Medical Imaging: MRI Acquisition Simulator

University of Houston, Fall 2023

Members of the Team:

Zachary Winter, 2022232, Department of Computer Science, Undergraduate

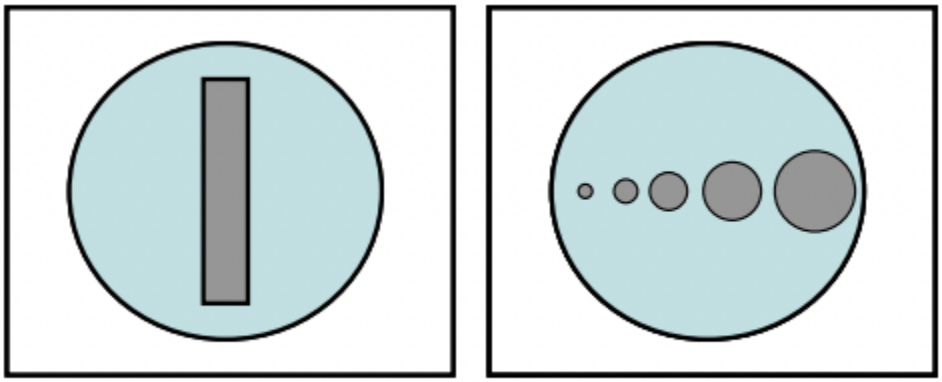
Ian Wulf, 1189329, Department of Computer Science, Undergraduate

Lisa Zuniga, 2052486, Department of Computer Science, Undergraduate

Meron Dessie, 2076296, Department of Computer Science, Undergraduate

Introduction:

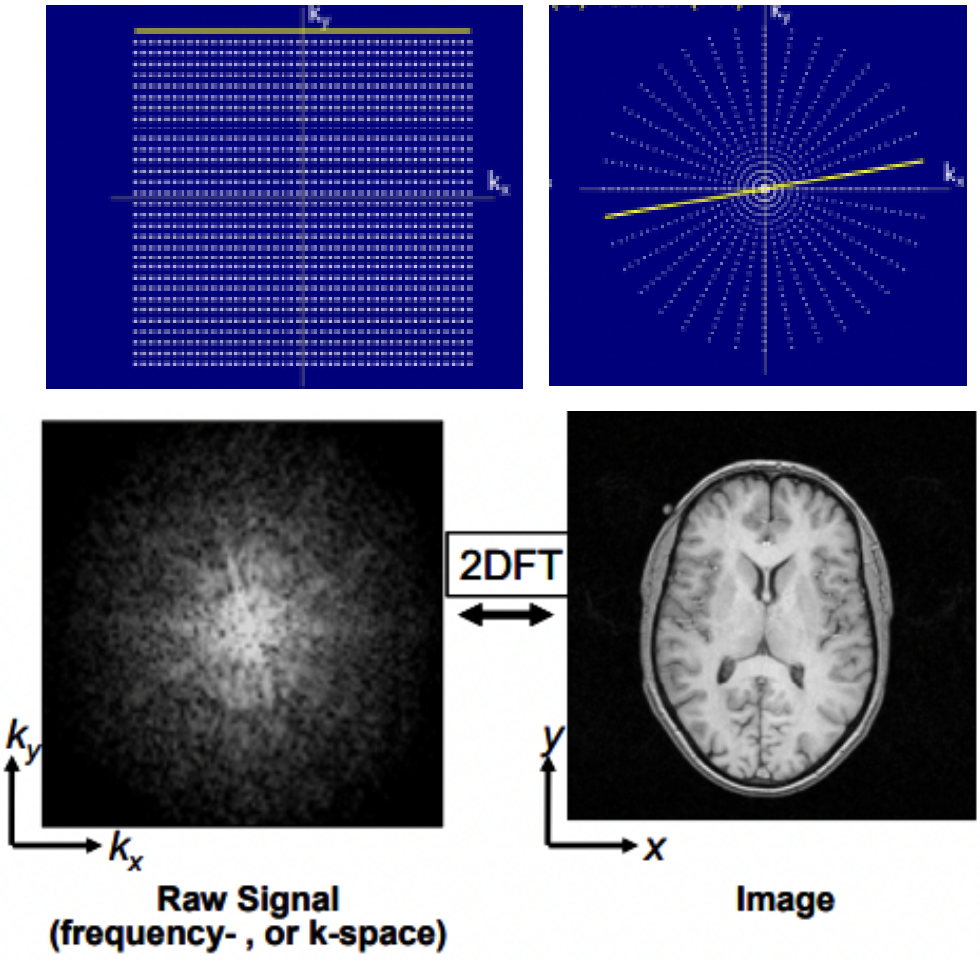
This project aims to develop software that simulates data acquisition for a virtual MRI scanner, crucial in understanding MRI data collection occurring in the frequency domain or k-space. The reciprocal relationship between real-space and k-space, navigated via forward or inverse Fourier transform (FT), forms the foundation. MATLAB's efficient FFT function serves as the basis for this simulation, coupled with a focus on exploring various digitization paths in k-space, known as k-space trajectories. The objective is to evaluate the impact of different k-space trajectories and collection strategies, along with acquiring selective portions of k-space, on the quality of reconstructed images. Validation involves utilizing two test phantoms—Phantom 1 and Phantom 2—comprising circles with internal structures, varying in intensities to simulate distinct tissue features.



Phantom 1 *Phantom 2*

Objective:

The primary goal is to assess the imaging outcomes by employing different k-space trajectories and sampling strategies in the simulation of MRI data acquisition. Specifically, the focus lies on evaluating the Cartesian trajectory (or 2DFT spinwarp) and the radial trajectory in sampling the k-space of the provided test phantoms. Within these trajectories, parameters such as the number/density of horizontal lines, density of points within a line, swapping horizontal and vertical lines, and the width of the acquisition matrix for Cartesian trajectory, and the number of radial lines and points along each line for the radial trajectory will be systematically modified and studied. The objective is to comprehensively understand how alterations in data acquisition parameters impact the quality and fidelity of reconstructed images from the acquired k-space data.



Methods:

Our code aims to perform various functions related to MRI simulation, including: generating phantoms, operating the scanner, reconstructing images, and conducting image analysis. Additionally, it aims to implement a GUI that combines these functionalities. Our code involves generating test/validation phantoms with specific structures, performing image analysis on the generated images, and investigating the impact of different acquisition parameters on the image quality.

- ***Generation of Phantoms*:** The code includes functions to generate two test/validation phantoms: **genPhantom1** and **genPhantom2**. These functions create circular structures with specific dimensions and values inside a background circle.
- ***Scanner Operation*:** The code simulates the operation of a scanner by performing Cartesian and radial sampling (**cartesian** and **radial** functions) of the generated phantoms. These functions utilize Fourier transforms to sample the k-space and perform image reconstruction based on different acquisition parameters.
- ***Image Analysis*:** The code implements image analysis functionalities such as signal intensity (SI) and contrast calculation, image difference analysis, and signal intensity profiles. These analyses aid in evaluating the accuracy of image reconstruction and identifying artifacts.
- ***GUI Implementation*:** While the code lacks the complete implementation of a GUI, it outlines functions for the scanner part of the GUI that allows users to change acquisition parameters (number of detectors, detector distance, type of array, etc.).

The GUI is intended to facilitate parameter adjustments and activate the acquisition process.

- ***Investigations and Testing*:** The code contains comments and functions intended for investigating the effects of changing acquisition parameters on the image quality. These investigations involve altering rotation angles, detector numbers/distance, analyzing circular structures, assessing rectangular structure edges, determining the overall image impression, and studying the impact of acquisition time on image quality.

Issues:

The initial process of generating images faced a significant obstacle - the resulting images were completely different from the original phantom. We attempted Cartesian sampling through the Fast Fourier Transform (FFT), establishing grids, and copying k-space values. The plan seemed promising in theory, but when implementation came – proved otherwise. When we finally loaded the reconstructed images, they hardly resembled the original phantom we had started with. The generated images appeared with little resemblance compared to the original phantom. This inconsistency disrupted our progress, indicating that the process required significant adjustments to capture the true essence of the initial phantom.

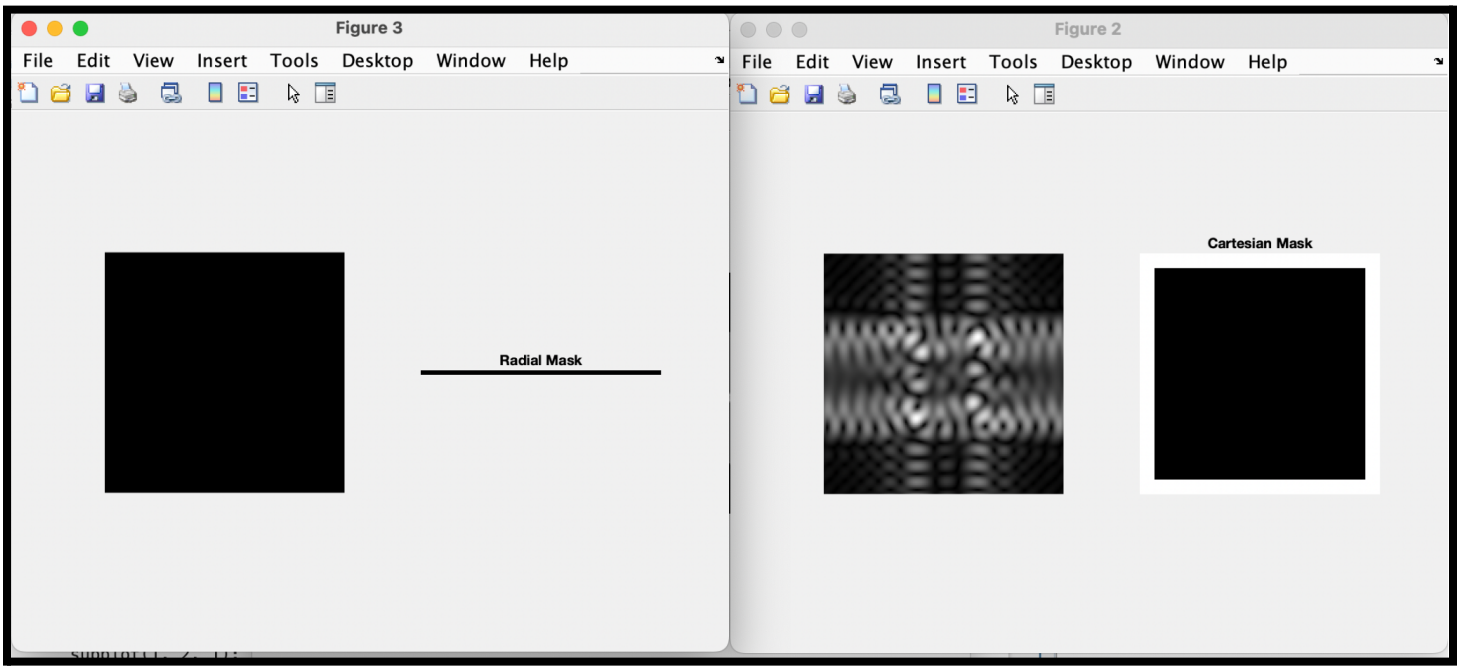


FIGURE ABOVE SHOWS INITIAL ATTEMPTS

Results and Discussion:

The generated images with varying backgrounds and colored phantoms are achieved by adjusting parameters within the cartesian() and radial() functions utilized in the main() function. The cartesian() and radial() functions reconstruct images from different phantoms (IM1 and IM2) employing either Cartesian or radial sampling methods. These functions allow for modifications in the background of the phantom images, such as altering the canvas size, setting different shapes (e.g., circles, rectangles), and changing their sizes and positions within the canvas. Additionally, adjustments in color and contrast are facilitated by manipulating parameters like color specifications and grayscale conversion. By varying these parameters within the functions called by main(), diverse phantom images are generated, showcasing different backgrounds, shapes, and color contrasts within the resultant images.

FIGURE 1:
IM1_C = cartesian(IM1, 100, 100)

FIGURE 2:
IM1_C = cartesian(IM1, 200, 200)

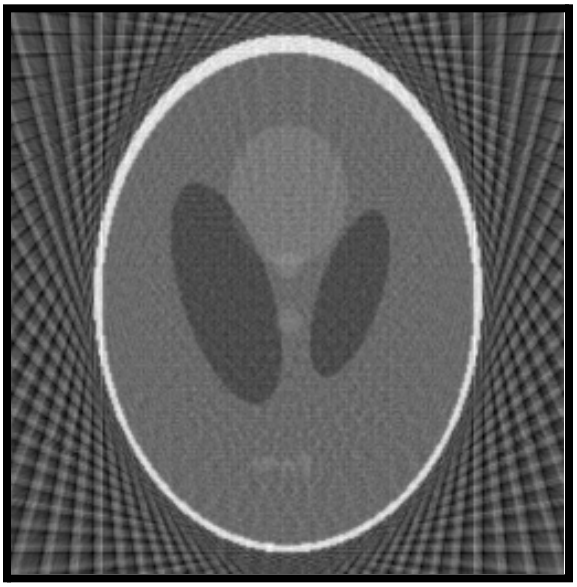


FIGURE 3:
IM1_C = radial(IM1, 100, 100)

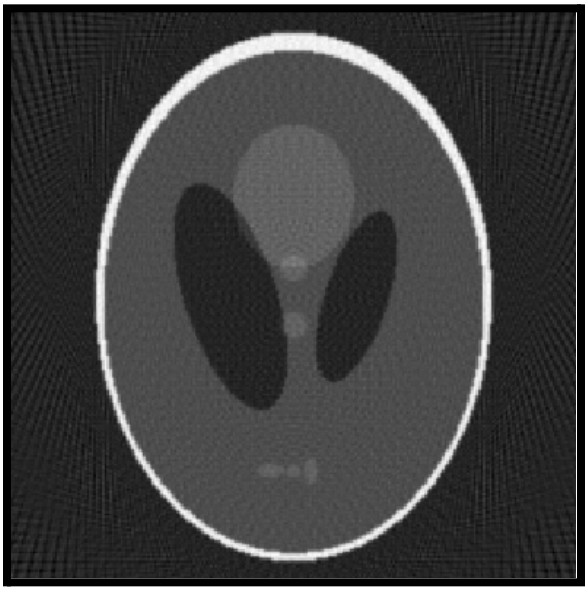


FIGURE 4:
IM1_C = radial(IM1, 200, 200)

Discussion:

The level of detail and sharpness in the images is directly influenced by the sampling parameters used in the cartesian() and radial() functions. Increasing the number of lines and points per line in the radial() function, such as moving from radial(IMC1100, 100) to radial(200, 200), enhances the image's clarity and precision. With higher sampling rates, the imaging process captures more information, resulting in increased resolution and finer details within the reconstructed image. This progression showcases how augmenting the sampling parameters significantly improves the image's fidelity and sharpness, allowing for a more detailed representation of the phantom structures. The figures below (FIGURE 5 and FIGURE 6) correspond to the above mentioned phantoms (FIGURE 1 and FIGURE 2 as well as FIGURE 7 and FIGURE 8 corresponding to FIGURE 3 and FIGURE 4).

FIGURE 5

FIGURE 6

FIGURE 7

FIGURE 8

Conclusion:

The code provided sets the stage for manipulating images by altering various parameters within functions like genPhantom1() and genPhantom2(), and we do this specifically using the radial() and cartesian() functions. These functions create test images with different shapes, sizes, and backgrounds—such as circles and rectangles—with the ability to adjust dimensions, positions, and color contrasts. The cartesian() and radial() functions further demonstrate the impact of parameter changes on image reconstruction techniques. By modifying parameters like sampling density, grid dimensions, and transformation methods, these functions yield distinct reconstructed images, showcasing how parameter adjustments influence image sharpness, detail, and overall quality. These parameter tweaks allow for exploration of different sampling methodologies and their effects on the resultant images, highlighting the importance of parameter manipulation in image generation and analysis.

main.m:

```
function main()
    [IM1, IM2] = genTestImages(); % generates both test phantom images
    IM3 = imread('brain.jpg'); % medical image of brain

    %{ Uncomment the trajectory you wish to test %}
    %{ Parameters: (image, number of lines, points per line %}
    %IM4 = cartesian(IM1, 200, 200);
    %IM4 = radial(IM1, 200, 200);

    imshow(IM4, []); % displays final reconstructed image in new window
end


function [i1, i2] = genTestImages()
    i1 = genPhantom1();
    i2 = genPhantom2();
end

function p1 = genPhantom1()
    canvasSize = [300, 300];
    mat = zeros(canvasSize,'uint8');

    % Calculating the center of the canvas
    centerPoint = [canvasSize(1)/2, canvasSize(2)/2];
    xc = centerPoint(1);
    yc = centerPoint(2);

    % Set radius of the background circle as 40% of the canvas's width
    radius = ceil(0.4 * canvasSize(1));

    %dimensions of rectangle (NOTE: change later for user entry)
    width = 40;
    height = 150;

    %coordinates of upper left corner of rectangle
    xr = xc - (width/2);
    yr = yc - (height/2);

    % Draw the background circle on the canvas with a specified color
    IM = insertShape(mat,"filled-circle",[xc yc radius],Color='white');
    IM2 = insertShape(IM,"filled-rectangle",[xr yr width height],Color='white');
    IM3 = rgb2gray(IM2);
    imwrite(IM3, "phantom1.jpg");
    p1 = IM3;
end
```

```
function p2 = genPhantom2()
    canvasSize = [300, 300];
    mat = zeros(canvasSize,'uint8');

    % Calculating the center of the canvas
    centerPoint = [canvasSize(1)/2, canvasSize(2)/2];
    xc = centerPoint(1);
    yc = centerPoint(2);

    % Set radius of the background circle as 40% of the canvas's width
    radius = ceil(0.4 * canvasSize(1));

    % Draw the background circle on the canvas with a specified color
    IM = insertShape(mat,"filled-circle",[xc yc radius],Color='white');

    % x-coordinates and radii for foreground circle series
    innerCircleDistance = ceil(radius/5.4);

    c1_x = centerPoint(2)- floor(4.5 * innerCircleDistance);
    c1_r = ceil( min(canvasSize)/55);

    c2_x = centerPoint(2)- 3 * innerCircleDistance;
    c2_r = floor(1.52 * c1_r);

    c3_x = centerPoint(2)- 1 * innerCircleDistance;
    c3_r = floor(1.52 * c2_r);

    c4_x = centerPoint(2) + innerCircleDistance;
    c4_r = floor(1.52 * c3_r);

    c5_x = centerPoint(2) + floor(3.5 * innerCircleDistance);
    c5_r = floor(1.52 * c4_r);

    % Draw series of circles of increasing size on top of background circle
    IM2 = insertShape(IM,"filled-circle",[c1_x yc c1_r],Color='white');
    IM3 = insertShape(IM2,"filled-circle",[c2_x yc c2_r],Color='white');
    IM4 = insertShape(IM3,"filled-circle",[c3_x yc c3_r],Color='white');
    IM5 = insertShape(IM4,"filled-circle",[c4_x yc c4_r],Color='white');
    IM6 = insertShape(IM5,"filled-circle",[c5_x yc c5_r],Color='white');

    IM7 = rgb2gray(IM6);
    imwrite(IM7, 'phantom2.jpg');
    p2 = IM7;
end
```

```
% Cartesian sampling and image reconstruction
function p = cartesian(IM, nlines, ppline)
    numberOfLines = nlines;
    pointsPerLine = ppline;

    % Generate Cartesian grid
    [x, y] = meshgrid(-numberOfLines:pointsPerLine, -numberOfLines:pointsPerLine);

    % Calculate radial distance from the center
    radialDistance = sqrt(x.^2 + y.^2);

    % Creating a circular mask with radius 15
    circularMask = (radialDistance < 15);

    % Performing Fourier Transform
    fftResult = fftshift(fft2(circularMask));

    % Computing the log magnitude spectrum
    logMagnitudeSpectrum = log(1 + abs(fftResult));

    % Visualize Sampled K-Space for Radial
    %imshow(im2uint8(logMagnitudeSpectrum / max(logMagnitudeSpectrum(:))), [], 'parent', app.SampledKSpaceImage);

    numLines = numberOfLines;
    numPoints = pointsPerLine;
    % Defining ranges for cropping
    startX = int64(640 - (numLines * 2.5));
    startY = int64(640 - (numPoints * 2.5));
    if startX == 0
        startX = 1;
    end
    if startY == 0
        startY = 1;
    end
    endX = int64(640 + (numLines * 2.5));
    endY = int64(640 + (numPoints * 2.5));

    %disp(startX);
    %disp(startY);
    %disp(endX);
    %disp(endY);

    imData = IM;
    imData = im2gray(imData);

    % Fourier transform with zero padding
    fourierTransform = fft2(imData, 1280, 1280);
    shiftedKSpace = fftshift(fourierTransform);

    % Cropping the k-space
    croppedKSpace = shiftedKSpace(startX:endX, startY:endY);
```

```
    % Inverse Fourier transform
    reconstruction = ifft2(croppedKSpace);

    % Cropping the reconstruction to specified size
    croppedReconstruction = reconstruction(1:numLines, 1:numPoints);

    % Scaling and resizing the image
    absScaled = abs(croppedReconstruction);
    stretchedImage = imresize(absScaled, [256, 256]);

    % Generating Sampled K-Space image for Cartesian sampling
    targetSize = [numLines * 5, numPoints * 5];
    win1 = centerCropWindow2d(size(shiftedKSpace), targetSize);
    B1 = imcrop(abs(log(shiftedKSpace)), win1);
    B1 = imresize(B1, [256, 256]);
    %imshow(B1, [], 'parent', app.SampledKSpaceImage);
    idiff = stretchedImage;
    % Display the reconstructed image for Cartesian
    %imshow(idiff, [], 'parent', app.ReconstructedImage);
    imshow(idiff, []);
    p = idiff;
end

% Radial sampling and image reconstruction
function p = radial(IM, nlines, ppline)
    numberOfLines = nlines;
    pointsPerLine = ppline;

    % Generate Cartesian grid
    [x, y] = meshgrid(-numberOfLines:pointsPerLine, -numberOfLines:pointsPerLine);

    % Calculate radial distance from the center
    radialDistance = sqrt(x.^2 + y.^2);

    % Creating a circular mask with radius 15
    circularMask = (radialDistance < 15);

    % Performing Fourier Transform
    fftResult = fftshift(fft2(circularMask));

    % Computing the log magnitude spectrum
    logMagnitudeSpectrum = log(1 + abs(fftResult));

    % Visualize Sampled K-Space for Radial
    %imshow(im2uint8(logMagnitudeSpectrum / max(logMagnitudeSpectrum(:))), [], 'parent', app.SampledKSpaceImage);

    % Read the phantom image
    %phantomImage = imread('./phantom.png');
    phantomImage = IM;
```

```
if (numberOfLines >= 16) && (numberOfLines < 64)
    if(pointsPerLine >= 16) && (pointsPerLine < 64)
        theta1 = 0:10:170;
    else
        theta1 = 0:10:180;
    end
    % Perform Radon transform
    [R1, ~] = radon(phantomImage, theta1);
    % size(R1, 2); returns number of columns in matrix R1
    % size(R1, 1); returns number of rows in matrix R2

    % Interpolate to 512 angles
    P512 = phantomImage;
    [~, ~] = radon(P512, theta1);
    %size(R512, 1); returns the number of rows in matrix R512
    outputSize = max(size(phantomImage));
    dTheta1 = theta1(2) - theta1(1);
    I1 = iradon(R1, dTheta1, outputSize);

    % Display the reconstructed image for Radial sampling
    idiff = I1;
    %imshow(idiff, [], 'parent', app.ReconstructedImage);
    imshow(idiff, []);
  elseif (numberOfLines >= 64) && (numberOfLines <= 128)
      if (pointsPerLine >= 64) && (pointsPerLine <= 128)
      theta2 = 0:5:175;
      else
          theta2 = 0:4.9:180;
      end

      % Perform Radon transform
      [R2, ~] = radon(phantomImage, theta2);
      %size(R2, 2);  returns number of column in matrix R2
      % size(R2, 1); returns number of rows in matrix R2

      % Interpolate to 512 angles
      P512 = phantomImage;
      [~, ~] = radon(P512, theta2);
      % size(R512, 1); returns number of rows in matrix R512
      outputSize = max(size(phantomImage));
      dTheta2 = theta2(2) - theta2(1);
      I2 = iradon(R2, dTheta2, outputSize);
      idiff = I2;

      % Display the reconstructed image of radial sampling
      %imshow(idiff, [], 'parent', app.ReconstructedImage);
      imshow(idiff, []);
  else
      if(numberOfLines > 128) && (pointsPerLine > 128)
          theta3 = 0:2:178;
      else
          theta3 = 0:2.5:178;
      end
```

```
            % Perform Radon transform
            [R3, ~] = radon(phantomImage, theta3);
            %size(R3, 2); returns number of columns in matrix R3
            %size(R3, 1); returns number of rows in matrix R3

            % Interpolate to 512 angles
            P512 = phantomImage;
            [~, ~] = radon(P512, theta3);
            %size(R512, 1); returns number of rows in matrix R512
            outputSize = max(size(phantomImage));
            dTheta3 = theta3(2) - theta3(1);
            I3 = iradon(R3, dTheta3, outputSize);
            idiff = I3;

            % Display the reconstructed image of radial sampling
            %imshow(idiff, [], 'parent', app.ReconstructedImage);
            imshow(idiff, []);
        end
end

%{
function result = cartesian(IM, slines, spoints)
    size_img = length(IM); % size of original phantom
    % ratios of phantom dimensions to sampling grid dimensions
    k = [size_img/slines, size_img/spoints];
    % dimensions of new image based on ratios
    new_dim = floor(size_img*k);
    % new padded image
    IM2 = zeros(new_dim(1), new_dim(2), 'uint8');
    IM2(1:size_img, 1:size_img) = IM;
    % fourier transformed image
    FT = fftshift(fft2(IM2));
    F_mag = sqrt(real(FT).^2 + imag(FT).^2);
    F_log = log(1 + FT);
    %imshow(F_log, []);
    % sample k-space
    SampleIM = zeros(new_dim(1), new_dim(2), 'uint8');

    pointStep = ;
    lineStep = ;
    for i = 0:slines
        for j = 0:spoints

        end
    end
    result = IM2;
end
%}
```