Project No 5: Point Set Registration

Nurefşan Altın
150210053
altinn21@itu.edu.tr

***Abstract*—This paper provides an implementation of the Kabsch-Umeyama algorithm and demonstrates its usage via visualized data**

***Index Terms*—Kabsch-Umeyama Algorithm, SVD, RMSD**

I. Introduction

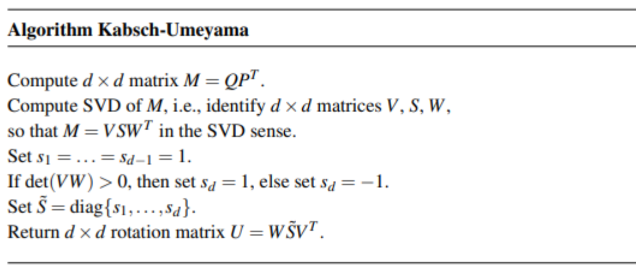
This paper focuses on how to construct and implement an effective algorithm to obtain a rotational matrix and a translation vector which alligns given 2 corresponding datasets optimally. The problem referred as “orthogonal Procrustes problem” is generally achieved by minimizing the root-mean-square deviation (RMSD), sum of the squared distances between corresponding data in two sets. This is an important problem that can be encountered in various applications. Some of those are Aligning spacecraft, obtaining correspondence of registration points in 3D model matching, matching structures in aerial and alignment of matched molecular and biochemical structures.

II. Implementation details

A. General information about the algorithm

The algorithm presented determines the best rotation by solving a least-squares problem, where orthogonality constraints are maintained using a Lagrange multiplier. Algorithm also involves calculating the eigenvalues and eigenvectors of a matrix in a singular value decomposition algorithm (SVD). Besides, special adjustments are made if there are reflections, since this situation may lead to improper rotation matrices.

B. How it is implemented?



Algorithm Kabsch-Umeyama

```
Compute $d \times d$ matrix $M = QP^T$.
Compute SVD of $M$, i.e., identify $d \times d$ matrices $V, S, W$,
so that $M = VSW^T$ in the SVD sense.
Set $s_1 = \ldots = s_{d-1} = 1$.
If $\det(VW) > 0$, then set $s_d = 1$, else set $s_d = -1$.
Set $\tilde{S} = \mathrm{diag}\{s_1, \ldots, s_d\}$.
Return $d \times d$ rotation matrix $U = W\tilde{S}V^T$.
```

Fig. 1. Kabsch-Umeyama Algorithm

Before the algorithm:

Before sending matrices P and Q to the algorithm kabsch-umeyama, datasets mat1 and mat2 should be updated according to correspondences list.

Inputs of the algorithm:

```
matrix_P: n x m matrix
matrix_Q: n x m matrix
```

Outputs of the algorithm:

```
R: Rotation matrix t: Translation vector
```

Firstly, centroids of the points are determined by calculating the mean of the points

```
centroid_P = np.mean(matrix_P, axis=0)
centroid_Q = np.mean(matrix_Q, axis=0)
```

Than, centroids are subtracted from columns of matrix_P and matrix_Q

```
mean_centered_P = matrix_P - centroid_P
mean_centered_Q = matrix_Q - centroid_Q
```

Before calculating matrix M, the transposes of matrices mean_centered_P and mean_centered_Q should be assigned to P and Q. Since a 3x3 rotation matrix is desired and given matrices are nx3, using their transposes will be more efficient and robust the algorithm.

```
P = mean_centered_P.T
Q = mean_centered_Q.T
```

Later, covariance matrix M is calculated with QPT . Singular value decomposition is computed to find optimal rotation between these points. From here, matrices V, S and W are constructed successfully.

```
M = Q @ (P.T)
V, S, W = svd(M)
```

S matrix is updated by replacing it with a diagonal matrix consists 1s in all of its diagonal entries.

```
S = np.eye(V.shape[0])
```

Since there might be an unwanted reflection case, there is a requirement to check the sign of the determinant of the multiplication (VW). This will help to detect and avoid any reflection. If (VW)’s determinant is negative, matrix S is updated so that its last diagonal element is -1. If it is not there is no need for any update.

```
if np.linalg.det(V@W)<0 :
    S [-1,-1] = -1
```

In the end, the rotation matrix R and translation vector t can be obtained by the result of the following multiplications:

```
R = (W.T) @ S @ (V.T)
t = centroid_P - R @ centroid_Q
```

After the algorithm:

After finding and saving R and t, an inverse rotation will be performed on mat2. RT and –t will be used for this computation.

In order to avoid dimension problems a matrix_t is constructed to use in this rotation.

```
matrix_t = np.tile(t[:,np.newaxis],
(1,matrix2.shape[0]))
```

Rotated matrix obtained by the following formula:

```
rotated_Q = (R.T @ (matrix2.T -
 matrix_t)).T
```

Now, two datasets mat1 and rotated_Q will be merged to recover full data. It is important to remove duplicates here; therefore, the correspondences list above used to check the overlapping points.

```
indices_duplicates = sorted(
 correspondences.astype(int)[:,1],
 reverse=False)
interleaved = []
i = 0
for row1, row2 in itertools.zip_longest(
rotated_Q, matrix1):
   if row1 is not None and i not in
   indices_duplicates:
       interleaved.append(row1)
   if row2 is not None:
       interleaved.append(row2)
   i +=1
mask = np.ones(len(interleaved),
 dtype=bool)
mask[indices_duplicates] = False
merged1=np.array(interleaved)[mask]
np.savetxt("merged.txt", interleaved)
```

III. Available data and experiments

The algorithm applied to the given data and the results are given below.

A. Plane Example

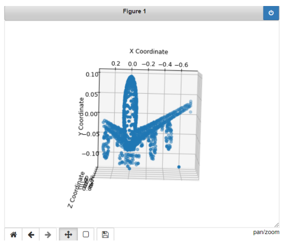


Fig. 2. Plane example: visualization of the data obtained after rotation

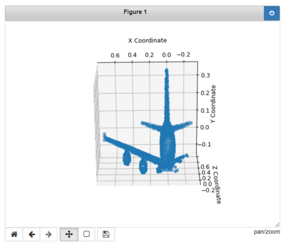


Fig. 3. Plane example: visualization of the data mat1

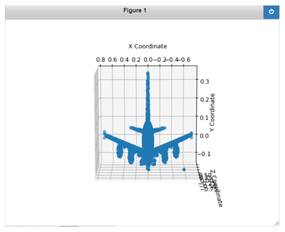


Fig. 4. Plane example: visualization of the data obtained after merging 2 sets

Rotation matrix:

$$\begin{pmatrix} 5.560262993955417121 \times 10^{-1} & 7.127843921584793296 \times 10^{-1} & -4.275195491083568222 \times 10^{-1} \\ 3.886942512888303947 \times 10^{-1} & 2.316589161819233300 \times 10^{-1} & 8.917672019483147139 \times 10^{-1} \\ 7.346789205726992344 \times 10^{-1} & -6.620215516405207890 \times 10^{-1} & -1.482455116800188988 \times 10^{-1} \end{pmatrix}$$

Translation vector:

$$\begin{pmatrix} 4.674446393667545330 \times 10^{0} \\ 2.953646508911610180 \times 10^{0} \\ 3.446977995480808410 \times 10^{0} \end{pmatrix}$$

B. Cup Example

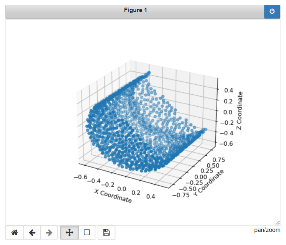


Fig. 5. Cup example: visualization of the data obtained after rotation

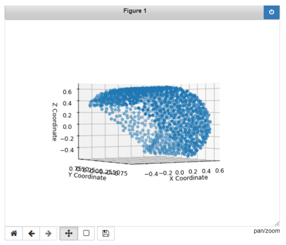


Fig. 6. Cup example: visualization of the data mat1

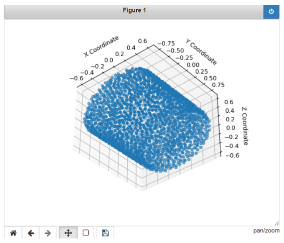


Fig. 7. Cup example: visualization of the data obtained after merging 2 sets

Rotation matrix:

$$\begin{pmatrix} -1.016113723318582129 \times 10^{-1} & -4.942767617590039242 \times 10^{-1} & -8.633420203131730863 \times 10^{-1} \\ -5.914822731301620090 \times 10^{-3} & 8.681222378807048612 \times 10^{-1} & -4.963218199942934228 \times 10^{-1} \\ 9.948067687707965456 \times 10^{-1} & -4.531647339655474338 \times 10^{-2} & -9.113464639065925466 \times 10^{-2} \end{pmatrix}$$

Translation vector:

$$\begin{pmatrix} 1.510827336883428407 \times 10^{0} \\ 1.892943471009337797 \times 10^{0} \\ 1.896294824763006925 \times 10^{0} \end{pmatrix}$$

C. Chair Example

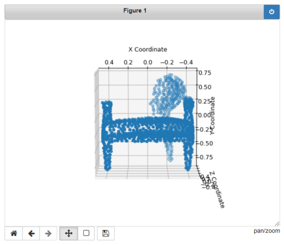


Fig. 8. Chair example: visualization of the data obtained after rotation

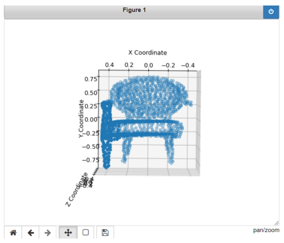


Fig. 9. Chair example: visualization of the data mat1

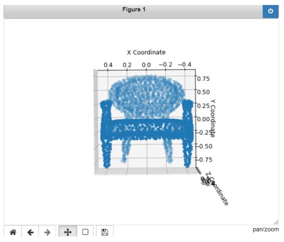


Fig. 10. Chair example: visualization of the data obtained after merging 2 sets

Rotation matrix:

$$\begin{pmatrix} 1.280179061139603280 \times 10^{-1} & -1.473679308931441223 \times 10^{-1} & -9.807623149149865549 \times 10^{-1} \\ -4.535255099826810410 \times 10^{-1} & -8.881443995735217589 \times 10^{-1} & 7.425415136143659334 \times 10^{-2} \\ -8.820013686823988674 \times 10^{-1} & 4.352942834256338100 \times 10^{-1} & -1.805334347624757974 \times 10^{-1} \end{pmatrix}$$

Translation vector:

$$\begin{pmatrix} 1.435318936110261845 \times 10^{0} \\ 1.420364300227839660 \times 10^{0} \\ 1.330339624587647762 \times 10^{0} \end{pmatrix}$$

D. Bottle Example

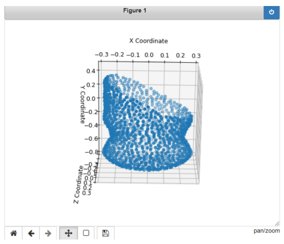


Fig. 11. Bottle example: visualization of the data obtained after rotation

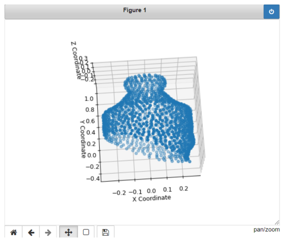


Fig. 12. Bottle example: visualization of the data mat1

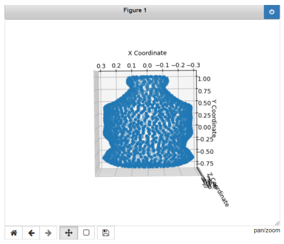


Fig. 13. Bottle example: visualization of the data obtained after merging 2 sets

Rotation matrix:

$$\begin{pmatrix} -7.468569661044899277 \times 10^{-1} & -3.886820098630694997 \times 10^{-1} & 5.395654693241300759 \times 10^{-1} \\ -5.664618683368113361 \times 10^{-1} & -5.314821328769757380 \times 10^{-2} & -8.223734391915404895 \times 10^{-1} \\ 3.483182091393348667 \times 10^{-1} & -9.198377527522879626 \times 10^{-1} & -1.804747936793187346 \times 10^{-1} \end{pmatrix}$$

Translation vector:

$$\begin{pmatrix} 1.530861734766091553 \times 10^{0} \\ 1.232733409325580842 \times 10^{0} \\ 1.011400513255998224 \times 10^{0} \end{pmatrix}$$

Appendix - SVD

Singular value decomposition is a matrix decomposition that factorizes a matrix A into 3 matrices V,S and W. They are defined in below:

```
V: left singular matrix, orthogonal matrix
W: right singular matrix, orthogonal matrix
S: diagonal matrix with singular values in
its diagonal
A = V @ S @ W.T
```

SVD can be calculated by utilizing a powerful algorithm called 'power method'. This method is actually used for computing the largest eigenvalue of a matrix A and its corresponding eigenvector. The algorithm utilizes a random unit vector x and updates this vector iteratively. This vector is updated according to the formula:

```
 Xn+1 = AXn/ norm of (AXn)
```

The procedure continues until the result become less than the error value(1e-10 used here). It is important that this method assumes A has a dominant eigenvalue and it is much more greater than its other eigenvalues. After finding the dominant eigenvalue and its vector, other eigenvalues can be determined by eliminating this eigenvalue from the matrix and repeating the steps for further findings. Since we started with a random unit vector and try to find the eigenvalues iteratively by checking a threshold, this method will have an error possibility and may not give the exact solution precisely for this problem. However, even with this possibility it has high accuracy and error rates are considerably low.

References

[1] Liu P, Agrafiotis DK, Theobald DL. Fast determination of the optimal rotational matrix for macromolecular superpositions. J Comput Chem. 2010 May;31(7):1561-3. doi: 10.1002/jcc.21439. PMID: 20017124; PMCID: PMC2958452.

[2] Hanson, A. J. (2020). Acta Cryst. A76, 432-457.

[3] Jim Lawrence, Javier Bernal, and Christoph Witzgall. A purely algebraic justification of the kabsch-umeyama algorithm. Journal of Research of the National Institute of Standards and Technology, 124, October 2019.

[4] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on Pattern Analysis &Machine Intelligence, 13(04):376–380, 1991.