1 Python correction

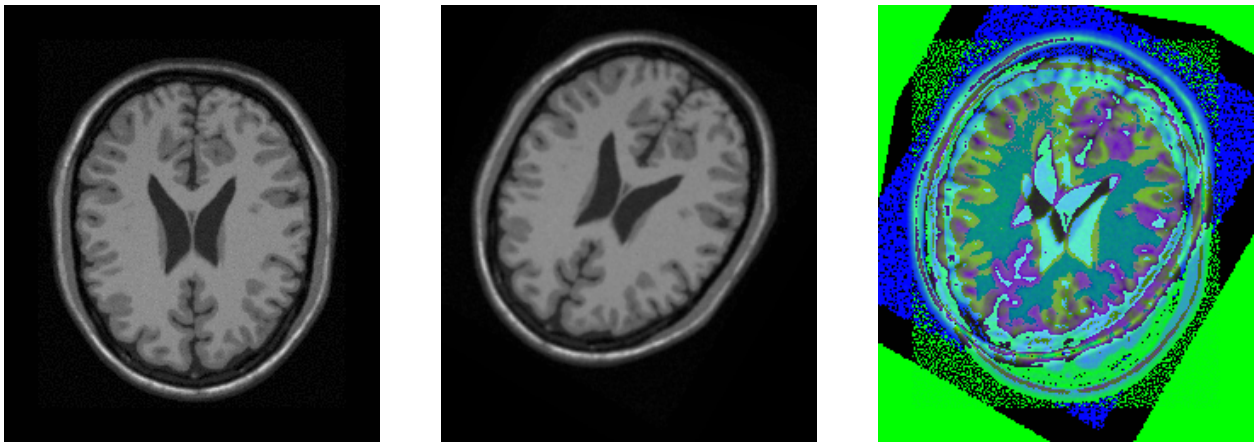
```python
from scipy import misc
import matplotlib.pyplot as plt
import numpy as np
import cv2
from scipy.spatial import cKDTree
```

1.1 Transformation estimation based on corresponding points

1.1.1 Image visualization

The following code is used to display the images (see Fig.1).

```python
# Read images and display
A=misc.imread("brain1.bmp")
B=misc.imread("brain2.bmp")
plt.imshow(A,cmap='gray');
plt.show();
plt.imshow(B,cmap='gray');
plt.show();

# define control points
A_points = np.array([[136, 100], [127, 153], [96, 156], [87, 99]]);
B_points = np.array([[144, 99], [109, 140], [79, 128], [100, 74]]);
```



(a) Moving image. (b) Source image. (c) Superimposition.

Figure 1: Initial images.

If you want to display and save the fusion of both images, you can use this function:

```python
def superimpose(G1, G2, filename=None):
    """
    superimpose 2 images, supposing they are grayscale images and of same shape
    """
    r,c=G1.shape;
    S = np.zeros((r,c,3));
    S[:,:,0] = np.maximum(G1-G2, 0)+G1;
    S[:,:,1] = np.maximum(G2-G1, 0)+G2;
    S[:,:,2] = (G1+G2) / 2;
    S = 255 * S / np.max(S);
    S = S.astype('uint8');
    plt.imshow(S);
    plt.show()
    if filename!=None:
        cv2.imwrite(filename, S);
    return S
```

1.1.2 Manual selection of corresponding points

With openCV, there is not built-in function to make a manual selection of pairs of control points. The following code uses a global variable I in order to manage the display of the points, which are finally stored into the pts variable.

```python
pts = [];
def on_mouse(event, x, y, flags, param):
    """
    callback method for detecting click on image
    It draws a circle on the global variable image I
    """
    global pts, I;
    if event == cv2.EVENT_LBUTTONUP:
        pts.append((x, y));
        cv2.circle(I,(x,y), 2, (0,0,255), -1)

def cpselect():
    """
    method for manually selecting the control points
    It waits until 'q' key is pressed.
    """
    cv2.namedWindow("image")
    cv2.setMouseCallback("image", on_mouse)
    print("press 'q' when finished")
    # keep looping until the 'q' key is pressed
    while True:
      # display the image and wait for a keypress
      cv2.imshow("image", I)
      key = cv2.waitKey(1) & 0xFF

      # if the 'c' key is pressed, break from the loop
      if key == ord("q"):
          break

    # close all open windows
    cv2.destroyAllWindows()
    return pts;
```

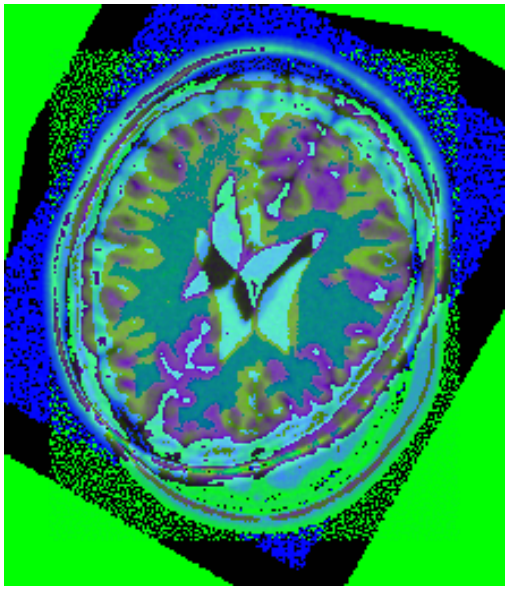
1.1.3 Transformation estimation

The rigid transformation is estimated from the corresponding points by the following function:

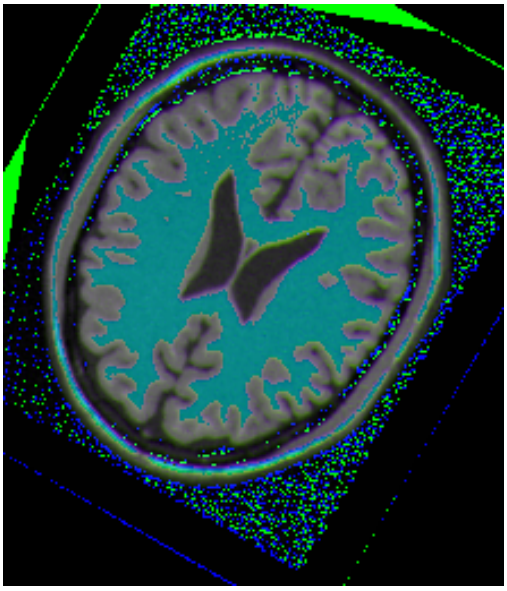
```python
def rigid_registration(data1, data2):
    """
    Rigid transformation estimation between n pairs of points
    This function returns a rotation R and a translation t
    data1 : array of size nx2
    data2 : array of size nx2
    returns transformation matrix T of size 2x3
    """
    data1 = np.array(data1);
    data2 = np.array(data2);

    # computes barycenters, and recenters the points
    m1 = np.mean(data1,0);
    m2 = np.mean(data2,0);
    data1_inv_shifted = data1-m1;
    data2_inv_shifted = data2-m2;

    # Evaluates SVD
    K = np.matmul(np.transpose(data2_inv_shifted), data1_inv_shifted);
    U,S,V = np.linalg.svd(K);

    # Computes Rotation
    S = np.eye(2);
    S[1,1] = np.linalg.det(U)*np.linalg.det(V);
    R = np.matmul(U,S);
    R = np.matmul(R, np.transpose(V));

    # Computes Translation
    t = m2-np.matmul(R, m1);

    T = np.zeros((2,3));
    T[0:2,0:2] = R;
    T[0:2,2] = t;
    return T;
```

Then, you can apply this function to the manually selected points.

```python
# 1st case, rigid registration, with pairs of points in the correct order
T = rigid_registration(A_points, B_points);

# Apply transformation on control points and display the results
data_dest = applyTransform(A_points, T);
I = B.copy();
for pb,pa in zip(data_dest, B_points):
    cv2.circle(I,totuple(pa), 1, (255,0,0), -1)
    cv2.circle(I,totuple(pb), 1, (0,0,255), -1)
plt.imshow(I);
plt.show();
# Apply transformation on image
rows, cols= B.shape;
dst = cv2.warpAffine(A,T, (cols, rows));
plt.imshow(dst);
superimpose(dst, B, "rigid_manual.png");
```



(a) Without registration.



(b) With registration.

Figure 2: Result of the registration for the manually selected control points.

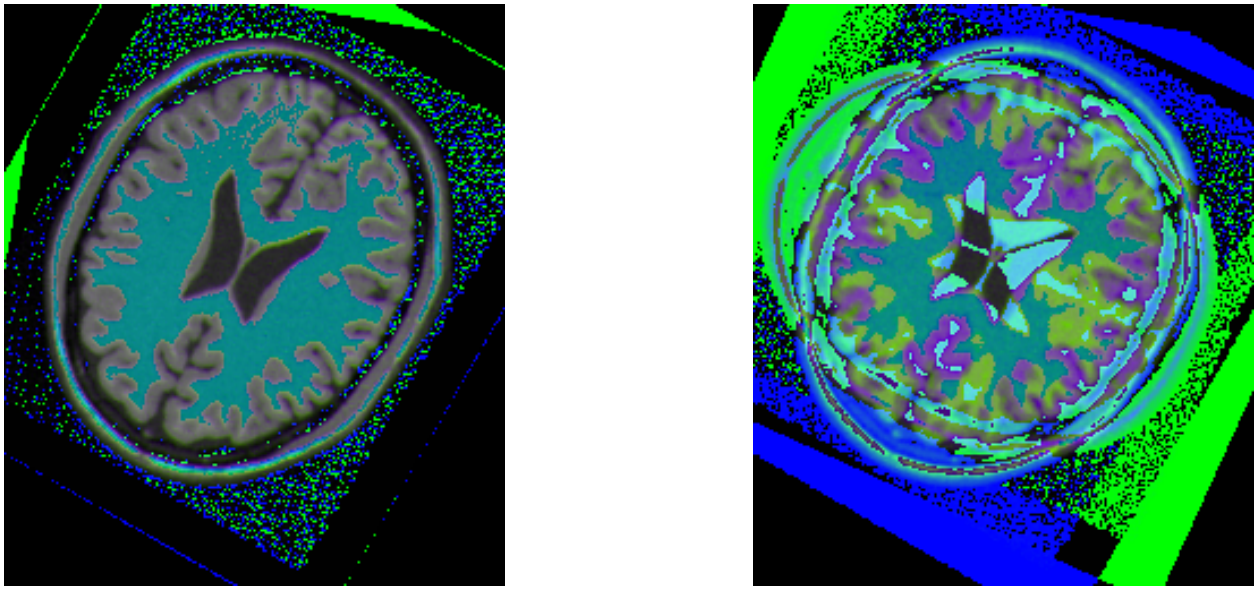
The result is good, because the manual selection of the points is good (the points are given in the correct order for both images).

1.2 ICP registration

1.2.1 Random permutation of points

The following code randomly shuffles the points of the first vector. The result is of course a wrongly registered image, see Fig.3.

```
# random permutation of the points
p = np.random.permutation(np.arange(4));
A_points = A_points[p];
T = rigid_registration(A_points, B_points);
```



(a) Matching pairs of points.

(b) Permulation of the points.

Figure 3: Result of the registration for when the control points are not found in the same order.

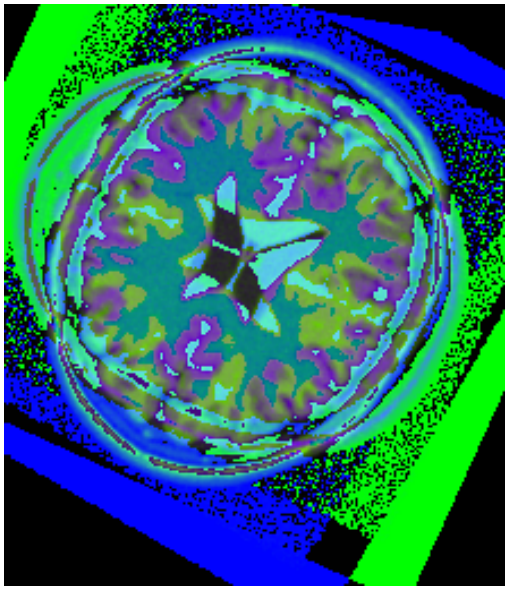
1.2.2 ICP

The previous operations simulates a general non-manual selection of the control points: there is not reason to finding the points by matching pairs. Thus, a reordering is necessary. This propositions implies that the number of points is exactly the same in order to perform the registration process, and that these are matching points (every point has a matching point in the other image). The ICP method (see code for icp_transform) reorders the points via a nearest neighbor rule.

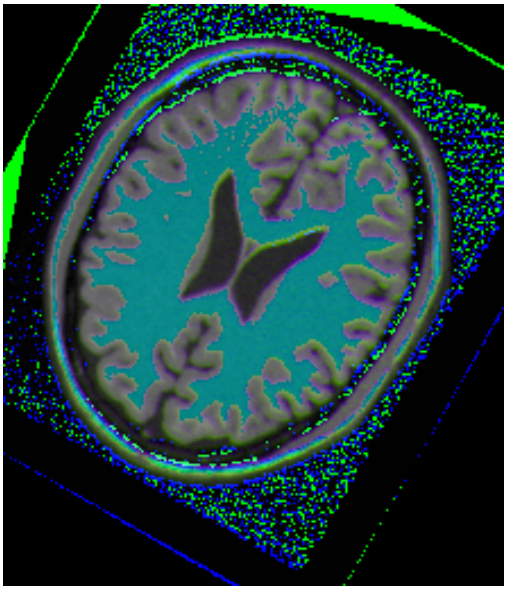
```python
def icp_transform(dataA, dataB):
    """
    Find a transform between A and B points
    with an ICP (Iterative Closest Point) method.
    dataA and dataB are of size nx2, with the same number of points n
    returns the transformation matrix of shape 2x3
    """
    data2A = dataA.copy();
    data2B = np.zeros(data2A.shape);
    T = np.zeros((2,3));
    T[0:2,0:2] = np.eye(2);
    T[0:2,2] = 0;

    nb_loops=5;
    tree = cKDTree( dataB );

    for loop in range(nb_loops):
        # search for closest points and reorganise array of points
            ↪ accordingly

        d, inds = tree.query( data2A);

        data2B = dataB[inds,:];
        # find rigid registration with reordered points
        t_loop = rigid_registration(data2A, data2B);

        T = composeTransform(t_loop, T);

        # evaluates transform on control points, to make the iteration
        data2A = applyTransform(dataA, T);

    return T;
```

1.2.3 Automatic extraction of corner points

Generally, the points are automatically detected, and thus, there is no warranty that they are found in the same order, nor that each pair of point correspond to matching points (some points –called outliers– need to be eliminated to compute the correct transformation). In this tutorial, we do not address the problem of outliers. Please notice that with these parameters and images, by chance, the same points are detected in the correct order.



(a) Random shuffle of points and direct rigid transformation estimation.

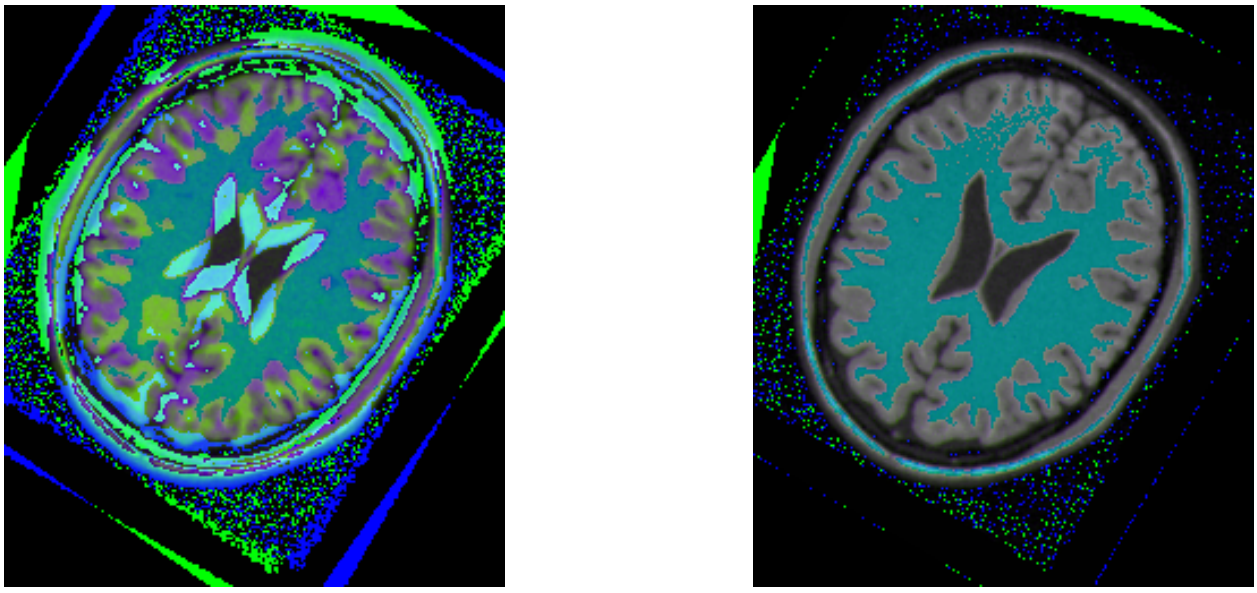


(b) ICP registration on the same points.

Figure 4: Result of the registration for when the control points are not found in the same order. The ICP algorithm reorders the points and gives a good result.

```python
# Automatic extraction of corner points
# Harris corners detection or Shi and Tomasi
# this method do not ensure the correct order in the points (A_points and
# B_points)
nb_points = 4; # number of points to detect
corners = cv2.goodFeaturesToTrack(A,nb_points,0.01,10, blockSize=3,
    ↪ useHarrisDetector=False)
corners = np.int0(corners)
a, b, c = corners.shape;
A_points = np.reshape(corners, (a,c));

corners = cv2.goodFeaturesToTrack(B,nb_points,0.01,10, blockSize=3,
    ↪ useHarrisDetector=False)
corners = np.int0(corners)
a, b, c = corners.shape;
B_points = np.reshape(corners, (a,c));
```

The Fig.5 displays the results with the automatic detection of corners (in this case, the method from Shi and Tomasi). Notice that by chance, the points in both images match. In other cases, one would have to remove outliers.



(a) (b)

Figure 5: Shi and Tomasi corners detection. By chance, the points correspond and the ICP method gives a correct result.