Impact of Undersampling the k-Space Data and Solutions For Minimizing Artifacts

Öykü Özbirinci - 22002320, Efe Özdilek - 22003139
Department of Electrical and Electronics Engineering, *İhsan Doğramacı Bilkent University*, Ankara, Turkey
January 6, 2025

***Abstract*—One of the major challenges of Magnetic Resonance Imaging (MRI) is the extended scanning time required for image acquisition. This prolonged duration is primarily attributed to the necessity of sampling the entire k-space with sufficient data points. Sampling rates above the Nyquist rate are required to avoid aliasing and artifacts in the spatial domain. In this paper, the main aim is to minimize artifacts and aliasing effects due to undersampling. Several undersampling schemes will be implemented in the k-space of various MRI images with contrast modalities T1, T2, and Proton Density (PD). These schemes will be compared regarding their efficiency in minimizing artifacts and reducing aliasing effects. Compressed Sensing (CS) techniques will be utilized when reconstructing the undersampled data. Magnetic Resonance Imaging and its sparsity in the frequency and wavelet domains will be introduced in the introduction. The introduction will discuss the implications of non-ideal sampling of k-space data in MRI. Methods section will provide a detailed explanation of the undersampling schemes and Compressed Sensing techniques along with the primary sources utilized. The datasets will also be described in detail in the methods section. The results section will present the outcomes corresponding to each undersampling scheme. Finally, the discussion section will compare the performance of various undersampling schemes and Compressed Sensing techniques, identifying the most efficient ones for each specific context.**

I. Introduction

Magnetic Resonance Imaging (MRI) is one of the most widely utilized imaging techniques in the world today. It relies on the magnetization of hydrogen spins, and with the application of a strong magnetic field and RF excitation, tissues can be imaged for medical purposes. Different tissues exhibit varying T1 and T2 relaxation times and proton densities, enabling contrast differentiation in the resulting images.

MRI images are primarily generated using a static, large magnetic field, an RF excitation pulse, and gradient coils. These gradient coils facilitate the acquisition of spatial resolution in the x-y plane. The primary MRI signal of interest is the transverse component of the bulk magnetization of the spins, which can be detected following RF excitation using techniques such as "Spin Echo" or "Gradient Echo." By employing gradient coils, the frequency domain of the image—referred to in MRI as "k-space"—can be sampled. With sufficient sampling, such that the sampling rate exceeds the Nyquist rate, the image can be accurately reconstructed from its k-space representation.

However, this sampling process is time-intensive, requiring full k-space coverage to avoid aliasing and artifacts. To shorten the acquisition time, the number of samples taken from k-space can be reduced, known as k-space undersampling. While undersampling decreases acquisition time, it poses challenges in maintaining image quality, as the Nyquist Theorem indicates that a lower sampling rate increases the risk of artifacts and aliasing. Nonetheless, several advanced techniques have been developed to mitigate these issues, enabling the reconstruction of high-quality MRI images even with reduced k-space sampling. These techniques mainly utilize the fact that MRI images are sparse in the frequency domain [5]. Sparsity in the frequency domain indicates that a significant portion of the data in MRI images consists of values close to zero when represented in this domain. This property can be effectively utilized in undersampling strategies and Compressed Sensing techniques to improve image acquisition efficiency and reconstruction quality.

II. Methods

A. Non-Ideality of MRI

In Magnetic Resonance Imaging (MRI), numerous assumptions are made to simplify computations and facilitate a better understanding of the process. For instance, it is often assumed that the data acquisition window is sufficiently short to neglect the effects of T2* decay or that the subject remains stationary throughout the MRI procedure, ensuring accurate sampling of k-space data.

In this study, we challenge the assumption that a sufficient number of k-space samples is always available for image reconstruction. The implications of undersampling k-space data are explored, and the resulting outcomes are analyzed and discussed. Therefore, in this study, MRI images are asummed to be taken from stationary patients and the effects of T2* decay are ignored, i.e. data acquisition window is assumed to be small.

$$\frac{f_s}{2} \geq \frac{\gamma G_x x_{\max}}{2\pi} \quad [16] \tag{1}$$

This equation is the well-known Nyquist Sampling Criterion [16] for MRI images. From (1), it can be inferred that an insufficient number of k-space samples introduces significant artifacts and aliasing effects in the reconstructed image, which can complicate the diagnostic or investigative utility of MRI. Nevertheless, various methods exist to mitigate these artifacts, improving image quality even under undersampled conditions.

B. Undersampling Methods

Different undersampling techniques were employed because, for each diagnosis, certain regions of the image hold greater significance, making the resolution in those regions or directions more critical. Consequently, specific samples in k-space are prioritized over others, allowing less important samples to be ignored. Owing to these considerations, eight distinct undersampling techniques were utilized in this study.

Random Undersampling: In random undersampling, half of the k-space is randomly selected and sampled. The image is then reconstructed using this sampled half of the k-space. (See Appendix A.)

Variable Density Undersampling: MRI images exhibit sparsity in the frequency domain, meaning high-frequency regions contain relatively less valuable information than low-frequency regions, which hold critical details. Utilizing this property, Variable Density Undersampling prioritizes the sampling of low-frequency regions with a higher probability while assigning lower probabilities to high-frequency regions. As with random undersampling, half of the k-space is sampled in this method. (See Appendix B.)

Partial Fourier Method: MRI images are mostly real-valued, meaning they lack complex components in the image domain. For real-valued signals, their frequency domain representation exhibits complex conjugate symmetry.

$$\text{If } f(x,y) = f^*(x,y) \rightarrow F(u,v) = F^*(-u,-v) \qquad (2)$$

Based on this property, the Partial Fourier Method allows for the reconstruction of the entire k-space using samples from only the upper half [11]. The lower half of the k-space is generated by applying the complex conjugate symmetry of the upper half [11]. If the MRI image is purely real-valued and free from artifacts such as chemical shifts and field inhomogeneities, the reconstruction is expected to be accurate and free of distortion. (See Appendix C.)

Center-Weighted Undersampling: Utilizing the sparsity of MRI data, the center-weighted undersampling technique focuses on sampling low-frequency regions of k-space, typically in the form of a square region, while ignoring high-frequency regions entirely [12]. This approach is analogous to passing the image through a low-pass filter [12]. As a result, the low-frequency regions provide an approximate reconstruction of the image, capturing essential structural information. However, this technique sacrifices high-resolution details, as these are predominantly contained in the high-frequency regions. (See Appendix D.)

Radial Undersampling: In radial undersampling, the low-frequency regions of k-space are sampled more densely compared to the high-frequency regions [13]. The sampling in the high-frequency regions is performed using a polar scanning approach [13]. The primary motivation for this technique is to mitigate motion artifacts in MRI. When motion occurs during the imaging process, the acquired k-space data may appear rotated. Radial undersampling is advantageous in this scenario, as it inherently distributes the effects of such rotation across the image, reducing their impact and preserving image quality [13]. (See Appendix E.)

Uniform Undersampling & Anti-Aliasing Filter: In uniform undersampling, samples are taken at regular intervals in both the x and y directions. However, this approach leads to aliasing in the reconstructed image, as described by (1). To mitigate aliasing, an anti-aliasing filter can be applied to the image before undersampling. This filter should eliminate the edges of the image responsible for aliasing and improve the quality of the reconstructed image, despite the cost of reduced resolution. (See Appendix F.)

$$G(u,v) = [F(u,v) * H(u,v)]\delta_s(u,v,\Delta u,\Delta v) \qquad (3)$$

Partial Fourier and Variable Density Undersampling: In the method proposed by the authors of this paper, a combination of the Partial Fourier Method and Variable Density Undersampling was employed. Initially, k-space was sampled using Variable Density Undersampling, after which only the upper half of the sampled data was retained. As a result, only approximately 25% of the total k-space samples were utilized, effectively reducing the sampling time by a factor of 4. Using the symmetry property described in (2), the lower half of k-space samples can be generated from the upper half, enabling image reconstruction. (See Appendix G.)

Cartesian Undersampling: In Cartesian undersampling, low-frequency regions are sampled more densely due to the inherent sparsity of MRI data, similar to previously discussed methods. However, in this approach, resolution in one direction is prioritized over the other. In the mask utilized in this study, the y-direction resolution is emphasized. Conversely, there are cases where the x-direction resolution is given higher priority, depending on the clinical application. For the diagnosis of certain diseases, achieving higher resolution in the y-direction can be particularly significant, as it provides better detail in the structures aligned along that axis. (See Appendix H.)

C. Compressed Sensing Methods

Compressed Sensing is a technique that exploits the sparsity of a signal in the frequency, image, or wavelet domain to reconstruct the signal using fewer samples than required by the Nyquist rate [10]. Compressed sensing algorithms typically employ an iterative approach to reconstruct the signal by minimizing the number of required samples, i.e. L1 norm of the signal, while ensuring that the residual error remains below a predefined threshold [10].

$$\hat{x} = \underset{x}{\operatorname{argmin}} \|x\|_1 \quad \text{subject to} \quad \|Ax - y\|_2 < \epsilon \qquad (4)$$

Minimizing $\hat{x}$ is the main objective in compressed sensing.

Total Variation Denoising: Total variation refers to the change in intensity between adjacent pixels in an image [15]. In compressed sensing, the goal is to achieve low total variation while minimizing the difference between the reconstructed image and the measured randomly undersampled data. By reducing these values, better contrast and resolution in the images can be achieved. Minimizing total variation is not challenging, as MRI images are inherently sparse [5, 15].

$$\hat{x} = \underset{x}{\operatorname{argmin}} \|\text{TV}(x)\|_1 \quad \text{subject to} \quad \|Ax - y\|_2 < \epsilon \quad (5)$$

Wavelet Transform Sparsity Regularization: In this method, the primary objective is to promote sparsity in the wavelet domain while minimizing the difference between the reconstructed image and the measured randomly undersampled data [14]. Compared to Total Variation Denoising, Wavelet Sparsity Regularization also provides good resolution when parameters are appropriately tuned [14]. This is because the minimization is performed in the wavelet domain, which inherently captures spatio-temporal information.

$$\hat{x} = \underset{x}{\operatorname{argmin}} \|Wx\|_1 \quad \text{subject to} \quad \|Ax - y\|_2 < \epsilon \quad (6)$$

D. Datasets

The dataset primarily used in this study is the BrainWeb: Simulated Brain Database from McGill University [9]. This simulated dataset offers various imaging modalities, including T1, T2, and Proton Density (PD), allowing for comprehensive investigations across different contrasts [9]. It also provides the flexibility to manipulate noise levels and slice thickness, enabling the evaluation of the effects of undersampling schemes under varying conditions [9]. Furthermore, for patients with tumorous brain MRIs, the data is collected from Br35H, Brain Tumor Detection Dataset, including samples from real patients [8].

E. Sources

ChatGPT and Claude were employed during the code development process, significantly accelerating the implementation of the algorithms utilized in this project [6, 7]. For designing the undersampling masks and the algorithm for the Compressed Sensing, ChatGPT and Claude were used as the primary sources [6, 7]. Additionally, several research articles were referenced to gain an understanding of the objectives and principles behind the various undersampling schemes.

III. RESULTS

Fig. 1 illustrates eight distinct undersampling methods applied to T2-weighted MRI images.

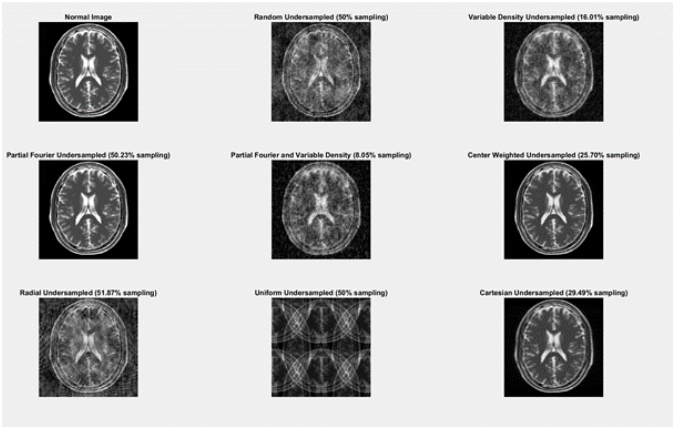


Fig. 1: Undersampling techniques implemented on a T2-weighted brain MRI

Furthermore, Fig. 2 and Fig. 3 depict patients with brain tumors. These subjects were selected as it is crucial to evaluate the ability to distinguish the tumor from surrounding tissue under different undersampling scenarios.

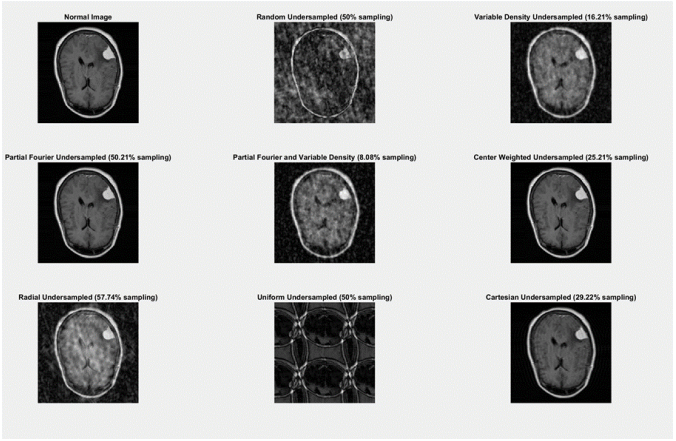


Fig. 2: Undersampling techniques implemented on a brain with a tumor

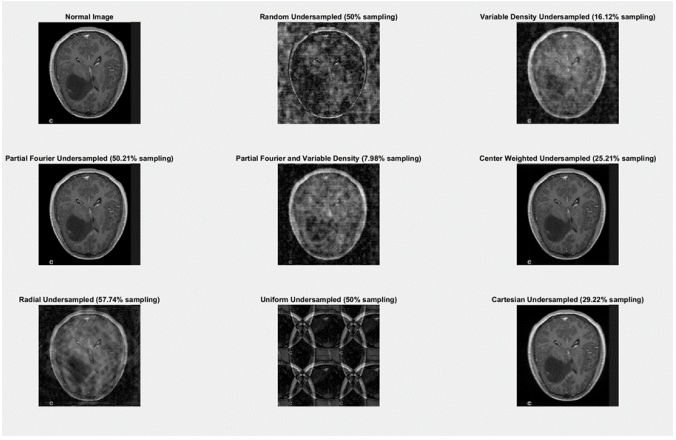


Fig. 3: Undersampling techniques implemented on another brain with tumor

Fig.1, Fig. 2, and Fig. 3 demonstrate that the Partial Fourier Method achieves perfect image recovery, as the images in this study are assumed to be real. Except for Uniform Undersampling and Random Undersampling, the tumor can be clearly distinguished in most cases. However, the Radial Undersampling Method also encounters some difficulty in depicting the tumor. Despite this, most methods maintain good soft tissue contrast. In the case of Uniform Undersampling, aliasing artifacts are present in the image domain due to the conditions outlined in Eq. 1. Notably, the Variable Density and Partial Fourier Methods achieve both good resolution and contrast while utilizing only 25% of the k-space samples, highlighting their efficiency.

Fig. 4 illustrates an example of a compressed sensing technique known as Total Variation Denoising. With a random sampling rate of 0.4, the initial reconstruction results in a very blurry image, making it impossible to distinguish the tumor. However, after applying Total Variation Regularization, the image quality improves significantly, enabling effective detection and diagnosis of the tumor and artifacts are dramatically reduced.

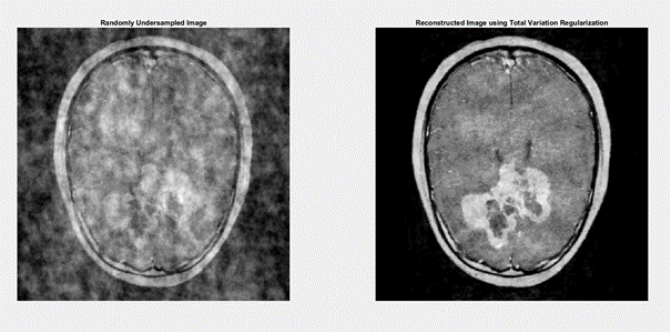


Fig. 4: Total Variation Regularization implemented on a randomly undersampled MRI image

Fig. 5 presents an example of a brain MRI with a tumor. On the left, the k-space of the image is undersampled at a rate of 0.4, resulting in a degraded reconstruction. On the right, the image is reconstructed using Wavelet Sparsity Regularization. As observed, the soft tissue contrast is significantly improved in the right image, allowing the tumorous tissue to be clearly identified.

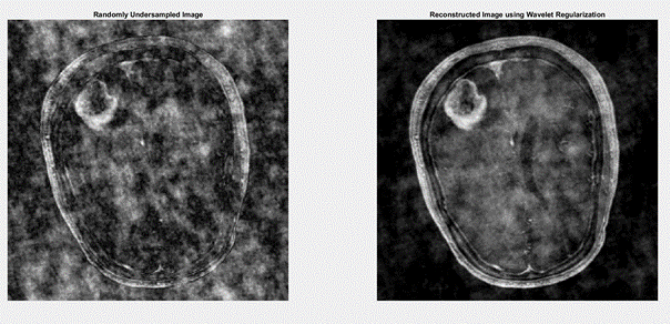


Fig. 5: Wavelet Sparsity Regularization implemented on a randomly undersampled MRI image

In Fig. 1, 2, and 3, aliasing artifacts are observed in the context of uniform undersampling. To mitigate the effects of aliasing, an anti-aliasing filter was applied prior to the undersampling process. Following the application of the anti-aliasing filter and the subsequent undersampling, a Low-Pass Filter (LPF) was employed in the spatial domain to reconstruct the image. Fig. 6 illustrates how the anti-aliasing filter effectively prevents aliasing during the undersampling process.

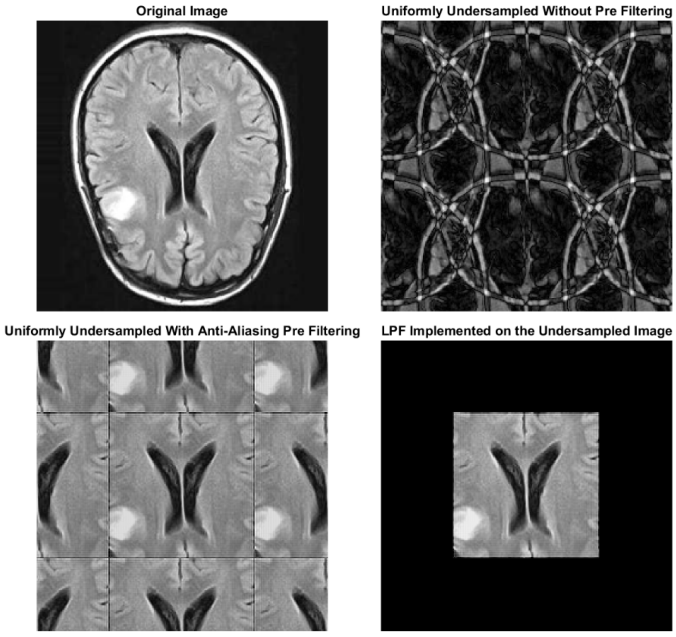


Fig. 6: Anti-Aliasing Filter implemented prior to uniform undersampling

Fig. 6 demonstrates that uniform undersampling in the frequency domain results in aliasing artifacts in the image domain. By applying edge filtering in the image domain prior to undersampling, these aliasing effects are effectively mitigated. Furthermore, a low-pass filter (LPF) was implemented in the spatial domain to isolate the image.

IV. DISCUSSION

The primary objective of this paper was to demonstrate that artifacts caused by undersampling can be minimized by applying various undersampling schemes and Compressed Sensing techniques. Most of these schemes performed as expected, while some required adjustments to be utilized effectively. The most accurate results were achieved using the Partial Fourier Method, as MRI images in this study were assumed to be real. This method successfully reconstructed the image without artifacts. Additionally, methods such as Center Weighted Undersampling, Cartesian Undersampling, and Variable Density Undersampling, prioritizing sampling the central regions of k-space, also produced mostly accurate results across different modalities.

The poorest performance was observed with Uniform Undersampling due to aliasing in the image domain. The sampling in the frequency domain resulted in spatial domain shifts,

but the sampling frequency was insufficient to resolve the images fully. Consequently, overlapping artifacts appeared, leading to a low-resolution image with significant distortions. However, with the help of an Anti-Aliasing Fourier Domain Filter applied beforehand, aliasing could be prevented with the cost of fine details. In Fig. 6, while we observe a loss of detail at the edges of the image, we achieve improved soft tissue contrast and resolution. Despite the edge-related losses, the brain image processed with anti-aliasing filtering allows for the visualization of a tumor that was indistinguishable in the brain image obtained through uniform undersampling without prefiltering.

Random Undersampling also performed poorly. By assigning equal importance to low-frequency and high-frequency regions, this method often undersampled the low-frequency regions, which are critical for MRI due to MRI's sparsity in high frequency regions. As a result, the reconstructed images lacked sufficient quality. However, with the help of Compressed Sensing techniques such as Total Variation Denoising and Wavelet Transform Sparsity Regularization, the randomly undersampled image can be recovered accurately.

Center Weighted Undersampling, which samples the low-frequency regions in a square pattern, yielded images with noticeable blur but preserved the overall structure. In Cartesian Undersampling, the y-directed resolution was nearly as good as the original image, while the x-directed resolution was sacrificed due to the undersampling mask. This method also produced an image resembling the original but with additional blur. Similarly, Variable Density Undersampling prioritized low-frequency regions, resulting in images with reduced sharpness due to blur, yet retaining structural integrity. The combination of the Partial Fourier Method and Variable Density Undersampling exceeded expectations. This approach leveraged the real nature of MRI images, allowing for high-resolution reconstruction despite using only 25% of the k-space samples. Radial Undersampling demonstrated robust performance in scenarios involving motion. Its ability to account for rotational shifts in k-space, common in moving subjects, allowed it to handle motion artifacts effectively. In summary, most undersampling techniques enabled the visualization of the tumor, with sufficient resolution and contrast for evaluation. The study highlights the potential of tailored undersampling schemes to optimize MRI acquisition while maintaining diagnostic image quality.

REFERENCES

[1] C. A. Cocosco, V. Kollokian, R.-S. Kwan, and A. C. Evans, "BrainWeb: Online Interface to a 3D MRI Simulated Brain Database," NeuroImage, vol. 5, no. 4, part 2/4, p. S425, 1997, presented at the 3rd International Conference on Functional Mapping of the Human Brain, Copenhagen, May 1997.

[2] R.-S. Kwan, A. C. Evans, and G. B. Pike, "MRI simulation-based evaluation of image-processing and classification methods," IEEE Transactions on Medical Imaging, vol. 18, no. 11, pp. 1085–1097, Nov. 1999.

[3] R.-S. Kwan, A. C. Evans, and G. B. Pike, "An Extensible MRI Simulator for Post-Processing Evaluation," Lecture Notes in Computer Science, vol. 1131, Visualization in Biomedical Computing (VBC'96), Springer-Verlag, pp. 135–140, 1996.

[4] D. L. Collins, A. P. Zijdenbos, V. Kollokian, J. G. Sled, N. J. Kabani, C. J. Holmes, and A. C. Evans, "Design and Construction of a Realistic Digital Brain Phantom," IEEE Transactions on Medical Imaging, vol. 17, no. 3, pp. 463–468, Jun. 1998.

[5] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," Magnetic Resonance in Medicine, vol. 58, no. 6, pp. 1182–1195, Dec. 2007.

[6] OpenAI, "ChatGPT," OpenAI, San Francisco, CA, USA, 2025. [Online]. Available: https://chat.openai.com.

[7] Anthropic, "Claude," Anthropic, San Francisco, CA, USA, 2025. [Online]. Available: https://www.anthropic.com.

[8] A. Hamada, "Br35H: Brain Tumor Detection 2020," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection?resource=download.

[9] McConnell Brain Imaging Centre, "BrainWeb: Simulated Brain Database," Montreal Neurological Institute, McGill University. [Online]. Available: https://brainweb.bic.mni.mcgill.ca/.

[10] S. Kojima, H. Shinohara, T. Hashimoto, and S. Suzuki, "Undersampling patterns in k-space for compressed sensing MRI using two-dimensional Cartesian sampling," Radiological Physics and Technology, vol. 11, no. 3, pp. 303–319, Sep. 2018.

[11] MRIQuestions, "Partial Fourier Imaging," [Online]. Available: https://mriquestions.com/partial-fourier.html#/.

[12] Pipe JG, "Reconstructing MR images from undersampled data: data-weighting considerations," Magn Reson Med, vol. 43, no. 6, pp. 867–875, Jun. 2000.

[13] M. Murad et al., "Radial Undersampling-Based Interpolation Scheme for Multislice CSMRI Reconstruction Techniques," Journal of Healthcare Engineering, vol. 2021, Article ID 6638588, 2021.

[14] İ. Bayram, "Sparsity Regularization in Signal Processing," [Online]. Available: https://web.itu.edu.tr/ibayram/ehb372e/SparsityReg/.

[15] I. W. Selesnick, "Lecture Notes: Total Variation Denoising and MM Algorithms," [Online]. Available: https://eeweb.engineering.nyu.edu/iselesni/lecture_notes/TVDmm/.

[16] L. Levesque, "Nyquist sampling theorem: Understanding the illusion of a spinning wheel captured with a video camera," Physics Education, vol. 49, no. 6, pp. 697–701, Nov. 2014.

GITHUB REPOSITORY

The source code and additional details can be found in the following GitHub repository:

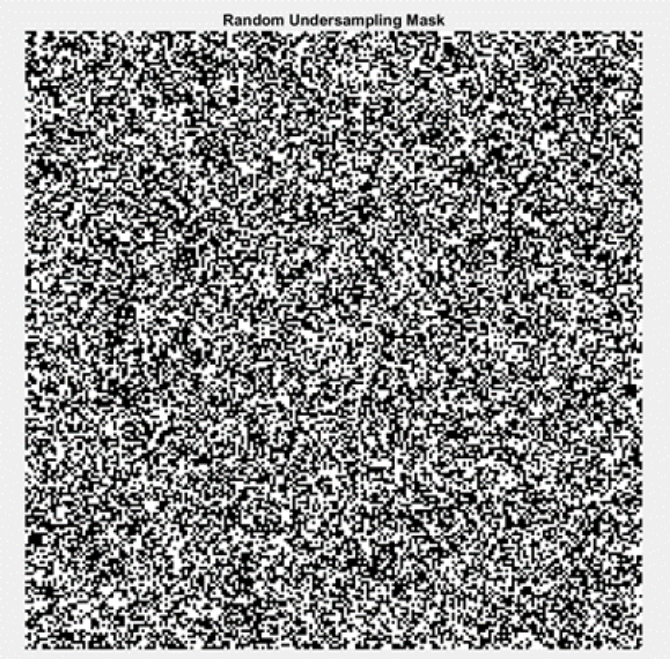
GitHub Repository: UndersamplingArtifactReduction

VIDEO PRESENTATION

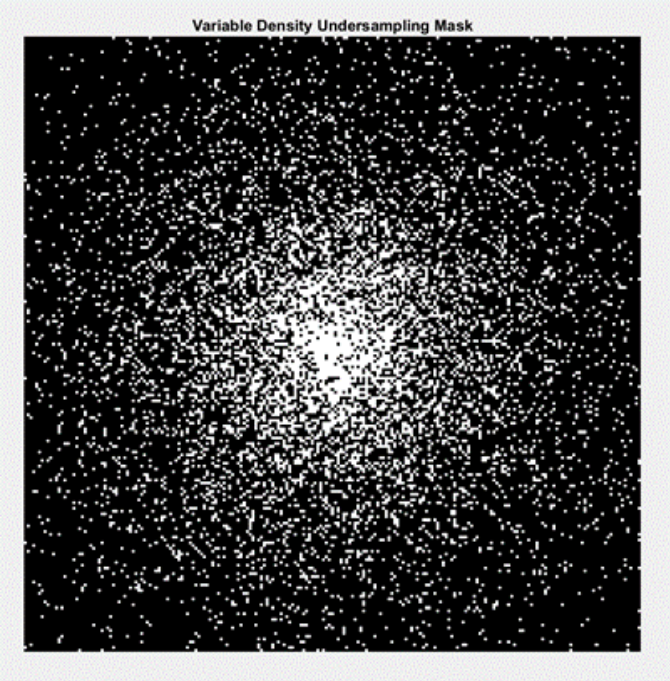
Youtube link for the video presentation can be found in the following link:

Youtube Link: Video Presentation

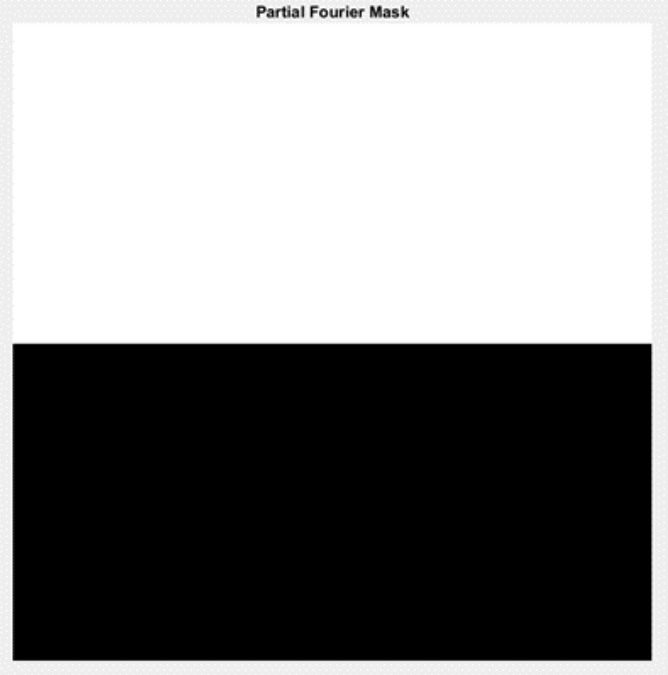
APPENDIX A
APPENDIX A: RANDOM UNDERSAMPLING MASK



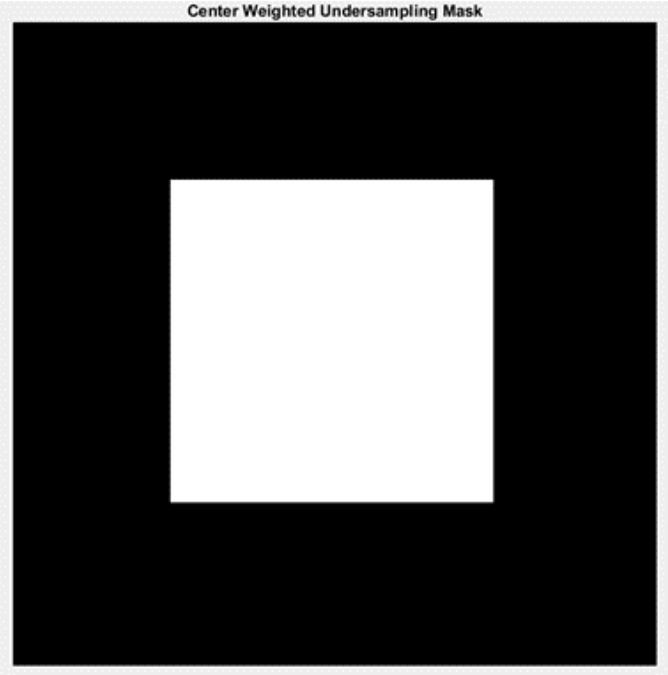
APPENDIX B
APPENDIX B: VARIABLE DENSITY UNDERSAMPLING MASK



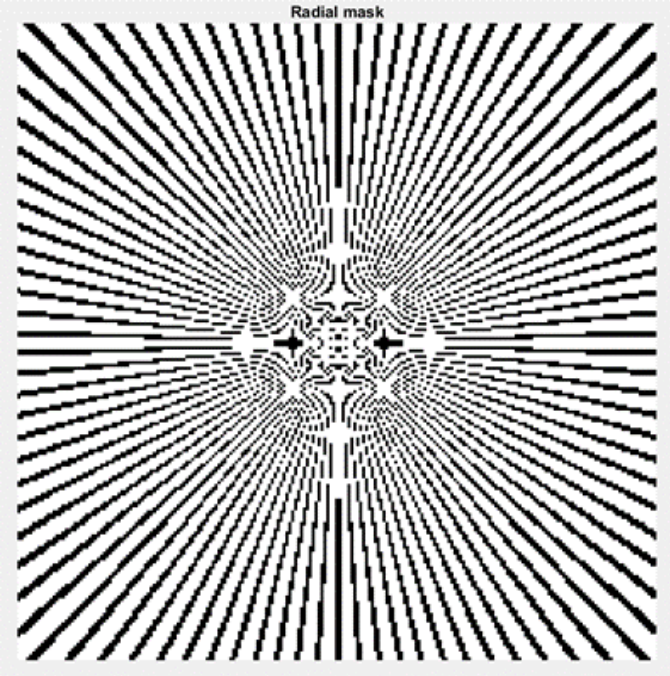
APPENDIX C
APPENDIX C: PARTIAL FOURIER METHOD UNDERSAMPLING MASK



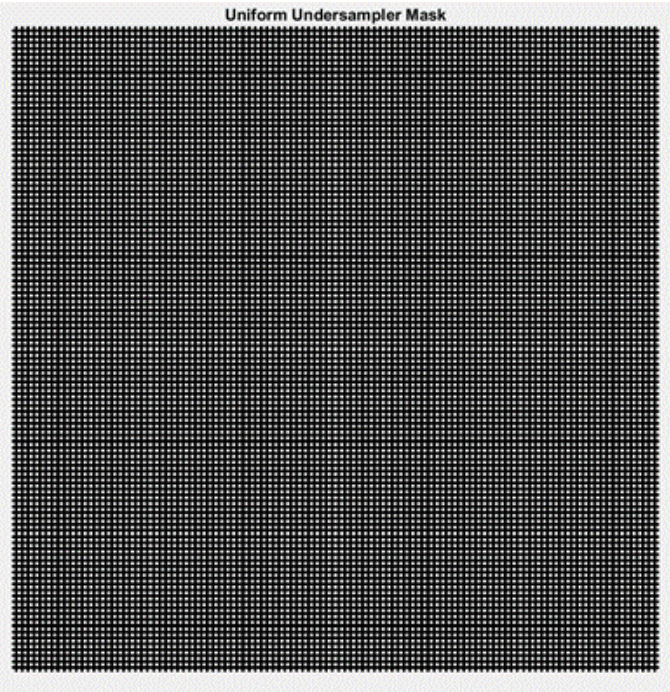
APPENDIX D
APPENDIX D: CENTER-WEIGHTED UNDERSAMPLING MASK



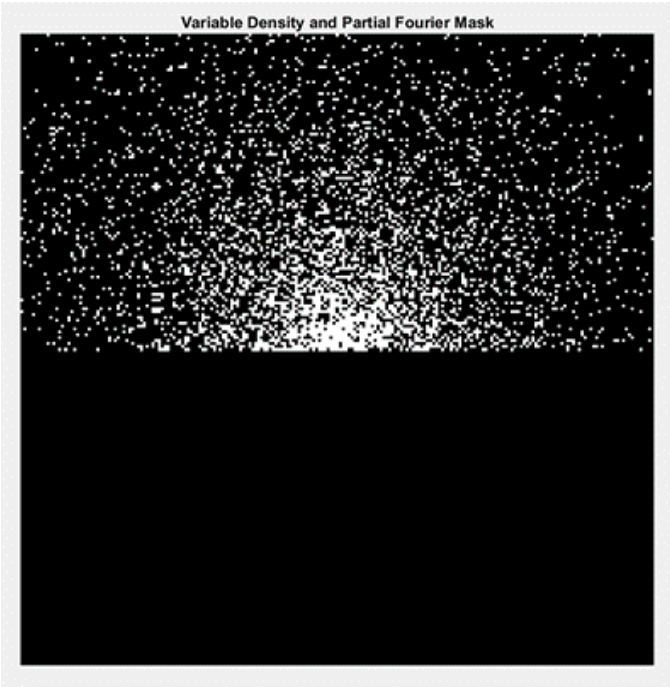
APPENDIX E
APPENDIX E: RADIAL UNDERSAMPLING MASK



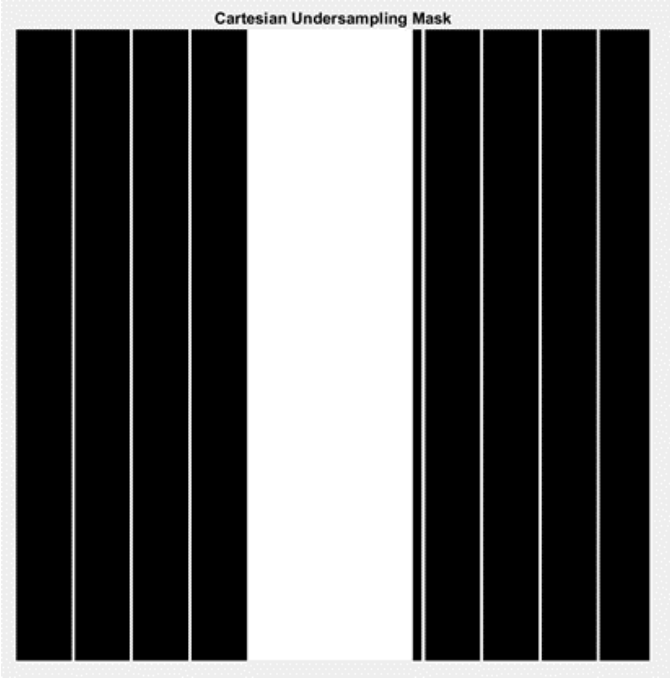
APPENDIX F
APPENDIX F: UNIFORM UNDERSAMPLING MASK



APPENDIX G
APPENDIX G: PARTIAL FOURIER AND VARIABLE DENSITY UNDERSAMPLING MASK



APPENDIX H
APPENDIX H: CARTESIAN UNDERSAMPLING MASK



APPENDIX I: CODE FOR ANTI-ALIASING FILTER

```
close all; clear; clc;
image = imread('y255.jpg');
% Get the size of the image
[height, width, ~] = size(image);
% Calculate the padding needed to make the image
    square
pad_size = (height - width) / 2;
% Add padding to both sides of the width dimension
if pad_size > 0
   % Add padding to make it square
   padded_image = padarray(image, [0,
       floor(pad_size), 0], 'replicate', 'pre');
   padded_image = padarray(padded_image, [0,
       ceil(pad_size), 0], 'replicate', 'post');
else
   disp('Image is already square in width');
end
% Resize the square image to 243x243
resized_image = imresize(padded_image, [243, 243]);
im1 = double(resized_image(:,:,1));
% Design of the Spatial Anti-Aliasing Filter
filter = zeros(size(im1));
[rows, cols] = size(im1);
width = 1/4;
filter(rows*width:rows*(1-width),
    cols*width:cols*(1-width)) = 1;
% Implementation of Anti-Aliasing Filter
c = im1.* filter;
% Uniformly Sampled Without Anti-Aliasing Filter
aliased = UniformUndersampler(im1);
% Uniformly Sampled With Anti-Aliasing Filter
anti_aliased = UniformUndersampler(c);
filtered = anti_aliased .* filter;
figure;
subplot(2,2,1); imshow(im1, []); title('Original
    Image');
subplot(2,2,2); imshow(aliased, []);
    title('Uniformly Undersampled Without Pre
    Filtering');
subplot(2,2,3); imshow(anti_aliased, []);
    title('Uniformly Undersampled With
    Anti-Aliasing Pre Filtering');
subplot(2,2,4); imshow(filtered, []); title('LPF
    Implemented on the Undersampled Image');
```

APPENDIX J: CODE FOR TOTAL VARIATION REGULARIZATION

```
close all; clear; clc;
image = imread('y268.jpg');
% Get the size of the image
[height, width, ~] = size(image);
% Calculate the padding needed to make the image
    square
pad_size = (height - width) / 2;
% Add padding to both sides of the width dimension
if pad_size > 0
   % Add padding to make it square
   padded_image = padarray(image, [0,
       floor(pad_size), 0], 'replicate', 'pre');
   padded_image = padarray(padded_image, [0,
       ceil(pad_size), 0], 'replicate', 'post');
else
   disp('Image is already square in width');
end
% Resize the square image to 243x243
resized_image = imresize(padded_image, [243, 243]);
im1 = double(resized_image(:,:,1));
k_space = fft2c(im1);
% Parameters
lambda = 0.01; % Regularization parameter
num_iters = 700; % Maximum number of iterations
sampling_ratio = 0.4; % Fraction of k-space samples
% Generate sampling mask and implement it on the
    k-space
mask = rand(size(k_space)) < sampling_ratio;
undersampled_k_space = k_space .* mask;
undersampled_image =
    abs(ifft2c(undersampled_k_space)); % Reconstruct
% Initial guess (zero-filled reconstruction)
x = abs(ifft2c(undersampled_k_space));
% Total Variation Regularization using Split Bregman
for iter = 1:num_iters
   % Step 1: Data fidelity (update x to fit
       undersampled k-space)
   x_kspace = fft2c(x);
   x_kspace(mask == 1) = undersampled_k_space(mask
       == 1);
   x = abs(ifft2c(x_kspace));
   % Step 2: Total Variation Minimization
   [dx, dy] = gradient(x); % Compute gradients
   grad_magnitude = sqrt(dx.^2 + dy.^2);
   dx = dx ./ (grad_magnitude + eps); % Normalize
       gradients
   dy = dy ./ (grad_magnitude + eps); % Normalize
       gradients
   x = x - lambda * divergence(dx, dy); % TV update
       step
   % Display progress
   if mod(iter, 10) == 0
       disp(['Iteration ', num2str(iter), '
           completed']);
   end
end
% Display final reconstructed image
subplot(1,2,1); imshow(undersampled_image, []);
    title('Randomly Undersampled Image');
subplot(1,2,2); imshow(x, []); title('Reconstructed
    Image using Total Variation Regularization');
```

APPENDIX K: CODE FOR WAVELET SPARSITY REGULARIZATION

```
image = imread('y110.jpg');
% Get the size of the image
[height, width, ~] = size(image);
% Calculate the padding needed to make the image
    square
pad_size = (height - width) / 2;
% Add padding to both sides of the width dimension
if pad_size > 0
   % Add padding to make it square
   padded_image = padarray(image, [0,
       floor(pad_size), 0], 'replicate', 'pre');
   padded_image = padarray(padded_image, [0,
       ceil(pad_size), 0], 'replicate', 'post');
else
   disp('Image is already square in width');
end
% Resize the square image to 243x243
resized_image = imresize(padded_image, [243, 243]);
im1 = double(resized_image(:,:,1));
k_space = fft2c(im1);
% Parameters
lambda = 0.005; % Regularization parameter
num_iters = 300; % Maximum number of iterations
sampling_ratio = 0.4; % Fraction of k-space samples
% Generate sampling mask and implement it on the
    k-space
mask = rand(size(k_space)) < sampling_ratio;
undersampled_k_space = k_space .* mask;
undersampled_image =
    abs(ifft2c(undersampled_k_space)); % Reconstruct
% Initial guess (zero-filled reconstruction)
x = abs(ifft2c(undersampled_k_space));
% Define wavelet transform and its inverse
wavelet_name = 'db1'; % Daubechies wavelet
```

```
decomposition_level = 2; % Number of levels for
    wavelet decomposition
% Iterative reconstruction
for iter = 1:num_iters
   % Step 1: Data fidelity
   x_kspace = fft2c(x);
   x_kspace(mask == 1) = undersampled_k_space(mask
       == 1);
   x = abs(ifft2c(x_kspace));
   % Step 2: Wavelet sparsity regularization
   [c, s] = wavedec2(x, decomposition_level,
       wavelet_name); % Wavelet decomposition
   c = sign(c) .* max(abs(c) - lambda, 0); % Soft
       thresholding
   x = waverec2(c, s, wavelet_name); % Reconstruct
       from thresholded coefficients
   % Display progress
   if mod(iter, 10) == 0
       disp(['Iteration ', num2str(iter), '
           completed']);
   end
end
% Display final reconstructed image
subplot(1,2,1); imshow(undersampled_image, []);
    title('Randomly Undersampled Image');
subplot(1,2,2); imshow(x, []); title('Reconstructed
    Image using Wavelet Regularization');
```

APPENDIX L: CODE FOR IMPLEMENTATION OF THE UNDERSAMPLING SCHEMES FOR TUMOROUS AND T2-WEIGHTED IMAGES

```
close all; clear; clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% rgb image processing %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% load the rgb image (243x205x3)
image = imread('y710.jpg');
% get image dimensions
[height, width, ~] = size(image);
% calculate padding for square image
pad_size = (height - width) / 2;
% make the image square using padding
if pad_size > 0
   padded_image = padarray(image, [0,
       floor(pad_size), 0], 'replicate', 'pre');
   padded_image = padarray(padded_image, [0,
       ceil(pad_size), 0], 'replicate', 'post');
else
   disp('image is already square in width');
end
% resize to 243x243
resized_image = imresize(padded_image, [243, 243]);
% extract first channel and show original
im1 = double(resized_image(:,:,1));
figure;
imshow(im1, []);
% apply undersampling methods
random1 = RandomUndersampler(im1);
[pfourier1, pfourier_mask] =
    PartialFourierMethod(im1);
[vardens_pfourier1, vardens_pfourier_mask] =
    VarDensityandPartialFourier(im1);
[rec1, vardens_mask] =
    VariableDensityUndersampler(im1);
[radial1, sampling_mask] = RadialUndersampler(im1);
[center_weighted, center_mask] =
    CenterWeighted(im1);
uniform1 = UniformUndersampler(im1);
[cartesian1, cartesian_mask] =
    CartesianUndersampler(im1);
subplot(3,3,1);
imshow(im1, []);
title('Normal Image');
subplot(3,3,2);
imshow(random1, []);
title('Random Undersampled (50% sampling)');
subplot(3,3,3);
imshow(rec1, []);
sampling_rate =
    sum(vardens_mask(:))/numel(vardens_mask);
title(sprintf('Variable Density Undersampled
    (%.2f%% sampling)', sampling_rate*100));
subplot(3,3,4);
imshow(pfourier1, []);
sampling_rate =
    sum(pfourier_mask(:))/numel(pfourier_mask);
title(sprintf('Partial Fourier Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,5);
imshow(vardens_pfourier1, []);
sampling_rate =
    sum(vardens_pfourier_mask(:))/numel(vardens_pfourier_mask);
title(sprintf('Partial Fourier and Variable Density
    (%.2f%% sampling)', sampling_rate*100));
subplot(3,3,6);
imshow(center_weighted, []);
sampling_rate =
    sum(center_mask(:))/numel(center_mask);
title(sprintf('Center Weighted Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,7);
imshow(radial1, []);
sampling_rate =
    sum(sampling_mask(:))/numel(sampling_mask);
title(sprintf('Radial Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,8);
imshow(uniform1, []);
title('Uniform Undersampled (50% sampling)');
subplot(3,3,9);
imshow(cartesian1, []);
sampling_rate =
    sum(cartesian_mask(:))/numel(cartesian_mask);
title(sprintf('Cartesian Undersampled (%.2f%%
    sampling)', sampling_rate*100));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% t2 weighted image processing %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
% read raw data
filename = 't2_icbm_normal_3mm_pn0_rf0.raws';
fileID = fopen(filename, 'r');
raw_data = fread(fileID, [181, 217*181], 'int16');
fclose(fileID);
imageInfo = imageInfo(filename);
% reshape volume
volume = reshape(raw_data, [181, 217, 60]);
% extract and process middle slice
axial_slice = rot90(volume(:,:,31), 1);
[h, w] = size(axial_slice);
if h > w
   axial_padded = padarray(axial_slice, [0
       floor((h-w)/2)], 'replicate', 'both');
else
   axial_padded = padarray(axial_slice,
       [floor((w-h)/2) 0], 'replicate', 'both');
end
% apply undersampling methods
im1 = axial_padded;
random1 = RandomUndersampler(im1);
[pfourier1, pfourier_mask] =
    PartialFourierMethod(im1);
[vardens_pfourier1, vardens_pfourier_mask] =
    VarDensityandPartialFourier(im1);
[rec1, vardens_mask] =
    VariableDensityUndersampler(im1);
[radial1, sampling_mask] = RadialUndersampler(im1);
[center_weighted, center_mask] =
    CenterWeighted(im1);
uniform1 = UniformUndersampler(im1);
```

```
[cartesian1, cartesian_mask] =
    CartesianUndersampler(im1);
% show results
subplot(3,3,1);
imshow(im1, []);
title('Normal Image');
subplot(3,3,2);
imshow(random1, []);
title('Random Undersampled (50% sampling)');
subplot(3,3,3);
imshow(rec1, []);
sampling_rate =
    sum(vardens_mask(:))/numel(vardens_mask);
title(sprintf('Variable Density Undersampled
    (%.2f%% sampling)', sampling_rate*100));
subplot(3,3,4);
imshow(pfourier1, []);
sampling_rate =
    sum(pfourier_mask(:))/numel(pfourier_mask);
title(sprintf('Partial Fourier Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,5);
imshow(vardens_pfourier1, []);
sampling_rate =
    sum(vardens_pfourier_mask(:))/numel(vardens_pfouri
title(sprintf('Partial Fourier and Variable Density
    (%.2f%% sampling)', sampling_rate*100));
subplot(3,3,6);
imshow(center_weighted, []);
sampling_rate =
    sum(center_mask(:))/numel(center_mask);
title(sprintf('Center Weighted Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,7);
imshow(radial1, []);
sampling_rate =
    sum(sampling_mask(:))/numel(sampling_mask);
title(sprintf('Radial Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,8);
imshow(uniform1, []);
title('Uniform Undersampled (50% sampling)');
subplot(3,3,9);
imshow(cartesian1, []);
sampling_rate =
    sum(cartesian_mask(:))/numel(cartesian_mask);
title(sprintf('Cartesian Undersampled (%.2f%%
    sampling)', sampling_rate*100));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% rgb image processing %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% load the rgb image (243x205x3)
image = imread('y143.jpg');
% get image dimensions
[height, width, ~] = size(image);
% calculate padding for square image
pad_size = (height - width) / 2;
% make the image square using padding
if pad_size > 0
   padded_image = padarray(image, [0,
       floor(pad_size), 0], 'replicate', 'pre');
   padded_image = padarray(padded_image, [0,
       ceil(pad_size), 0], 'replicate', 'post');
else
   disp('image is already square in width');
end
% resize to 243x243
resized_image = imresize(padded_image, [243, 243]);
% extract first channel and show original
im1 = double(resized_image(:,:,1));
figure;
imshow(im1, []);
% apply undersampling methods
random1 = RandomUndersampler(im1);
[pfourier1, pfourier_mask] =
    PartialFourierMethod(im1);
[vardens_pfourier1, vardens_pfourier_mask] =
    VarDensityandPartialFourier(im1);
[rec1, vardens_mask] =
    VariableDensityUndersampler(im1);
[radial1, sampling_mask] = RadialUndersampler(im1);
[center_weighted, center_mask] =
    CenterWeighted(im1);
uniform1 = UniformUndersampler(im1);
[cartesian1, cartesian_mask] =
    CartesianUndersampler(im1);
subplot(3,3,1);
imshow(im1, []);
title('Normal Image');
subplot(3,3,2);
imshow(random1, []);
title('Random Undersampled (50% sampling)');
subplot(3,3,3);
imshow(rec1, []);
sampling_rate =
    sum(vardens_mask(:))/numel(vardens_mask);
title(sprintf('Variable Density Undersampled
    (%.2f%% sampling)', sampling_rate*100));
subplot(3,3,4);
imshow(pfourier1, []);
sampling_rate =
    sum(pfourier_mask(:))/numel(pfourier_mask);
title(sprintf('Partial Fourier Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,5);
imshow(vardens_pfourier1, []);
sampling_rate =
    sum(vardens_pfourier_mask(:))/numel(vardens_pfourier_mask);
title(sprintf('Partial Fourier and Variable Density
    (%.2f%% sampling)', sampling_rate*100));
subplot(3,3,6);
imshow(center_weighted, []);
sampling_rate =
    sum(center_mask(:))/numel(center_mask);
title(sprintf('Center Weighted Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,7);
imshow(radial1, []);
sampling_rate =
    sum(sampling_mask(:))/numel(sampling_mask);
title(sprintf('Radial Undersampled (%.2f%%
    sampling)', sampling_rate*100));
subplot(3,3,8);
imshow(uniform1, []);
title('Uniform Undersampled (50% sampling)');
subplot(3,3,9);
imshow(cartesian1, []);
sampling_rate =
    sum(cartesian_mask(:))/numel(cartesian_mask);
title(sprintf('Cartesian Undersampled (%.2f%%
    sampling)', sampling_rate*100));

\end{verbatim}

\section*{Appendix M: Code For Functions of The
    Undersampling Techniques}
\begin{verbatim}
function [undersampled_image] =
    RandomUndersampler(image1)
   % convert to k-space
   k_space = fft2c(image1);

   % create random sampling pattern (50% sampling
       rate)
   undersampling_mask = rand(size(k_space)) > 0.5;

   % apply mask to k-space
   undersampled_k_space = k_space .*
       undersampling_mask;

   % show the sampling pattern
   figure;
   imshow(undersampling_mask, []);
```

```
   title('random sampling mask');

   % convert back to image space
   undersampled_image =
       abs(ifft2c(undersampled_k_space));
end
function [reconstructed_image, undersampling_mask]
    = VariableDensityUndersampler(image1)
   % convert to k-space
   k_space = fft2c(image1);
   % get image dimensions
   nx = size(image1,1);
   ny = size(image1,2);
   % create coordinate grid for k-space
   [x, y] = meshgrid(-nx/2:nx/2-1, -ny/2:ny/2-1);
   % calculate distance from center for each point
   distance_from_center = sqrt(x.^2 + y.^2);
   max_distance = max(distance_from_center(:));
   % create probability map (higher chance to
       sample near center)
   sampling_probability = exp(-distance_from_center
       / (max_distance / 4));
   % generate random sampling pattern based on
       probability
   vardens_undersampling_mask = rand(nx, ny) <
       sampling_probability;
   % calculate actual sampling percentage
   sampling_rate =
       sum(vardens_undersampling_mask(:)) /
       numel(vardens_undersampling_mask);
   % show the sampling pattern
   figure;
   imshow(vardens_undersampling_mask, []);
   title(sprintf('variable density mask (%.2f%%
       sampling)', sampling_rate*100));
   % apply mask to k-space
   vardens_sampled_k_space = k_space .*
       vardens_undersampling_mask;
   % convert back to image space
   reconstructed_image =
       abs(ifft2c(vardens_sampled_k_space));
   % save mask for later use
   undersampling_mask = vardens_undersampling_mask;
end
function [undersampled_image, sampling_mask] =
    PartialFourierMethod(image)
   % get image size
   [nx, ny] = size(image);
   % convert to k-space
   k_space = fft2c(image);
   % create mask for upper half of k-space
   mask = zeros(nx, ny);
   mask(1:ceil(nx/2), :) = 1;
   % calculate sampling percentage
   sampling_rate = sum(mask(:)) / numel(mask);
   % show the sampling pattern
   figure;
   imshow(mask, []);
   title(sprintf('partial fourier mask (%.2f%%
       sampling)', sampling_rate*100));
   % apply mask to k-space
   undersampled_k_space = k_space .* mask;
   % use symmetry to fill in bottom half
   for x = ceil(nx/2)+1:nx
       for y = 1:ny
           % find matching point in top half
           sym_x = nx - x + 1;
           sym_y = ny - y + 1;
           % check if point exists and copy it
           if sym_x >= 1 && sym_x <= nx && sym_y >=
               1 && sym_y <= ny
               undersampled_k_space(x, y) =
                   conj(undersampled_k_space(sym_x,
                   sym_y));
           end
       end
   end
   % convert back to image space
   undersampled_image =
       abs(ifft2c(undersampled_k_space));
   % save mask for later use
   sampling_mask = mask;
end
function [undersampled_image, sampling_mask] =
    CenterWeighted(image)
   % get image size
   [nx, ny] = size(image);
   % convert to k-space (frequency domain)
   k_space = fft2c(image);
   % create empty mask
   mask = zeros(nx, ny);
   % calculate center region boundaries (middle 50%
       of k-space)
   center_start_x = round(nx/4);
   center_end_x = round(3*nx/4);
   center_start_y = round(ny/4);
   center_end_y = round(3*ny/4);
   % fill in center region of mask
   mask(center_start_x:center_end_x,
       center_start_y:center_end_y) = 1;
   % calculate how much of k-space we're actually
       sampling
   sampling_rate = sum(mask(:)) / numel(mask);
   % show the sampling pattern
   figure;
   imshow(mask, []);
   title(sprintf('center weighted mask (%.2f%%
       sampling)', sampling_rate*100));
   % apply mask to k-space data
   undersampled_k_space = k_space .* mask;
   % convert back to image space
   undersampled_image =
       abs(ifft2c(undersampled_k_space));
   % save mask for later use
   sampling_mask = mask;
end
function [undersampled_image, sampling_mask] =
    RadialUndersampler(image)
   % convert to k-space
   k_space = fft2c(image);
   [nx, ny] = size(k_space);
   % calculate spoke parameters for ~50% sampling
   radius = min(nx, ny)/2;
   num_spokes = round(0.17 * (nx*ny)/(2*radius));
   angles = linspace(0, 180, num_spokes);
   % set up radial sampling pattern
   [kx, ky] = meshgrid(-nx/2:nx/2-1, -ny/2:ny/2-1);
   mask = zeros(nx, ny);
   % convert coordinates to angles
   theta = atan2d(ky, kx);
   theta(theta < 0) = theta(theta < 0) + 180;
   % create the spoke pattern
   for angle = angles
       tolerance = 1.3;
       mask(abs(theta - angle) < tolerance) = 1;
   end
   % calculate how much we're sampling
   sampling_rate = sum(mask(:))/numel(mask);
   % show the sampling pattern
   figure;
   imshow(mask, []);
   title(sprintf('radial mask (sampling rate:
       %.2f%%)', sampling_rate*100));
   % apply mask and reconstruct
   undersampled_k_space = k_space .* mask;
   undersampled_image =
       abs(ifft2c(undersampled_k_space));
   sampling_mask = mask;
end
function [undersampled_image] =
    UniformUndersampler(image)
   % get image dimensions
   [nx, ny] = size(image);
   % convert to k-space
```

```
    k_space = fft2c(image);
    % set uniform sampling rate (take every other
        point)
    sampling_factor = 2;
    % create uniform sampling pattern
    mask = zeros(nx, ny);
    mask(1:sampling_factor:end,
        1:sampling_factor:end) = 1;
    % show the sampling pattern
    figure;
    imshow(mask, []);
    title('uniform sampling mask');
    % apply mask and reconstruct
    undersampled_k_space = k_space .* mask;
    undersampled_image =
        abs(ifft2c(undersampled_k_space));
end
function [undersampled_image, sampling_mask] =
    VarDensityandPartialFourier(image)
    % convert to k-space
    k_space = fft2c(image);
    [nx, ny] = size(image);
    % set up coordinate grid for variable density
        sampling
    [x, y] = meshgrid(-nx/2:nx/2-1, -ny/2:ny/2-1);

    % calculate distance from center for each point
    distance_from_center = sqrt(x.^2 + y.^2);
    max_distance = max(distance_from_center(:));

    % create probability map (higher chance to
        sample near center)
    sampling_probability = exp(-distance_from_center
        / (max_distance / 4));

    % create random sampling pattern
    vardens_undersampling_mask = rand(nx, ny) <
        sampling_probability;

    % create mask for upper half of k-space
    partial_fourier_mask = zeros(nx, ny);
    partial_fourier_mask(1:ceil(nx/2), :) = 1;

    % combine both sampling patterns
    total_mask = vardens_undersampling_mask .*
        partial_fourier_mask;

    % calculate actual sampling percentage
    sampling_rate = sum(total_mask(:)) /
        numel(total_mask);

    % show the combined sampling pattern
    figure;
    imshow(total_mask, []);
    title(sprintf('variable density + partial
        fourier mask (%.2f%% sampling)',
        sampling_rate*100));

    % apply mask to k-space
    undersampled_k_space = k_space .* total_mask;

    % use symmetry to fill in bottom half
    for x = ceil(nx/2)+1:nx
        for y = 1:ny
            % find matching point in top half
            sym_x = nx - x + 1;
            sym_y = ny - y + 1;

            % check if point exists and copy it
            if sym_x >= 1 && sym_x <= nx && sym_y >=
                1 && sym_y <= ny
                undersampled_k_space(x, y) =
                    conj(undersampled_k_space(sym_x,
                    sym_y));
            end
        end
    end

    % convert back to image space
    undersampled_image =
        abs(ifft2c(undersampled_k_space));

    % save mask for later use
    sampling_mask = total_mask;
end
function [undersampled_image, sampling_mask] =
    CartesianUndersampler(image)
    % convert to k-space
    k_space = fft2c(image);
    [nx, ny] = size(image);
    % set sampling parameters
    center_fraction = 0.25;    % fully sample the
        middle 25%
    outer_fraction = 0.05;     % sample 5% of the
        outer regions
    % create empty mask
    mask = zeros(nx, ny);
    % define central region
    center_start = round((1 - center_fraction) * ny
        / 2);
    center_end = round((1 + center_fraction) * ny /
        2);
    % sample all columns in the center
    mask(:, center_start:center_end) = 1;
    % calculate how often to sample in outer regions
    outer_sampling_factor = round(1 /
        outer_fraction);
    % sample left side columns
    for col = 1:center_start-1
        if mod(col, outer_sampling_factor) == 0
            mask(:, col) = 1;
        end
    end
    % sample right side columns
    for col = center_end+1:ny
        if mod(col, outer_sampling_factor) == 0
            mask(:, col) = 1;
        end
    end
    % calculate actual sampling percentage
    sampling_rate = sum(mask(:)) / numel(mask);
    % show the sampling pattern
    figure;
    imshow(mask, []);
    title(sprintf('cartesian sampling mask (%.2f%%
        sampling)', sampling_rate*100));
    % apply mask to k-space
    undersampled_k_space = k_space .* mask;
    % convert back to image space
    undersampled_image =
        abs(ifft2c(undersampled_k_space));
    % save mask for later use
    sampling_mask = mask;
end
```