Medical Imaging Project

Members of team:

Adriel Arias – 1621983 (soft number), Computer Science, Senior undergraduate

Jose Alvarez – 1587271 (soft number), Computer Science, Senior undergraduate

Introduction:

The purpose of our project is to be able simulate data acquisition. By analyzing this data and applying certain algorithms to them like Fourier transforms we should be able to more accurately depict the data in a manner that is user friendly.

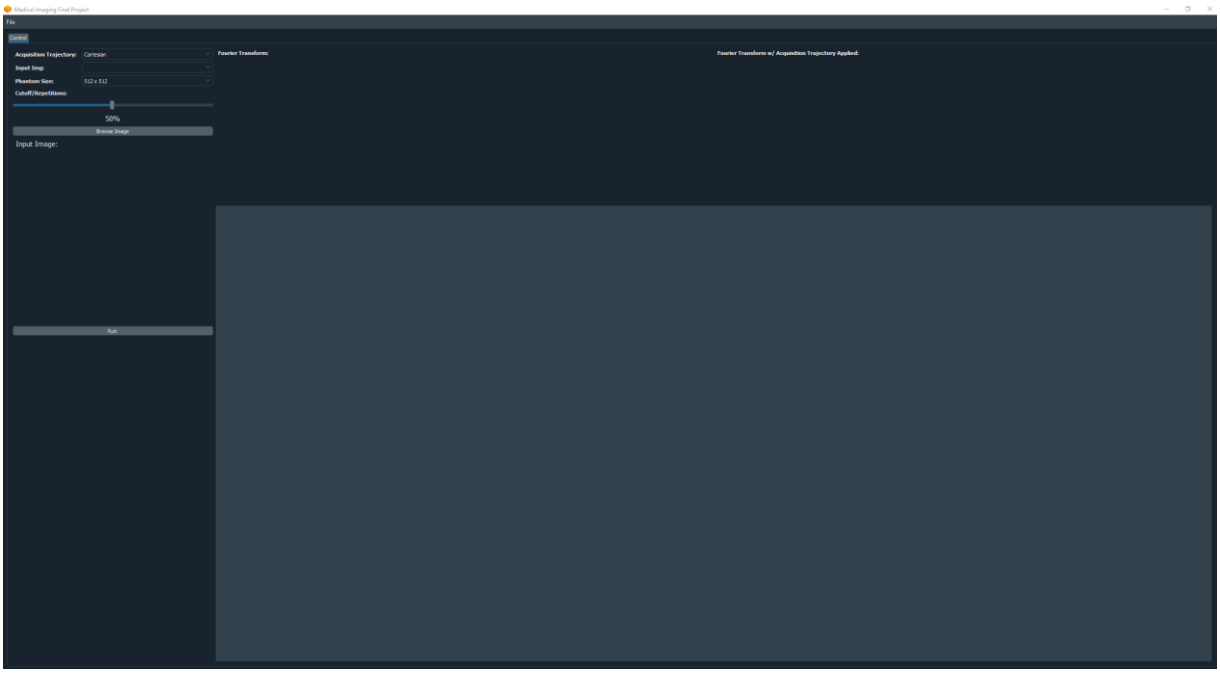
Aims:

1. To develop a user friendly, yet visually pleasing, GUI.
2. Generate Phantoms for testing and validation.
3. To apply Fourier transforms and evaluate what types of images are created from Cartesian and Radial trajectories.
4. Study the effects or the reconstructed images depending on the acquisition parameters.

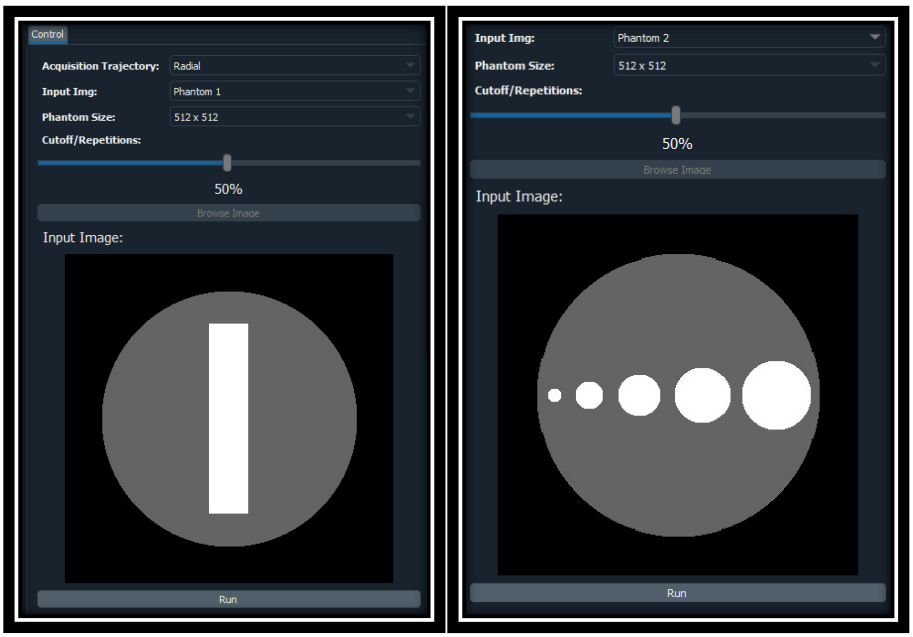
Methods

To develop our GUI (1) we used PyQT5. This allowed us to add the option for the user to select what acquisition trajectory they wanted to use. Through the GUI, the user can either choose one of our generated phantoms or input their own image with a “browse” button. Furthermore, the user can see the generated k-space before and after an acquisition trajectory was applied (3), as well as how changing their parameters, with a slider, affects the outcome (4). In order to generate the phantoms, we used Skimage and created the sizes of the objects relative to that of the image with the functions phantom1() and phantom2() (2). We then used the function shiftedDFT() which used numpy fft functions to calculate the Fourier Transform of the image and shift it before returning. From this we can output to the GUI how the k-space looks (3). Then, we generate a mask, depending on the user trajectory selection, to multiply with the k-space and output back to the GUI (3). From this we can now generate the original image after it has been modified with the revertDFT() function.

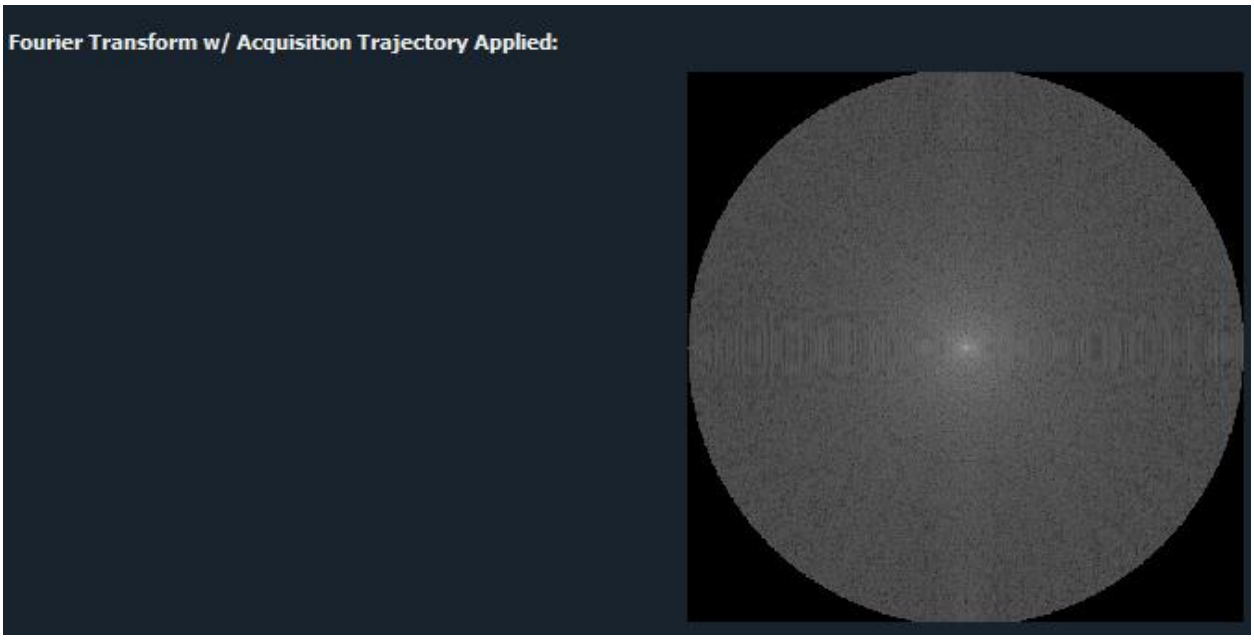
Results and Discussion:



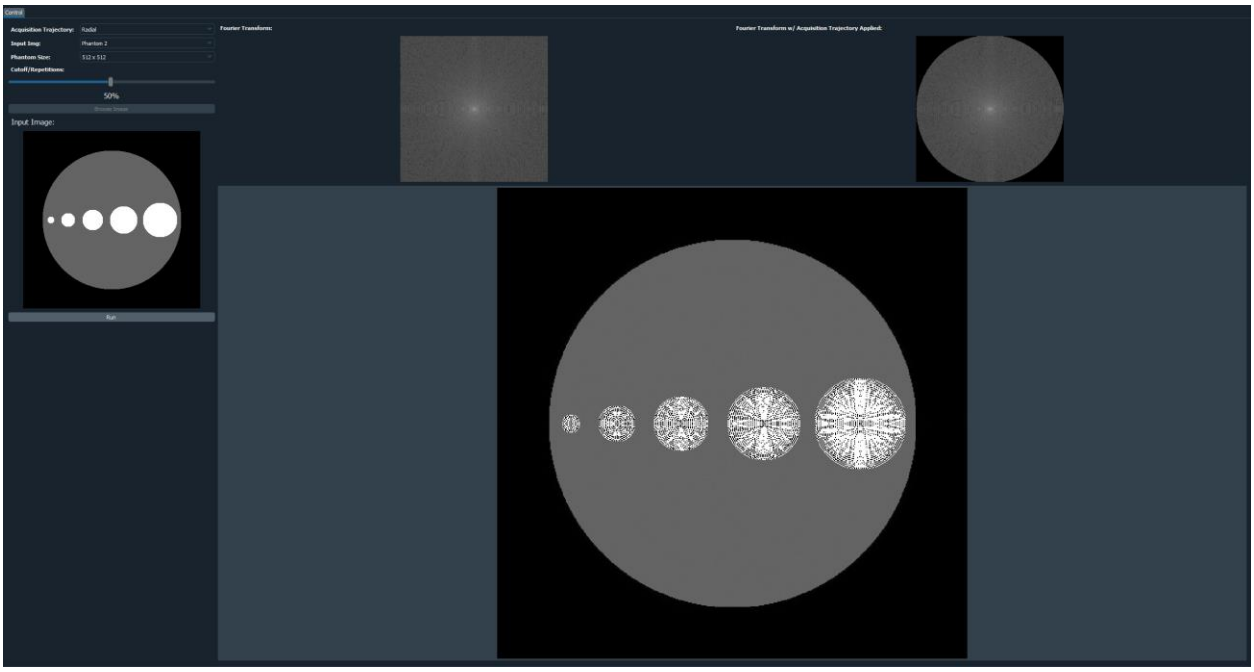
(Aim 1 and Aim 2) We developed an easy to use GUI that gives the user the option to input their custom image or generate a phantom1/phantom 2 image. The user will also be given the option to choose the phantom size (512x512/1024x1024/2048x2048/4096x4096) and the acquisition trajectory. This will then show the user the selected image as a small preview.



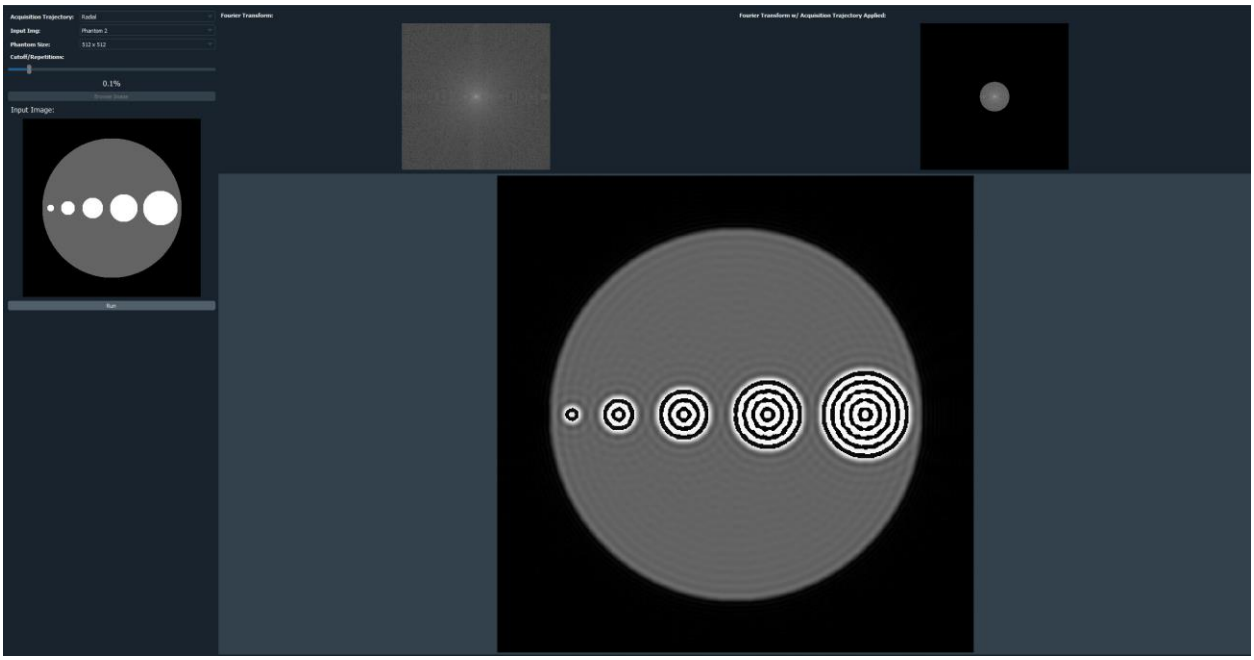
(Aim 3) In our GUI we can also display to the user the k-space generated from the input image and the collected k-space according to the acquisition trajectory they previously set. The image below shows the radial acquisition trajectory being applied with the radius being 50% of the image



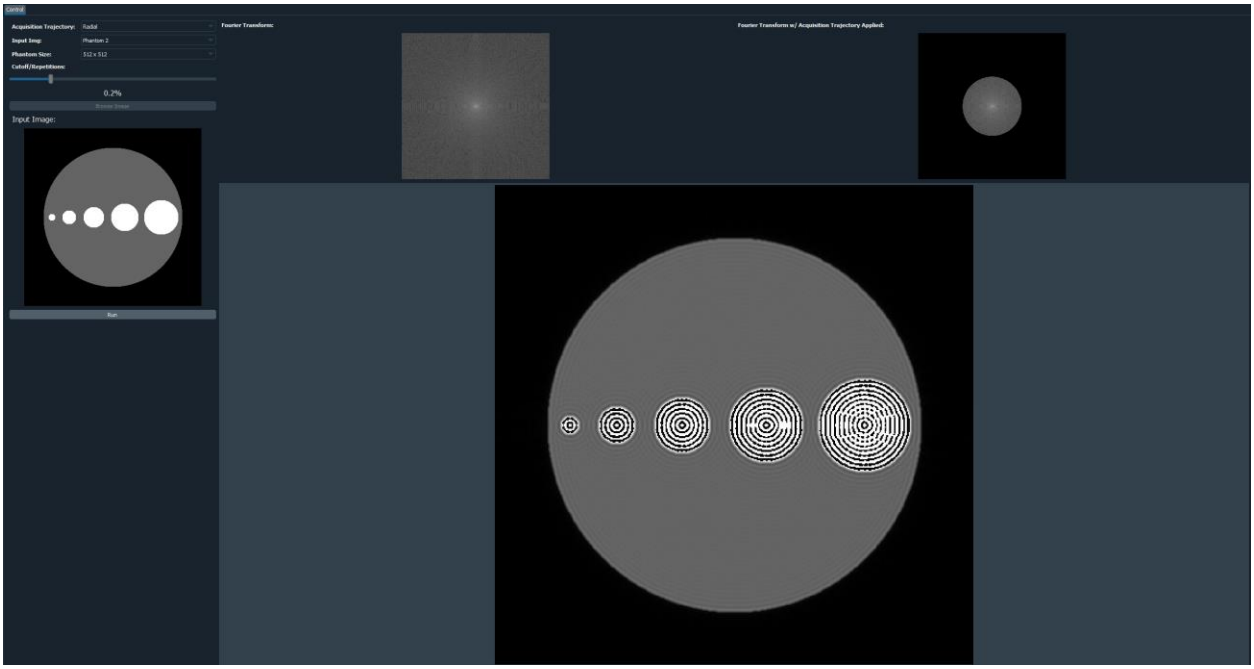
This is how the final GUI interface would look to the user after the final image is calculated from the acquisition trajectory:



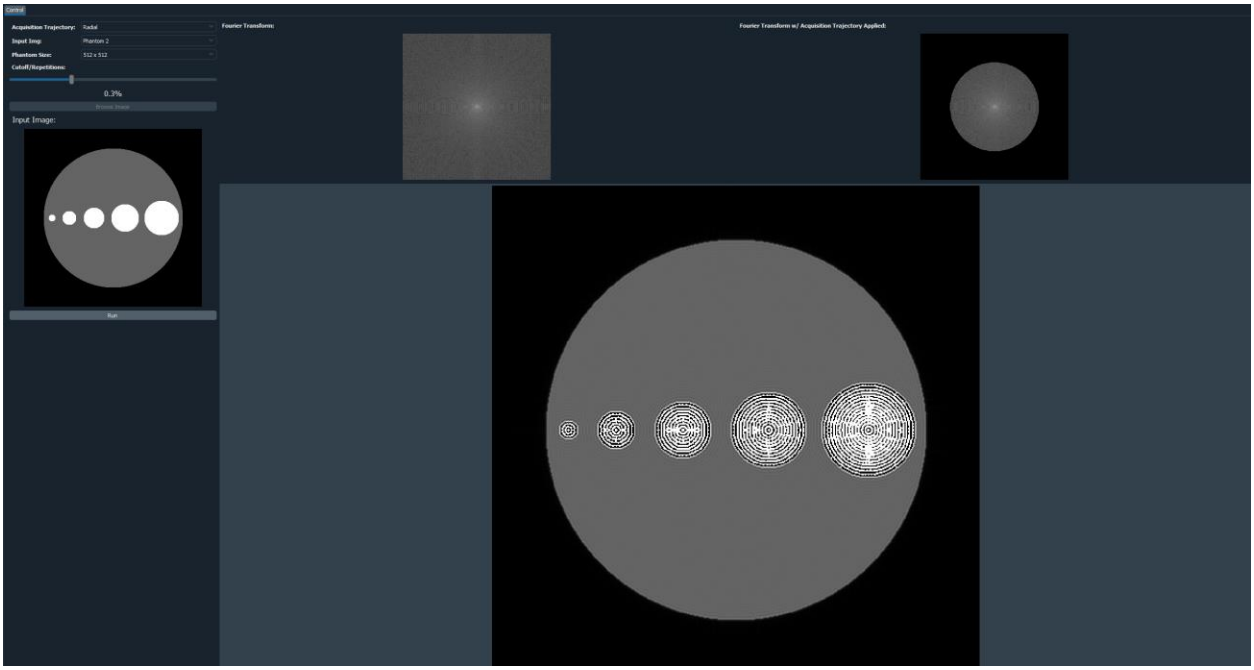
This allows the user to compare the final and original phantom. If one wants to see how changing the acquisition trajectory, one can simply change the slide under “cutoff/repetitions”. The images below demonstrate how changing this slider can impact the final result:



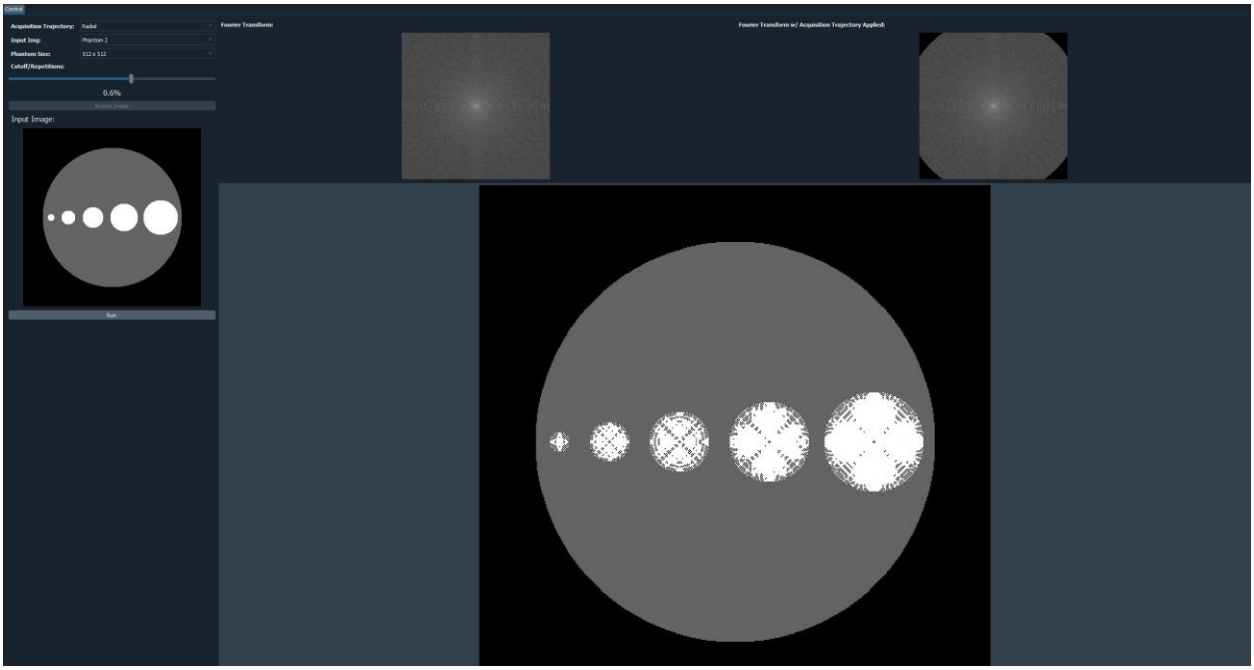
Radial Acquisition Trajectory at 10%



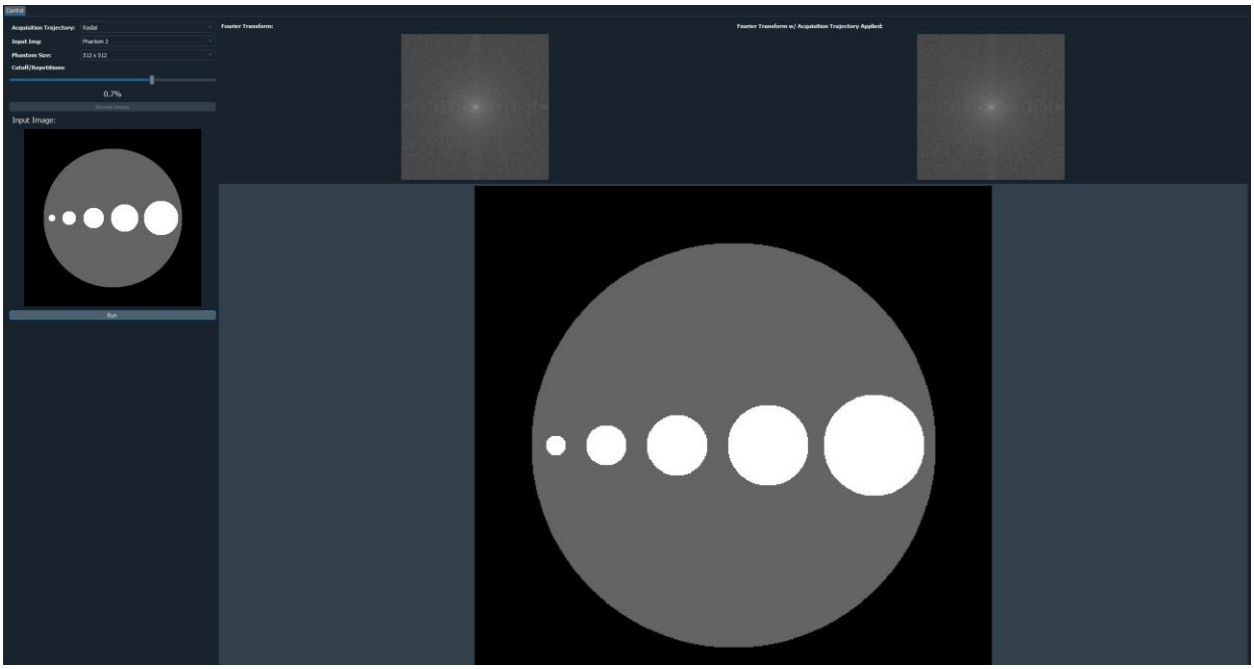
Radial Acquisition Trajectory at 20%



Radial Acquisition Trajectory at 30%

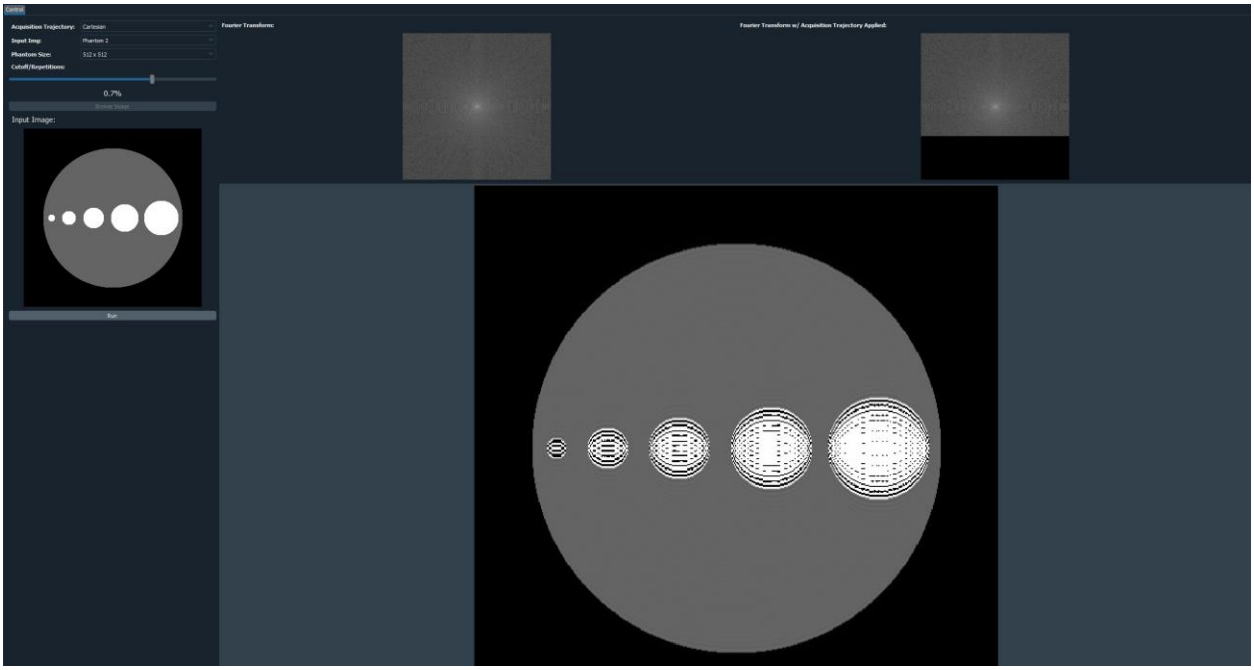


Radial Acquisition Trajectory at 60%



Radial Acquisition Trajectory at 70%

The same can be done for Cartesian!



Cartesian Acquisition Trajectory at 70%

Specifics:

8. Using the test/validation phantom investigate the effect of changing acquisition parameters. In all cases the "ground truth" is your original phantom! You will compare the images generated by your code relative to this phantom! Use the different tools in section 5 to investigate how your scanner works. Compare the image and the validation phantom for different acquisition parameters:
   - How the edges of the rectangular structure the first phantom are appear in the image? Any improvements?
   - In the phantom with the circular structures, what is the effect on them?
   - What is the overall image impression?
   - What parameters give you the best possible image?
   - Add a time parameter and (e.g. this will be the duration of data acquisition per horizontal line for the Cartesian acquisition, and per radial line for the radial acquisition) and investigate how "the longer the acquisition time the better the collected images"

Answers:

- There seems to be no improvement from collecting certain k-spaces.
- The phantom with the circular structures seems to improve the longer the acquisition time is
- The overall image impression is that the parameters depend a lot on the image quality
- The parameters that give the best result are the radial acquisitions with a longer time.
- It seems that the longer the acquisition the better because it allows more data in

Conclusions

We were able to successfully execute all our aims. The user is can choose what trajectory they want with the image of their choice to analyze the reconstructed images. To further analyze the reconstructed images, the user can also change the parameters and see how changing the trajectory affects the final result.

CODE:

```
import numpy as np
import cv2
from skimage.draw import circle
from skimage.draw import rectangle
from math import floor
from PIL import Image

class DFT:

    #generates the first phantom image
    def phantom1(self, x, y):
        #begins the first image with the size (x,y)
        img = np.zeros((y,x),dtype='uint8')

        rr, cc = circle(y/2,x/2,floor(x*.39))
        img[rr,cc] = 100

        recHeight = floor(y*.58)
        recWidth = floor(x*.12)

        originY = floor((y/2)-(recHeight/2))
        originX = floor((x/2)-(recWidth/2))
        #coordinate of the origin
        start = (originY, originX)
        extent = (floor(y*.58),floor(x*.12))
        rr, cc = rectangle(start, extent=extent,shape=img.shape)

        img[rr,cc] = 255

        # cv2.imwrite('phantom1.jpg',img)
        # cv2.imread('phantom1.jpg')
        return img

    #generates the second phantom image
    def phantom2(self, x, y):

        img = np.zeros((y,x),dtype='uint8')

        #main circle holding others
        rr, cc = circle(x*.5,y*.5,int(round(x*0.390625)))


        img[rr,cc] = 100
```

```
        #generates multiple circles at a ratio

        #vertical position, horizontal position, diameter
        rr, cc = circle(int(y*.5), int(round(x*0.15625)),
int(round(x*0.01953125)) )
        img[rr,cc] = 255

        rr, cc = circle(int(y*.5), int(round(x*0.25390625)),
int(round(x*0.0390625)))
        img[rr,cc] = 255

        rr, cc = circle(int(y*.5), int(round(x*0.390625)),
int(round(x*0.05859375)))
        img[rr,cc] = 255

        rr, cc = circle(int(y*.5), int(round(x*0.56640625)),
int(round(x*0.078125)))
        img[rr,cc] = 255

        rr, cc = circle(int(y*.5), int(round(x*0.771484375)),
int(round(x*0.09765625)))
        img[rr,cc] = 255

        return img


    def load_display(self, image):

        #Create a output window
        cv2.namedWindow("Image", cv2.WINDOW_AUTOSIZE)

        #shows image
        cv2.imshow("Image", image)

        cv2.waitKey(0)

        #destroy windows
        cv2.destroyWindow("Image")

    def magnitude(self, image):
        #gets the width and height of the input image
        height, width = image.shape
        #iterates and ensures values aren't negative
        for h in range(height):
            for w in range(width):
                # if negative convert to positive
                if image[h][w] < 0:
                    image[h][w] *= -1
        return image

    def radial(self, image, cutoff):

        """Computes a Ideal low pass mask
        takes as input:
        shape: the shape of the mask to be generated
        cutoff: the cutoff frequency of the ideal filter
        returns a ideal low pass mask"""
```

```
        height = image.shape[0] # P
        width = image.shape[1] # Q


        mask = np.copy(image)

        for u in range(height):
            for v in range(width):
                #used the pythagorean theorem to check if the coordinate
position is within range
                duv = ((u - height/2)**2 + (v - width/2)**2)**(1/2)
                if duv <= cutoff:
                    mask[u, v] = 1 #If in range leave at 1
                elif duv > cutoff:
                    mask[u, v] = 0 #If out of range change to black

        maskedimg = np.multiply(image, mask)

        return maskedimg

    def cartesianMask (self, image, repetitions):

        height,width = image.shape
        mask = image.copy()
        #0 is black
        #iterates through image depending on the user input
        for h in range (height):
            for w in range(width):
                if h < repetitions:
                    mask[h,w] = 1
                elif h >= repetitions:
                    mask[h,w] = 0

        maskedimg = np.multiply(image, mask)

        return maskedimg
    def shiftedDFT(self, img):
        # run dft on image
        img = np.fft.fft2(img)
        img = np.fft.fftshift(img)
        return img

    def revertDFT(self, img):
        #revert DFT to get back calculated image
        img = np.fft.ifftshift(img)
        img = np.fft.ifft2(img)
        return img


    def prepareFinalOutput(self, img):
        # Steps before outputting!----------------------------------------

        # this image will be returned
        magDFT = img.copy()
        magDFT = self.magnitude(magDFT)
        # magDFT = 10 * np.log(magDFT)
        magDFT = magDFT.astype("uint8") #Makes it a type that can output
```

```
    # --------------------------------
    return magDFT

def prepareOutput(self, img):
    #Output for DFT k-spaces
    # this image will be returned
    magDFT = img.copy()
    magDFT = self.magnitude(magDFT)
    magDFT = 10 * np.log(magDFT)
    magDFT = magDFT.astype("uint8") #Makes it a type that can output

    # --------------------------------
    return magDFT
```