Bishop's university(Medical Imaging)
(Dr Russell Butler – Summer 2022)
Members:

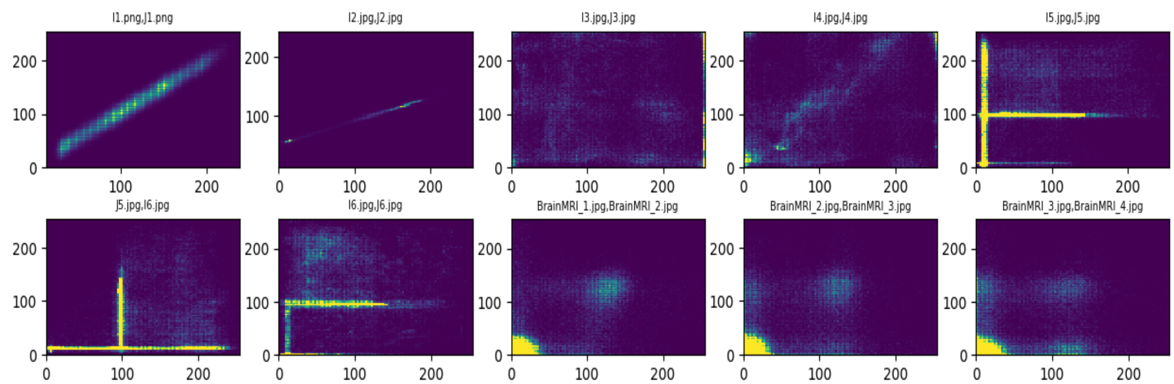
Razieh Shahsavar (002341606)   Maryam Bayatzadeh (002338161)

Assigment2

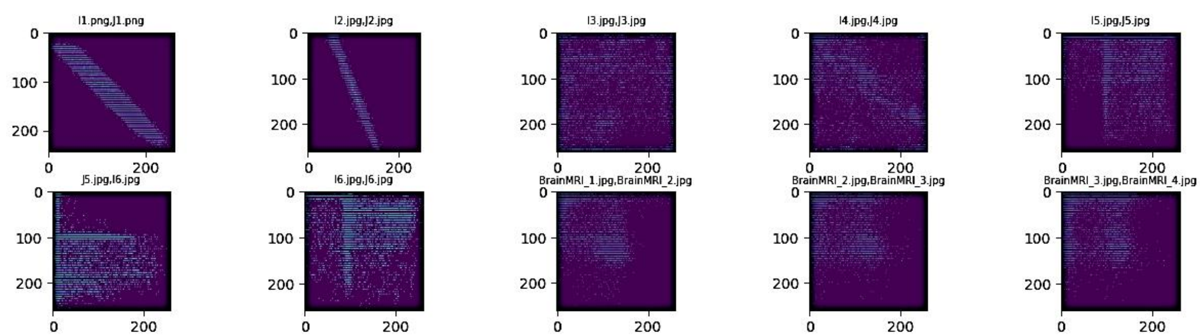
Part1)

Part1-a)

Output of joint histogram(by hist2d function(existing function)):



Output of joint histogram(by manual code):



Code:

```
#part1---------------------------------
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os

# paths of all images
paths = ['D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\I1.png',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\J1.png',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\I2.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
```

```
russell\\assigment2\Assignment_2_data\Data\J2.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\I3.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\J3.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\I4.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\J4.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\I5.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\J5.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\I6.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\J6.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_1.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_2.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_3.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_4.jpg']

# Read Data of all images
pic = []
for i in range(len(paths)):
    pic.append(cv2.imread(paths[i]))
#----------------manual function for show the joint histogram and calculate
prepare bins for each pair of images
def joinhist(p1, p2):
 p1 = np.array(p1)
 p2 = np.array(p2)

 x_min = np.min(p1)
 x_max = np.max(p1)

 y_min = np.min(p2)
 y_max = np.max(p2)

 # calculate bin for each image
 listI = np.linspace(x_min, x_max,50)
 listJ = np.linspace(y_min, y_max,50)

 h=np.zeros((int(x_max+5),int(y_max+5)))

 for n in range(len(listI)):
  I=np.round(listI[n])
  for x in range(p1.shape[0]):
    for y in range(p1.shape[1]):
     if np.round(p1[x,y,1])==I:
      J1=np.round(p2[x,y,1])
      I1=int(I)
```

```
      # print(I1,J1)
      count=0
      for x1 in (x,p1.shape[0]-1):
       for y1 in range(y,p1.shape[1]):
         if np.round(p1[x1,y1,1])==I1 and np.round(p2[x1,y1,1])==J1:
           count+=1
           print(f'i1={I1},j1={J1} , count={count}')
      h[int(I1),int(J1)]=count
      #return h
 plt.imshow(h)
#-----------------------------------manual function
#-----------------------------------Ready  function
# def joinhist(p1, p2, n):
#     p1 = np.array(p1)
#     p2 = np.array(p2)
#
#     x_min = np.min(p1)
#     x_max = np.max(p1)
#
#     y_min = np.min(p2)
#     y_max = np.max(p2)
#
#     x_bins = np.linspace(x_min, x_max, 100)
#     y_bins = np.linspace(y_min, y_max, 100)
#     #------1
#     if (n > 3):
#         plt.hist2d(p1.ravel(), p2.ravel(), bins=[x_bins, y_bins], vmax=200)
#     else:
#         plt.hist2d(p1.ravel(), p2.ravel(), bins=[x_bins, y_bins])
#----------------------------------------------Ready function
# call joint histogram  for each pair images and show all of them
plotn = 1
for i in range(len(pic)):
    try:
        x, y, z = pic[i - 1].shape
        x1, y1, z1 = pic[i].shape
        if (x == x1 and y == y1 and z == z1):
            plt.subplot(4, 5, plotn)
            plotn += 1
            plt.title(f'{os.path.basename(paths[i -
1])},{os.path.basename(paths[i])}', fontsize=7)
            joinhist(pic[i - 1], pic[i])
        else:
            print("two images are not same")
    except:
        print("there is no image")
plt.show()
```

Part1-b) verify by np.sum(histogram)==np.size(Image)

```
"D:\laptop\Bishops\bishops lectures win 2022\term2\MI Dr russell\assigment2\ass
size of histogeram and image are equal for:I1.png,J1.png
size of histogeram and image are equal for:I2.jpg,J2.jpg
size of histogeram and image are equal for:I3.jpg,J3.jpg
size of histogeram and image are equal for:I4.jpg,J4.jpg
size of histogeram and image are equal for:I5.jpg,J5.jpg
size of histogeram and image are equal for:J5.jpg,I6.jpg
size of histogeram and image are equal for:I6.jpg,J6.jpg
size of histogeram and image are equal for:BrainMRI_1.jpg,BrainMRI_2.jpg
size of histogeram and image are equal for:BrainMRI_2.jpg,BrainMRI_3.jpg
size of histogeram and image are equal for:BrainMRI_3.jpg,BrainMRI_4.jpg
```

Code:

```
#verify size of histogram is equal to size of image
for i in range(len(pic)):
 try:
  x, y, z = pic[i].shape
  x1, y1, z1 = pic[i+1].shape
  if (x == x1 and y == y1 and z == z1):
   hist_2_image=joinhist(pic[i],pic[i+1])
   if np.sum(hist_2_image)==np.size(pic[i]):
        print(f'size of histogeram and image are equal 
for:{os.path.basename(paths[i])},{os.path.basename(paths[i+1])}')
   else:
        print("two images are not same")
 except:
   print("there is no image")
```

Part1-c)Observe:

I1,j1: It seems 2 images are similar

I2,J2: These two images are approximately similar

I3,J3: These two images are not the same , only in some points

I4,J4: These two images are a little bit the same , in diagonal

I5,J5: There is a mask for other image with misalignment

J5,I6: There is a mask for other image with misalignment

I6,J6: There is a exactly mask for another image

BrainMri1, BrainMri2: These two images are nearly the same with a little bit misalignment

BrainMri2, BrainMri3: These two images are a approximately the same with misalignment

BrainMri3, BrainMri4: These two images are a little bit the same with misalignment

Part2)

Code for call part a,b,c and show ssd ,p_corr,MI:

```
# part2-a,b,c)

# call images for apply ssd,correlation coefficient,Mutual information to them
for i in range(len(pic)):
    try:
      x, y, z = pic[i].shape
      x1, y1, z1 = pic[i+1].shape
      if (x == x1 and y == y1 and z == z1):
          print(f'for( {os.path.basename (paths[i])} ,{os.path.basename (paths
S[i+1])})----->')
          print(f'SSD={calculate_ssd(pic[i], pic[i+1])}')
          print(f'P_corr={np.round(calculate_p_corr(pic[i], pic[i + 1]),4)}')
          print(f'MI={np.round(mutual_information(pic[i], pic[i + 1]),4)}')

      else:
          print("two images have not the same shape")
    except:
        print("There is no more image")
```

Part2-a)

```
Run:    main
"D:\laptop\Bishops\bishops lectures win 2022\term2\MI Dr russell\assigment1\pythonProject\venv\Scr
SSD(I1.png,J1.png)=79722472.0
Images dont have the same shape
SSD(I2.jpg,J2.jpg)=1420137600.0
Images dont have the same shape
SSD(I3.jpg,J3.jpg)=2612255488.0
Images dont have the same shape
SSD(I4.jpg,J4.jpg)=1086784256.0
Images dont have the same shape
SSD(I5.jpg,J5.jpg)=2099666432.0
SSD(J5.jpg,I6.jpg)=2098221312.0
SSD(I6.jpg,J6.jpg)=1276608384.0
Images dont have the same shape
SSD(BrainMRI_1.jpg,BrainMRI_2.jpg)=261301072.0
SSD(BrainMRI_2.jpg,BrainMRI_3.jpg)=405957792.0
SSD(BrainMRI_3.jpg,BrainMRI_4.jpg)=577233536.0
Images dont have the same shape

Process finished with exit code 0
```

Code:

```
#part2------------------------
#part2-a)
# Computing the sum of squared differences (SSD) between two images.
def calculate_ssd(img1, img2):
  return np.sum((np.array(img1, dtype=np.float32) - np.array(img2,
dtype=np.float32))**2)
```

Part2-b)

```
"D:\laptop\Bishops\bishops lectures win 2022\term2\MI Dr russell\assigment2\a
P_corr(I1.png,J1.png)=0.9782
two images have not the same shape
P_corr(I2.jpg,J2.jpg)=0.9962
two images have not the same shape
P_corr(I3.jpg,J3.jpg)=0.1434
two images have not the same shape
P_corr(I4.jpg,J4.jpg)=0.564
two images have not the same shape
P_corr(I5.jpg,J5.jpg)=0.6564
P_corr(J5.jpg,I6.jpg)=0.6523
P_corr(I6.jpg,J6.jpg)=0.7802
two images have not the same shape
P_corr(BrainMRI_1.jpg,BrainMRI_2.jpg)=0.6998
P_corr(BrainMRI_2.jpg,BrainMRI_3.jpg)=0.5373
P_corr(BrainMRI_3.jpg,BrainMRI_4.jpg)=0.3387
There is no more image
```

Code:

```
# part2-b)
# Computing the pearson correlation coeficient between two images.
def calculate_p_corr(img1, img2):
    if img1.shape != img2.shape:
        print("The images are not the same size")
        return
    numerator = np.mean(np.multiply((img1 - np.mean(img1)), (img2 -
np.mean(img2))))
    denominator = (np.std(img1) * np.std(img2))
    return numerator / denominator
```

Part2-c)

```
main
"D:\laptop\Bishops\bishops lectures win 2022\term2\MI Dr russell\assigment1\p
MI(I1.png,J1.png)=1.5285334270679176
two images have not the same shape
MI(I2.jpg,J2.jpg)=2.044563004694763
two images have not the same shape
MI(I3.jpg,J3.jpg)=0.2944248207518547
two images have not the same shape
MI(I4.jpg,J4.jpg)=0.6006517475787867
two images have not the same shape
MI(I5.jpg,J5.jpg)=0.5005305456754068
MI(J5.jpg,I6.jpg)=0.5332754665098568
MI(I6.jpg,J6.jpg)=0.6153862064687785
two images have not the same shape
MI(BrainMRI_1.jpg,BrainMRI_2.jpg)=0.3449403924021491
MI(BrainMRI_2.jpg,BrainMRI_3.jpg)=0.20910363331064513
MI(BrainMRI_3.jpg,BrainMRI_4.jpg)=0.11883753000282023
There is no more image
```

Code:

```
#Calculate Mutual information for joint histogram
def mutual_information(img1,img2):
# Convert bins counts to probability values
 hist=hist2d(img1,img2)
 pxy = hist / float(np.sum(hist))
 px = np.sum(pxy, axis=1) # marginal for x over y
 py = np.sum(pxy, axis=0) # marginal for y over x
 px_py = px[:, None] * py[None, :] # Broadcast to multiply marginals
 # Now we can do the calculation using the pxy, px_py 2D arrays
 nzs = pxy > 0 # Only non-zero pxy values contribute to the sum
 return np.sum(pxy[nzs] * np.log(pxy[nzs] / px_py[nzs]))
#get histogram to calculate Mutual Information
def hist2d(img1,img2):
 p1 = np.array(img1)
 p2 = np.array(img2)

 x_min = np.min(p1)
 x_max = np.max(p1)

 y_min = np.min(p2)
 y_max = np.max(p2)

 x_bins = np.linspace(x_min, x_max, 100)
 y_bins = np.linspace(y_min, y_max, 100)

 hist, x, y = np.histogram2d(p1.ravel(), p2.ravel(), bins=[x_bins, y_bins])
 return hist
```

Part2-d)

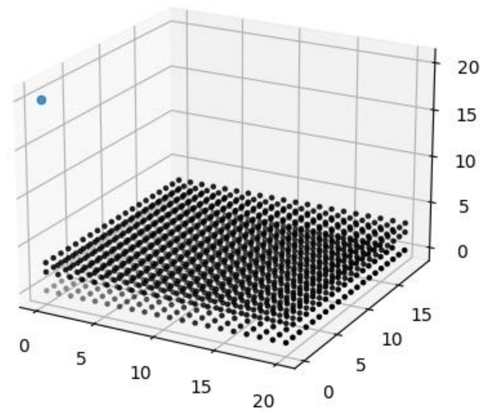
Image's name	SSD	P_correlation coeficie	Mutual information
(I1.png,J1.png)	79722472.0	0.9782	1.5285
(I2.jpg,J2.jpg)	1420137600.0	0.9962	2.0446
(I3.jpg,J3.jpg)	2612255488.0	0.1434	0.2944
(I4.jpg,J4.jpg)	1086784256.0	0.564	0.6007
(I5.jpg,J5.jpg)	2099666432.0	0.6564	0.5005
(J5.jpg,I6.jpg)	2098221312.0	0.6523	0.5333
(I6.jpg,J6.jpg)	1276608384.0	0.7802	0.6154
(BrainMRI_1.jpg,BrainMRI_2.jpg)	261301072.0	0.6998	0.3449
(BrainMRI_2.jpg,BrainMRI_3.jpg)	405957792.0	0.5373	0.2091
(BrainMRI_3.jpg,BrainMRI_4.jpg)	577233536.0	0.3387	0.1188

SSd: only considers the distance between respective pixels; the value of ssd is big even if two images have similar geometry but different colours.

P_corr: pictures with near geometry can be detected However, it is unable to detect noise.

MI: It appears to perform the best among these functions, as it can recognise images with near geometry and is noise sensitive.

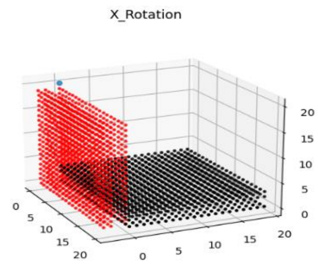
Part3) Part 3-a)

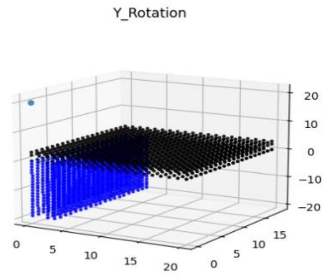


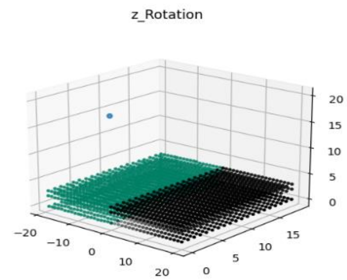
Code:

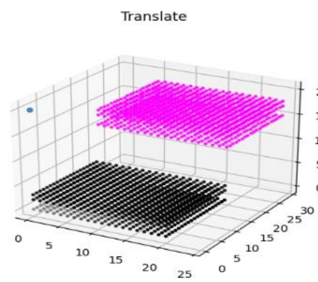
```
#part3-a-----------------------------------------------------------------
#create 3d empty figure
fig = plt.figure()
ax = plt.axes(projection='3d')

#set value for number of index for  x-axis , y-axis
v1=20
v2=20
# create 3 scene of array with index: x=20  , y=20
#create z_vec as a vector for z axis and y_vec as a vector for y axis and
x_vec as a vector for x axis
z_vec = np.linspace(0, 0,v1)
for ii in range(0, v2):
  y_vec=np.linspace(ii, ii, v1)
  x_vec=np.linspace(0, v2, v1)
  #create 3d scatter
  ax.scatter3D(x_vec, y_vec, z_vec, c=z_vec, cmap="gray",marker='.')
z_vec = np.linspace(2, 2,v1)
for ii in range(0,v2):
  y_vec=np.linspace(ii, ii, v1)
  x_vec=np.linspace(0, v2, v1)
  ax.scatter3D(x_vec, y_vec, z_vec, alpha=0.8, c=z_vec,
cmap="gray",marker='.')
z_vec = np.linspace(3,3,v1)
for ii in range(0, v2):
  y_vec=np.linspace(ii, ii, v1)
  x_vec=np.linspace(0, v2, v1)
  ax.scatter3D(x_vec, y_vec, z_vec, alpha=0.8, c=z_vec,
cmap="gray",marker='.')
#create one point on z-axis to show the tall z-axis
ax.scatter3D(0, 0, 20, alpha=0.8,cmap='white')
#show the scatter plots
plt.show()
```

Part3-b-i-ii-iii)Rigid Transform(theta=90,omega=90,phi-90,p=5,q=10,r=15)









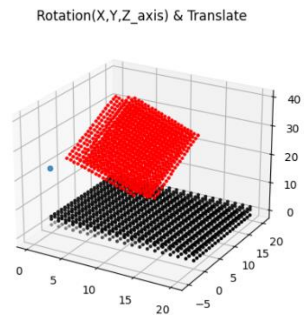
Code:

```
#create zero matrix as Rotation matrix and translation
rotmat_x = np.zeros([3,3])
rotmat_y = np.zeros([3,3])
rotmat_z = np.zeros([3,3])
trans_mat = np.zeros([3,1])

def rigid_transform(theta,omega,phi,p,q,r):
    # define matrix to Rotate in x axis(Rotate matrix)
    rotmat_x[0, 0] = 1
    rotmat_x[0, 1] = 0
    rotmat_x[0, 2] = 0
    rotmat_x[1, 0] = 0
    rotmat_x[1, 1] = np.cos(theta)
    rotmat_x[1, 2] = -np.sin(theta)
    rotmat_x[2, 0] = 0
    rotmat_x[2, 1] = np.sin(theta)
    rotmat_x[2, 2] = np.cos(theta)

    # define matrix to Rotate in y axis(Rotate matrix)
    rotmat_y[0, 0] = np.cos(omega)
    rotmat_y[0, 1] = 0
    rotmat_y[0, 2] = np.sin(omega)
    rotmat_y[1, 0] = 0
    rotmat_y[1, 1] = 1
    rotmat_y[1, 2] = 0
    rotmat_y[2, 0] = -np.sin(omega)
    rotmat_y[2, 1] = 0
    rotmat_y[2, 2] = np.cos(omega)
```

```
    # define matrix to Rotate in z axis(Rotate matrix)
    rotmat_z[0, 0] = np.cos(phi)
    rotmat_z[0, 1] = -np.sin(phi)
    rotmat_z[0, 2] = 0
    rotmat_z[1, 0] = np.sin(phi)
    rotmat_z[1, 1] = np.cos(phi)
    rotmat_z[1, 2] = 0
    rotmat_z[2, 0] = 0
    rotmat_z[2, 1] = 0
    rotmat_z[2, 2] = 1

    # define matrix to Translate
    trans_mat[0, 0] = p
    trans_mat[1, 0] = q
    trans_mat[2, 0] = r

    fig = plt.figure()
    ax = plt.axes(projection='3d')

    v1=20
    v2=20

    # generate and plot 3d grid
    #plot surface1
    z_vec = np.linspace(0, 0,v1)
    for ii in range(0, v2):
      y_vec=np.linspace(ii, ii, v1)
      x_vec=np.linspace(0, v2, v1)
      ax.scatter3D(x_vec, y_vec, z_vec, c=z_vec, cmap="gray",marker='.')

    #plot surface2
    z_vec = np.linspace(2, 2,v1)
    for ii in range(0,v2):
      y_vec=np.linspace(ii, ii, v1)
      x_vec=np.linspace(0, v2, v1)
      ax.scatter3D(x_vec, y_vec, z_vec, alpha=0.8, c=z_vec,
cmap="gray",marker='.')

    #plot surface3
    z_vec = np.linspace(3,3,v1)
    for ii in range(0, v2):
      y_vec=np.linspace(ii, ii, v1)
      x_vec=np.linspace(0, v2, v1)
      ax.scatter3D(x_vec, y_vec, z_vec, alpha=0.8, c=z_vec,
cmap="gray",marker='.')

    ax.scatter3D(0, 0, 20, alpha=0.8,cmap=plt.get_cmap("binary"))

    #calculate x_rotation
    for z in (0,2,3):
     for x in range(0,20):
      for y in range(0,20):
        #multiply 3 andices for each pixel to X rotation matrix ,then plot
them
        x1,y1,z1=np.matmul(rotmat_x,[x,y,z])
```

```
        ax.scatter3D(x1,y1,z1, alpha=0.8, c=z, cmap=plt.get_cmap("autumn"),
marker='.')
    plt.title("X_Rotation")

    #calculate y_rotation
    for z in (0,2,3):
     for x in range(0,20):
      for y in range(0,20):
        #multiply 3 andices for each pixel to Y rotation matrix,then plot
them
        x1,y1,z1=np.matmul(rotmat_y,[x,y,z])
        ax.scatter3D(x1,y1,z1, alpha=0.8, c=z, cmap=plt.get_cmap("winter"),
marker='.')
    plt.title("Y_Rotation")

    #calculate z_rotation
    for z in (0,2,3):
     for x in range(0,20):
      for y in range(0,20):
        #multiply 3 andices for each pixel to Y rotation matrix,then plot
them
        x1,y1,z1=np.matmul(rotmat_z,[x,y,z])
        ax.scatter3D(x1,y1,z1, alpha=0.8, c=z, cmap=plt.get_cmap("summer"),
marker='.')
    plt.title("Z_Rotation")

    #calculate Translate
    for z in (0,2,3):
     for x in range(0,20):
      for y in range(0,20):
        #sum 3 andices with p,q,r for each pixel to translate,then plot them
        ax.scatter3D(x+p,y+q,z+r, alpha=0.8, c=z,
cmap=plt.get_cmap("spring"), marker='.')
    plt.title("Translate")

rigid_transform(np.pi/2,np.pi/2,np.pi/2,5,10,15)
plt.show()
```

If we want to have all rigid transform with together,we add te below code at the end of rigid transform function: Rigid Transform(theta=45,omega=45,phi=45,p=5,q=-5,r=30)

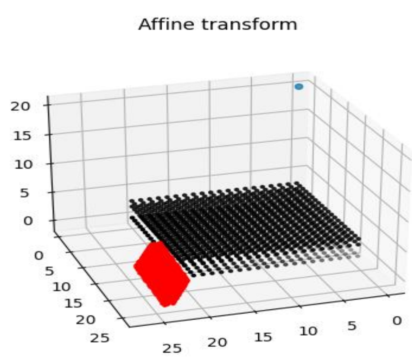



```
#calculate Rotation and Translate with togethear
    for z in (0,2,3):
     for x in range(0,20):
      for y in range(0,20):
        x0,y0,z0=np.matmul(rotmat_x,[x,y,z])
        x1, y1, z1 = np.matmul(rotmat_y, [x0, y0, z0])
        x2, y2, z2 = np.matmul(rotmat_z, [x1, y1, z1])
        #sum 3 andices with p,q,r for each pixel to translate,then plot them
        ax.scatter3D(x2+p,y2+q,z2+r, c=z, cmap=plt.get_cmap("autumn"),
marker='.')
    plt.title("Rotation(X,Y,Z_axis) & Translate")
```

And call rigid_transform function with below values:

```
#call rigid transform with this values
rigid_transform(45,45,45,5,-5,30)
```

Part3-c) Affine Transform(theta=90,omega=0,phi=45,p=20,q=20,r=0,s=0.35)



Code:

```
#create zero matrix as Rotation matrix and translation and Scaling
rotmat_x = np.zeros([3,3])
rotmat_y = np.zeros([3,3])
rotmat_z = np.zeros([3,3])
trans_mat = np.zeros([3,1])
scale_mat = np.zeros([3,3])
```

```
def affine_transform(theta,omega,phi,p,q,r,s):
    # define matrix to Rotate in x axis(Rotate matrix)
    rotmat_x[0, 0] = 1
    rotmat_x[0, 1] = 0
    rotmat_x[0, 2] = 0
    rotmat_x[1, 0] = 0
    rotmat_x[1, 1] = np.cos(theta)
    rotmat_x[1, 2] = -np.sin(theta)
    rotmat_x[2, 0] = 0
    rotmat_x[2, 1] = np.sin(theta)
    rotmat_x[2, 2] = np.cos(theta)

    # define matrix to Rotate in y axis(Rotate matrix)
    rotmat_y[0, 0] = np.cos(omega)
    rotmat_y[0, 1] = 0
    rotmat_y[0, 2] = np.sin(omega)
    rotmat_y[1, 0] = 0
    rotmat_y[1, 1] = 1
    rotmat_y[1, 2] = 0
    rotmat_y[2, 0] = -np.sin(omega)
    rotmat_y[2, 1] = 0
    rotmat_y[2, 2] = np.cos(omega)

    # define matrix to Rotate in z axis(Rotate matrix)
    rotmat_z[0, 0] = np.cos(phi)
    rotmat_z[0, 1] = -np.sin(phi)
    rotmat_z[0, 2] = 0
    rotmat_z[1, 0] = np.sin(phi)
    rotmat_z[1, 1] = np.cos(phi)
    rotmat_z[1, 2] = 0
    rotmat_z[2, 0] = 0
    rotmat_z[2, 1] = 0
    rotmat_z[2, 2] = 1

    # define matrix to Translate
    trans_mat[0, 0] = p
    trans_mat[1, 0] = q
    trans_mat[2, 0] = r

    # define matrix to Scaling
    scale_mat[0, 0] = s
    scale_mat[0, 1] = 0
    scale_mat[0, 2] = 0
    scale_mat[1, 0] = 0
    scale_mat[1, 1] = s
    scale_mat[1, 2] = 0
    scale_mat[2, 0] = 0
    scale_mat[2, 1] = 0
    scale_mat[2, 2] = s

    fig = plt.figure()
    ax = plt.axes(projection='3d')

    v1=20
```

```
    v2=20

    # generate and plot 3d grid
    #plot surface1
    z_vec = np.linspace(0, 0,v1)
    for ii in range(0, v2):
      y_vec=np.linspace(ii, ii, v1)
      x_vec=np.linspace(0, v2, v1)
      ax.scatter3D(x_vec, y_vec, z_vec, c=z_vec, cmap="gray",marker='.')

    #plot surface2
    z_vec = np.linspace(2, 2,v1)
    for ii in range(0,v2):
      y_vec=np.linspace(ii, ii, v1)
      x_vec=np.linspace(0, v2, v1)
      ax.scatter3D(x_vec, y_vec, z_vec, alpha=0.8, c=z_vec,
cmap="gray",marker='.')

    #plot surface3
    z_vec = np.linspace(3,3,v1)
    for ii in range(0, v2):
      y_vec=np.linspace(ii, ii, v1)
      x_vec=np.linspace(0, v2, v1)
      ax.scatter3D(x_vec, y_vec, z_vec, alpha=0.8, c=z_vec,
cmap="gray",marker='.')

    ax.scatter3D(0, 0, 20, alpha=0.8,cmap=plt.get_cmap("binary"))

    #calculate Affine Transform
    for z in (0,2,3):
     for x in range(0,20):
      for y in range(0,20):
        x0,y0,z0=np.matmul(rotmat_x,[x,y,z])
        x1, y1, z1 = np.matmul(rotmat_y, [x0, y0, z0])
        x2, y2, z2 = np.matmul(rotmat_z, [x1, y1, z1])
        x3, y3, z3 = np.matmul(scale_mat, [x2, y2, z2])
        #sum 3 andices with p,q,r for each pixel to translate,then plot them
        ax.scatter3D(x3+p,y3+q,z3+r, c=z, cmap=plt.get_cmap("autumn"),
marker='.')
    plt.title("Affine transform")

affine_transform(90,0,45,20,20,0,0.35)
plt.show()
```

Part3-d)

```
"D:\laptop\Bishops\bishops lectures win 2022\term2\MI Dr russell\assigment2\assigment2-part2\venv\
M1)scale is = 0.9999892299420029
Angle of rotations is= [0.3141608820210316, -0.31414489569845333, 0.3141694001781506]
Translations for p,q,r is=[10. 10. 10.]
M2)scale is = 0.3
Angle of rotations is= [0.0, -1.5707963267948966, 0.0]
Translations for p,q,r is=[-3.   1.5  0. ]
M3)scale is = -2.151507136869408З
Angle of rotations is= [1.6786569724750684, -0.3033033004324591, -1.9281305604187609]
Translations for p,q,r is=[1.8115 0.2873 0.7404]
```

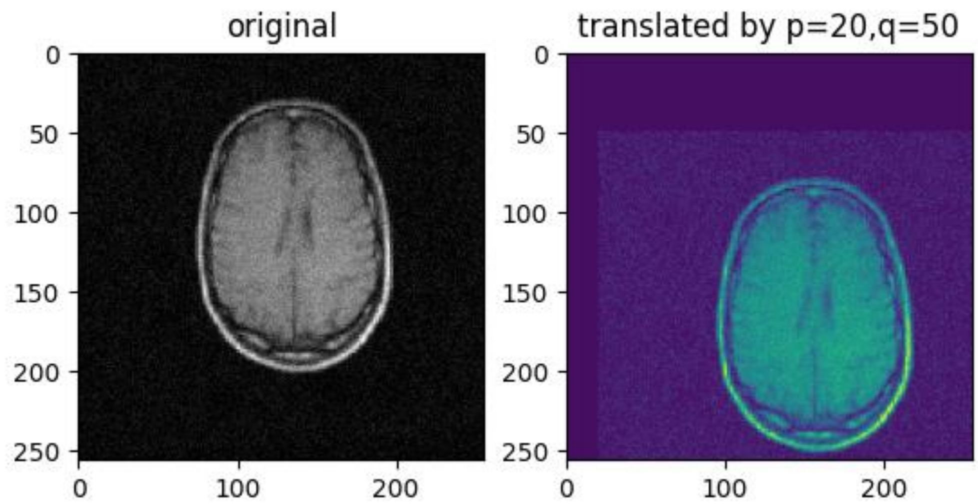
Code:

```
import numpy as np
import math

#define matrices in assigment2
M1=np.array([[0.9045,-0.3847,-0.1840,10],
             [0.2939,0.8750,-0.3847,10],
             [0.3090,0.2939,0.9045,10],
             [0,0,0,1]])
M2=np.array([[-0,-0.2598,0.1500,-3],
             [0,-0.1500,-0.2598,1.5000],
             [0.3000,-0,0,0],
             [0,0,0,1]])
M3=np.array([[0.7182,-1.3727,-0.5660,1.8115],
             [1.9236,-4.6556,-2.5512,0.2873],
             [-0.6426,-1.7985,-1.6285,0.7404],
             [0,0,0,1]])
#Return length, Euclidean norm, of ndarray along axis.
def vector_norm(data,axis=None):

    data = np.array(data, dtype=np.float64, copy=True)
    if data.ndim == 1:
        return math.sqrt(np.dot(data, data))
    data *= data
    out = np.atleast_1d(np.sum(data, axis=axis))
    np.sqrt(out, out)
    return out

# Return sequence of transformations from transformation matrix.
def decompose(matrix):
    # epsilon for testing whether a number is close to zero
    EPS = np.finfo(float).eps * 4.0

    M = np.array(matrix, dtype=np.float64, copy=True).T
    if abs(M[3, 3]) < EPS:
        raise ValueError('M[3, 3] is zero')
    M /= M[3, 3]
```

```
    P = M.copy()
    P[:, 3] = 0.0, 0.0, 0.0, 1.0
    if not np.linalg.det(P):
        raise ValueError('matrix is singular')

    sc = np.zeros((3,))
    sc1 = [0.0, 0.0, 0.0]
    rot = [0.0, 0.0, 0.0]

    if any(abs(M[:3, 3]) > EPS):
        per = np.dot(M[:, 3], np.linalg.inv(P.T))
        M[:, 3] = 0.0, 0.0, 0.0, 1.0
    else:
        per = np.array([0.0, 0.0, 0.0, 1.0])

    trans = M[3, :3].copy()
    M[3, :3] = 0.0

    row = M[:3, :3].copy()
    sc[0] = vector_norm(row[0])
    row[0] /= sc[0]
    sc1[0] = np.dot(row[0], row[1])
    row[1] -= row[0] * sc1[0]
    sc[1] = vector_norm(row[1])
    row[1] /= sc[1]
    sc1[0] /= sc[1]
    sc1[1] = np.dot(row[0], row[2])
    row[2] -= row[0] * sc1[1]
    sc1[2] = np.dot(row[1], row[2])
    row[2] -= row[1] * sc1[2]
    sc[2] = vector_norm(row[2])
    row[2] /= sc[2]
    sc1[1:] /= sc[2]

    if np.dot(row[0], np.cross(row[1], row[2])) < 0:
        np.negative(sc, sc)
        np.negative(row, row)

    rot[1] = math.asin(-row[0, 2])
    if math.cos(rot[1]):
        rot[0] = math.atan2(row[1, 2], row[2, 2])
        rot[2] = math.atan2(row[0, 1], row[0, 0])
    else:
        rot[0] = math.atan2(-row[2, 1], row[1, 1])
        rot[2] = 0.0
    return sc, sc1, rot, trans, per

# call decompose functoin foe each matrix(M1,M2,M3)
mat=[M1,M2,M3]
mat_name=["M1","M2","M3"]
for n in (0,1,2):
    sc, sc1, rot, trans, per = decompose(mat[n])
    print(f'{mat_name[n]})scale is = {sc[0]}\nAngle of rotations is=
{rot}\nTranslations for p,q,r is={trans}')
```

Part4)

Part4-a)



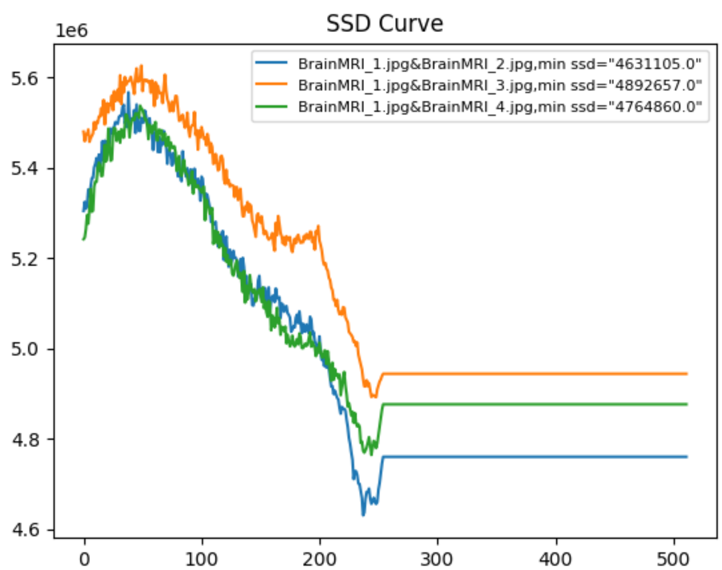
Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d

paths = ['D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_1.jpg']

#Read Data of all images
pic = []
for i in range(len(paths)):
    pic.append(cv2.imread(paths[i]))

#function for translate image based on p and q and input image
def translate_part4(img,p,q):
    pic_z = img[:,:,0]
    # sz_x = 200
    # sz_y = 200
    x = np.linspace(0,img.shape[0],img.shape[0])
    y = np.linspace(0,img.shape[1],img.shape[1])
    f = interp2d(x+p,y+q,pic_z,kind='cubic',fill_value=0)
    new_pic = f(x,y)
    plt.subplot(1,2,1)
    plt.imshow(img)
    plt.title("original")
    plt.subplot(1,2,2)
    plt.imshow(new_pic)
    plt.title(f'translated by p={p},q={q}')
```

```
#call translate function
translate_part4(pic[0],20,50)
plt.show()
```

Part4_b)

Justify: we can see that in all of the curve of two pair of the images ,first the SSD of two images rise and then it is strictly decreasing then fix in the constant value. The SSD for figures(Brain1,Brain2-the blue line) has the minimum value among the figures because it can be seen that brain3,brain4 have rotation and the translation does not good effect for registration for them.



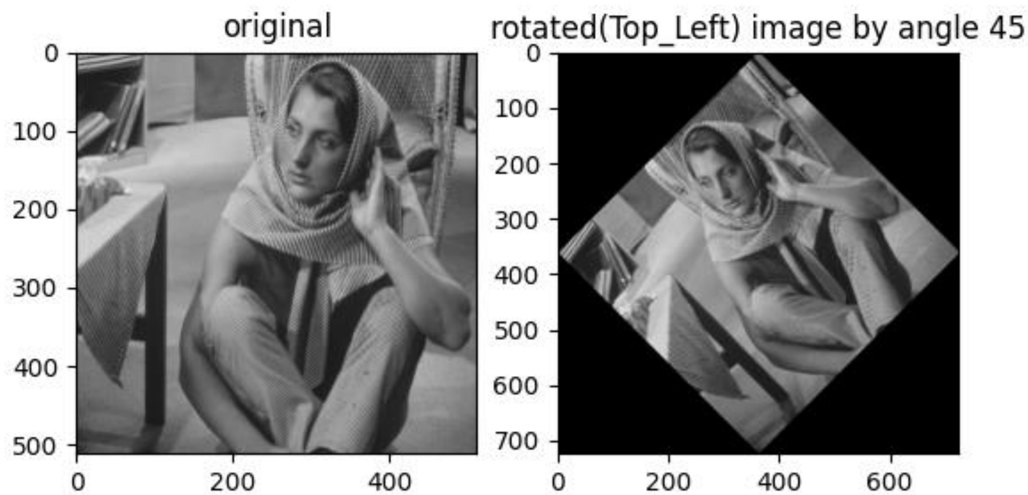
Code:

```
import matplotlib.pyplot as plt
import numpy as np
import os
from skimage.io import imread

paths = ['D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_1.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_2.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_3.jpg',
         'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_4.jpg']

#Read Data of all images
pic = []
for i in range(len(paths)):
    pic.append(imread(paths[i]))
```

```
# calculate ssd for two images
def ssd(img1, img2):
    return np.sum((img1 - img2) ** 2)

# translate the image by x,y step by step in numpy image array
def translateImageStep(img, p, q):
    width = img.shape[0]
    height = img.shape[1]
    # translate image in x axis
    for i in range(p):
        for j in range(width - 1):
            img[j, :] = img[j + 1, :]
    # translate image in y axis
    for i in range(q):
        for j in range(height - 1):
            img[:, j] = img[:, j + 1]
    #return translated image
    return img


# calculate of ssd registration for 2pair of 2d images
def ssd_registration(pic1, pic2):
    #read  2 images and convert them to numpy array
    img1 = np.array(pic[pic1])
    img2 = np.array(pic[pic2-1])
    width = img1.shape[0]+img1.shape[0]
    #create array to save list of ssd of 2 images
    img_ssd_array = np.zeros(width)
    #move image to the most left side to be ready for translate and
comparision
    img_translated = translateImageStep(img1,-width, 0)
    #set p and q as a step to translate image  in x_axis and y_axis
respectivly
    p = 1
    q=1
    #move the second image on the another image by width of the image by
pixel
    for x in range(width):
        #set condition to end of translation
        if width - x >= p:
            #call translation of one image by p(step for translation in
x_axis) and q(step for translation in y_axis)
            img_translated = translateImageStep(img_translated, p, q)
            #find ssd for main image with the second image that translated
and save each ssd in array of ssd to plot ssd curve
            img_ssd_array[x] = ssd(img2, img_translated)
    #find and print minimum ssd for 2 pair of images
    min_ssd=np.min(img_ssd_array)
    print(f'minimum of ssd for[{os.path.basename(paths[pic1])} ,
{os.path.basename(paths[pic2-1])}]={min_ssd}')
    return img_ssd_array,min_ssd

#call "ssd_registration" function to registration 2D image by ssd and plot
ssd for each 2 pair of images
for i in (2,3,4):
```

```
    ssd_array ,minSSd= ssd_registration(0, i)
    plt.plot(ssd_array,
label=f'{os.path.basename(paths[0])}&{os.path.basename(paths[i-1])},min
ssd="{minSSd}"')
    plt.legend(fontsize='8')
    plt.title("SSD Curve")
plt.show()
```

part 4_c)



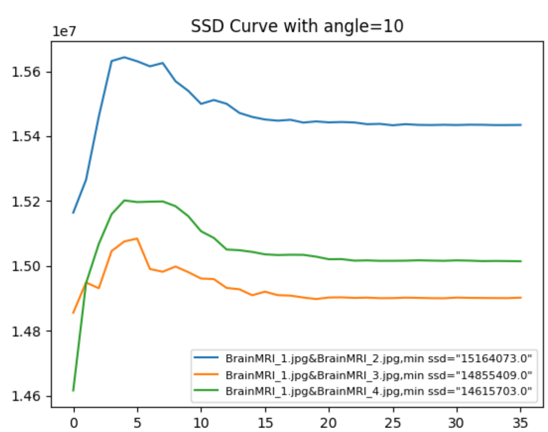
Code:

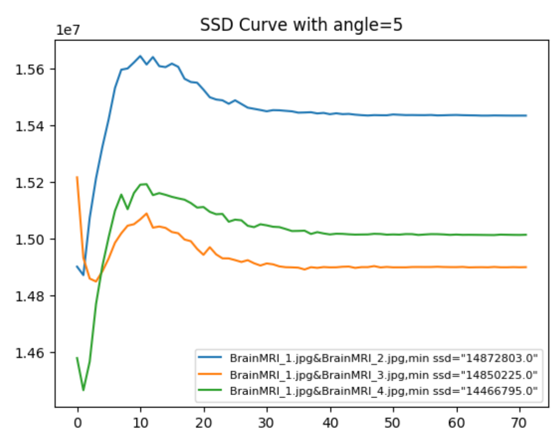
```
import numpy as np
import math
import cv2

#Read Data of all images
pic = []
for i in range(len(paths)):
    pic.append(cv2.imread(paths[i]))

# set values for variables
theta = 45
img = pic[0]

#rotate function of the image around the center by amount of theta
def Rotate_Top_Left_image(img, theta):
    #convert the degrees into radians
    rads = math.radians(theta)
    # find the height and width of the rotated image
    height_rot_img = round(abs(img.shape[0] * math.cos(rads))) + \
                     round(abs(img.shape[1] * math.sin(rads)))
    width_rot_img = round(abs(img.shape[1] * math.cos(rads))) + \
                    round(abs(img.shape[0] * math.sin(rads)))
    #create grid by size of rotation image
```

```
    rot_img = np.uint8(np.zeros((height_rot_img, width_rot_img,
img.shape[2])))
    # Finding the center point of the original image
    cx, cy = (img.shape[1] // 2, img.shape[0] // 2)
    # Finding the center point of rotated image.
    midx, midy = (width_rot_img // 2, height_rot_img // 2)
    # apply rotation matrix to the image
    for i in range(rot_img.shape[0]):
        for j in range(rot_img.shape[1]):
            x = (i - midx) * math.cos(rads) + (j - midy) * math.sin(rads)
            y = -(i - midx) * math.sin(rads) + (j - midy) * math.cos(rads)
            x = round(x) + cy
            y = round(y) + cx
            if (x >= 0 and y >= 0 and x < img.shape[0] and y < img.shape[1]):
                rot_img[i, j, :] = img[x, y, :]
    return rot_img



#call rotation function with 2 paramete
rotated_image = Rotate_Top_Left_image(img, theta)

plt.subplot(1,2,1)
plt.imshow( img)
plt.title("original")

plt.subplot(1,2,2)
plt.imshow( rotated_image)
plt.title(f'rotated image by angle {theta}')
plt.show()
```

part 4-d)

justify: For example, for images (Brain1, Brain2) we find that the second image can be adjusted (registered) to another image with a slight rotation at a low angle (for example 5) because ssd is minimal and ssd increases after further rotation. .



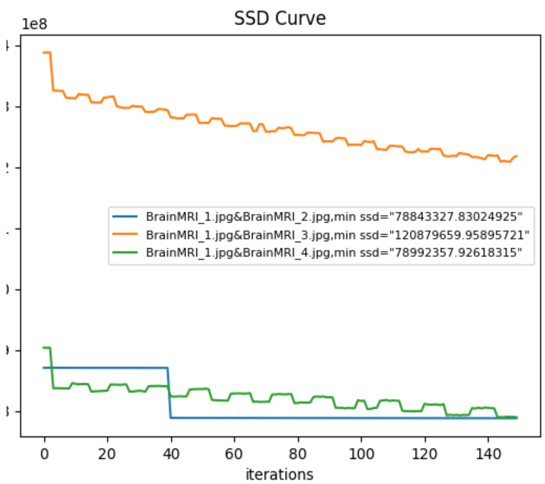


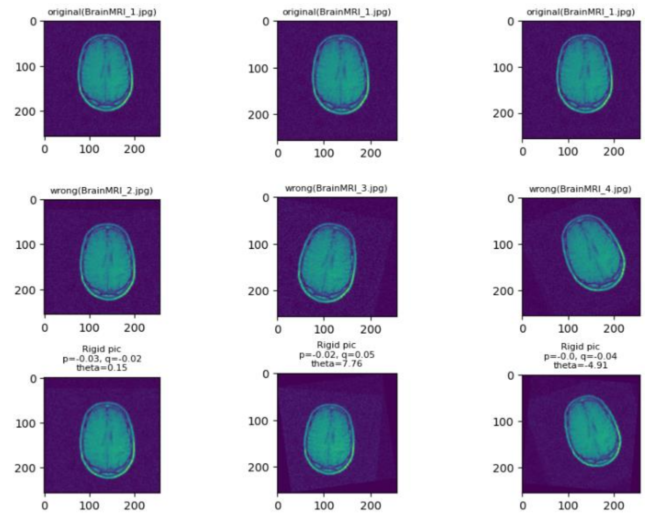
Code:

```
#part 4-d)
# calculate ssd for two images
def ssd(img1, img2):
    print(np.sum((img1 - img2) ** 2))
    return np.sum((img1 - img2) ** 2)

# Rotation the image by theta step by step in base of center of image
def RotationImageStep(img,theta):
    # convert the degrees into radians
    rads = math.radians(theta)
    # find the height and width of the rotated image
    height_rot_img = round(abs(img.shape[0] * math.cos(rads))) + \
                     round(abs(img.shape[1] * math.sin(rads)))
    width_rot_img = round(abs(img.shape[1] * math.cos(rads))) + \
                    round(abs(img.shape[0] * math.sin(rads)))
    # create grid by size of rotation image
    rot_img = np.uint8(np.zeros((height_rot_img, width_rot_img,
img.shape[2])))
    # Finding the center point of the original image
    cx, cy = (img.shape[1] // 2, img.shape[0] // 2)
    # Finding the center point of rotated image.
    midx, midy = (width_rot_img // 2, height_rot_img // 2)
    # apply rotation matrix to the image
    for i in range(rot_img.shape[0]):
        for j in range(rot_img.shape[1]):
            x = (i - midx) * math.cos(rads) + (j - midy) * math.sin(rads)
            y = -(i - midx) * math.sin(rads) + (j - midy) * math.cos(rads)
            x = round(x) + cy
            y = round(y) + cx
            if (x >= 0 and y >= 0 and x < img.shape[0] and y < img.shape[1]):
                rot_img[i, j, :] = img[x, y, :]
    # plt.imshow(rot_img)
    return rot_img

# adjustment the size of "old_size image" to size of "proper_image_size
image'
def adjustment_pic_size(old_size, proper_img_size):
    # get prpoerties of image's shape for goal image
    width = proper_img_size.shape[0]
    height =proper_img_size.shape[1]
    dim = (width, height)
    # resize image
    resized = cv2.resize(old_size, dim, interpolation=cv2.INTER_AREA)
    # print('Resized Dimensions : ', resized.shape)
    # cv2.imshow("Resized image", resized)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    return resized
```

```
# calculate of ssd registration for 2pair of 2d images for rotation
def ssd_registration_rotation(pic1, pic2,theta):
    #read  2 images and convert them to numpy array
    img1 = np.array(pic[pic1])
    img2 = np.array(pic[pic2-1])
    #create array to save list of ssd of 2 images
    img_ssd_array = np.zeros(int(360/theta))
    #set defaul for image_rotated
    img_rotated = RotationImageStep(img1,theta)
    #Rotate the second image to another 360-degree image divided by the input
angle
    for x in range(0,int(360/theta)):
            #call rotation function for one of images by theta(step for
rotation in theta degree)
            img_rotated = RotationImageStep(img_rotated,theta)
            #adjustment the size of 2 pair of images(size f img_rotated
convert equal to size of img2)
            img_rotated=adjustment_pic_size(img_rotated,img2)
            #find ssd for main image with the second image that rotationed
and save each ssd in array of ssd to plot ssd curve
            img_ssd_array[x] = ssd(img2, img_rotated)
    #find and print minimum ssd for 2 pair of images
    min_ssd=np.min(img_ssd_array)
    print(f'minimum of ssd for[{os.path.basename(paths[0])} ,
{os.path.basename(paths[1])}]={min_ssd}')
    return img_ssd_array,min_ssd

#call "ssd_registration_rotation" function to registration 2D image by ssd
and plot ssd for each 2 pair of images
for i in (2,3,4):
    theta=5
    ssd_array,minSSd= ssd_registration_rotation(0, i,theta)
    plt.plot(ssd_array,
label=f'{os.path.basename(paths[0])}&{os.path.basename(paths[i-1])},min
ssd="{minSSd}"')
    plt.legend(fontsize='8')
    plt.title(f'SSD Curve with angle={theta}')
plt.show()
```

Part4_e)





Justify:

Which registrations converge? Brain1,2 and brain 1,3

Which do not converge? Brain1,4

Why do you think some fail to converge? Because Brain 4 has changed in 3 properties (p, q, theta) compared to Brain 1.

Code:

```
import math
import os
import cv2 as cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d
from skimage.io import imread

paths = [
    'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_1.jpg',
    'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_2.jpg',
    'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_3.jpg',
    'D:\laptop\Bishops\\bishops lectures win 2022\\term2\MI Dr
russell\\assigment2\Assignment_2_data\Data\BrainMRI_4.jpg']

# Read Data of all images
pic = []
for i in range(0, len(paths)):
```

```
    pic.append(imread(paths[i]))


# rotate function of the image around the center by amount of theta
def Rotate_Top_Left_image(img, theta):
    # convert the degrees into radians
    rads = math.radians(theta)
    # find the height and width of the rotated image
    height_rot_img = round(abs(img.shape[0] * math.cos(rads))) + \
                     round(abs(img.shape[1] * math.sin(rads)))
    width_rot_img = round(abs(img.shape[1] * math.cos(rads))) + \
                    round(abs(img.shape[0] * math.sin(rads)))
    # create grid by size of rotation image
    rot_img = np.uint8(np.zeros((height_rot_img, width_rot_img)))
    # Finding the center point of the original image
    cx, cy = (img.shape[1] // 2, img.shape[0] // 2)
    # Finding the center point of rotated image.
    midx, midy = (width_rot_img // 2, height_rot_img // 2)
    # apply rotation matrix to the image
    for i in range(rot_img.shape[0]):
        for j in range(rot_img.shape[1]):
            x = (i - midx) * math.cos(rads) + (j - midy) * math.sin(rads)
            y = -(i - midx) * math.sin(rads) + (j - midy) * math.cos(rads)
            x = round(x) + cy
            y = round(y) + cx
            if (x >= 0 and y >= 0 and x < img.shape[0] and y < img.shape[1]):
                rot_img[i, j] = img[x, y]
    return rot_img


# function for translate image based on u(p and q) and input image
def translate(img, u):
    p = u[0]
    q = u[1]
    pic_z = img[:, :]
    x = np.linspace(0, img.shape[0], img.shape[0])
    y = np.linspace(0, img.shape[1], img.shape[1])
    # create new translated points
    f = interp2d(x + p, y + q, pic_z, kind='cubic', fill_value=0)
    # apply new points to the image
    new_pic = f(x, y)
    return new_pic


# create rigid_transform to rotation image then translate the rotated image
then return the image
def rigid_transform(img, theta, u):
    x = img.shape[0]
    y = img.shape[1]

    # call rotation function
    rotated_image = Rotate_Top_Left_image(img, theta)
    # call translate function
    new_pic = cv2.resize(translate(rotated_image, u), (x, y))
    return new_pic
```

```
# create function to calculate the  register function and calculate optimize
ssd
def register(self, j, mode, optimizer, lr, iters):
    mode1 = 'rotation'
    mode2 = 'translation_x'
    mode3 = 'translation_y'

    # set default value for theta and p and q
    u, t = np.zeros(2), 0

    # create empty array to save results of ssd and all of theta  and p and q
    ssd_history = np.zeros(iters)
    tt = np.zeros(iters)
    uu1 = np.zeros(iters)
    uu2 = np.zeros(iters)

    i = self
    x, y = np.arange(j.shape[0]), np.arange(j.shape[1])
    vu, vt = np.zeros(2), 0

    # calculate the optimize ssd in translate and rotation for each change of
image by fix image
    for itr in range(iters):
        # send image to translate and rotate  b theta and u(p,q) and give the
new image
        curr = rigid_transform(i, theta=t, u=u)

        # calculate the deviation for gain ssd minimum
        c, s = np.cos(np.deg2rad(vt)), np.sin(np.deg2rad(vt))
        gx, gy = np.gradient(curr)
        dx, dy, dt = 0., 0., 0.
        # when image was translated then u(p,q) will change then the mode
variable become equal to tranlation_x or y then we optimize  new p and q
        if mode2 == 'translation_x':
            dx = -((curr - j) * gx).sum() * 2
        if mode3 == 'translation_y':
            dy = -((curr - j) * gy).sum() * 2

        # when image was translated then u(p,q) will change then the mode
variable become equal to tranlation_x or y then we optimize  new p and q
        if mode1 == 'rotation':
            dt = -2 * ((curr - j) * (gy * (x * c - y * s) - gx * (x * s + y *
c))).sum()

        # apply new optimize points to the gradien descent equition to gain
new points of p,q,theta
        if optimizer == 'gd':
            # update translation vector in float format
            u[0] = u[0] - float(lr) * (float(dx))
            u[1] = u[1] - float(lr) * (float(dy))
            # update rotation angle  (in degrees)
            t -= lr * dt
        print(f'dt={dt}, dx={dx}, dy={dy}, u={u}, t={t}')
```

```
        # claculate ssd for each ieteration
        ssd_history[itr] = np.sum((curr - j) ** 2)

        # save ssd and theta and p and q for each iteration(to be used to
apply to incorrect image)
        tt[itr] = t
        uu1[itr] = u[0]
        uu2[itr] = u[1]
    return u, t, ssd_history, tt, uu1, uu2  # return translation vector and
rotation angle


# apply registeration for all brain's image
for ii in (1, 2, 3):
    # call register function  with parameters
    u, t, ssd_history, tt, uu1, uu2 = register(pic[ii], pic[0], mode='rigid',
optimizer='gd', lr=1e-10, iters=150)

    # finf minimum ssd for each 2 pair images
    min_ssd = ssd_history.min()
    # get index of min ssd to find the theta and p and q in its step
    min_ssd_index = np.where(ssd_history == min_ssd)
    # convert index to int
    min_ssd_index = int(min_ssd_index[0])

    # get the rotation angle and translation vector at the min ssd
    tt_min = tt[min_ssd_index]
    uu1_min = uu1[min_ssd_index]
    uu2_min = uu2[min_ssd_index]
    uu = [uu1_min, uu2_min]

    #     ## plot ssd_history for all images
    #     plt.plot(ssd_history,
label=f'{os.path.basename(paths[0])}&{os.path.basename(paths[ii])},min
ssd="{min_ssd}"')
    #     plt.legend(fontsize='8')
    #     plt.title("SSD Curve")
    #     plt.xlabel('iterations')
    #     plt.ylabel('ssd')
    # plt.show()
    # whan we want to plot below code , we should comment block code of "plot
ssd_history"
    # plt  rotate and translated image for each picture
    # plot correct image
    plt.subplot(3, 1, 1)
    plt.imshow(pic[0])
    plt.title(f"original({os.path.basename(paths[0])})", fontsize=8)

    # plot incorrect image
    plt.subplot(3, 1, 2)
    plt.imshow(pic[ii])
    plt.title(f"wrong({os.path.basename(paths[ii])})", fontsize=8)

    # plot fixed image (translate and rotation)
```

```
    plt.subplot(3, 1, 3)
    plt.imshow(rigid_transform(pic[ii], tt_min, uu))
    plt.title(f'Rigid pic\np={np.round(uu1_min, 2)}, q={np.round(uu2_min, 2)}\ntheta={np.round(tt_min, 2)}', fontsize=8)
    plt.show()
```

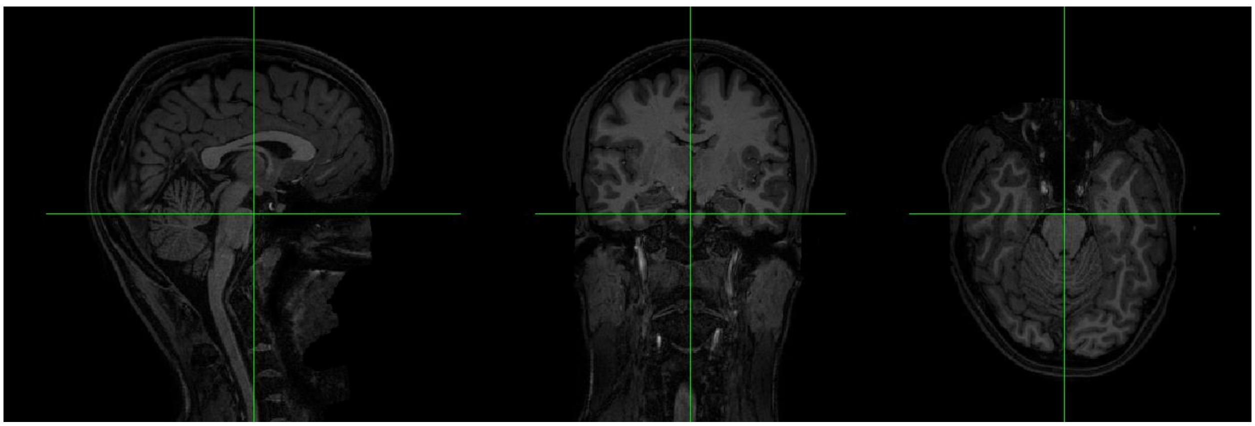
part5)

to solve this part, at the first we install FSLeyes on ubuntu 18 by instruction of the below link:

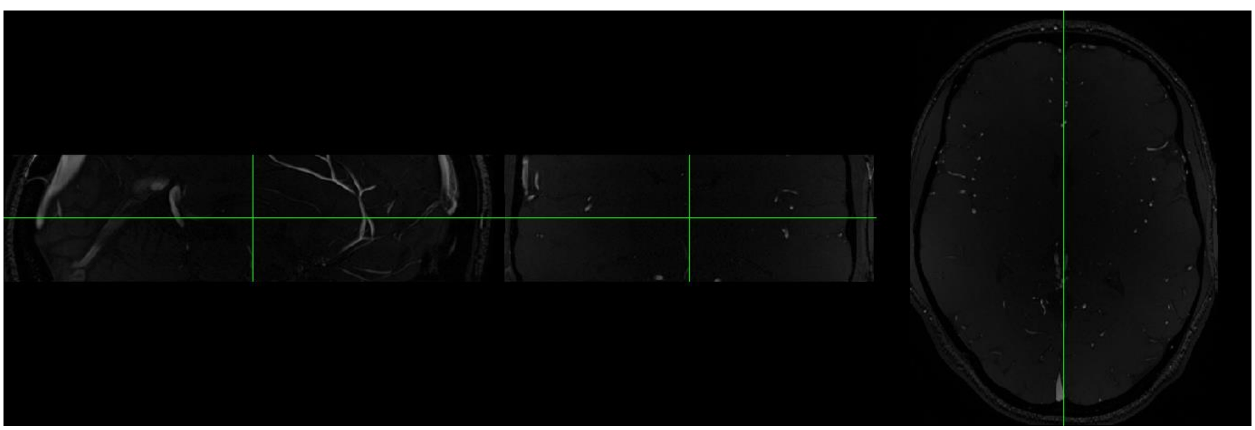
https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation/Windows

then in " ubuntu's shel"l we call the "fsl software" by command: fsl

visualize of t1.nii by fsleyes:

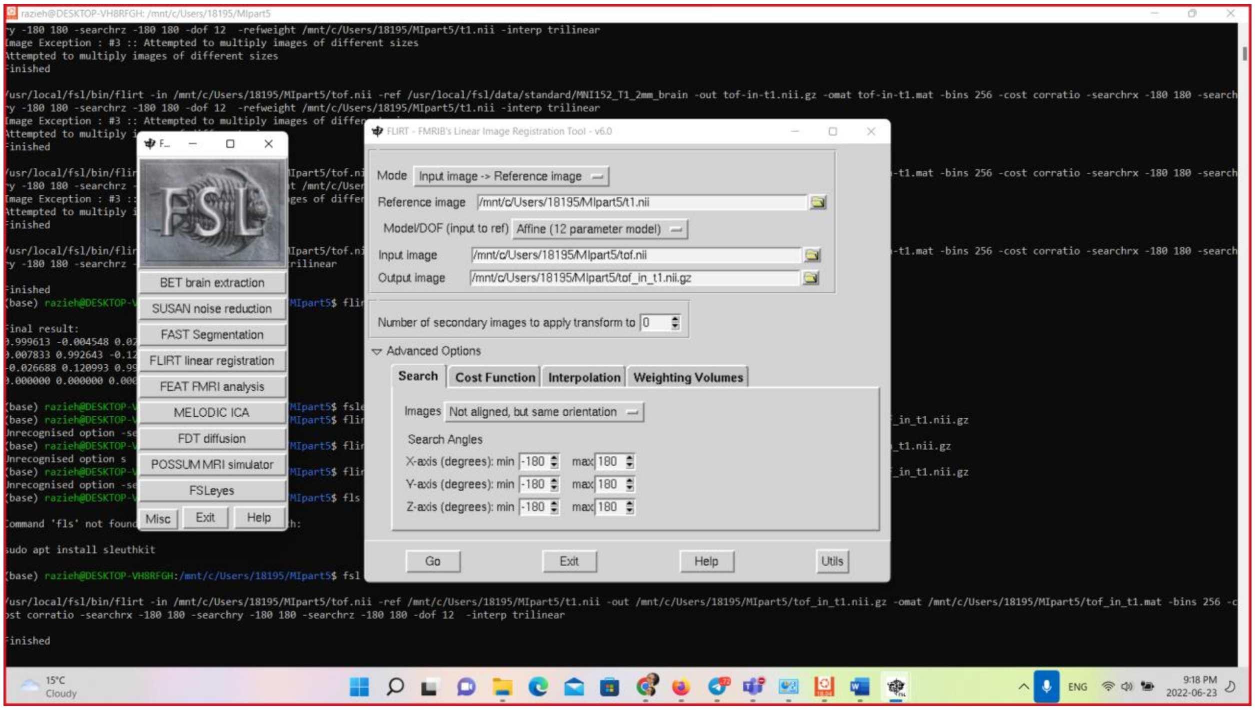


visualize of tof.nii by fsleyes:

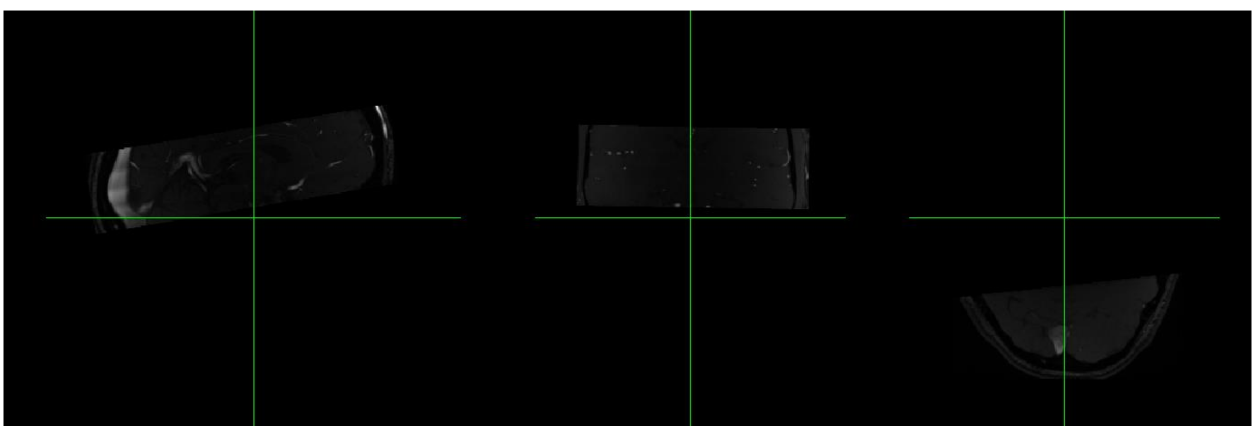


then we select "Flirt Linear Regression" and in the window we set the values like below:

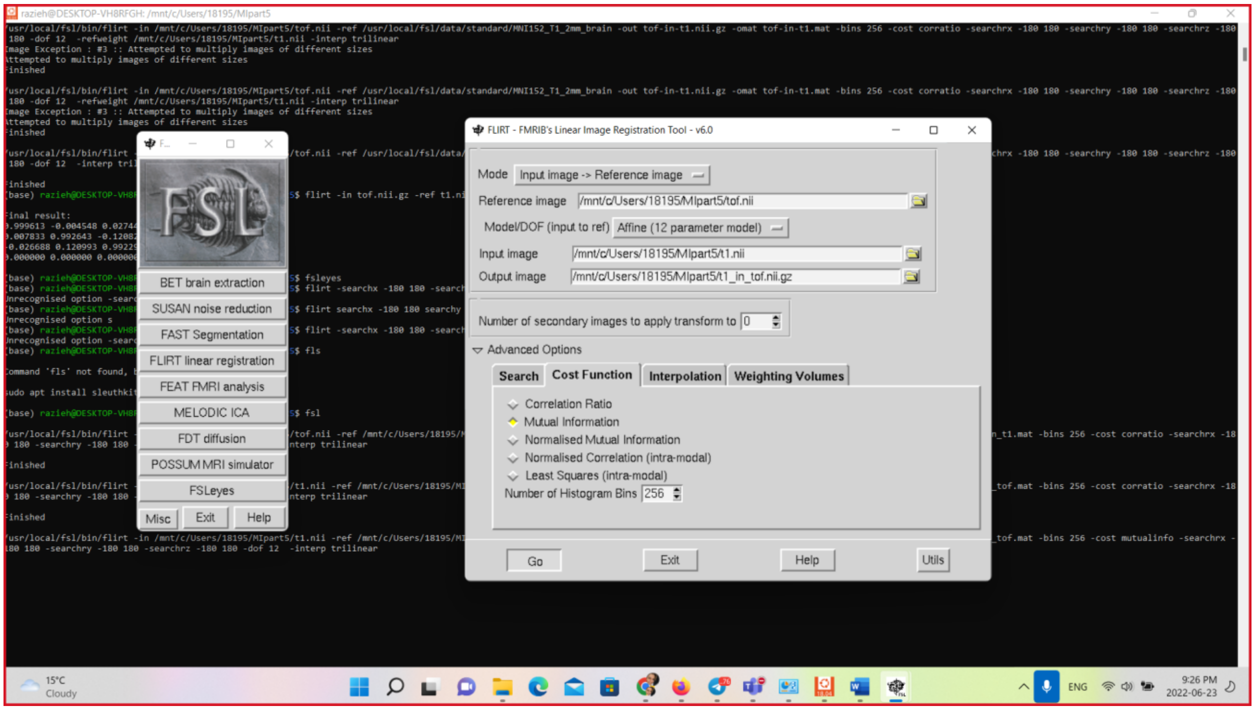
Allign tof in t1:



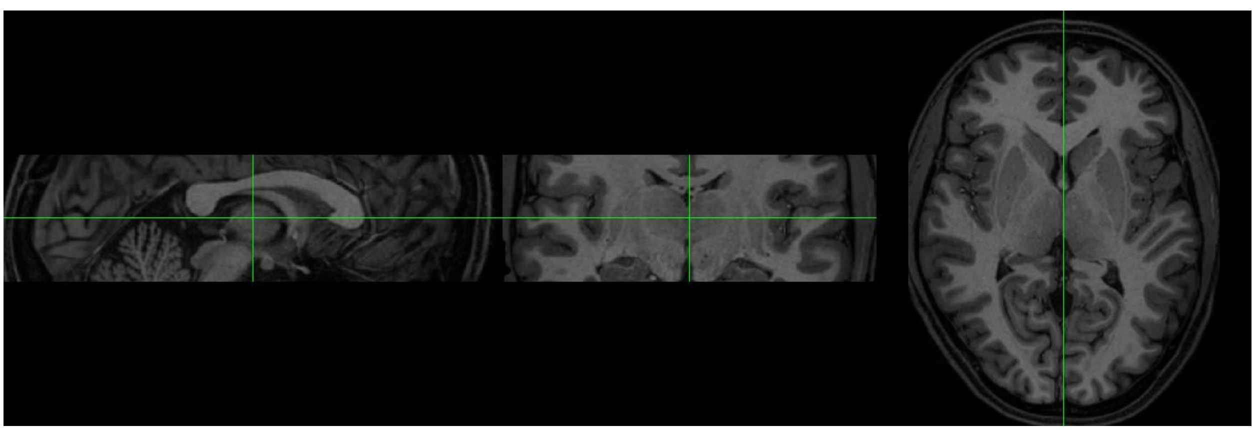
Select "FSLeyes button" to visualize the output (tof in t1):



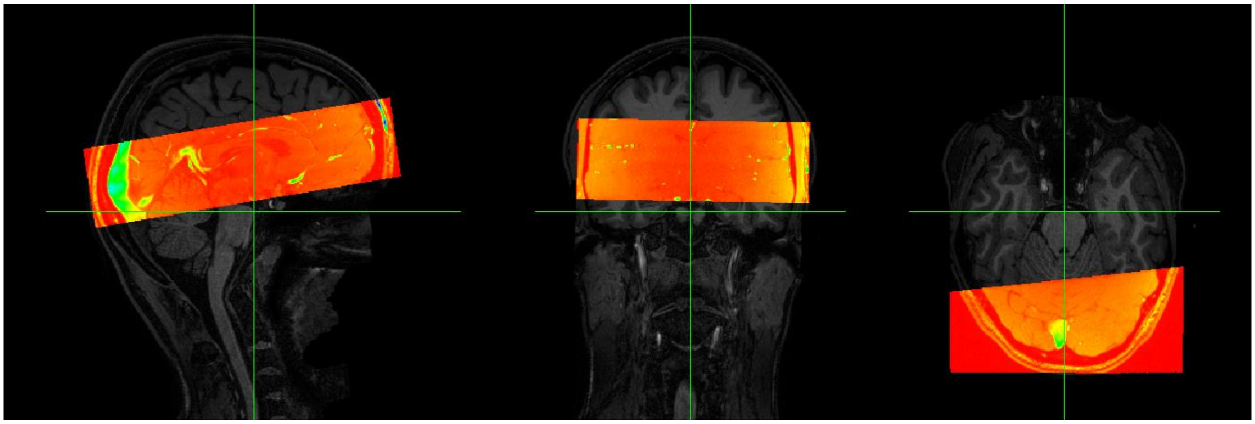
Align t1 in tof by cost=mutual information:



Select "FSLeyes button" to visualize the output (t1 in tof):



For visualize the registration of tof.nii on the t1.nii, first we add the original image of t1 into "fsleyes window" then we add the second image(aligned "tpf in t1") to this window, finally we change the color of the second image,that you can see below:



When we run the software, we could see below code in the ubuntu's shell:

```
(base) razieh@DESKTOP-VH8RFGH:/mnt/c/Users/18195/MIpart5$ fsl

/usr/local/fsl/bin/flirt -in /mnt/c/Users/18195/MIpart5/tof.nii -ref /mnt/c/Users/18195/MIpart5/t1.nii -out /mnt/c/Users/18195/MIpart5/tof_in_t1.nii.gz -omat /mnt/c/Users/18195/MIpart5/tof_in_t1.mat -bins 256 -cost corratio -searchrx -18
0 180 -searchry -180 180 -searchrz -180 180 -dof 12  -interp trilinear

Finished

/usr/local/fsl/bin/flirt -in /mnt/c/Users/18195/MIpart5/t1.nii -ref /mnt/c/Users/18195/MIpart5/tof.nii -out /mnt/c/Users/18195/MIpart5/t1_in_tof.nii.gz -omat /mnt/c/Users/18195/MIpart5/t1_in_tof.mat -bins 256 -cost corratio -searchrx -18
0 180 -searchry -180 180 -searchrz -180 180 -dof 12  -interp trilinear

Finished

/usr/local/fsl/bin/flirt -in /mnt/c/Users/18195/MIpart5/t1.nii -ref /mnt/c/Users/18195/MIpart5/tof.nii -out /mnt/c/Users/18195/MIpart5/t1_in_tof.nii.gz -omat /mnt/c/Users/18195/MIpart5/t1_in_tof.mat -bins 256 -cost mutualinfo -searchrx -
180 180 -searchry -180 180 -searchrz -180 180 -dof 12  -interp trilinear

Finished
```