of these differences revolve around resizing, as some tools available in python are more consistent and more direct than MATLAB's, which is what the writers of the paper used. These tools include numpy and cw2, and one major difference is that instead of using AND and compliment to zero out the LH array (you could just use XOR to reduce it to one step) and then using AND to combine it with the QR code, we can just use cw2 directly to resize the QR code to the dimensions of LH, ready for inverse DWT. One slight modification, though it comes from a lack of explanation from the paper, is in color images, where dwt has to be taken on a specific color component rather than the whole image, so arbitrarily the blue channel was selected to be subject to hiding. Lastly, an alpha factor was introduced as using the raw QR intensity values proved to be far too perceptible. As QR values are either 255 or 0, an alpha factor of .03 changes the values to either 0 or 8, which is rather imperceptible in the cover image. The step of changing intensity values in extracting is slightly modified to account for this, where if the value is 0 it stays so, and if not it is set to 255. The ability to save a bmp to a png file in the steganography process is also available to the user, but as discussed later, this results in poorer performance yet still readable in most every case.

Security:

The security of this method comes from the application of DWT in this method. The reason a QR image is used instead of pure text in image representation is that the pure black and white of a QR code can be scaled to levels of imperceptibility, while a byte representation of text cannot, and as shown earlier, the lack of scaling, represented by the alpha of 1.0, is very perceptable, and would be even more so if other colors are present. Imperceptibility can be reached with QR codes in DWT, preventing visual detection. Detection using histograms is discussed later, but in general the statistical properties of text are not present nor detectable, though some altering can be depending on the cover. In terms of preventing destruction, DWT embeds the QR into the LH band, which means that when embedding data in DWT coefficients, the frequency information is changed, not raw pixels. These coefficients affect broad areas of the image, so the changes are not concentrated in one place. This gives the method natural redundancy meaning that small losses won't wipe out all the embedded info. Protection against extraction is held in the fact that the specific transform and wavelet function is needed as the coefficients are modified according to the specifications of this set of mathematical operations, and if using a different wavelet or transform, the frequency changes are likely to distort the QR image to great extents, and even if extracted, a bit shift left is needed. If the LH subband is successfully extracted, there is still a need to set the 8 value bits to 255 to fully reveal the QR and have it be readable, so without the source code of this project, extraction proves the hardest level of failure to achieve.

Capacity:

The capacity for this method proves relatively low. The highest version of QR codes available is version 40, whose max capacity is 1,277 bytes. This is because of QR redundancy, where QR codes provide levels of error correction and robustness. This means that the program is able to scan QR codes no matter the dimensions of the cover image along with remaining relatively consistent in results, but this is traded off for capacity.

Robustness:

As mentioned in the above section, the robustness of the chosen method with a few caveats is what the method performs well in. The robustness comes from in part hiding in the LH subband rather than the HH subband, which is higher frequency information that is more volatile and subject to more change on slight modifications to the image. The other factor of robustness comes for the QR format, which even at the lowest level of error correction, Reed–Solomon error correction, redundant data layout, and finder & alignment patterns are present allowing the data to survive transformations in both embedding and transmission.

Performance:

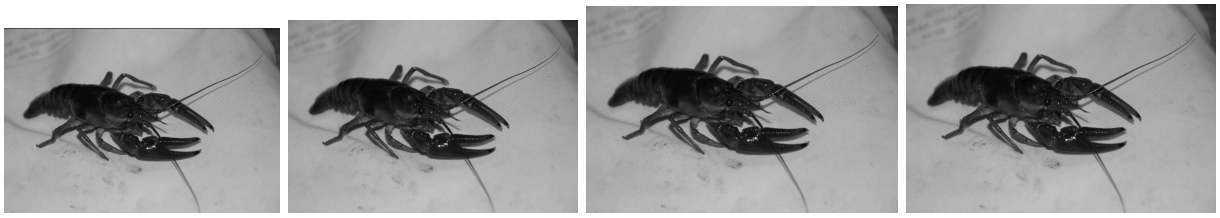
The following performance cases take an original image displayed on the left, with the next three images being the stego-images hiding sample.txt, random, and max.txt respectively.

Sample.txt contains the text “This is short sample text” which is 25 bytes; random is an option in the code to hide 128 random bytes; max.txt is a txt file of 1,277 bytes, the max this program was able to embed. All byte histograms are shown beneath the respective images, and for all the stego-images, the extracted QRs as well as the MSE and PSNR compared to the original are shown. MSE and PSNR was calculated by the following code, which compares the change of color information of the images to the original, where file type is of no effect in the performance metric calculations, though file type may be of effect in use of the method itself.

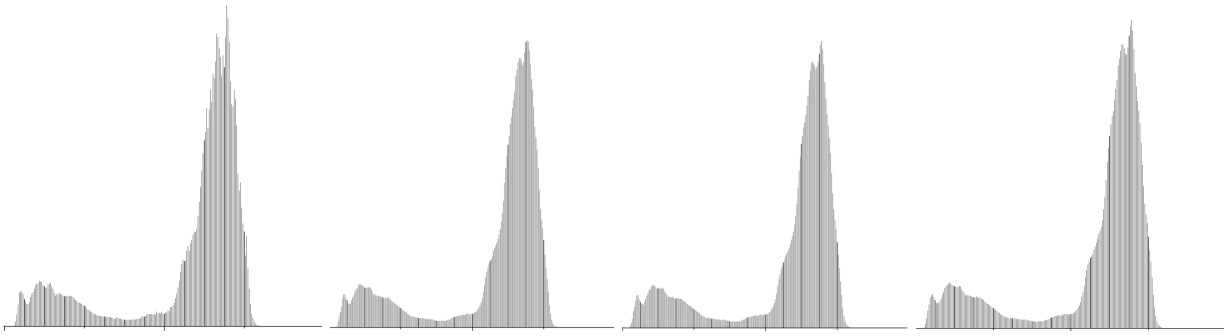
```
cover_np = np.array(cover, dtype=np.float64)
stego_np = np.array(stego, dtype=np.float64)

# Calculate MSE
mse = np.mean((cover_np - stego_np) ** 2)
print(f"MSE: {mse}")

# Calculate PSNR
if mse == 0:
    print("PSNR: Infinity (images are identical)")
else:
    psnr = 10 * math.log10((255 ** 2) / mse)
    print(f"PSNR: {psnr:.2f} dB")
```

MSE and PSNR calculation code

#1 grayscale bmp

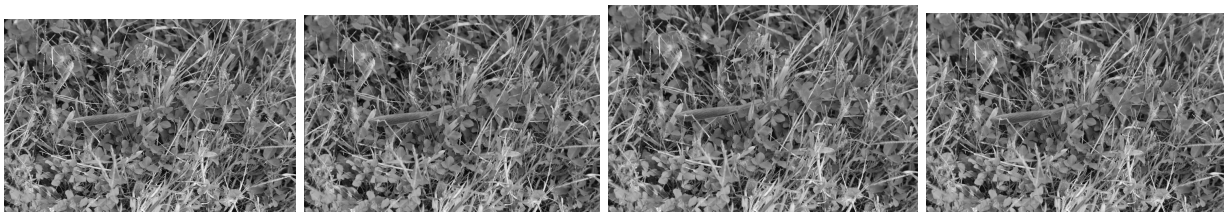


Name:	crawdad.bmp	crawdad_sample.bmp	crawdad_rnd.bmp	crawdad_max.bmp
Entropy:	6.699	6.706	6.704	6.704

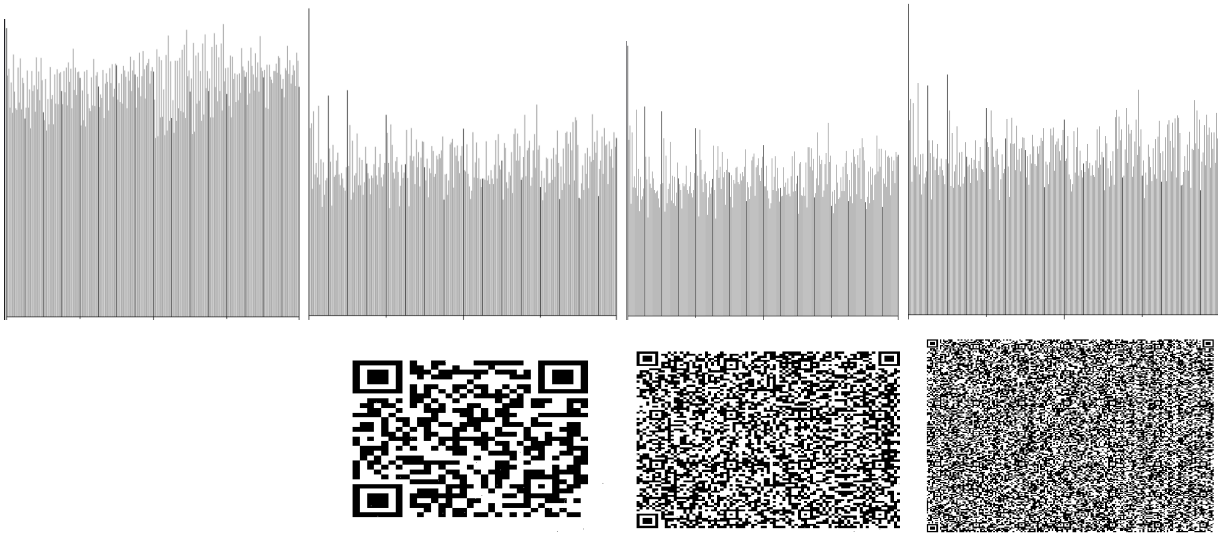


MSE:	7.358	6.959	6.801
PSNR:	39.46	39.71	39.81

#2 grayscale png



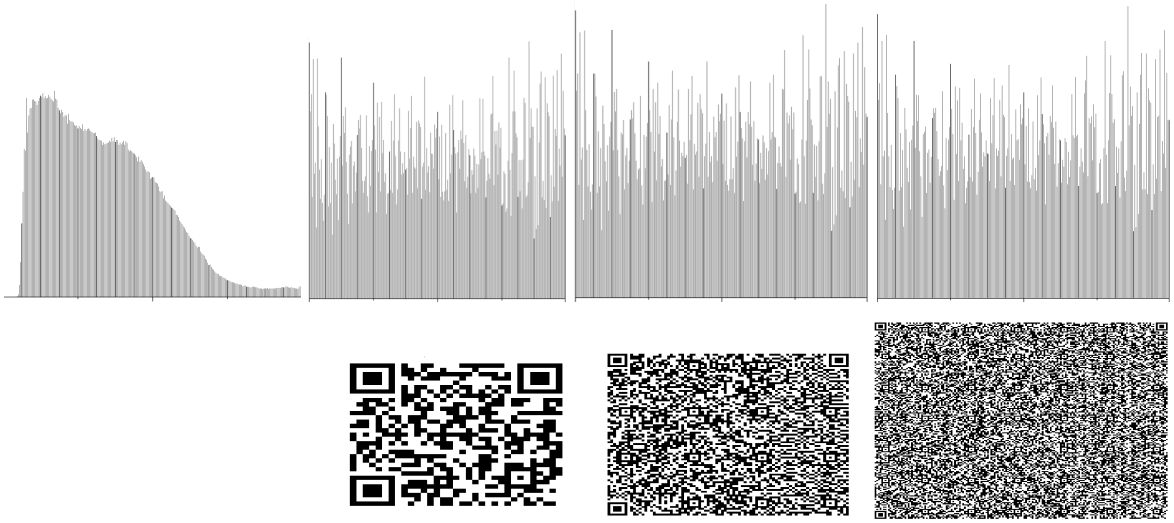
Name:	critter.png	critter_sample.png	critter_rnd.png	critter_max.png
Entropy:	7.993	7.982	7.982	7.983



MSE:	58.708	58.422	58.188
PSNR:	30.44	30.46	30.48

#3 grayscale bmp to png

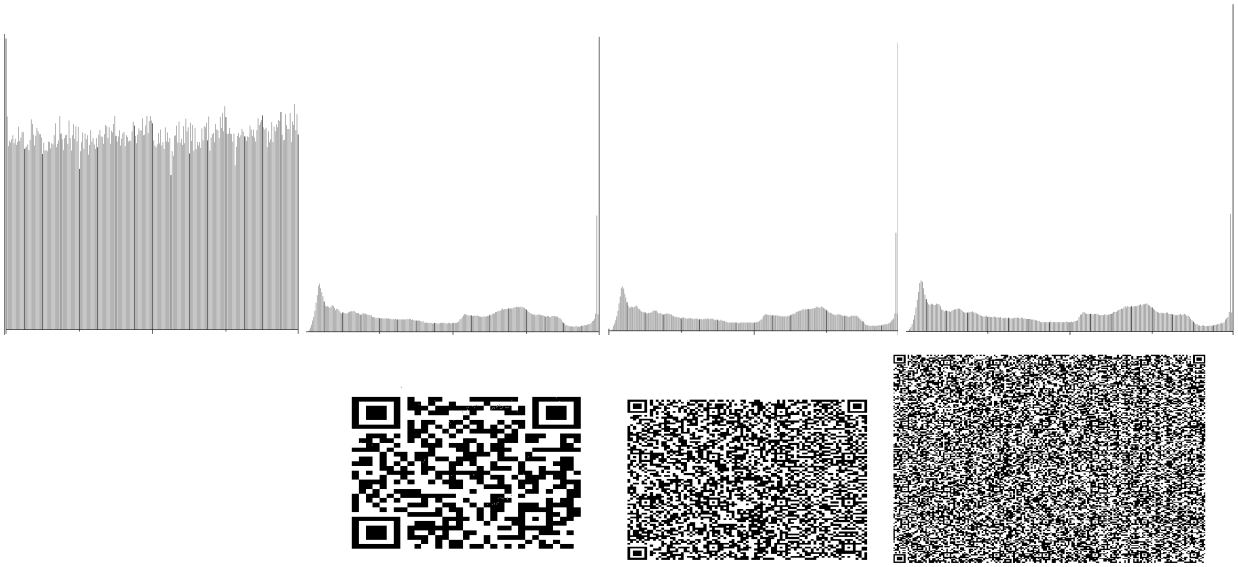
Name:	owl.bmp	owl_sample.png	owl_rnd.png	owl_max.png
Entropy:	7.462	7.945	7.945	7.947



MSE:	13.765	13.412	13.251
PSNR:	36.74	36.86	36.91

#4 grayscale png to bmp

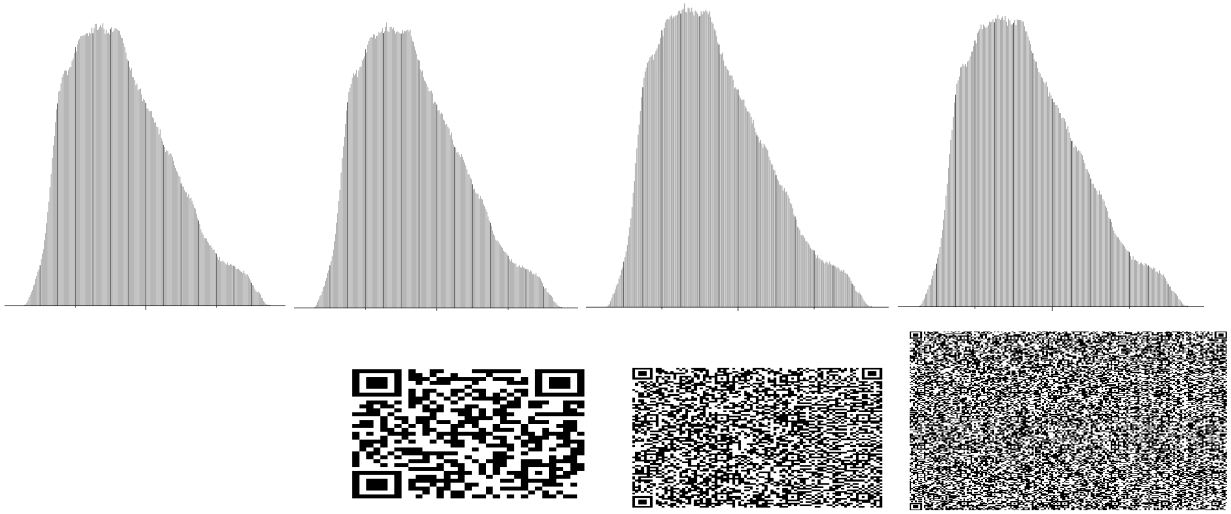
Name:	flowers.png	flowers_sample.bmp	flowers_rnd.bmp	flowers_max.bmp
Entropy:	7.997	7.650	7.646	7.647



MSE:	22.317	21.942	21.876
PSNR:	34.64	34.72	34.73

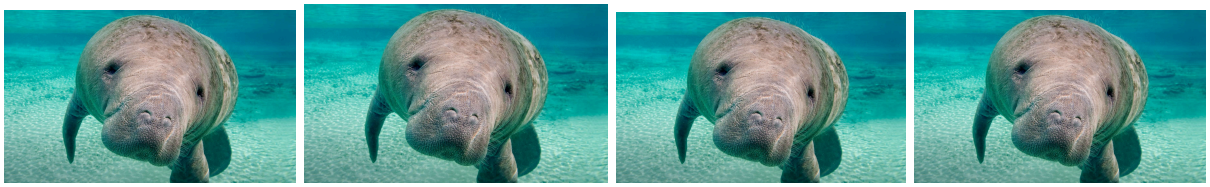
#5 color bmp

Name:	Mantis.bmp	mantis_sample.bmp	mantis_rnd.bmp	mantis_max.bmp
Entropy:	7.412	7.411	7.411	7.411

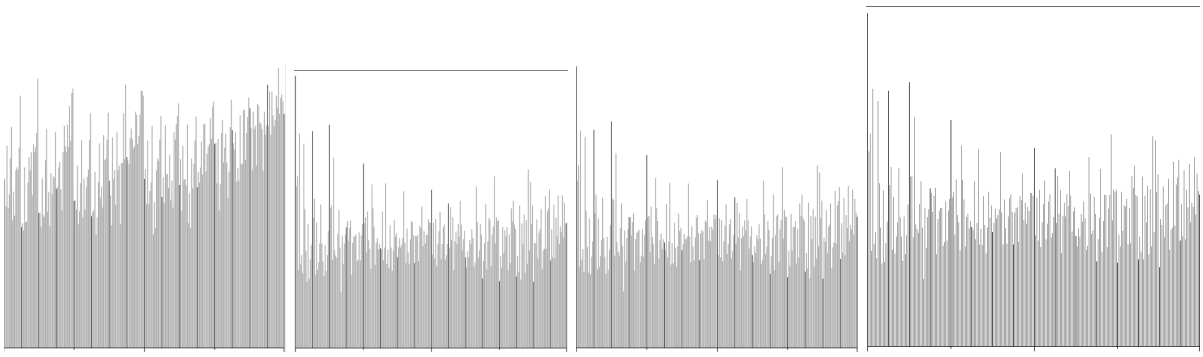


MSE:	6.012	5.876	5.832
PSNR:	40.34	40.44	40.47

#6 color png example 1



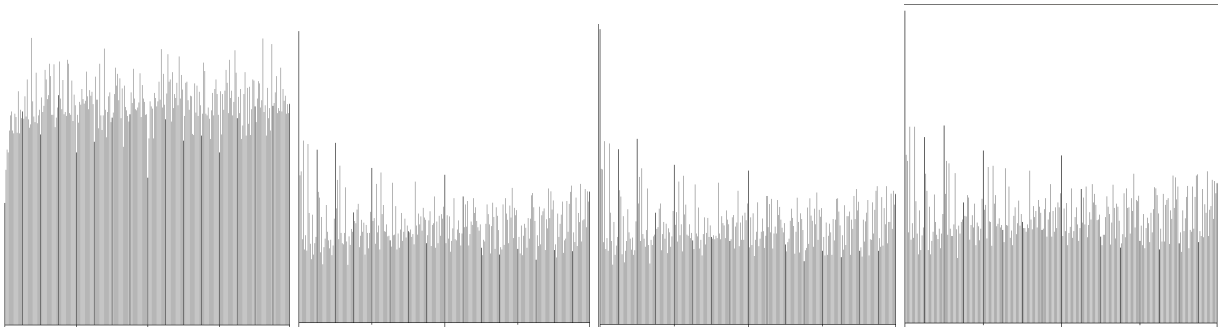
Name:	Manatee.png	manatee_sample.png	manatee_rnd.png	manatee_max.png
Entropy:	7.971	7.956	7.957	7.957



MSE:	6.551	6.442	6.395
PSNR:	39.97	40.04	40.07

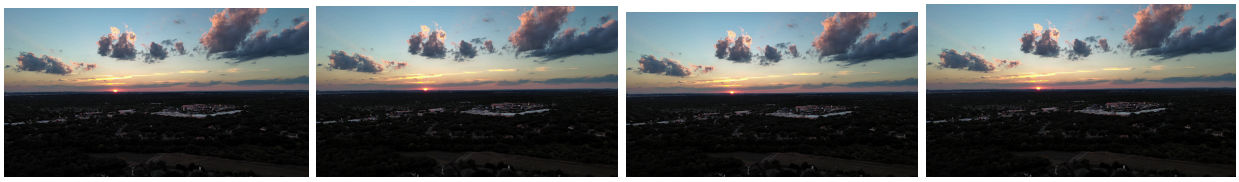
#7 color png example 2

Name:	football.png	football_sample.png	football_rnd.png	football_max.png
Entropy:	7.992	7.959	7.957	7.960

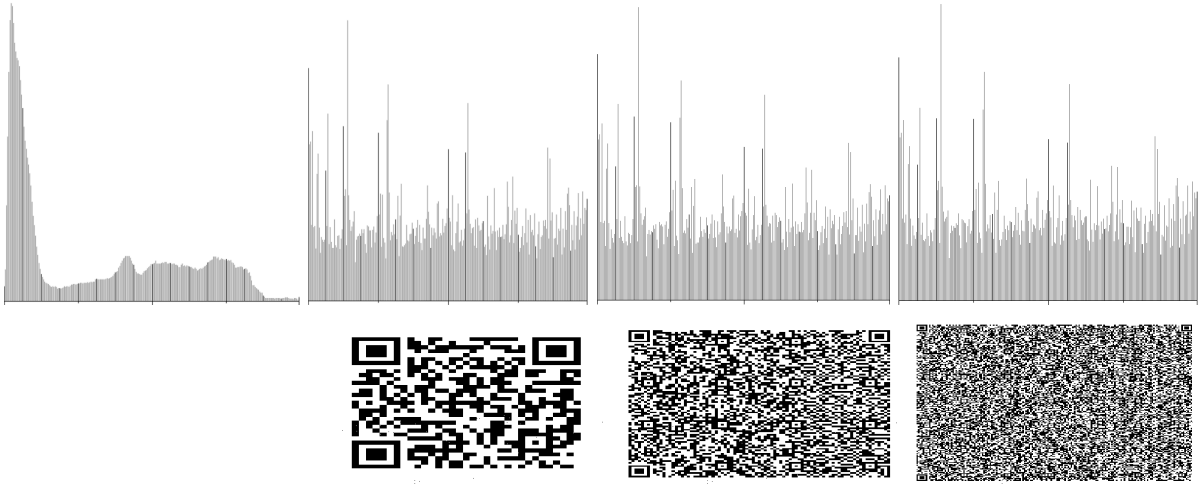


MSE:	23.912	23.859	23.845
PSNR:	34.34	34.35	34.36

#8 color bmp to png



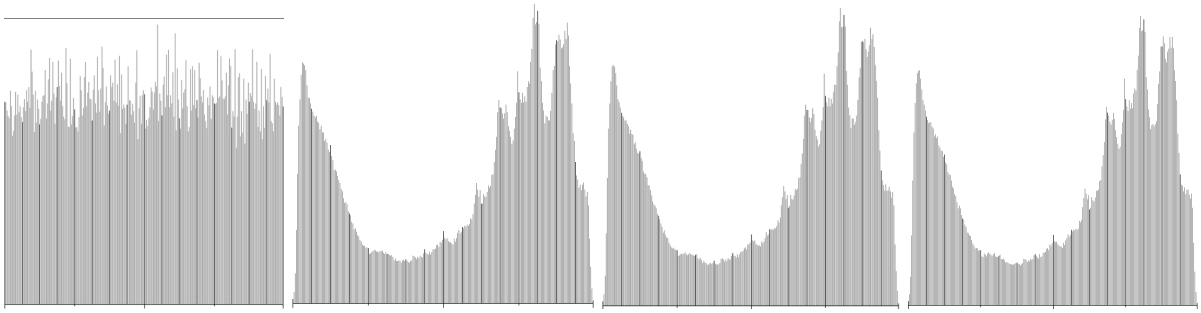
Name:	sky.bmp	sky_sample.png	sky_rnd.png	sky_max.png
Entropy:	7.268	7.905	7.907	7.911



MSE:	4.732	4.612	4.576
PSNR:	41.38	41.49	41.53

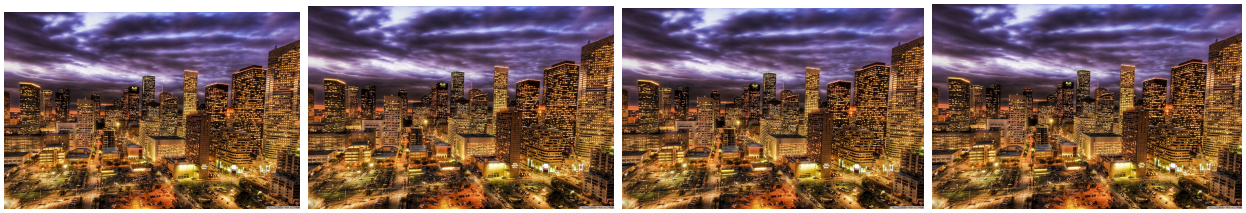
#9 color png to bmp

Name:	Moose.png	Moose_sample.bmp	Moose_rnd.bmp	Moose_max.bmp
Entropy:	7.991	7.744	7.743	7.743

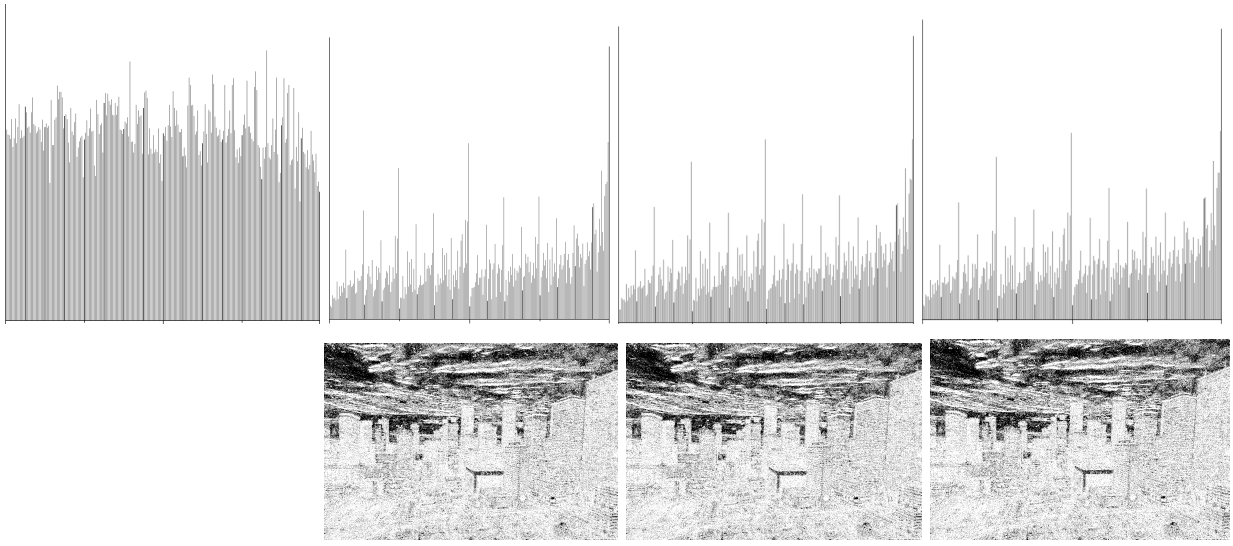


MSE:	8.198	8.054	8.042
PSNR:	38.99	39.07	39.08

#10 jpg



Name:	Houston.jpg	Houston_sample.jpg	Houston_rnd.jpg	Houston_max.jpg
Entropy:	7.986	7.785	7.785	7.787



MSE:	2.140	2.128	2.121
PSNR:	44.83	44.85	44.86

Method performance notes:

The performance metrics reveal several key aspects of the chosen method's behavior and limitations. One notable trend is that higher QR code versions consistently yield better MSE and PSNR values, regardless of the baseline image quality. This is because higher versions have smaller modules, which distribute embedding changes more evenly across the cover image. When applying the inverse DWT, these fine patterns blend more naturally with image textures

than the large, high-contrast blocks of lower QR versions, which create more visible and measurable distortion.

This behavior also explains why a small increase in payload size can significantly change performance metrics: crossing a version threshold to a higher-capacity QR code can improve embedding smoothness. However, higher versions are also more sensitive to distortion — if noise or transformation occurs, the finer modules are more prone to decoding errors. This is evident in cases such as *football_max.png* and *Moose_max.bmp*, where maximum-capacity embedding resulted in unreadable QR codes. Therefore, a balance between payload size and robustness is critical. In practical terms, typical QR use cases such as URLs (e.g., "https://www.utsa.edu/students/wellbeing/quick/covid-19-student-guidance.html#_ga=2.199455470.1393096259.1754528837-1634659062.1696193149") provide an ideal payload length: complex enough to be robust yet small enough to remain discreet and achieve good metrics.

File type also plays a significant role in performance. Across the tests, the ranking from best to worst was generally: color BMPs, color PNGs / grayscale BMPs, then grayscale PNGs. While results vary based on image content, PNG covers consistently showed greater histogram change from their originals compared to BMP covers of the same image. PNGs also experienced a slightly greater drop in entropy, and in nearly all cases this correlated with slightly worse MSE/PSNR performance.

An additional finding is that BMP→PNG conversions performed better than PNG→BMP conversions, and that grayscale PNGs produced the worst results overall. This behavior is consistent with how PIL (the Python Imaging Library) handles PNGs: when reading PNGs, PIL may apply gamma correction or brightness scaling based on embedded metadata, and rounding errors can occur depending on image dimensions. These effects alter the pixel values returned by np.array() compared to the original embedding space. BMP files, by contrast, store raw pixel values without modification, which explains their more consistent performance.

Despite these limitations, the DWT-based method still succeeds in transmitting the hidden message even under PNG-specific transformations, due to its reliance on frequency-domain embedding. Imperceptibility remains high in all cases, with no visible artifacts. However, while PNGs can produce excellent results in certain cases (e.g., Case #6), their inherent inconsistencies in reading and writing make them less predictable than BMPs for performance metrics.

Cover performance notes:

As seen in the performance metrics, some file types produce better results than others due to properties of the steganography method, but characteristics of the image itself can also significantly affect performance. Complex images generally work better than simple or

limited-color images because there is more detail in the HL and HH subbands to mask the effects of altering the LH subband.

Some covers can produce a readable but “spotty” QR code that is not perfectly clean. This can result from:

- Unusual pixel dimensions that are not evenly divisible by 4.
- Small image sizes, which reduce the resolution available for the LH subband.
- A QR version that does not fit cleanly into the LH subband’s dimensions.

Since DWT divides the cover image into four subbands (each ¼ the original size), any mismatch between the LH subband’s dimensions and the QR code’s module structure can cause small amounts of data loss or distortion. These effects are usually minor, but for highly complex QRs (e.g., version 40 from hiding *max.txt*), even slight errors can make the code unreadable—as seen with *Moose_max.png* and *football_max.png*. These covers had dimensions of 684×455 and 783×439, respectively, which likely contributed to decoding errors.

Another observation is that covers with significant blue-channel complexity—meaning strong variation and detail in blue tones—tend to yield the best imperceptibility, seen in case #6 with the manatee. In these cases, the HL and HH subbands of the color image already contain enough complex information that the QR pattern introduced into the LH subband does not significantly disrupt perceived detail.

Suggestions for future work:

Some improvements became conceptualized when coding and adapting the paper process. These include changing from hiding in the LH subband to the HH subband, forming a black and white image with the binary equivalence of the message data and hiding that image instead of the QR, and using all of the color components of the color pictures for hiding. As discussed earlier, there are issues with these ideas, but an exploration of any one of these ideas on their own would likely prove useful. The alpha value could undergo a more rigorous investigation as to what value gives the best balance between imperceptibility and function. Also, the Haar wavelet function could be replaced with another, as there exists other wavelet functions, and while Haar is the most computationally simple of these it is worth exploring other functions to observe possible performance improvement. Lastly, the QR image could be resized back into a square according to the version, but as the data is printed out the terminal if readable and QR codes can be stretched, this extra function is not essential but convenient.

Team member contributions:

Luke Alvarado: Concept and code framework for i/o and dwt process

Preston Mumphrey: Code for usage in a terminal and installation/platform management
Aradhnna Reddy: Deliverable and expectation management, team organizer and deadline assurance
Carlos Osuna: Optimizations, code consistency and format, and documentation/comments

Bibliography:

Priya, K., et al. “DWT based QR Steganography.” Journal of Physics: Conference Series, vol. 1917, no. 1, 1 June 2021, p. 012020, https://doi.org/10.1088/1742-6596/1917/1/012020.

Lawson, S, and J Zhu. “Image compression using wavelets and JPEG2000: a tutorial.” Electronics & Communication Engineering Journal, June 2002, pp. 112–121, https://engineering.purdue.edu/~ee538/Image_compression_wavelets_jpeg2000.pdf.

Introduction to the Discrete Wavelet Transform (DWT), Feb. 2004, https://mil.ufl.edu/nechyba/www/eel6562/course_materials/t5.wavelets/intro_dwt.pdf.

Schulfer, Scott. “How Do QR Codes Work? QR Code Technical Basics.” SproutQR, 14 Sept. 2020, www.sproutqr.com/blog/how-do-qr-codes-work.

“The Python Standard Library.” Python Documentation, Python Software Foundation, docs.python.org/3/library/.

“NumPy Introduction.” w3schools, Refsnes Data, www.w3schools.com/python/numpy/numpy_intro.asp.

Zyprian, Florian. “CV2: OpenCV Guide for Python Developers.” Konfuzio, Helm & Nagel, 9 Mar. 2025, https://konfuzio.com/en/cv2/

“Wavelet Transforms in Python#.” PyWavelets, The PyWavelets Developers, https://pywavelets.readthedocs.io/en/latest/

“Find, Install and Publish Python Packages with the Python Package Index.” PyPI, https://pypi.org/.

“Python: Pillow (a Fork of PIL).” GeeksforGeeks, GeeksforGeeks, 28 Apr. 2025, https://www.geeksforgeeks.org/python/python-pillow-a-fork-of-pil/.