Project Report for

**RE4012 - MACHINE VISION**

Image Reconstruction from a "Sinogram"

Revision Timestamp: 14/03/2025 14:30:09

## Project Team

| Student Name | Student Id |
|---|---|
| ZEXIN LI | 23202637 |
| QINYUAN LIU | 20137095 |

## Contents

## 1. Introduction

Sinogram reconstruction is a common technique used in CT scanners, where one or multiple rotational beams of X-rays pass through an object, generating an image output on a sensing semiconductor. This output, commonly known as a **sinogram**, represents the raw projection data collected from different angles.

The sinogram produced by a CT scanner can be transformed into **2D cross-sectional slices** of the scanned object, revealing its internal structure. This transformation is typically performed using **Filtered Back Projection (FBP)**, a widely used reconstruction algorithm in medical imaging.

This project report focuses on the **Python implementation** of image reconstruction from a given sinogram (CT raw sinogram output) using the **FBP algorithm**. We will first explain the **organizational approach** taken in structuring the implementation, followed by a detailed discussion of the **technical aspects** of the project.

## 2. Approach

The project is undertaking by 2 individuals, with attempt of recruit new members made. Unfortunately, we have not yet received a reply from suitable candidates. The project is implemented in 2 phases: Code development and Report writing. Zexin Li has undertaken the main part of the code development, and Qinyuan Liu has taken responsibility for code review and Report writing.

## 3. Mathematical principles

Fourier transformation and inverse Fourier Transform are fundamental in Filtered Back Projection (FBP). The Fourier Transform helps in converting the sinogram projections into the frequency domain, allowing us to apply Ramp Filtering, which enhances high-frequency components and reduces blurring effects in the reconstructed image.

Ramp filtering is an essential step in FBP because direct back projection without filtering results in blurry reconstructions. The Ramp Filter acts as a high-pass filter that compensates for the excessive low-frequency components inherent in the projection data.

The following table shows the Mathematical Formula used and it’s associated code:

| Step | Mathematical Formula | Python Code |
|---|---|---|
| **1D Fourier Transform** | $F(k) = \sum f(n) e^{-j2\pi kn/N}$ | F = rfft(row_data) |
| **Ramp Filtering** | $\mathrm{F_{filtere}d(k) = F(k)} \cdot \lvert k \rvert$ | k |
| **Inverse Fourier Transform** | $\mathrm{f(n)} = \frac{1}{N} \sum \mathrm{F_{filtere}(k) e^{j2\pi kn/N}}$ | recon_image=iradon(filtered_sino, theta=theta, filter_name=None, circle=True) |
| **Backprojection** | $\mathrm{I(x, y)} = \int \mathrm{Sfiltered(\theta, p) d\theta}$ | recon_image = iradon(filtered_sino, theta=theta, filter_name=None, circle=True) |

*Table 1 Formula Table*

## 4. Workflow of the Code

The workflow of the reconstruction process is as follows:

1. Load sinogram
    a. Read the sinogram file (sinogram.png) in RGB format, which usually consists of image projection data and is used for tomography reconstruction.

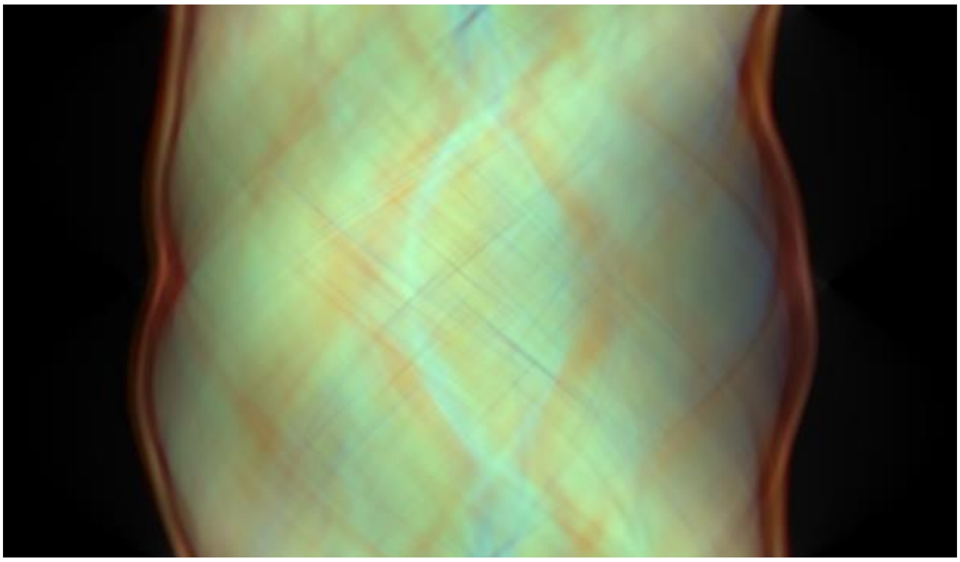

*Table 2 Sinogram.png*

2. Separate color channels
    a. Split the RGB image into three independent channels: red (R), green (G), and blue (B), and process the projection data of each channel separately.
3. Frequency domain stimulation
    a. Apply a ramp filter (implemented in the frequency domain) to each channel. This step requires a Fourier transform (FFT) first, multiplying with the ramp filter (high-frequency compensation), and then converting it back to the spatial domain through an inverse Fourier transform (iFFT) to eliminate projection blur.
4. Backprojection reconstruction
    a. Project the auxiliary projection data into the image space and generate a two-dimensional reconstructed image for each channel by integration.
5. Post-processing
6. Crop region cancellation: remove the black border at the edge of the projected image (cancel region data).
7. Adjust aspect ratio: correct the image ratio to avoid display distortion.
8. Normalize to 8 bits: linearly scale the pixel values to the range of 0-255, equipped with a standard image format.
9. Merge color channels

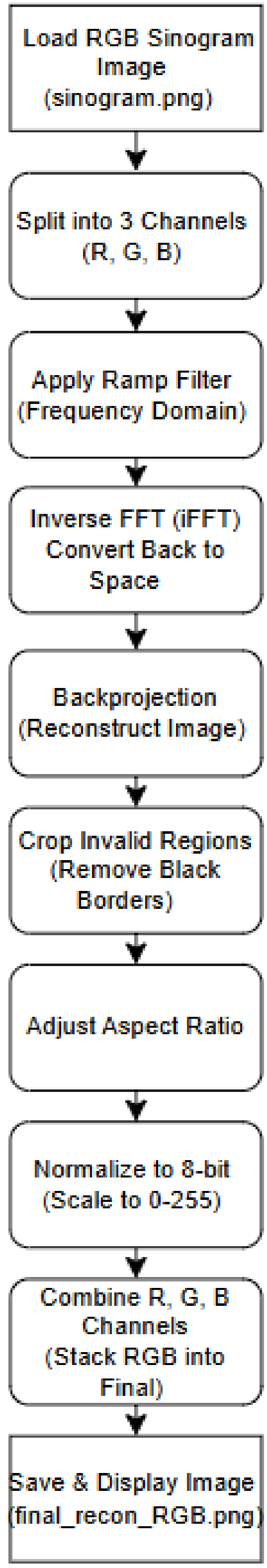


*Figure 1 Code Workflow*

   a. Restore the brightness of the processed R, G, and B channels to an RGB color image.

10. Save the final reconstructed image (final_recon_RGB.png) and display it.

## 5. Code Implementation:

```
import numpy as np
import imageio.v3 as iio
import matplotlib.pyplot as plt
from numpy.fft import rfft, irfft
from skimage.transform import rotate, iradon
from skimage.transform import resize

def ramp_filter_1d_fftrow(row_data):
    """
    1) rFFT
    2) ramp
    3) iFFT
    """
    N = len(row_data)
    #rFFT
    F = rfft(row_data)  # shape: (N//2 + 1,)

    #ramp
    freq = np.arange(len(F))
    F_filtered = F * freq

    #irFFT
    filtered_row = irfft(F_filtered, n=N)
    return filtered_row


def hamming_window_1d(ffts):
    hamming = np.hamming(ffts.shape[0])
    return ffts * hamming


def ramp_filter_1d_hamming_fftrow(row_data):
    # row_data must be 1d
    N = len(row_data)

    F = np.fft.rfft(row_data)
    print(row_data.shape)

    freq = np.fft.rfftfreq(N, d=1.0)

    ramp = np.abs(freq)

    F_filtered = F * ramp
```

```
    F_filtered = hamming_window_1d(F_filtered)

    filtered_row = irfft(F_filtered, n=N)
    return filtered_row



def ramp_filter_1d_hann_fftrow(row_data):
    N = len(row_data)
    F = rfft(row_data)

    freq = np.fft.rfftfreq(N, d=1.0)
    ramp = np.abs(freq)

    hann_win = np.hanning(len(F))

    F_filtered = F * ramp * hann_win
    filtered_row = irfft(F_filtered, n=N)
    return filtered_row


def apply_filter_to_sinogram(sinogram_2d, filter_type='none'):
    """
    use none, ramp, hamming, hann to specify which filter to use
    """
    rows, cols = sinogram_2d.shape
    output = np.zeros_like(sinogram_2d, dtype=np.float64)

    if filter_type == 'none':
        return sinogram_2d.astype(np.float64)

    for i in range(rows):
        row_data = sinogram_2d[i, :]
        if filter_type == 'ramp':
            output[i, :] = ramp_filter_1d_fftrow(row_data)
        elif filter_type == 'hamming':
            output[i, :] = ramp_filter_1d_hamming_fftrow(row_data)
        elif filter_type == 'hann':
            output[i, :] = ramp_filter_1d_hann_fftrow(row_data)
        else:
            output[i, :] = row_data

    return output


def reconstruct_from_sinogram(sinogram_2d):
    rows, cols = sinogram_2d.shape
    theta = np.linspace(0., 180., cols, endpoint=False)

    recon_image = iradon(sinogram_2d, theta=theta, filter_name=None, circle=False)
    return recon_image
```

```python
def crop_circle_region(image, threshold=0.01):
    rows, cols = image.shape
    mask = (image > threshold)
    row_indices = np.where(np.any(mask, axis=1))[0]
    col_indices = np.where(np.any(mask, axis=0))[0]
    if len(row_indices) == 0 or len(col_indices) == 0:
        return image
    rmin, rmax = row_indices[0], row_indices[-1]
    cmin, cmax = col_indices[0], col_indices[-1]
    cropped = image[rmin:rmax + 1, cmin:cmax + 1]
    return cropped


def apply_aspect_ratio(image, aspect_ratio_str):
    if ':' in aspect_ratio_str:
        w_ratio, h_ratio = aspect_ratio_str.split(':')
        w_ratio = float(w_ratio)
        h_ratio = float(h_ratio)
        ratio = w_ratio / h_ratio
    else:
        ratio = 1.0

    height, width = image.shape
    new_width = int(round(height * ratio))

    image_resized = resize(image, (height, new_width),
                           preserve_range=True,
                           anti_aliasing=False)
    return image_resized


def float_to_8bit(image):
    im_min, im_max = image.min(), image.max()
    if im_max == im_min:
        return np.zeros_like(image, dtype=np.uint8)
    norm = (image - im_min) / (im_max - im_min)
    out = (norm * 255.0).astype(np.uint8)
    return out


metadata = iio.immeta("sinogram.png")
aspect_ratio_str = metadata.get("AspectRatio", "1:1")

sinogram_rgb = iio.imread('sinogram.png')
R, G, B = sinogram_rgb[:, :, 0], sinogram_rgb[:, :, 1], sinogram_rgb[:, :, 2]

iio.imwrite('sinogram_R.png', R)
iio.imwrite('sinogram_G.png', G)
iio.imwrite('sinogram_B.png', B)
```

```
sinogram_R = iio.imread("sinogram_R.png").T  # shape: (W, H)
sinogram_G = iio.imread("sinogram_G.png").T
sinogram_B = iio.imread("sinogram_B.png").T

# ========== (a) Reconstruction without Filtering ==========

sino_R_nofilter = apply_filter_to_sinogram(sinogram_R, filter_type='none')
sino_G_nofilter = apply_filter_to_sinogram(sinogram_G, filter_type='none')
sino_B_nofilter = apply_filter_to_sinogram(sinogram_B, filter_type='none')

recon_R_nofilter = reconstruct_from_sinogram(sino_R_nofilter)
recon_G_nofilter = reconstruct_from_sinogram(sino_G_nofilter)
recon_B_nofilter = reconstruct_from_sinogram(sino_B_nofilter)

# ========== (b) Reconstruction with Pure Ramp Filtering ==========

sino_R_ramp = apply_filter_to_sinogram(sinogram_R, filter_type='ramp')
sino_G_ramp = apply_filter_to_sinogram(sinogram_G, filter_type='ramp')
sino_B_ramp = apply_filter_to_sinogram(sinogram_B, filter_type='ramp')

recon_R_ramp = reconstruct_from_sinogram(sino_R_ramp)
recon_G_ramp = reconstruct_from_sinogram(sino_G_ramp)
recon_B_ramp = reconstruct_from_sinogram(sino_B_ramp)

# ========== (c) Reconstruction with Ramp Filtering + Hamming Window ==========

sino_R_hamming = apply_filter_to_sinogram(sinogram_R, filter_type='hamming')
sino_G_hamming = apply_filter_to_sinogram(sinogram_G, filter_type='hamming')
sino_B_hamming = apply_filter_to_sinogram(sinogram_B, filter_type='hamming')

recon_R_hamming = reconstruct_from_sinogram(sino_R_hamming)
recon_G_hamming = reconstruct_from_sinogram(sino_G_hamming)
recon_B_hamming = reconstruct_from_sinogram(sino_B_hamming)

# ========== (Optional) Reconstruction with Ramp Filtering + Hann Window ==========

sino_R_hann = apply_filter_to_sinogram(sinogram_R, filter_type='hann')
sino_G_hann = apply_filter_to_sinogram(sinogram_G, filter_type='hann')
sino_B_hann = apply_filter_to_sinogram(sinogram_B, filter_type='hann')

recon_R_hann = reconstruct_from_sinogram(sino_R_hann)
recon_G_hann = reconstruct_from_sinogram(sino_G_hann)
recon_B_hann = reconstruct_from_sinogram(sino_B_hann)


# ========== Post-processing: Crop, Adjust Aspect Ratio, Convert to 8-bit, and Merge into RGB ==========

def postprocess_and_merge(r_img, g_img, b_img, aspect_ratio_str):
    R_crop = crop_circle_region(r_img)
    G_crop = crop_circle_region(g_img)
```

```
    B_crop = crop_circle_region(b_img)

    R_resized = apply_aspect_ratio(R_crop, aspect_ratio_str)
    G_resized = apply_aspect_ratio(G_crop, aspect_ratio_str)
    B_resized = apply_aspect_ratio(B_crop, aspect_ratio_str)

    R_8bit = float_to_8bit(R_resized)
    G_8bit = float_to_8bit(G_resized)
    B_8bit = float_to_8bit(B_resized)

    min_rows = min(R_8bit.shape[0], G_8bit.shape[0], B_8bit.shape[0])
    min_cols = min(R_8bit.shape[1], G_8bit.shape[1], B_8bit.shape[1])
    R_final = R_8bit[:min_rows, :min_cols]
    G_final = G_8bit[:min_rows, :min_cols]
    B_final = B_8bit[:min_rows, :min_cols]

    final_RGB = np.dstack([R_final, G_final, B_final])
    return final_RGB


# Generate four sets of results (No Filtering / Ramp / Hamming / Hann)
final_nofilter_RGB = postprocess_and_merge(recon_R_nofilter, recon_G_nofilter,
recon_B_nofilter, aspect_ratio_str)
final_ramp_RGB = postprocess_and_merge(recon_R_ramp, recon_G_ramp, recon_B_ramp,
aspect_ratio_str)
final_hamming_RGB = postprocess_and_merge(recon_R_hamming, recon_G_hamming,
recon_B_hamming, aspect_ratio_str)
final_hann_RGB = postprocess_and_merge(recon_R_hann, recon_G_hann, recon_B_hann,
aspect_ratio_str)

# Save results
iio.imwrite("final_recon_no_filter.png", final_nofilter_RGB)
iio.imwrite("final_recon_ramp.png", final_ramp_RGB)
iio.imwrite("final_recon_hamming.png", final_hamming_RGB)
iio.imwrite("final_recon_hann.png", final_hann_RGB)

# ========== Display ==========

plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.imshow(final_nofilter_RGB)
plt.title("No Filter Reconstruction")
plt.axis("off")

plt.subplot(2, 2, 2)
plt.imshow(final_ramp_RGB)
plt.title("Ramp Filter Reconstruction")
plt.axis("off")

plt.subplot(2, 2, 3)
```

```
plt.imshow(final_hamming_RGB)
plt.title("Ramp + Hamming Reconstruction")
plt.axis("off")

plt.subplot(2, 2, 4)
plt.imshow(final_hann_RGB)
plt.title("Ramp + Hann Reconstruction")
plt.axis("off")

plt.tight_layout()
plt.show()

print("Reconstruction results saved. Comparison displayed.")
```

## 6. Results

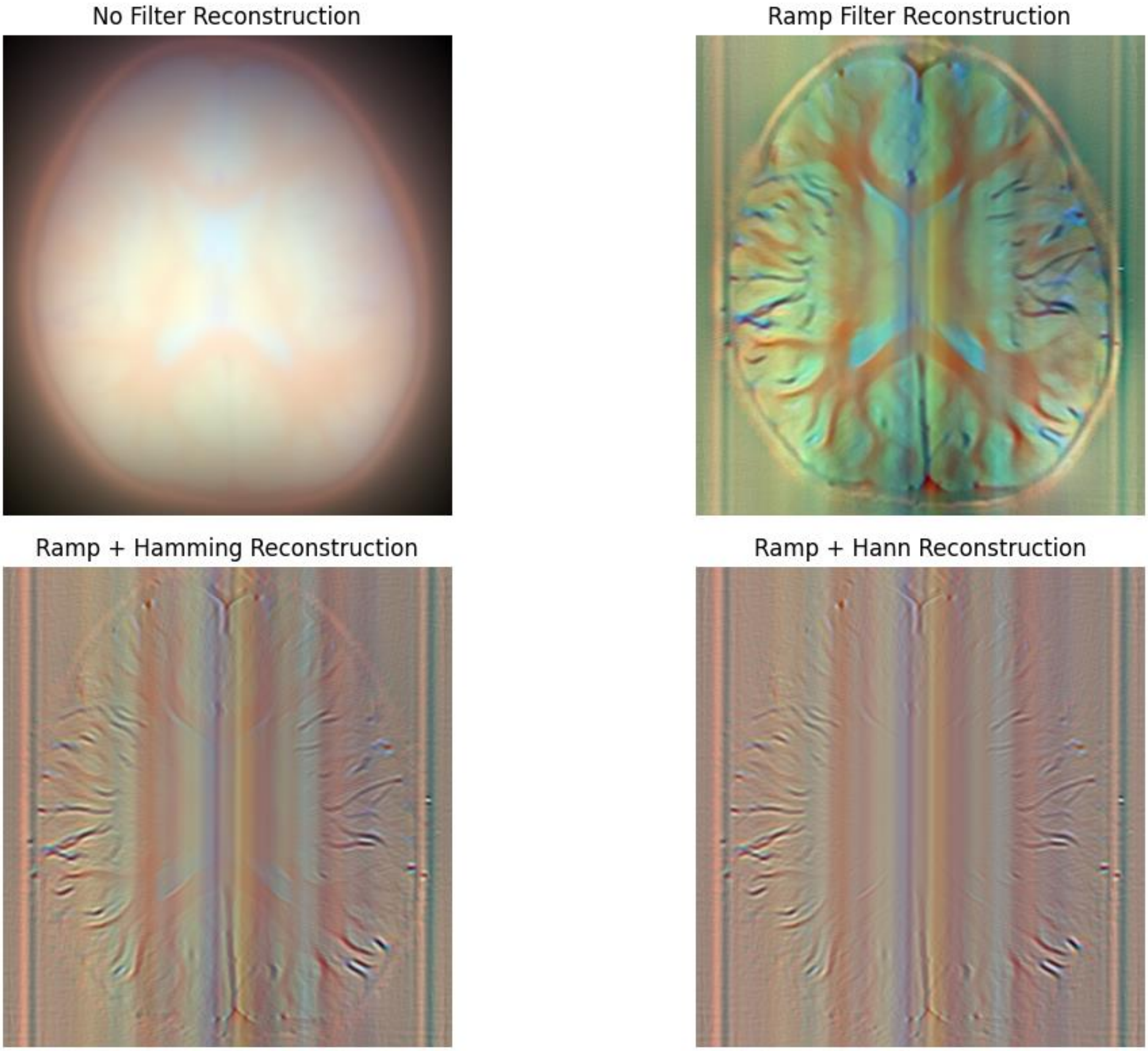


*Table 3 Reconstructed image*

## 7. Conclusion

The project demonstrates the students understanding of the FBP algorism, and the practical skills of the implementation in python environment.

The no-filter reconstruction illustrate the reconstructed image of sinogram.png, while the different windows (Filters) demonstrate that window function’s capability in digital processing.

Visually, the ramp filter provides the best outcome in terms of brain region distinction, while the Ramp + Hann is best for edge detection.

## 8. References

[1] C. Flanagan, "Image Reconstruction from a Sinogram," RE4012/7 Machine Vision Project 1, 2024-5.

[2] C. Flanagan, "Radon Transform: Image Reconstruction from Projections," 2024-5.

[3] "Filtered Backprojection (FBP): Illustrated Guide for Radiologic Technologists," How Radiology Works, 2024. [Online]. Available: https://howradiologyworks.com/filtered-backprojection-fbp-illustrated-guide-for-radiologic-technologists/. [Accessed: 14-Mar-2025].

[4] [4] 谢钧, "现代医学成像（3）——CT（基本原理与图像重建）," 知乎, Jan. 24, 2020. [Online]. Available: https://zhuanlan.zhihu.com/p/90571757. [Accessed: Mar. 14, 2025].