DIGITAL HOLOGRAPHY

Charmaine S. Tolledo

CONTENTS

Objectives

Methods

Results

Analysis

Reflection

OBJECTIVES

01

Extraction

To extract amplitude and phase information from a digital hologram using MATLAB.

02

Visualization

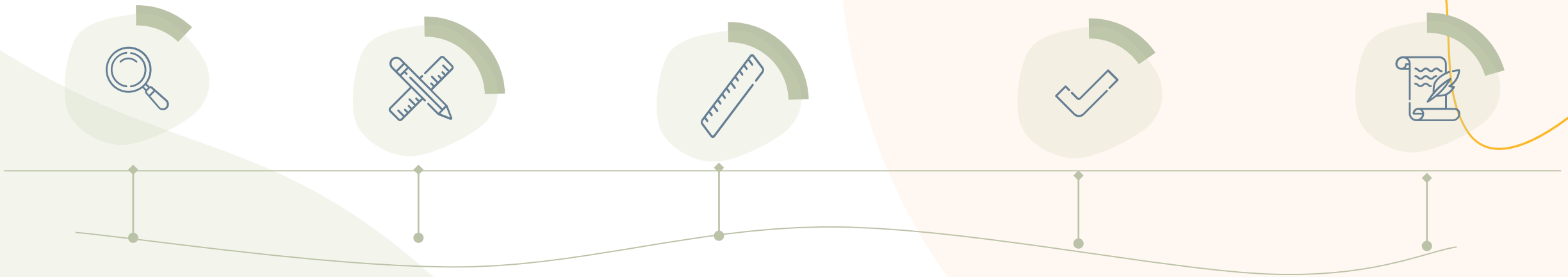
To visualize and analyze the extracted amplitude and phase images.

03

Comprehension

To understand the principles of digital holography and its applications.

METHODOLOGY



INITIALIZATION

- Initiating the process by loading and preparing the digital hologram image for analysis.

FREQUENCY DOMAIN CONVERSION

- Applying Fourier Transform (FFT) to convert image to frequency domain.

ROI EMPHASIS

- Emphasizing specific areas of interest by selecting a Region of Interest (ROI) for extracting phase and amplitude information.

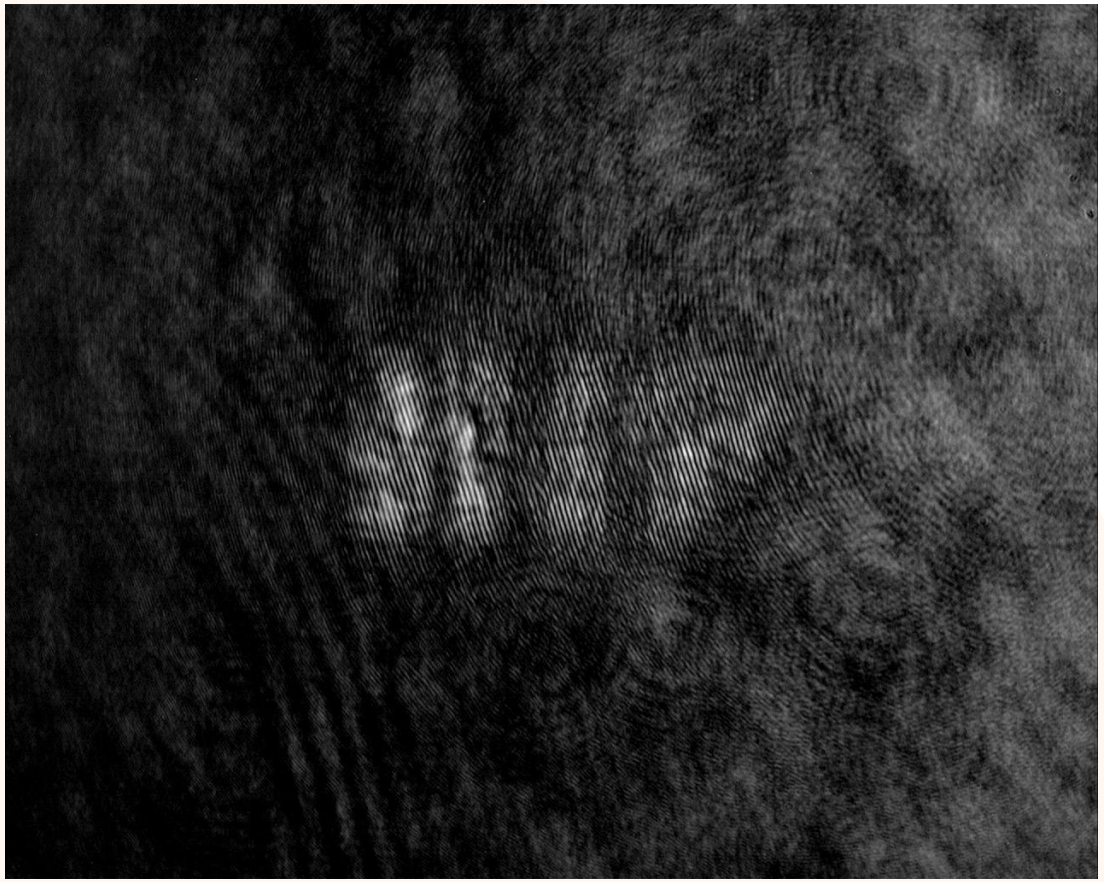
RECONSTRUTION VIA INVERSE FFT

- Bringing together the extracted phase and amplitude information to reconstruct the original holographic image using Inverse FFT.

RESULTS ASSESSMENT

- Assessing the reconstructed holographic image, interpreting findings, and drawing conclusions from the analysis.

DIGITAL HOLOGRAM IMAGE



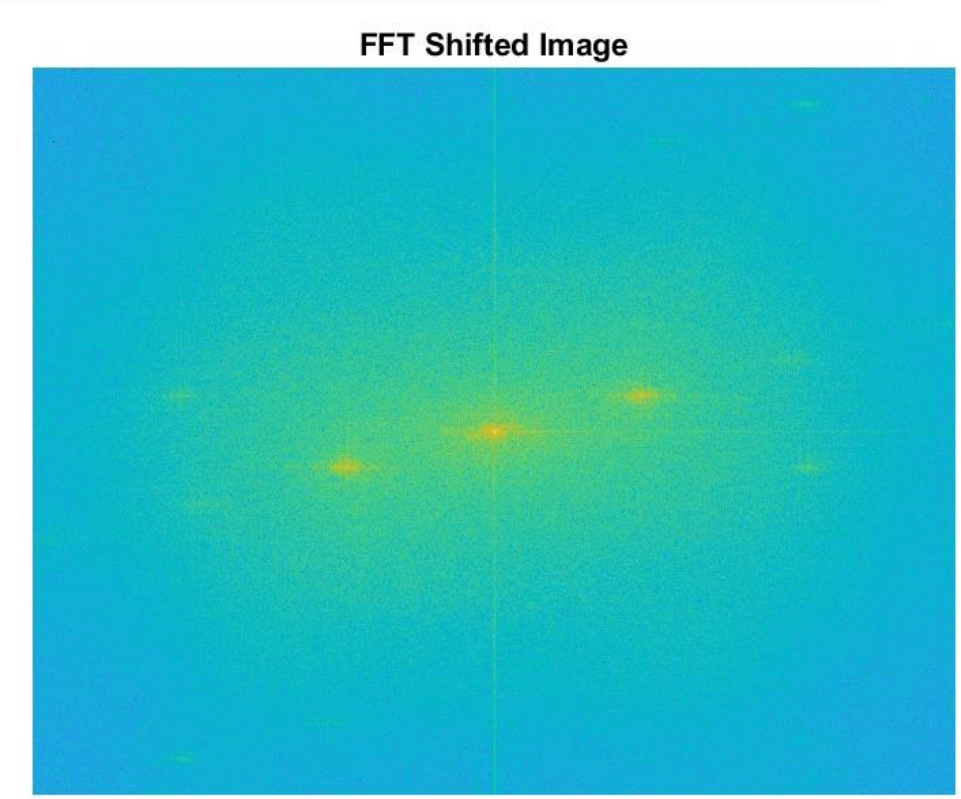
Digital holography uses a pair of synchronized light beams. One provides illumination for the object, while the other functions as a point of reference. Their interaction generates a pattern that is captured by a camera. This pattern accurately represents both the brightness and the physical structure of the light, similar to a unique identifier of the object's surface.

The image on the left is an example of a digital hologram recorded by Dr. Percival Almoro. This is the image we will extract the phase and amplitude from.

MAIN PROCESS USING MATLAB

FREQUENCY DOMAIN CONVERSION

```
% Initialization
I = im2gray(imread("digitalholographysample.jpg"));
FI = fft2(double(I));
FIshift = fftshift(FI);
imagesc(log(abs(FIshift)));
```

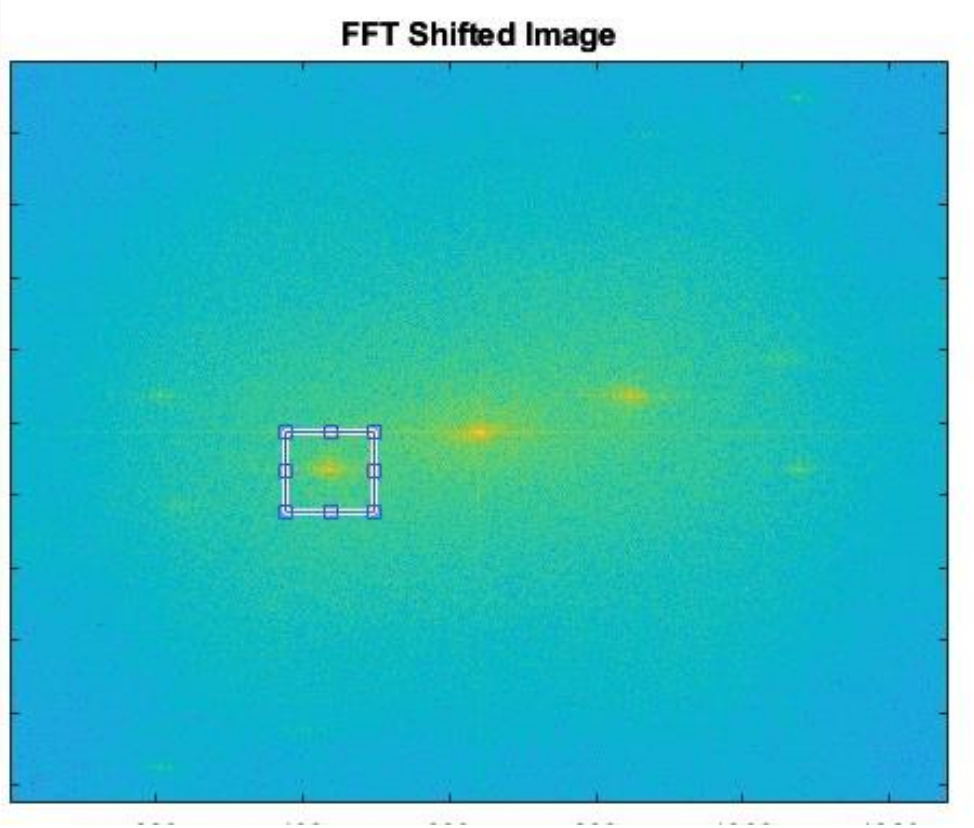


We use fft2 to execute a two-dimensional Fast Fourier Transform (FFT) to examine the image's frequency content. This converts spatial picture data to frequency. For better display, we center the low-frequency components with the fftshift function.

The resulting frequency spectrum, as seen in the figure, has a central bright spot which corresponds to the to the DC component, which represents the average intensity. Around this central point, we can see that there are two symmetric patterns emerge.

REGION OF INTEREST (ROI) SELECTION

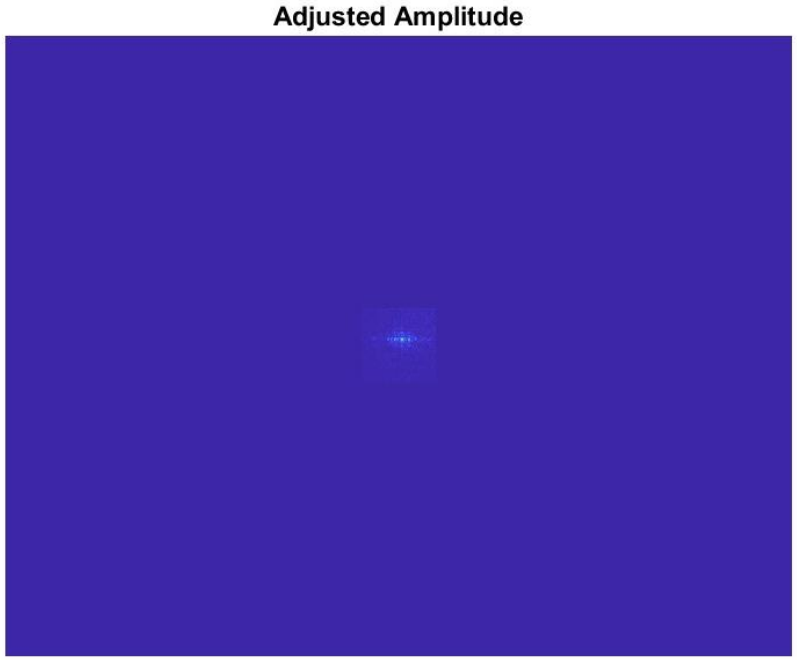
```
%ROI selection
[J,rectout] = imcrop;
Fabs = imcrop(abs(FIshift),rectout);
Fang = imcrop(angle(FIshift),rectout);
```

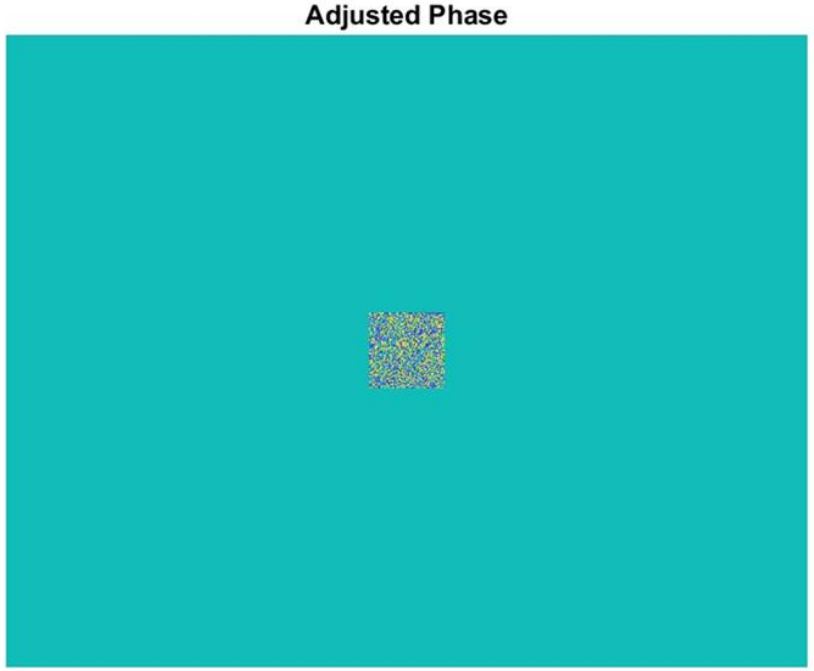


This section precisely targets specific features within the image's frequency domain. Using ‘imcrop’, we extracted both amplitude and phase information – ‘Fabs’ and ‘Fang’ - for the chosen Region of Interest (ROI). This direct access to critical data eliminates unnecessary calculations and allows focused ROI frequency component analysis for targeted image processing.

AMPLITUDE & PHASE ADJUSTMENT

```
% Amplitude and Phase Adjustment
s = size(I);
Aabs = zeros(s);
Aangle = zeros(s);
w = round(rectout(3));
h = round(rectout(4));
wI = s(2); hI = s(1);
uprow = round(hI/2-h/2);
upcol = round(wI/2-w/2);
Aabs(uprow:uprow+h-1, upcol:upcol+w-1) = Fabs;
Aangle(uprow:uprow+h-1, upcol:upcol+w-1) = Fang;
A = Aabs.*exp(1j.*Aangle);
```



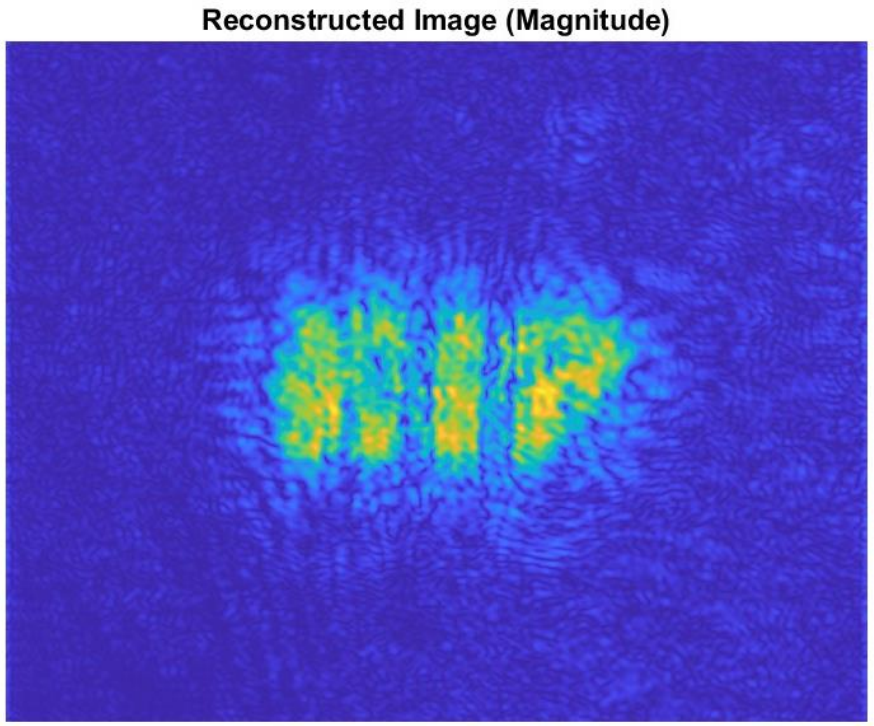


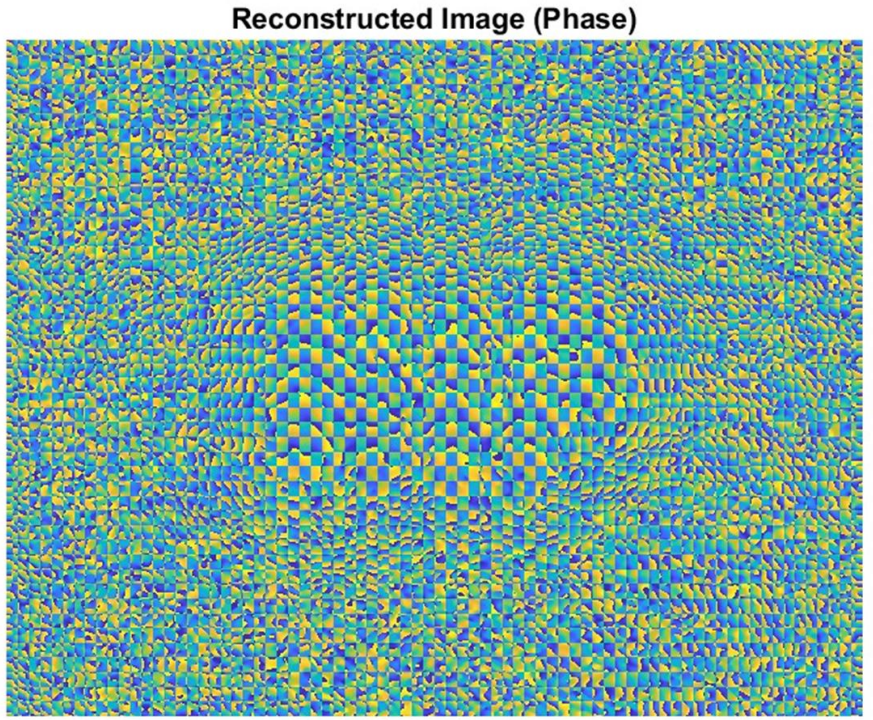
Here, the extracted frequency information was positioned within a blank matrix mirroring the original image size. The adjusted amplitude and phase angle values were stored in the ‘Aabs’ and ‘Aangle’ matrices. Next, accurate computations centered ROI data in these matrices. Amplitude (‘Fabs’) and phase angle (‘Fang’) information were allocated to the selected regions in ‘Aabs’ and ‘Aangle’, respectively. The final adjusted matrix A is obtained by combining these adjusted amplitude and phase components.

The two images above show the adjusted amplitude and the adjusted phase. The first one visualizes the distribution of adjusted frequency magnitudes within the designated ROI and the other one maps the relative shifts in wave positions within the ROI.

RESULT: IMAGE RECONSTRUCTION

```
% Inverse FFT and Image Reconstruction
AFIshift = ifft2(A);
figure;
imagesc(abs(AFIshift));
figure;
imagesc(angle(AFIshift));
```





Here, the inverse Fourier Transform was used to translate the adjusted frequency information back into the spatial domain which, in turn, generated two reconstructed images as seen above.

First image shows the magnitude of the reconstructed hologram and the absolute value of the inverse transform that graphically maps intensity values. Brighter spots have greater adjusted frequency amplitudes while darker areas reflect lower-amplitude frequencies. On the other hand, the second image, which emphasizes the phase information, gives an additional viewpoint by mapping the relative shifts of wave positions within the reconstructed image. The phase map illustrates the alignment or opposition of frequency components. These two images are proof that we were able to extract the amplitude and phase of our image.

BONUS PART:
2D PHASE UNWRAPPING

RELIABILITY CALCULATION

```
% Get reliability
rel = zeros(size(img));

% Get the shifted images (N-2, N-2)
 img_im1_jm1 = img(1:end-2, 1:end-2);
 img_i_jm1   = img(2:end-1, 1:end-2);
 img_ip1_jm1 = img(3:end  , 1:end-2);
 img_im1_j   = img(1:end-2, 2:end-1);
 img_i_j     = img(2:end-1, 2:end-1);
 img_ip1_j   = img(3:end  , 2:end-1);
 img_im1_jp1 = img(1:end-2, 3:end  );
 img_i_jp1   = img(2:end-1, 3:end  );
 img_ip1_jp1 = img(3:end  , 3:end  );

% Calculate the difference
gamma = @(x) sign(x) .* mod(abs(x), pi);
H  = gamma(img_im1_jm1   - img_i_jm1) - gamma(img_i_jm1 - img_ip1_j  );
V  = gamma(img_i_jm1   - img_i_j) - gamma(img_i_j - img_i_jp1  );
D1 = gamma(img_im1_jm1 - img_i_j) - gamma(img_i_j - img_ip1_jp1);
D2 = gamma(img_im1_jp1 - img_i_j) - gamma(img_i_j - img_ip1_jm1);

% Convert variables to double
H_double = double(H);
V_double = double(V);
D1_double = double(D1);
D2_double = double(D2);

% Calculate the second derivative
D = sqrt(abs(H_double).^2 + abs(V_double).^2 + abs(D1_double).^2 + abs(D2_double).^2);

% Assign the reliability as 1 / D
rel(2:end-1, 2:end-1) = 1./D;

% Assign all NaN's in rel with non-NaN in img to 0
% Also assign the NaN's in img to NaN
rel(isnan(rel) & ~isnan(img)) = 0;
rel(isnan(img)) = nan;
```

Now that we have the reconstructed phase image, we unwrap this using 2D phase unwrapping to produce 3D image. And the first step was reliability calculation.

We started with a blank matrix to hold reliability values. For difference computations in various directions, we created eight shifted copies of the image. The phase image's second derivative was calculated by comparing the differences in pixel values in horizontal, vertical, and diagonal directions. The inverse of the calculated second derivative was used to evaluate the dependability matrix (rel).

EDGE DETECTION & SORTING

```
% Get the edges
[Ny, Nx] = size(rel);
h_edges = [rel(1:end, 2:end) + rel(1:end, 1:end-1), nan(Ny, 1)];
v_edges = [rel(2:end, 1:end) + rel(1:end-1, 1:end); nan(1, Nx)];

% Combine all edges and sort it
edges = [h_edges(:); v_edges(:)];
edge_bound_idx = Ny * Nx; % if i <= edge_bound_idx, it is h_edges
[~, edge_sort_idx] = sort(edges, 'descend');
```

We then proceeded to locate and prioritize edges within the phase image. Edges were identified by summing adjacent reliability values both horizontally (‘h_edges’) and vertically (‘v_edges’). Subsequently, these edges were combined into a single vector (‘edges’). A boundary index was then calculated to distinguish between the two edge types within the combined matrix. Finally, the edges were sorted in descending order of their reliability scores. This ensured a systematic and reliable unwrapping procedure by prioritizing regions with higher reliability for more accurate phase unwrapping.

GROUP LABELING & UNWRAPPING PROPAGATION

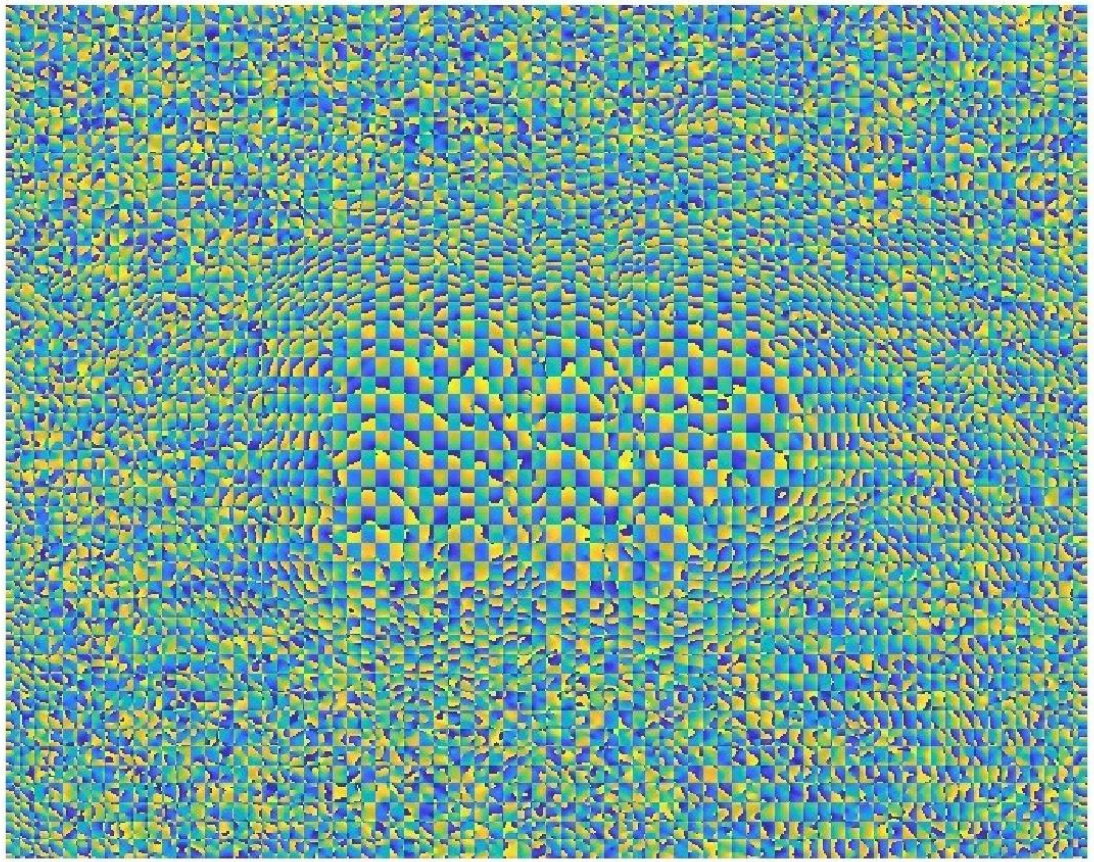
```
% Get the indices of pixels adjacent to the edges
idxs1 = mod(edge_sort_idx - 1, edge_bound_idx) + 1;
idxs2 = idxs1 + 1 + (Ny - 1) .* (edge_sort_idx <= edge_bound_idx);

% Label the group
group = reshape([1:numel(img)], Ny*Nx, 1);
is_grouped = zeros(Ny*Nx,1);
group_members = cell(Ny*Nx,1);
for i = 1:size(is_grouped,1)
    group_members{i} = i;
end
num_members_group = ones(Ny*Nx,1);
```

```
% Propagate the unwrapping
res_img = img;
num_nan = sum(isnan(edges)); % Count how many NaNs and skip them
for i = num_nan+1 : length(edge_sort_idx)
    % Get the indices of the adjacent pixels
    idx1 = idxs1(i);
    idx2 = idxs2(i);

    % Skip if they belong to the same group
    if (group(idx1) == group(idx2)) continue; end

    % idx1 should be ungrouped (swap if idx2 ungrouped and idx1 grouped)
    % Otherwise, activate the flag all_grouped.
    % The group in idx1 must be smaller than in idx2. If initially
    % group(idx1) is larger than group(idx2), then swap it.
    all_grouped = 0;
    if is_grouped(idx1)
        if ~is_grouped(idx2)
            idxt = idx1;
            idx1 = idx2;
            idx2 = idxt;
        elseif num_members_group(group(idx1)) > num_members_group(group(idx2))
            idxt = idx1;
            idx1 = idx2;
            idx2 = idxt;
            all_grouped = 1;
        else
            all_grouped = 1;
        end
    end
```

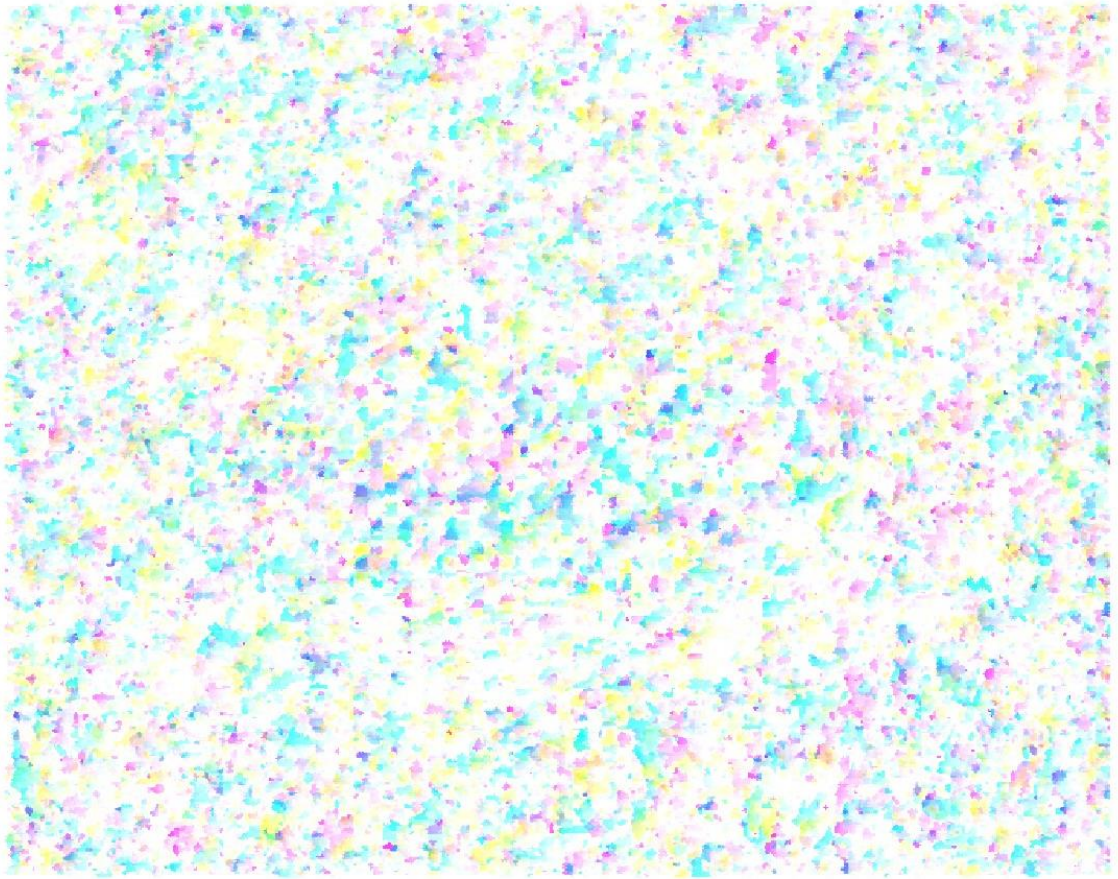
```
    % Calculate how much we should add to the idx1 and group
    dval = floor((res_img(idx2) - res_img(idx1) + pi) / (2*pi)) * 2*pi;

    % Which pixel should be changed
    g1 = group(idx1);
    g2 = group(idx2);
    if all_grouped
        pix_idxs = group_members{g1};
    else
        pix_idxs = idx1;
    end

    % Add the pixel value
    if dval ~= 0
        res_img(pix_idxs) = res_img(pix_idxs) + dval;
    end

    % Change the group
    len_g1 = num_members_group(g1);
    len_g2 = num_members_group(g2);
    group_members{g2}(len_g2+1:len_g2+len_g1) = pix_idxs;
    group(pix_idxs) = g2; % Assign the pixels to the new group
    num_members_group(g2) = num_members_group(g2) + len_g1;

    % Mark idx1 and idx2 as already being grouped
    is_grouped(idx1) = 1;
    is_grouped(idx2) = 1;
end
```

After edge detection and sorting, we proceeded to get the indices of pixel adjacent to the edges. We then assigned each pixel with a unique group label. Meanwhile, Unwrapping propagation, the subsequent step, focused on iteratively modifying pixel values to ensure phase consistency across adjacent regions. During the iteration, adjustments were made only for pixels belonging to different groups. The unwrapping propagation method removed phase ambiguities from the wrapped image, which resulted to a coherent and interpretable unwrapped phase image.

Original Wrapped Phase



Unwrapped Phase



REFLECTION

I appreciated the fact that the code was provided, which made it rather easy for me to execute this activity. However, the bonus part was the challenging part. In fact, have just attempted it and am still uncertain as to whether or not I was successful in unwrapping the phase image. Nevertheless, since I was able to meet the objective, which is to extract the phase and amplitude from an actual digital hologram, I will still give myself a perfect score of **100/100.**

REFERENCES

- M. A. Herraez, D. R. Burton, M. J. Lalor, and M. A. Gdeisat, "Fast two-dimensional phase-unwrapping algorithm based on sorting by reliability following a noncontinuous path", Applied Optics, Vol. 41, Issue 35 pp. 7437-7444 (2002).
- M. F. Kasim, "Fast 2D phase unwrapping implementation in MATLAB https://github.com/mfkasim91/unwrap_phase/ (2017).
- Dr. Maricor Soriano, Physics 167, Activity 8 Digital Holography Lab Manual.

THANK YOU!