Faculty of Engineering
Cairo University

MRI Task # 3 : Phantom for K-Space

Supervised by: Dr. Inas A. Yassine

Team members:

- Alaa Gamal
- Sherif Esam
- Marwa Adel
- Muhammed Ehab

Abstract:

Magnetic resonance imaging (**MRI**) is a medical imaging technique used in radiology to form pictures of the anatomy and the physiological processes of the body. MRI scanners use strong magnetic fields, magnetic field gradients, and radio waves to generate images of the organs in the body. In most medical applications, hydrogen nuclei, which consist solely of a proton, that are in tissues create a signal that is processed to form an image of the body in terms of the density of those nuclei in a specific region. Given that the protons are affected by fields from other atoms to which they are bonded, it is possible to separate responses from hydrogen in specific compounds. To perform a study, the person is positioned within anMRI scanner that forms a strong magnetic field around the area to be imaged. First, energy from an oscillating magnetic field is temporarily applied to the patient at the appropriate resonance frequency. Scanning with X and Y gradient coils causes a selected region of the patient to experience the exact magnetic field required for the energy to be absorbed. The excited atoms emit a radio frequency (RF) signal, which is measured by a receiving coil. The RF signal may be processed to deduce position information by looking at the changes in RF level and phase caused by varying the local magnetic field using gradient coils. As these coils are rapidly switched during the excitation and response to perform a moving line scan, they create the characteristic repetitive noise of an MRI scan as the windings move slightly due to magnetostriction. The contrast between different tissues is determined by the rate at which excited atoms return to the equilibrium state.

Each tissue returns to its equilibrium state after excitation by the independent relaxation processes of T1 (spin-lattice; that is, magnetization in the same direction as the static magnetic field) and T2 (spin-spin; transverse to the static magnetic field). To create a T1-weighted image, magnetization is allowed to recover before measuring the MR signal by changing the repetition time (TR). This image weighting is useful for assessing the cerebral cortex, identifying fatty tissue, characterizing focal liver lesions, and in general obtaining morphological information, as well as for post-contrast imaging. To create a T2-weighted image, magnetization is allowed to decay before measuring the MR signal by changing the echo time (TE). This image weighting is useful for detecting edema and inflammation, revealing white matter lesions, and assessing zonal anatomy in the prostate and uterus.

Task 1 Recap

In task 1, we've submitted an algorithm that implements all of:

- A function that rotates the **bulk magnetization vector**.
- The **bulk magnetization's trajectory** by generating a figure representing the results.
- A function that can load images, show the original figure and calculate its **FFT.**
- A function that simulates the nonuniformity effect.

And in this report we will complete our implementation in order to get more closer and be more familiar with all the **MRI** functionalities.

Task 2 Recap

In task 2, we've submitted an algorithm that implements all of:

- A function that can load images, show the original figure and plot it in the **K-Space** domain.
- A function that depends on the nonuniformity effect and implements all of:
    - Angular frequency difference between the molecules of water.
    - The **bulk magnetization's trajectory** by generating a figure representing the results.

Task 3 Requirements

In task 3, we're required to submit an algorithm that implements all of:

- A function that can load an image, and create a phantom that is made of oil and water for this image.
- A function that generates the K-space for this image.
- A function that can add the non-uniformity and regenerate the K-space.
- A function that represents the trajectory at any point in the space.

Code Explanation

T1 & T2 Assignment

```
image = cv2.imread('test.jpg', 0)
size_image=(len(image))
row=size_image
col=size_image
for i in range(row):
   for j in range(col):

       if (image[i,j] >= 0 and image[i,j] <20):
           T1[i,j]=500
           T2[i,j]=100
       elif ( image[i,j]>=20 and image[i,j]<180):
           T1[i,j]=1000
           T2[i,j]=120
       elif ( image[i,j]>=180 and image[i,j]<255):
           T1[i,j]=1500
           T2[i,j]=150
```

- After reading the image in **Gray_Scale** and calculating its size, we are rotating all over the length, considering we've a range in color vision from **[0:255]** graded from the black **[0:20]** to the white **[180:255]** passing through all the gray shades.

- For every range of these ranges, we are assigning for **T1** and **T2** specific values to be distinguished for each range. Considering that **T1 > T2.**

Rotate & Decay Functions

- This function takes theta and phantom and calculates the RF pulses, then applies these pulses to the phantom to start to decay with the given theta.
- We specified theta = 90 [In radians] in the code.

```
def rotate(theta,phantom):
RF=([[np.cos(theta),0,np.sin(theta)],[0,1,0],[-np.sin(theta),0,np.cos(theta
)]])
    phantom=np.dot(RF,phantom)
    return phantom
```

- This function takes phantom, TE and T2 as parameters to
  - Calculate the decay.
  - Apply the decay equation to the given phantom.

```
def decay(phantom,TE,T2):
   dec=np.exp(-TE/T2)
   phantom=np.dot(dec,phantom)
   return phantom
```

- This function calls the previous two functions in a double for loop that iterates on the phantom and apply the rotate and decay on phantom's vectors one-by-one.

```
def rotate_decay(phantom,theta,TE,T2,row,col):
   for i in range(row):
       for j in range(col):
           phantom[i,j,:]=rotate(theta,phantom[i,j,:])
           phantom[i,j,:]=decay(phantom[i,j,:],TE,T2[i,j])
   return phantom
```

Creating Phantom

```
def createPhantom(row,col):
   phantom=np.zeros((row,col,3))
   for i in range(row):
       for j in range(col):
           phantom[i,j,2]=1

   return phantom
```

- This function creates the phantom for that's used all over the algorithm.
- We create a phantom that contains 3D vectors in the same width and length of the image array.
- And all these vectors have a value in the **Z-Axis** only =1 **[Uniform]**

Rotate & Recovery functions

```
def recovery(phantom,t,T1,T2,larmarFreq):
   E1 = math.exp(-t/T1)
   E2 = math.exp(-t/T2)
   A= np.array([[E2,0,0],[0,E2,0],[0,0,E1]])
   B= np.array([0, 0, 1-E1])
   phi =2*math.pi*larmarFreq*t/1000

Rz=np.array([[math.cos(phi),-math.sin(phi),0],[math.sin(phi),math.cos(phi),0],[0,
0, 1]])
   Afb = np.matmul(A,Rz)
   phantom =np.matmul(Afb,phantom)+B
   return phantom
```

- After decaying, by applying this function, the recovery process will take place.
- It takes phantom, t, T1, T2, larmerfreqency as parameters.
- This function calculates the precession of the trajectory.
- The recovery function returns the phantom after applying an Equation on the form [AX+B] where X is the precession, A & B are vectors calculated from T1 and T2.

```
mag_field = np.random.uniform(low= 0 , high= 5, size = 1000)

def rotate_recovery(phantom,theta,t,T1,T2,row,col):
   larmarFreqOil= mag_field*42.6
   larmarFreqWater= mag_field*5.8
   k=0
   for i in range(row):
       for j in range(col):
           if(i<2 and j<2):
phantom[i,j,:]=recovery(phantom[i,j,:],t,T1[i,j],T2[i,j],larmarFreqOil[k])
           else:
phantom[i,j,:]=recovery(phantom[i,j,:],t,T1[i,j],T2[i,j],larmarFreqWater[k]
)
           if k>=999:
               k = 0
           else:
               k=k+1
   return phantom
```

- This function takes some parameters to apply it on the recovery trajectory.
- In this function we set the values of the larmel frequency of water and oil and pass them to the recovery function.
- Setting the larmel frequency needs to apply a different larmel frequency for each vector to apply the non uniformity in the trajectory.
- So, the variable mag_field is some random values applied to get the different larmel frequency.

Spin Echo Function

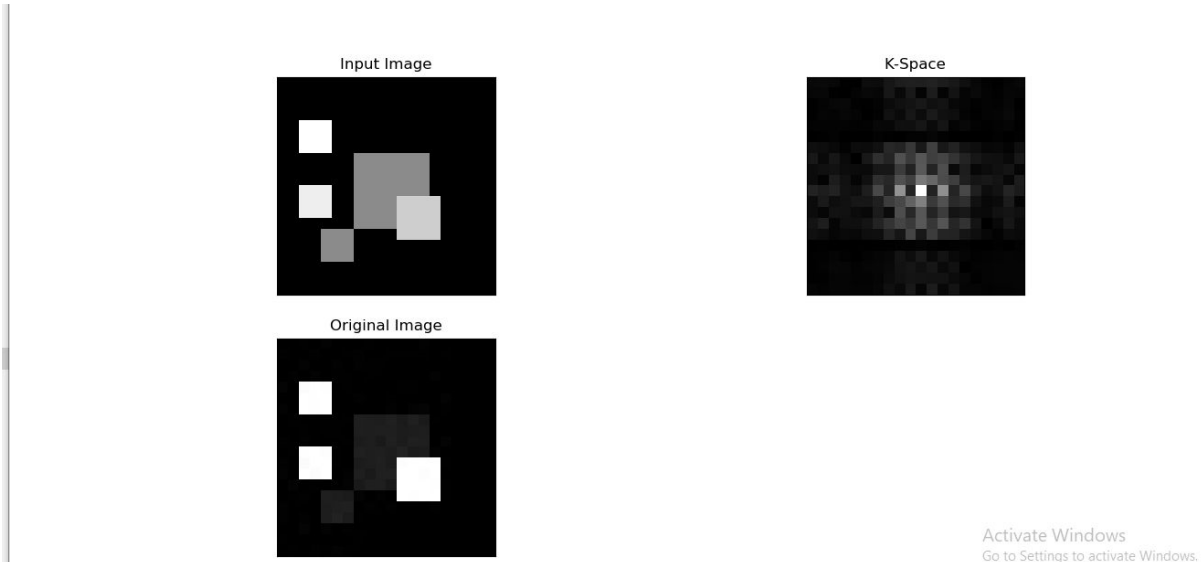
```
def spin_Echo():
    flip_angle=90
    theta=np.radians(flip_angle)
    Gy=2*math.pi
    phantom = createPhantom(row,col)
    Kspace=np.zeros((image.shape[0],image.shape[1]),dtype=np.complex_)
    t=0
    Check=True
    doubleCheck=False
    phantom=rotate_decay(phantom,np.radians(90),TE/2,T2,row,col)
    while (Check):
        phantom=rotate_recovery(phantom,theta,t,T1,T2,row,col)
        print(phantom)
        t=t+1
        for i in range(row):
            for j in range(col):
                if(phantom[i,j,2]<0.9):
                    Check=True
                    doubleCheck=True
                    break
                else:
                    Check=False
            if (doubleCheck==True):
                break

    phantom=rotate_decay(phantom,np.radians(180),TE/2,T2,row,col)

    for r in range(Kspace.shape[0]):  #rows
        for c in range(Kspace.shape[1]): #columns
            Gx_step=((2*math.pi)/row)*r
            Gy_step=((Gy)/col)*c
            for ph_row in range(row):
                for ph_col in range(col):
                    Toltal_theta=(Gx_step*ph_row)+(Gy_step*ph_col)

Mag=math.sqrt(((phantom[ph_row,ph_col,0])*(phantom[ph_row,ph_col,0]))+((phantom[ph_
row,ph_col,1])*(phantom[ph_row,ph_col,1])))
                    Kspace[r,c]=Kspace[r,c]+(Mag*np.exp(-1j*Toltal_theta))
```

- To generate the K-Space, we follow the spin_Echo sequence.
- We generate an empty array of zeros for the K-Space with the same shape of the original image.

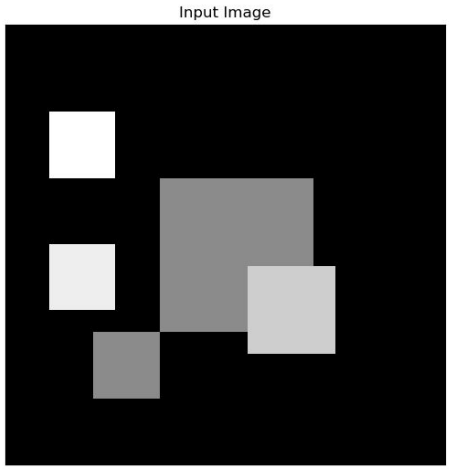
- The first two for loops are to append values to K-space and the other two for loops are to iterate all over the rows and columns of the image.
- The 4 for loops are taking gradients in X and Y directions.
- The first while loop is to check on the returned value for the phantom from the recovery function if it's more than 0 and less than 1.
- After finishing all the loops, we are calculating the summation of the K-Space.

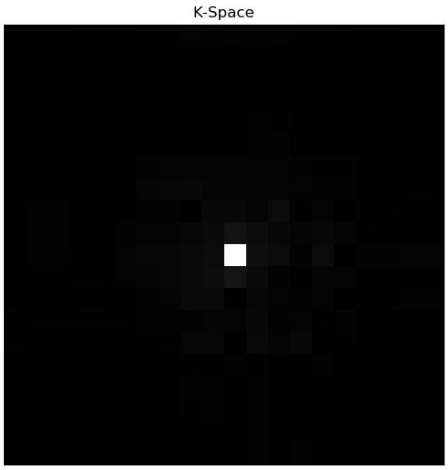
The Results:

- We can plot the K-Space of the **Gray Scale** image and plot its inverse After Adding The nonuniformity effect.

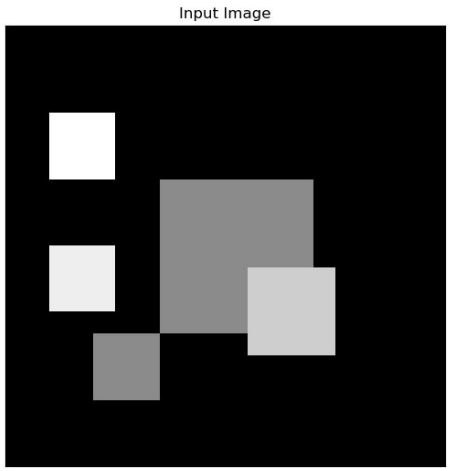


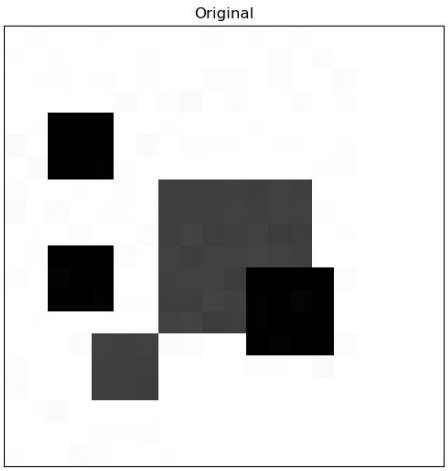
- Without the nonuniformity effect we got this result in the K-Space:





- And this result after getting the inverse:





- We can represent the trajectory at any point of the space by printing the phantom, but we couldn't plot it. We faced some problems because the previous two tasks we created a function that plot only one vector. But now we are required to plot the phantom, that's created of hundreds of vectors.