Bonn-Aachen International Center for Information Technology
University of Bonn
Master Programme in Life Science Informatics
Master Thesis

Brain Tractography Registration with Nonrigid ICP

Mohammed Abdelgadir Awadelkareem Hassan

Supervisor: Prof. Thomas Schultz
Bonn-Aachen International Center for Information Technology
University of Bonn

Supervisor: Dr. Martin Vogt
Bonn-Aachen International Center for Information Technology
University of Bonn

Date: February 20, 2019

ACKNOWLEDGMENT

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The registration problem or sometimes called alignment or absolute orientation is one of fundamental problem in computer vision. If we have two objects and we need to align them that means we should reduce the distance between them by making one object fix and move the other one to the closest distance, this simple form of alignment called rigid transformation because it doesn't change the shape of the template object, where as the non-rigid transformation is when deformation occur to the shape of the template object. Due to fundamental importance of registration in computer vision, it is necessary step in many different applications, for instance: object recognition, tracking, range data fusion, graphics, medical image alignment, robotics and structural bioinformatics etc [1].

As we mention, the registration is important in medical image alignment, we are going to apply it on Brain fiber pathways, which is 3D images data. This type of data is not easy to align due to slightly different shape of the same fiber pathway in different brains, and sometimes alignment occur between left side and right side of the fiber pathway.

One of the famous methods to solve this problem is Iterative Closest Point (ICP), which we apply in this thesis. The main idea is in each point on the moving object we find the closest point on the fixed object, with respect to threshold, and try to minimize the overall distance between them, the minimal overall distance can be reached by solving the cost function that represent the distance and stiffness, which we will discuss later in this thesis. Iterative Closest Point (ICP) can be applied by different algorithms, in our case we use Least squares (LSQR) algorithm to solve our problem and we have sufficient results.

1

INTRODUCTION

-complexity of brain -Data set and usage of registration (all the details about the data) and the 3D model -Importance of registration (differences case and control) -motivation/Proposal of the thesis - comparison between non-rigid and rigid image registration -Medical application

- brain anatomy
- how we take MRI
- how we build 3D model
- The difficulty of aligning pathways

As we mention before, registration is important step on many application including bioinformatics, which is our case, because our data is brain pathways (sometimes in this thesis we call the bundles or tracts for simplicity). Our method will help comparing passion and control brain pathways to distinguish the deformation occur on patients brains due to disease or outside affects. we apply ICP method refer to [2] with some altering to fit the spacial case of our data, because the method on [2] applied for surface registration and our data is tracts (streamlines).

Our data as we mentioned is brain pathways which are part of white matter (brain consists of white and gray matter as we will explain later),

2

METHOD

1 THE ICP FRAMEWORK

The main subject of this thesis is demonstrating how the ICP framework can be extended to nonrigid registration, whilst keeping properties of the original algorithm. The principle idea is the application of the work presented in [2]. The extended ICP framework uses different regularisations to control the incrementally deformation of template towards the target whilst decreasing the distance and stiffness weight iteratively, recovering the whole range of global and local deformations.

As has been defined in the introduction, vertex registration is a problem in which two or more datasets of points are given and the task is to optimally align them by estimating a best transformation. In our case, we use a dense registration method to find a mapping from each point in the template onto the target while sparse methods find correspondence only for selected feature points. This is done by deforming the template, locally moving it closer in each iteration to the target in order to wrap them together with respect to stiffness.

ICP moves the template S towards the target T step by step. In each iteration, it minimizes the difference between the template S and the target, T as illustrated in figure {1}, to reach the minimal value by solving the main equation (1):

$$||Ax - b||^2 \tag{1}$$

In equation (1) x is a list of X_i, each X_i is a 3×4 affine matrix which uses homogeneous coordinates $[x, y, z, 1]$ in 3D Euclidean space. By stacking X_i together we get matrix x with size $4n \times 3$ as shown in equation (2):

$$X = [X_1, X_2, ..., X_n]^T \tag{2}$$

If we were to simply solve the equation (1) by assigning A as the template and b as the target, we will get exactly $Ax = b$, which means the difference between them is zero. That leads to the deforming of A and a complete loss of its shape, therefore, we need to add the stiffness part to the equation that will prevent this deformation by keeping the vertices originally as close to each other as possible which we will later explain in this chapter.

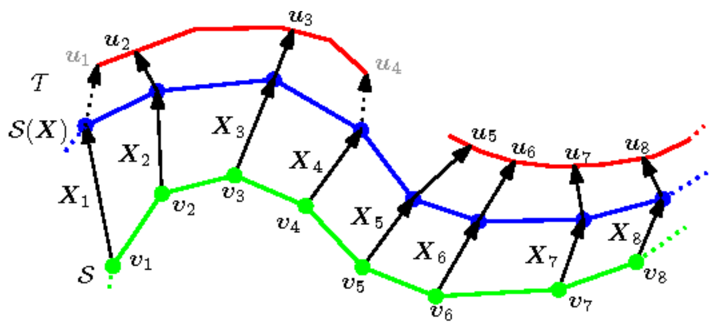


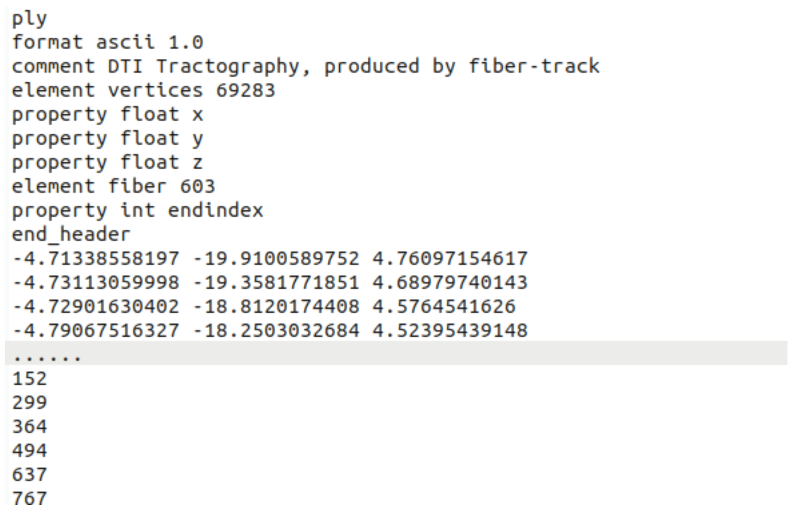
Figure 1: The template graph S (green) is deformed by locally affine transformations (X_i) onto the target graph T (red). The algorithm determines closest points (u_i) for each displaced source vertex $(X_i v_i)$ and finds the optimal deformation for the stiffness used in this iteration. This is repeated until a stable state is found. The process then continues with a lower stiffness. Due to the stiffness constraint the vertices do not move directly towards the target graph, but may move parallel along it. The correspondences u_1 and u_4 are dropped as they lie on the border of the target [2].

2 SETUP DATA

The method presented in [2] deals with surface graph representations of the data $S = (V, E)$ where S, V and E are assigned to graph, vertices and edges, respectively. As described in the introduction, our data contains human brain fiber bundles (pathways) saved in a ***ply*** data format, as shown in the simplified figure {4}. The ***ply*** format consists of three parts; the first, uppermost part is the header of the file which contains the file description (i.e. format, comment, etc) and the major components of the header element parts which describe how many elements each part has. In the sample in figure {4}, there are 69283 vertices in x,y,z and 603 fibers. The second part of the ***ply*** files contains vertices in 3D Euclidean coordinate space while the last part of the ***ply*** file represents the end index of each fiber. As the method in [2] require the template to be in graph format, we develop a tool to read ***ply*** files into a numpy array of arrays which can subsequently be used as a graph. To be able to use ***numpy ndarray*** as a graph, we put each tract in a separate array with respect to the link between points (Edges E). To do so, we put each point linked together next to each other, and then wrap all the tracts belonging to the same bundle (i.e ATR, CC, genu, splenium, etc) in a new ***numpy ndarray***.

3 PREPARING GRAPHS FOR ICP

After we read the data in graph format, we must align the template graph to the target graph as closely as possible. In order to do so, we use *Principal Components Analysis (PCA)*. Before applying PCA, we scale the template graph and the target graph to a $[0, 1]$ scale to match each other. This is done by



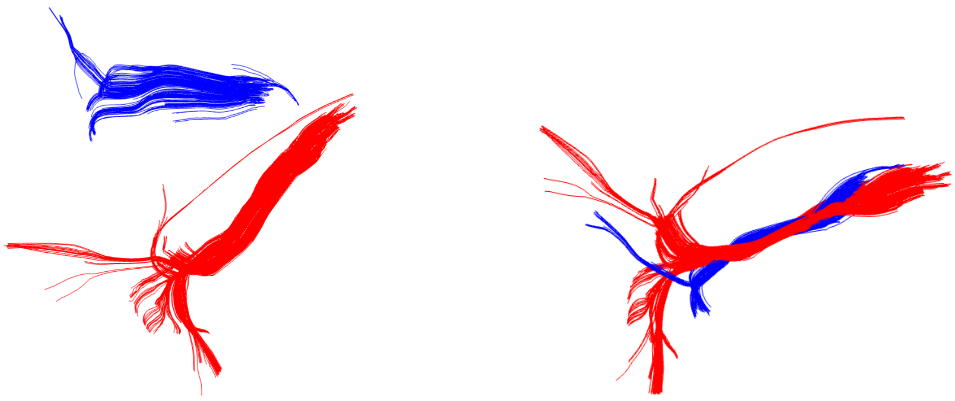
```
ply
format ascii 1.0
comment DTI Tractography, produced by fiber-track
element vertices 69283
property float x
property float y
property float z
element fiber 603
property int endindex
end_header
-4.71338558197 -19.9100589752 4.76097154617
-4.73113059998 -19.3581771851 4.68979740143
-4.72901630402 -18.8120174408 4.5764541626
-4.79067516327 -18.2503032684 4.52395439148
......
152
299
364
494
637
767
```

Figure 2: Simplified ***ply*** file sample

subtracting all points from the minimum value in the graph and then dividing them by the maximum value in the graph. Next, we apply PCA and use $[-1,1]^3$ cube combination and measure the distance between each point in the template graph to the closest point in the target graph $\sum ||v_i - v_j||_F^2$ where $v_i \in S, v_j \in T$ eight times to select the combination with the minimum distance as is illustrated in table {1}.

1	2	3	4	5	6	7	8
(x,y,z)	(x,y,-z)	(x,-y,z)	(x,-y,-z)	(-x,-y,z)	(-x,y,-z)	(-x,y,z)	(-x,-y,-z)

Table 1: $[-1,1]^3$ cube combination



(a) Original position (b) After PCA

Figure 3: PCA alignment

4 DISTANCE MEASUREMENT

In all processes in our code, we use K-D tree (or k-dimensional) to measure the euclidean distance between the points, including in PCA transformation, ICP or any other step in which we measure the distance.

4.1 *The K-D tree*

"I will right something about K-D tree"

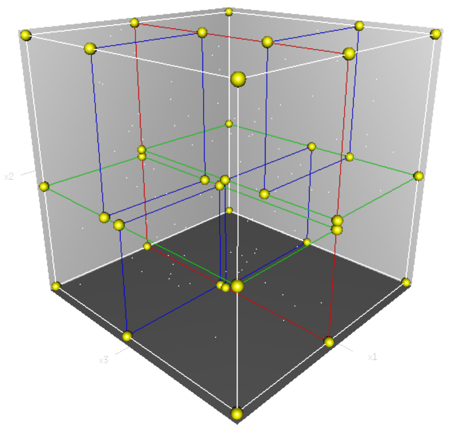


Figure 4: A 3-dimensional k-d tree. The first split (the red vertical plane) cuts the root cell (white) into two subcells, each of which is then split (by the green horizontal planes) into two subcells. Finally, four cells are split (by the four blue vertical planes) into two subcells. Since there is no more splitting, the final eight are called leaf cells [3].

5 FINDING THE AFFINE TRANSFORMATION MATRICES

After we prepare our data, we refer back to [2] and determine how we need to build the cost function. As we are going to explain, it consists of three parts: the distance term, stiffness term and landmark term.

As we mention above, the cost function, as shown in equation (3), consists of three terms, each of which represent different concepts: the distance term, stiffness term and landmark term. The distance term represents the distance between the template graph S and target graph T which has to be as minimal as possible (i.e., which must be minimized). The stiffness term regularizes the template graph to prevent it from being exactly the target graph. The final term is the landmark term which gives the initial case for the template graph.

$$E(X) = E_d(X) + \alpha E_s(X) + \beta E_t(X) \tag{3}$$

Let us illustrate the first term, **the distance term**. As we mentioned before, it is the euclidean distance between points in the template graph S and points in the target graph T, which can be written as represented in equation (4). This distance has to be minimized as much as possible to align the template graph S to the target graph T with respect to the original shape of the template graph S. That means we need to ensure (by using the stiffness term, which we will discuss later in this section) that all points which are close to each other stay as near to each other as possible.

$$E_d(X) = \sum_{v_i \in V} w_i dist^2(T, X_i v_i) \tag{4}$$

In equation (4), v_i represent the vertices V in the template graph S in homogeneous coordinate $v_i = [x, y, z, 1]^T$, such that they can be multiplied by affine matrices X_i. These affine matrices are 3×4 matrices including the transformation part, as illustrated in equation (1). The distance between each vertex $v_i \in V$ in the template graph S and its closest vertex $u_i \in U$ in target graph T noted with $dist^2(T, X_i v_i)$, and $w \in [0, 1]$ is weight which is *one* if the correspondence for this vertex v_i is found in the target graph S, and *zero* otherwise. This concept will let each vertex v_i in the template graph T moves toward its correspondence vertex u_i in the target graph T if it is found, otherwise, it will move with its neighbors because of the stiffness. Essentially, we use a distance threshold as a parameter to distinguish correspondances.

Now we come to the second term, **the stiffness term**, which regularizes the shape deformation, represented in equation (5):

$$E_s(X) = \sum_{i,j \in E} ||(X_i - X_j)G||_F^2 \tag{5}$$

Regularizing the deformation occurs by penalising the weighted difference of the transformations of neighbouring vertices under the Frobenius norm $||.||_F$ using a weighting matrix $G := diag(1, 1, 1, \gamma)$.

The parameter γ is used to balance the rotational and skew against translation while transforming the template graph S [2]. The value of parameter γ depends on the units of the data and on the type of deformation that shall be expressed. In our case we select it to be one as we have already scaled our data into the $[-1, 1]^3$ cube.

We use twelve parameters per vertex in 3×4 shape. This will allow us to write the cost function as a quadratic function which can be solved directly [2].

The Frobenius norm in equation (5) defined for matrix A is demonstrated in equation (6):

$$||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2} \tag{6}$$

By applying this equation we keep the vertices which are linked together (i.e., there is an edge between them or they are in the same tract) or neighbors close to each other.

The α parameter in equation (3) is a constant that manages the effect of the stiffness term; when it is high the neighbor vertices doesn't move far from eachother and when it is low, a greater deformation can occur.

The final part of the equation (3) is the **Stiffness term** which is demonstrated in the equation below (7):

$$E_l(X) = \sum_{(vi,l)\in L} ||(X_i v_i - l)||^2 \tag{7}$$

Amberg et al. mention the landmark initializes and guides the registration. Given a set of landmarks $L = \{(v_{i1}, l_1), (v_{i2}, l_2), ..., (v_{il}, l_l)\}$ mapping template graph S vertices V_i into the target graph T vertices U_i.

The registration can be found even without landmarks. Without landmarks, the cost function has global minima where the template is collapsed onto a point on the target surface, but the local minimum corresponding to the correct registration can be found for a wide range of initial conditions.

In our case, we use PCA transformation as the initial step to align two graphs, therefore, we omit the landmark term later on from our cost function This is considered to be a first optimization to the cost function due to a reduction of computational effort and time to calculate the landmark term in our code.

We illustrate now the cost function in more depth and transform the graphs in matrices that suit our cost function to ease implementation of the method.

The **distance term** in equation (3) and (4) become:

$$\bar{E}_d(X) = \sum_{v_i \in V} w_i ||X_i v_i - u_i||^2 \tag{8}$$

$$= \left\| (W \otimes I_3) \left(\begin{bmatrix} X_1 & & \\ & \ddots & \\ & & X_n \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} - \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \right) \right\|^2$$

where $W = diag(w_1, \ldots, w_n)$ corresponds to the weight in the diagonal matrix multiplied by Kronecker product $\otimes$ to I_3 identity matrix. X is diagonal matrix of X_i that we need to solve and that is multiplied by V which are vertices of template graph S, subtracted by the corresponding vertices U of the target graph T.

the **Kronecker product**, denoted by $\otimes$, is an operation on two matrices of arbitrary size resulting in a block matrix. It is a generalization of the outer product, (denoted by the same symbol) from vectors to matrices, and gives the matrix of the tensor product with respect to a standard choice of basis [4].

For instance, If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the Kronecker product $A \otimes B$ is the $mp \times nq$ block matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

Thus, the result of $W \otimes I_3$ is a $3n \times 3n$ matrix, where matrix X has size $3n \times 4n$, V has a shape $4n \times 1$ and U has shape $3n \times 1$.

We continue to reform the equation to be easily differentiated by converting it into canonical form. We swap the position of the X and V matrices. For matrix V we define a sparse matrix D which contains the the vertices $v_i \in V$ in diagonal position as demonstrated below:

$$D = \begin{bmatrix} v_1^T & & & \\ & v_2^T & & \\ & & \ddots & \\ & & & v_n^T \end{bmatrix} \tag{9}$$

The new format of the equation becomes:

$$\bar{E}_d(X) = ||W(DX - U)||_F^2 \tag{10}$$

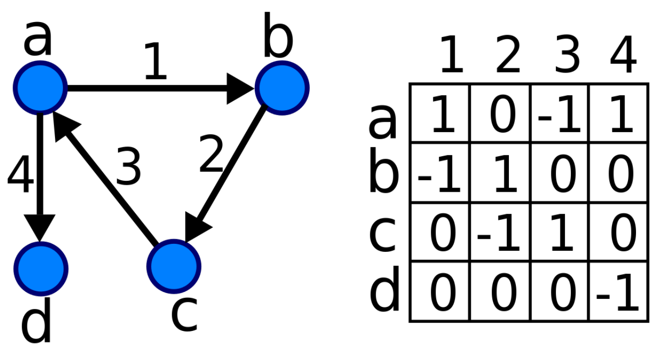
Where the matrix W has size $n \times n$, the matrix D has size $n \times 4n$ (vertices v_i are in homogeneous coordinates $[x, y, z, 1]^T$), X has size $4n \times 3$, and U doesn't change with shape $n \times 3$ (vertices u_i are in 3D coordinate).

We continue to reform **the stiffness term** which penalises differences between the transformation matrices (affine matrices) X assigned to neighboring vertices. The simplified form of the stiffness term will be:

$$E_s(X) = ||(M \otimes G)X||_F^2 \tag{11}$$

where M is a node-arc incidence matrix defined for directed graphs. The number of rows in this matrix equal the number of nodes (vertices) and the number of columns equal the number of arcs (edges) on the graph and the values has to be *zeros, ones and/or minus ones*. The value will be *zero* if the edge on this column is not connected to the vertex on this row where the value lies. Otherwise, the value must be either 1 or -1; it will be 1 if the edge direction is coming towards the vertex and -1 otherwise. As illustrated in the figure {5}, the matrix M has size $e \times n$ where $eandn$ are the number of edges and vertices respectively. $G = diag(1, 1, 1, y)$ is the diagonal matrix and the result of $M \otimes G$ is a matrix with size $4e \times 4n$.

The final term is **the landmark term** which needs to be reformed as well; the process of doing so is similar to that of the distance term. We only consider



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline a & 1 & 0 & -1 & 1 \\ b & -1 & 1 & 0 & 0 \\ c & 0 & -1 & 1 & 0 \\ d & 0 & 0 & 0 & -1 \end{array}$$

Figure 5: Oriented graph with corresponding incidence matrix [5]

those vertices from D (D *from the distance term* (8)) which are landmarks in a new matrix called D_L, and vertices from U (which are vertices from target graph T) which are corresponded to those landmarks. These in turn are put in a new matrix denoted $U_L = [l_1, l_1, \dots, l_l]^T$.

The final shape of the cost function is a quadratic function, as shown below:

$$\bar{E}(X) = \left\| \begin{bmatrix} \alpha M \otimes G \\ WD \\ \beta D_L \end{bmatrix} X - \begin{bmatrix} 0 \\ WU \\ U_L \end{bmatrix} \right\|^2 = ||AX - B||_F^2 \tag{12}$$

The current form of the function can be minimized by setting its derivative to zero and solving it as linear equation. The minimum value of $\bar{E}(X)$ is when $X = (A^T A)^{-1} A^T B$. A is also a sparse matrix and most of its values are zeros. However, it is still a large matrix and requires the most time in solving the equation. In order to solve this problem, we will remove the landmark term as mentioned before because we already have the initial step by applying PCA. Once we omit the landmark term, the function will look like this:

$$\bar{E}(X) = \left\| \begin{bmatrix} \alpha M \otimes G \\ WD \end{bmatrix} X - \begin{bmatrix} 0 \\ WU \end{bmatrix} \right\|^2 = ||AX - B||_F^2 \tag{13}$$

6 APPLYING THE TRANSFORMATIONS

Let us review the principle of affine transformation briefly. If we have a point $[x, y, 1]$ in homogeneous coordinate in 2D euclidean space and we want to move this point three units through the X *axis*, $[x, y, 1]^T a = [x+3, y, 1]^T$, then in this case, a is called the affine matrix.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x+3 \\ y \\ 1 \end{bmatrix}$$

The example mentioned above is only for translation; we still have some other moves (i.e. sheer, rotation and scale) which can be performed, as demonstrated in figure {6} for 2D shape.

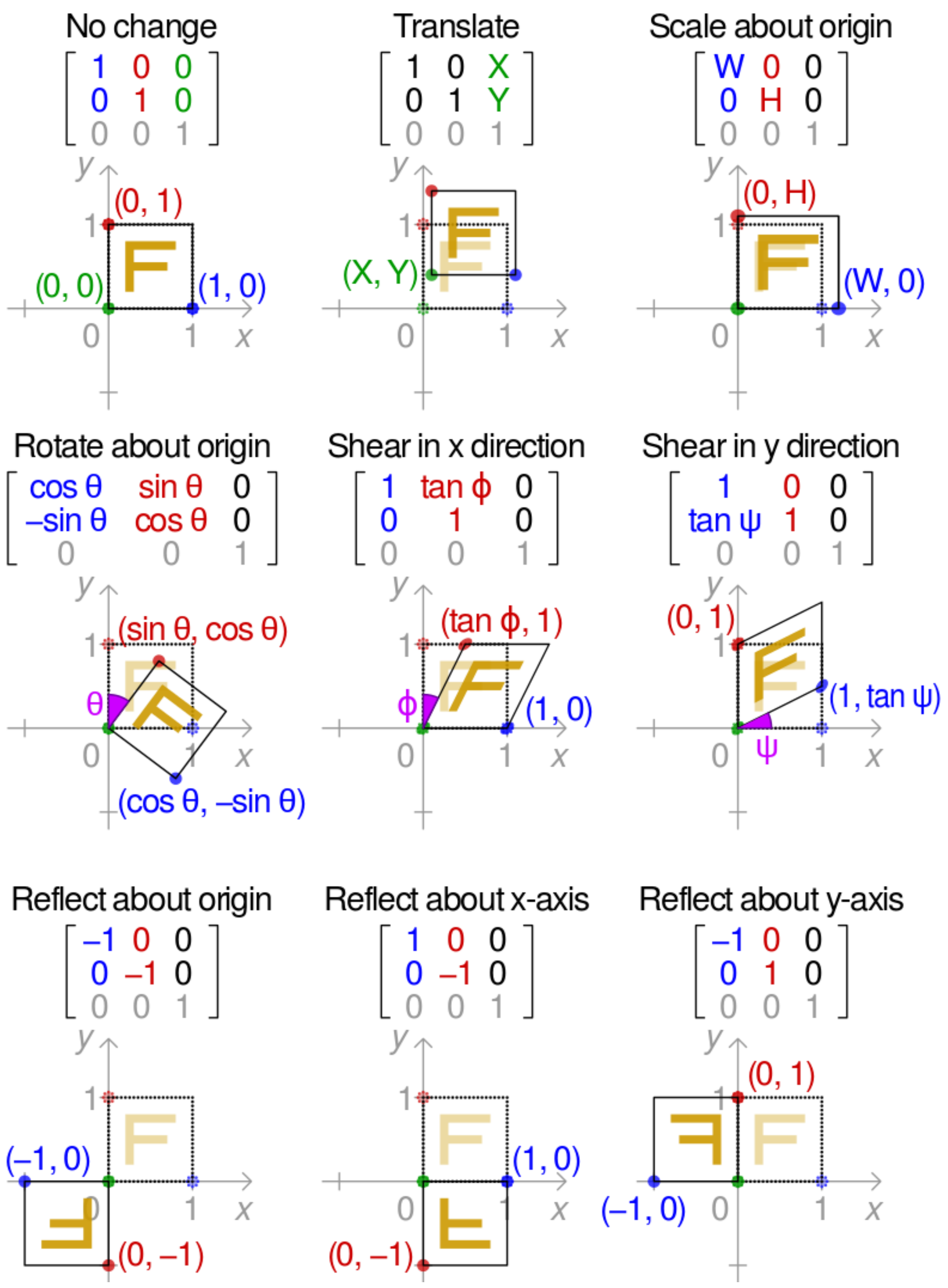


Figure 6: 2D affine transformation matrices [6]

We have shown how 2D transformation occurs; the same principles apply to 3D as well, as demonstrated below:

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \\ 1 \end{bmatrix} \begin{bmatrix} a & b & c & e \\ f & g & h & i \\ j & k & l & m \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

However, this is still for one point; if we have more points as in our case, the equation above becomes:

$$\begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \dots & \hat{x}_n \\ \hat{y}_1 & \hat{y}_2 & \dots & \hat{y}_n \\ \hat{z}_1 & \hat{z}_2 & \dots & \hat{z}_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} a & b & c & e \\ f & g & h & i \\ j & k & l & m \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

7 NONRIGID OPTIMAL ICP ALGORITHMS

As we mentioned in the introduction, ***Iterative Closest Point (ICP)*** is an algorithm employed to minimize the distance between two or more points clouds. To achieve this goal, we use ***Sparse Least Squares algorithm (LSQR)*** which is an iterative method for solving $Ax = b$ and $min||Ax - b||^2$, where the matrix A is large and sparse (as in our case). The method is based on the bidiagonalization procedure of Golub and Kahan [7]. It is analytically equivalent to the standard method of conjugate gradients, but possesses more favorable numerical properties [7].

We will demonstrait the idea of using *LSQR* to solve our cost function in the next chapter implementation.

8 IMPLEMENTATION

To implement the method we have discussed before, we wrote the code in *Python Programming Language* and *Spyder* version 3 and *Integrated Development Environment (IDE)* are used.

As we mentioned before, our data is in ***ply*** format as shown in figure {4}, to read this format we wrote our own method using *plyfile* library due to special case of our data which has different arrangement files. The method we wrote return data in *numpy ndarray* structure where each array tract is putted in an array and all tracts belong to the same bundle were putted together with respect to the sequences of the neighboring vertices.

Then we apply PCA transformation (*sklearn.decomposition* library) to the data and generate histogram (*matplotlib.pyplot* library) for distances between each vertex in the template graph S and its closet vertex in target graph T, before and after PCA transformation to compare the distance difference, because in some cases where the two graphs (template and target) are already in the best alignment before ICP, PCA transformation increase the distance between them as we will show in the result chapter. If the distance between two graphs is increased we drop the PCA transformation step. For distance measurement we use *K-D tree* from *scikit learn*.

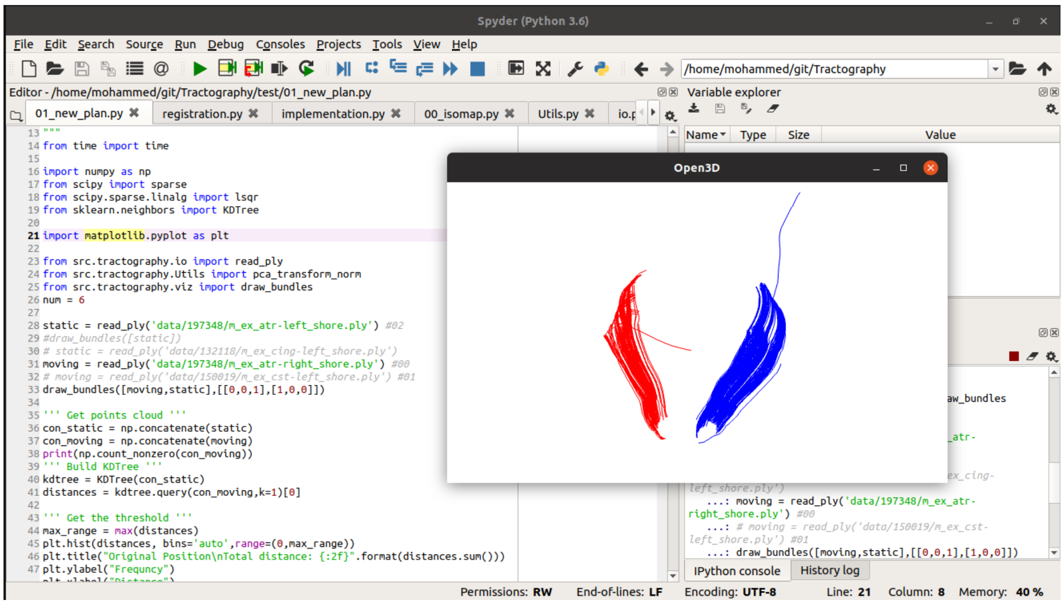
We continue to build variables as shown in equation (**??**), to do so, we use histograms generated in the previous step to determine the threshold for the maximum distance we need to consider for W in the **distance term** for cost function, if the value is equal or below the threshold $w_i = 1$, otherwise $w_i = 0$. Then we build W as diaginal sparse matrix using *scipy.sparse* library. Then we continue using the same library (*scipy.sparse*) to build D sparse matrix and calculate WD and WU.

Now we have **the distance term** of the cost function, we need to build the **stiffness term**, we use the same library (*scipy.sparse*) to build M sparse matrix and G diagonal sparse matrix and calculate Kronecker product ($M \otimes G$).

The final step of building the cost function is to vertically stack MG and WD to have A, and vertically stack *zero* sparse matrix and WU to have B as require $||Ax - b||_F^2$.

The last step is solving the cost fuction $||Ax - b||_F^2$ by using ***LSQR*** from *scipy.sparse.linalg* library. As mentioned in the *scipy.sparse.linalg.lsqr* documentation [8], b in equation $Ax - b$ must be a vector, and as b in our case is a matrix size $n \times 3$, we use matrices fundamental concept to solve this problem as we use **lsqrt** three times for each column and horizontally stack the result to get the affine matrices combination.

Finally for visualization we customize functions from *open3d* library to view brain bundles due to special case of our data.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/mohammed/git/Tractography
Editor - /home/mohammed/git/Tractography/test/01_new_plan.py
01_new_plan.py registration.py implementation.py 00_isomap.py Utils.py io.p
Variable explorer
Name Type Size Value
Open3D
IPython console History log
Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 21 Column: 8 Memory: 40 %

```python
"""
from time import time

import numpy as np
from scipy import sparse
from scipy.sparse.linalg import lsqr
from sklearn.neighbors import KDTree

import matplotlib.pyplot as plt

from src.tractography.io import read_ply
from src.tractography.Utils import pca_transform_norm
from src.tractography.viz import draw_bundles
num = 6

static = read_ply('data/197348/m_ex_atr-left_shore.ply') #02
#draw_bundles([static])
# static = read_ply('data/132118/m_ex_cing-left_shore.ply')
moving = read_ply('data/197348/m_ex_atr-right_shore.ply') #00
# moving = read_ply('data/150019/m_ex_cst-left_shore.ply') #01
draw_bundles([moving,static],[[0,0,1],[1,0,0]])

''' Get points cloud '''
con_static = np.concatenate(static)
con_moving = np.concatenate(moving)
print(np.count_nonzero(con_moving))
''' Build KDTree '''
kdtree = KDTree(con_static)
distances = kdtree.query(con_moving,k=1)[0]

''' Get the threshold '''
max_range = max(distances)
plt.hist(distances, bins='auto',range=(0,max_range))
plt.title("Original Position\nTotal distance: {:2f}".format(distances.sum()))
plt.ylabel("Frequncy")
```

```
aw_bundles
_atr-
ex_cing-
left_shore.ply')
   ...: moving = read_ply('data/197348/m_ex_atr-
right_shore.ply') #00
   ...: # moving = read_ply('data/150019/m_ex_cst-
left_shore.ply') #01
   ...: draw_bundles([moving,static],[[0,0,1],[1,0,0]])
```

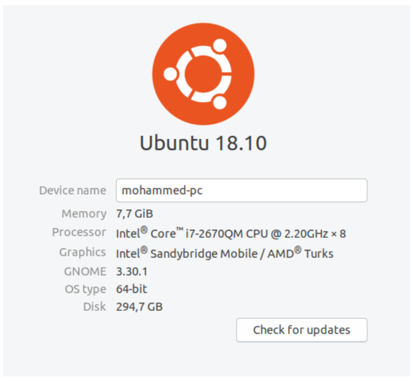
Figure 7: IDE and visualization tool

3

RESULTS

Outline

- The device specification
- Number of sample and description
- Data source (from where we get it)
- Number of points and tracts
- The algorithm and method to compare with
- Comparison :
    - ISO MAP [to be discuss]
    - Time
    - Distance
    - Visual



Ubuntu 18.10

Device name	mohammed-pc
Memory	7,7 GiB
Processor	Intel® Core™ i7-2670QM CPU @ 2.20GHz × 8
Graphics	Intel® Sandybridge Mobile / AMD® Turks
GNOME	3.30.1
OS type	64-bit
Disk	294,7 GB

Figure 8: To be edited

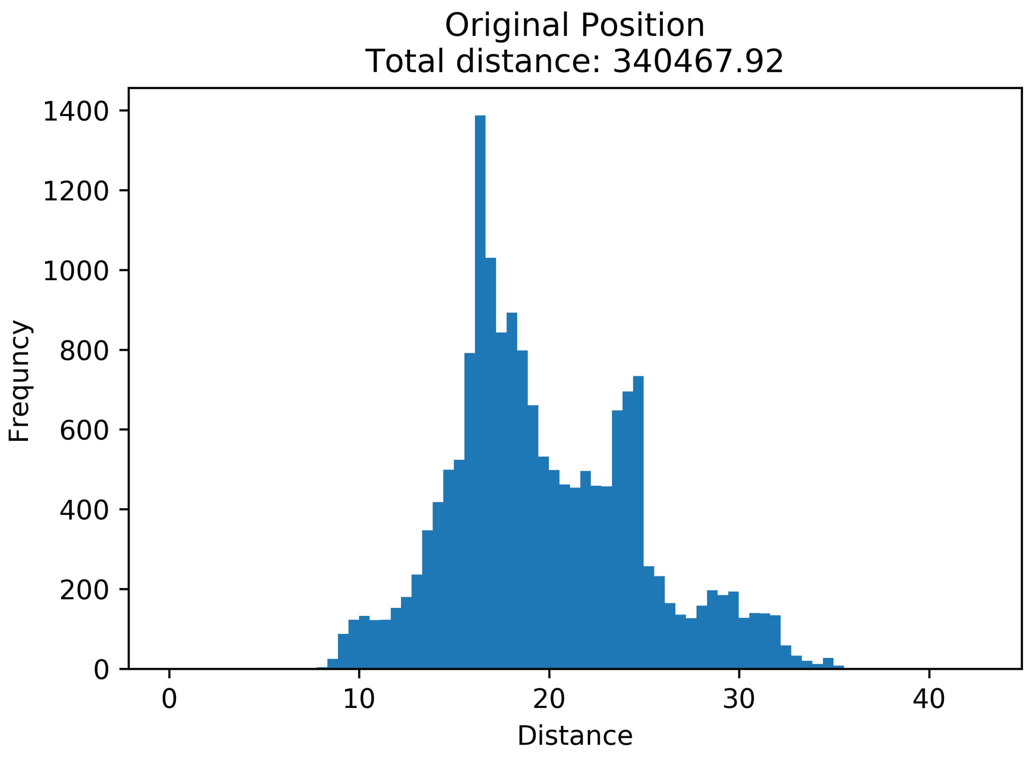


Figure 9: To be edited

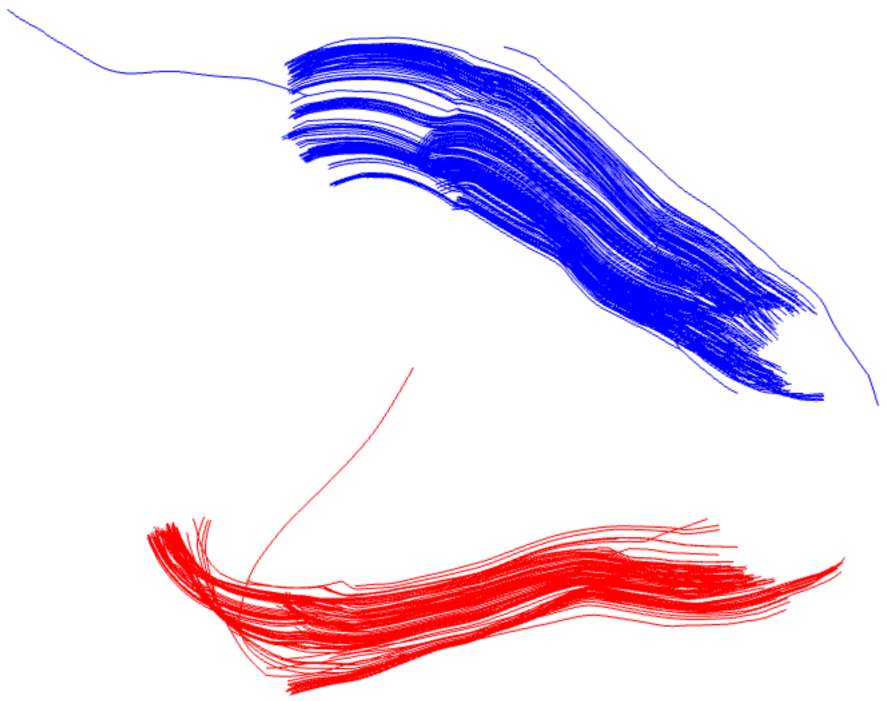


Figure 10: To be edited

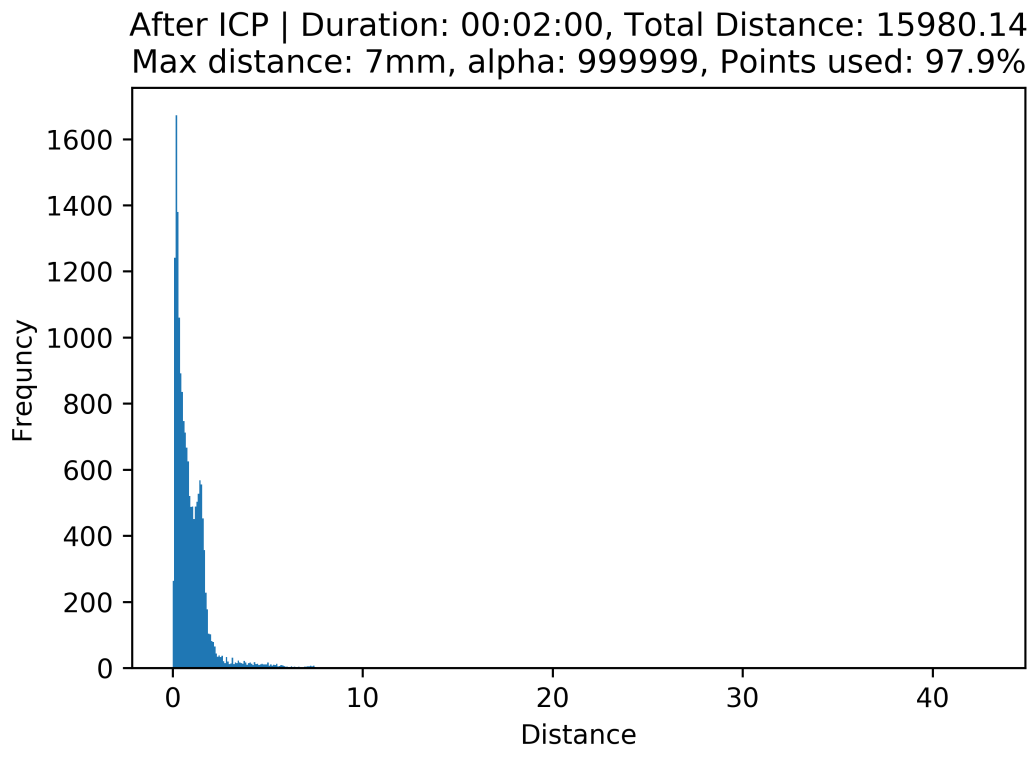


Figure 11: To be edited

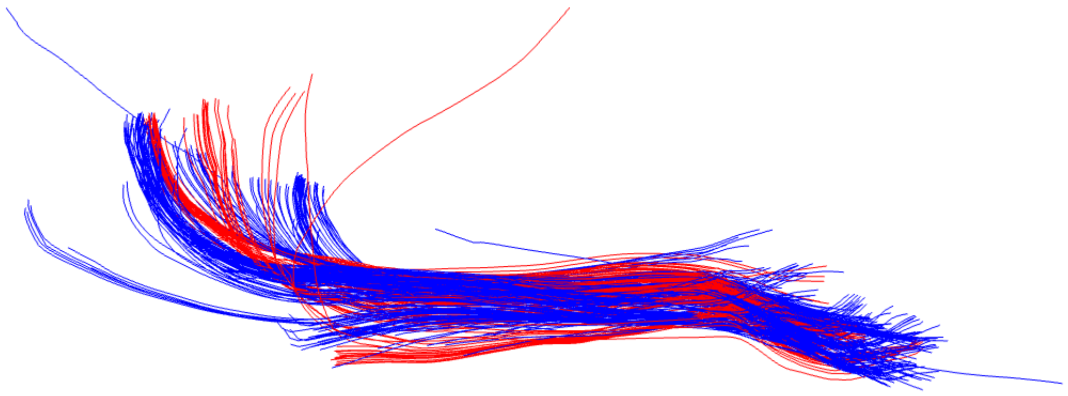


Figure 12: To be edited

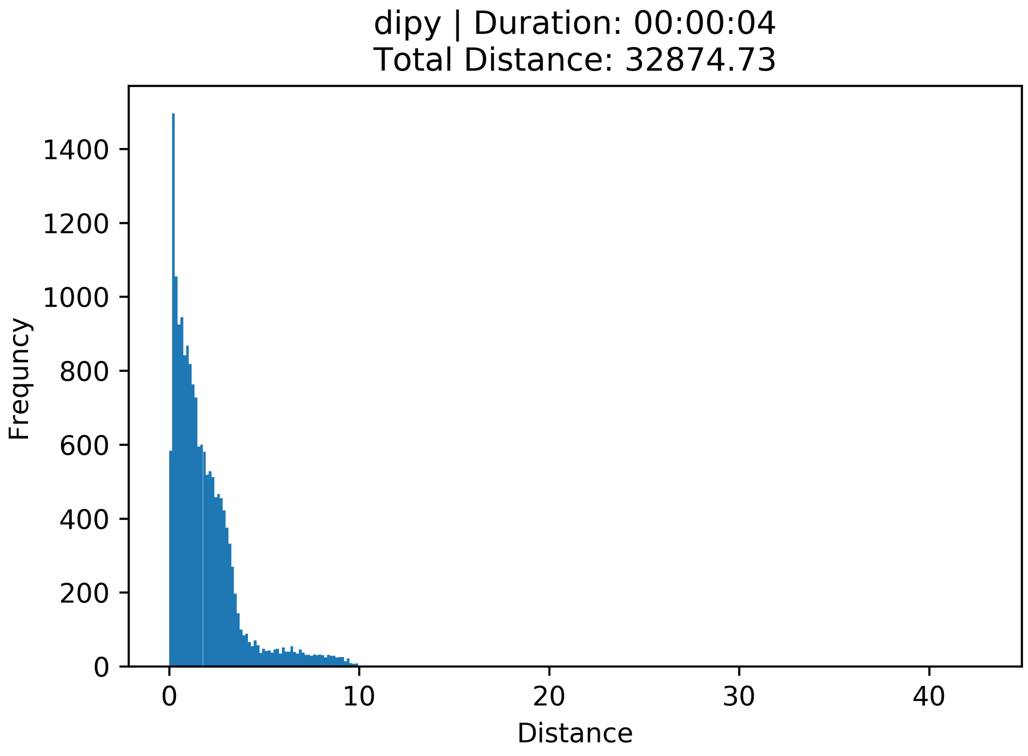


Figure 13: To be edited

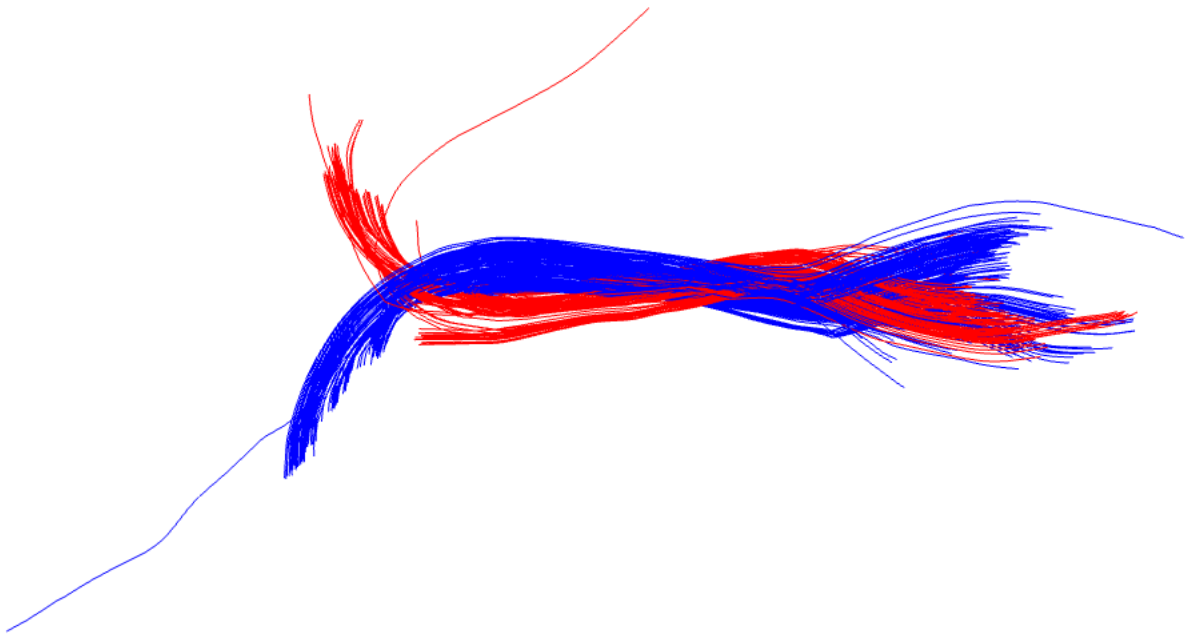


Figure 14: To be edited

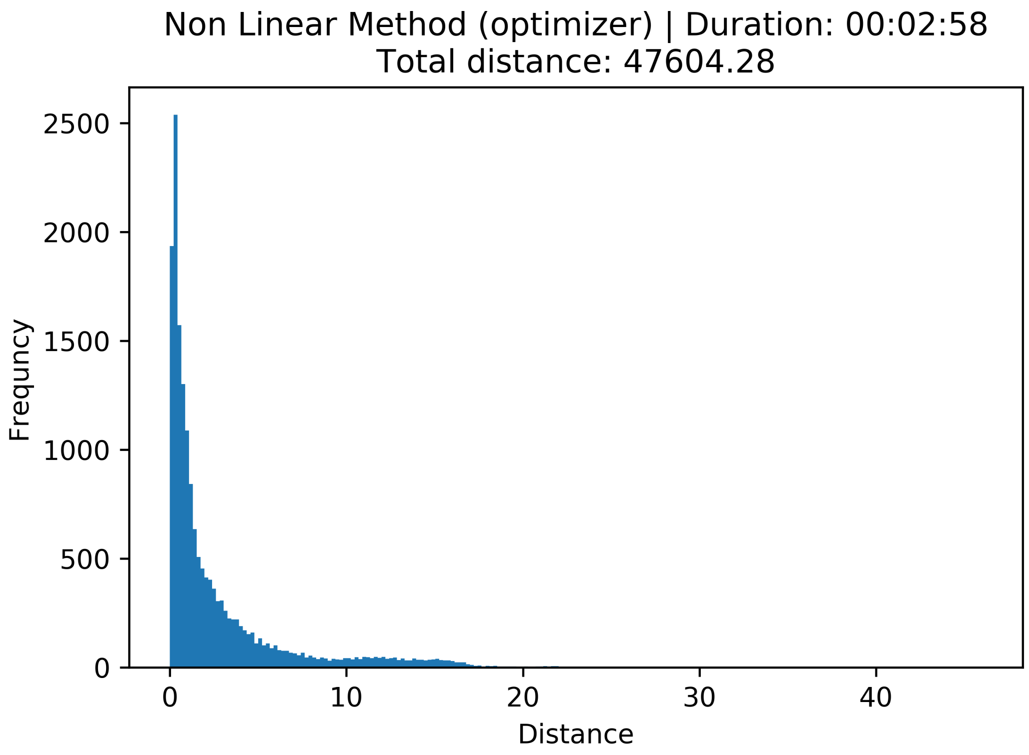


Figure 15: To be edited

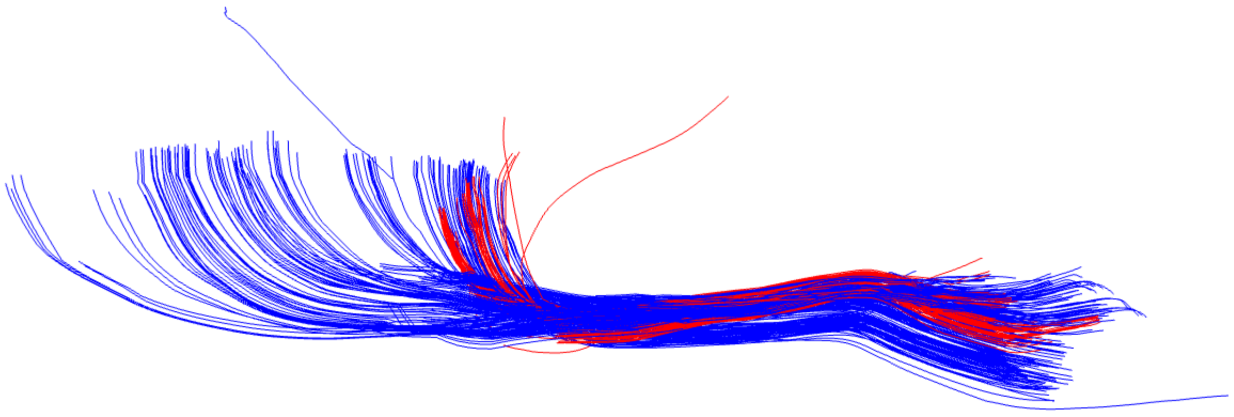


Figure 16: To be edited

4 CONCLUSION

BIBLIOGRAPHY

[1] Hongdong Li and Richard Hartley. The 3d-3d registration problem revisited. In *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007.

[2] Brian Amberg, Sami Romdhani, and Thomas Vetter. Optimal step nonrigid icp algorithms for surface registration. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[3] Wikipedia, the free encyclopedia. 3d tree, February 2006. [Online; accessed Feb 9, 2019].

[4] Wikipedia contributors. Kronecker product, February 2019. [Online; accessed 9-February-2019].

[5] Wikipedia, the free encyclopedia. Oriented graph with corresponding incidence matrix, 2010. [Online; accessed Feb 9, 2019].

[6] Wikipedia, the free encyclopedia. 2d affine transformation matrices, 2016. [Online; accessed Feb 10, 2019].

[7] Christopher C Paige and Michael A Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. [Online; accessed February 11th, 2019].