Symmetry and Orbit Detection via Lie-Algebra Voting

Tong Zhao

March 17, 2019

Contents

1 Introduction **2**

2 Algorithm **2**

2.1 Pipeline 2

2.2 Sampling 3

2.3 Local Feature Computation 3

2.4 Point Pairing 3

2.5 Clustering 3

2.6 Pattern Patching 4

3 Implementation **5**

3.1 2D Implementation 5

3.2 3D Implementation 8

4 Experiment **9**

4.1 Parameter Study 9

4.2 Some Examples 10

5 Perspective **11**

5.1 Advantages 11

5.2 Disadvantages 11

5.3 Future works 12

6 Conclusion **12**

1 Introduction

With the huge improvement of 3D modeling techniques, the analysis of raw point clouds draws more and more attention since it allows us to better understand the intrinsic properties and structures of the data. While many local properties - such as the linearity, the verticality and so on - are well studied, the global descriptors - such as the symmetry, the orbit and so on - remain a huge challenge.

In this project, we focus on the detection of the symmetry and the orbit, which exist everywhere in our life, both in man-made objects and in natural objects. They help us simplify the structure of the point cloud and make the model physically balanced, thus they have many applications in various fields, like architecture design, biomedical analysis and physical modelling.

Although it is obvious for a man to detect them from a point cloud, it is much more complicated for a computer. The point cloud is an unorganized collection sampled from an object with noise added, which demands a large calculating quantity. What's more, we have neither information about the conneticity nor the correspondance, which makes geometric features useless. Thus an efficient algorithm needs to extract pointwise local features and deduces the symmetries and orbits.

The goal of this project is as following:

- Study the article: Symmetry and Orbit Detection via Lie-Algebra Voting [7].
- Adapt the algorithm to 2D point clouds and implement it in Python, to facilitate the visualization and parameter tuning.
- Implement the algorithm on 3D point clouds in C++, which allows a fast computation on big point clouds.
- Study the influence of different parameters and analyze the algorithm.

2 Algorithm

In order to detect symmetries and orbits automatically from a raw point cloud, the proposed algorithm is designed based on a voting method. They can be considered as the invariance under a set of transformations, which is a pairwise local property. Thus an efficient algorithm needs to extract pairwise local features and deduce the global symmetries and orbits from them. In the article, they adopted the pipeline proposed in [6] and a Lie-algebra based voting method is designed to improve it.

2.1 Pipeline

The whole algorithm can be splited into five steps, namely subset sampling, local features computation, point pairing, clustering and pattern patching. The reprinted figure 1 shows the pipeline:

The most significant contribution of this article is the embedding method of transformations. Each step will be detailed in following paragraphs.

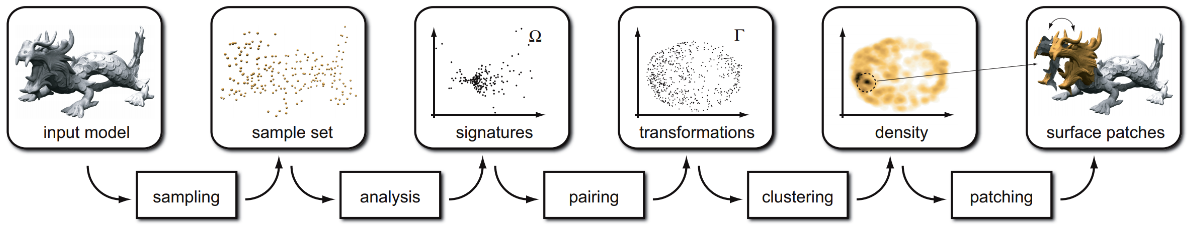


Figure 1: Detection Pipeline. (reprinted from [6])

2.2 Sampling

The size of the candidate point set has a big influence on the complexity of the algorithm. A large point set leads to a high computational cost in quadratic order during the pairing phase. The points set P are required to be sampled uniformly from the raw point cloud P_0 to avoid causing potential bias when we compute alignment errors in pairing phase.

2.3 Local Feature Computation

Local features, namely the oriented normals and the principal curvatures, are then computed for each point in P. Then point pruning is performed to eliminate degenerate points which may cause problems in the pairing phase. This case only exists in 3D point clouds, where the two principal curvatures $k_1 = k_2$. To avoid any ambiguity, we discard all the points whose principal curvatures $|k_1/k_2| < \gamma$ and thus get a new point set with features P'.

2.4 Point Pairing

Given P', we choose randomly a subset $\tilde{P}$ with $|\tilde{P}| \approx |P'|/5$. For each point $p_i \in \tilde{P}$, each point $p_j \in P'$, a transformation is estimated according to their local features. In a 2D point set, the normal together with the curvature direction decide the rotation matrix of the local frame, and the ratio between two curvatures decides the scaling coefficient. In a 3D point set, the normal with two principal curvatures decide the local frame and the scaling coefficient is decided by $\big(k_{i,1}/k_{j,1} + k_{i,2}/k_{j,2}\big)/2$.

2.5 Clustering

Embedding Unlike [6], whose authors embed the transformations in a $\mathbb{R}^7$ euclidean space, the authors of article [7] embed at first the transformations in SIM(3) and then convert it to the corresponding Lie algebra $\mathfrak{sim}(3)$. SIM(3) is a Lie group which contains rotation, translation and uniform scaling. The matrix logarithm maps bijectively each element in SIM(3) to an element in $\mathfrak{sim}(3)$. It can be written in the format:

$$T = \begin{pmatrix} R & t \\ 0 & w^{-1} \end{pmatrix} \rightarrow \log(T) = \begin{pmatrix} \omega & u \\ 0 & -s \end{pmatrix}$$

where $R \in \mathbb{R}^{d\times d}$ the rotation matrix, $t \in \mathbb{R}^d$ the translation vector, w the scaling coefficient, $\omega \in \mathbb{R}^{d\times d}$ the infinitesimal rotation in tangent space, $u \in \mathbb{R}^d$ the infinitesimal translation in tangent space and s the infinitesimal scale factor. What's more, ω can be written as a $d \times d$ skew-symmetric matrix whose degree of liberty is 1 when $d = 2$ and is 3 when $d = 3$. Thus the final embedding for each transformation is a vector $v = \big(\omega_1, \cdots, \omega_d, u_1, \cdots, u_d, s\big)$.

Distance Metric A well defined distance metric is the basis of following algorithms. A naïve choice is a weighted euclidean norm of the difference of two transformations. For any pair of transformations F and G, we have the difference is $\delta = FG^{-1}$ and thus $log(\delta) = log(F) - log(G)$, then we have the following notion:

$$d^2(F,G) = log(\delta)^t \begin{pmatrix} \alpha Id & 0 & 0 \\ 0 & \beta Id & 0 \\ 0 & 0 & \gamma \end{pmatrix} log(\delta) = \|log(\delta)\|_E^2 = \|log(F) - log(G)\|_E^2$$

However, this notion is not stable enough in terms of its dependence on the location in Euclidean space or the scaling. An variational invariance is proposed to deal with the problem. This distance is not studied in this project but one can prove that the optimal distance is coincidently equivalent to the previous distance notion in 2D point sets.

Symmetry Detection Corresponding points on two symmetric patches should have similar transformations thus the pairs should be centered closely in one region. Mean shift algorithm [3] is then used to find the clustering center. Mean shift is fairly robust to initializations and it converges fast in practice. After finding the nearest transformation to the center in $\mathfrak{sim}(d)$, we apply a region growing algorithm on both sides to extract the symmetric patches.

Orbit Detection One of the benefits of $\mathfrak{sim}(3)$ is that the orbits are converted to linear structures in the embedding space. Given a group of transformations $\{G_1, \cdots, G_k\}$, if they generate a common orbit, we have:

$$log(T) = log(\Pi_i G_i^{r_i}) = \sum_i r_i log(G_i)$$

Thus RANSAC [4] is a good choice to find the orbits. And then region growing algorithm is applied again to find all patches.

2.6 Pattern Patching

Once the candidates in embedding space are detected, the symmetric patches and orbits can be found in SIM(3). For the symmetry detection, the two points of the transformation which is nearest to the clustering center are selected, and we perform a Region Growing algorithm [1] on two points, respectively. In terms of the orbits, all the transformations on the linear sub-space are chosen and the Region Growing algorithm is performed on each corresponding point in $\mathbb{R}^d$ space.

3 Implementation

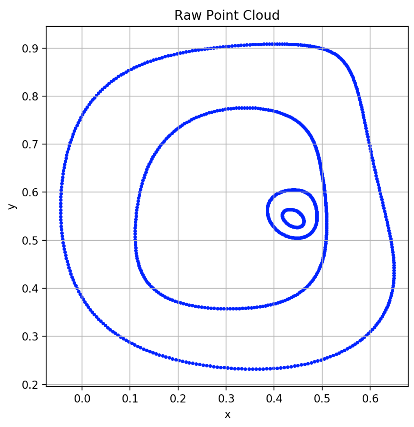
The codes are implemented in both Python and C++ for different purposes.

I adapted the described algorithm to 2D point cloud and implemented it in Python from scratch. Only Numpy and basic functions from Sklearn (i.e. KDTree, pairwise_distance) are used to accelerate the algorithm. The purpose is to understand the algorithm and to give a good visualization for each step.

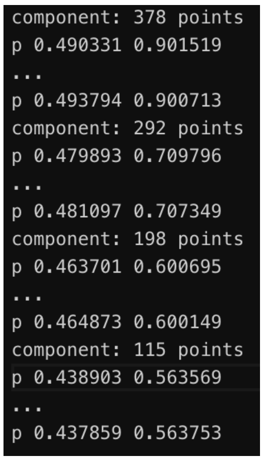
The 3D version is implemented in C++, with Eigen [5] and CGAL [8] library. It is a more general version, which enables to deal with raw point clouds in a high speed.

3.1 2D Implementation

I/O The format of the input file is *pslg*, whose points are written in order for each component. It allows us to find the connectivity which will be used in local feature computation. The function *read_pslg* in *utils.py* takes as input the file and converts it to a numpy array.



(a) Raw Point Cloud in 2D (4 components)



```
component: 378 points
p 0.490331 0.901519
...
p 0.493794 0.900713
component: 292 points
p 0.479893 0.709796
...
p 0.481097 0.707349
component: 198 points
p 0.463701 0.600695
...
p 0.464873 0.600149
component: 115 points
p 0.438903 0.563569
...
p 0.437859 0.563753
```

(b) Input pslg file

Local Features In this step we intend to find local features for each point.

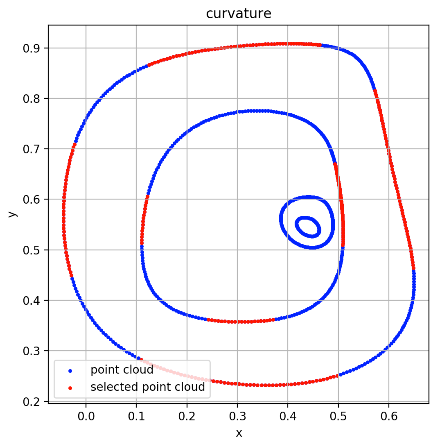
The unoriented normals are computed by a local PCA. For a point p_i, we take all points in a centered ball whose radius is r and then calculate its eigenvectors. The eigenvector corresponding to the largest eigenvalue is chosen as the normal vector n_i.

In order to orient all normals, we compute the inner product between the normal vector and the vector $p_i^r - p_i^l$, where p_i^r is the right neighbor of p_i and p_i^l the left one. The normal vector is inversed if the inner product is negative.

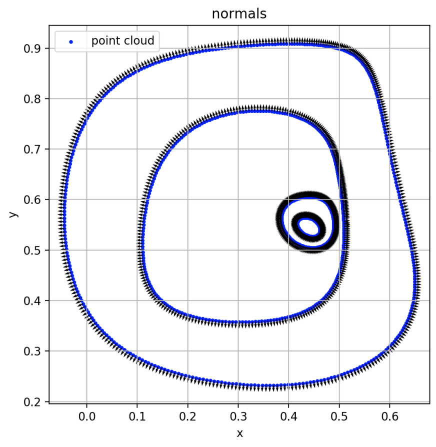
For a curve given parametrically in Cartesian coordinates as $(x(t), y(t))$, the curvature is computed as following:

$$k = \frac{|x'y'' - y'x''|}{(x'^2 + y'^2)^{3/2}}$$

With the help of given connectivity, we are able to estimate the gradients by using first order Taylor expansion and thus we get the curvature k_i for each point p_i.



(a) Points whose curvatures are larger than 3



(b) Estimated normals

Pairing From the whole point cloud, we choose two subsets $\tilde{P}$ and $\tilde{P}'$ where $\tilde{P}$ is sampled uniformly from P and $\tilde{P}'$ is randomly sampled from $\tilde{P}$. For each pair of points (p_i, p_j) where $p_i \in \tilde{P}'$ and $p_j \in \tilde{P}$, we estimate a transformation T_{ij}. The neighbors of p_i within a ball of radius r are then transformed by applying T_{ij}. The alignment error is calculated by:

$$\sum_{p_l \in n_i} \min_{p_k \in n_j} \|p_k - p_l\|$$

where n_i is the neighbors of p_i and n_j is the neighbors of p_j. Any transformation whose alignment error is larger than a give threshold is rejected and the others are saved. A dictionary is created in this step to keep track of the relation between the index of the transformation and the indices of the pair of points.

Clustering An element in SIM(2) and the corresponding one in $\mathfrak{sim}(2)$ has following relationship:

$$T = \begin{pmatrix} \cos\theta & -\sin\theta & v_1 \\ \sin\theta & \cos\theta & v_2 \\ 0 & 0 & s \end{pmatrix} \rightarrow \log(T) = \begin{pmatrix} 0 & -\theta & x_1 \\ \theta & 0 & x_2 \\ 0 & 0 & \lambda \end{pmatrix}$$

The logarithm map is then:

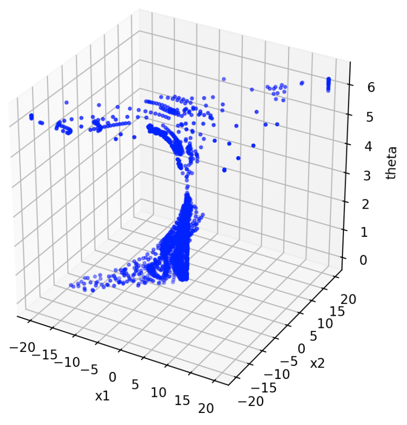
$$\begin{cases} \lambda = \log s \\ x_1 = \dfrac{\theta \sin\theta}{2(1-\cos\theta)} v_1 + \dfrac{\theta}{2} v_2 \\ x_2 = \dfrac{\theta \sin\theta}{2(1-\cos\theta)} v_2 - \dfrac{\theta}{2} v_1 \end{cases}$$

And the exponential map is:

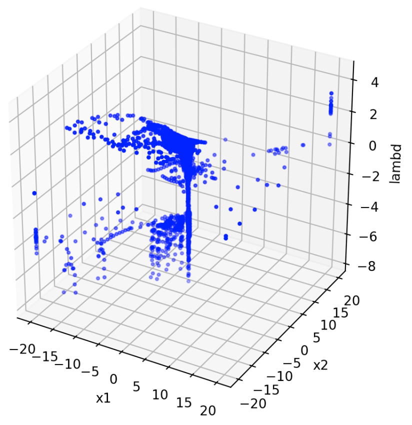
$$\begin{cases} s = \exp\lambda \\ v_1 = \dfrac{\sin\theta}{\theta} x_1 - \dfrac{1-\cos\theta}{\theta} x_2 \\ v_2 = \dfrac{\sin\theta}{\theta} x_2 + \dfrac{1-\cos\theta}{\theta} x_1 \end{cases}$$

Now we can represent all elements in $\mathfrak{sim}(2)$ by vectors $v \in \mathbb{R}^4$, i.e. $(x_1, x_2, \theta, \lambda)$.

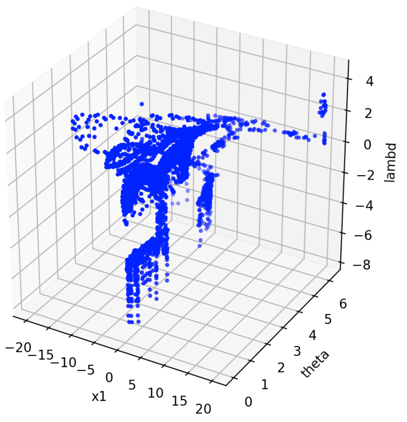
The following four figures show the distributions of transformations in $\mathfrak{sim}(2)$ along each three axes.



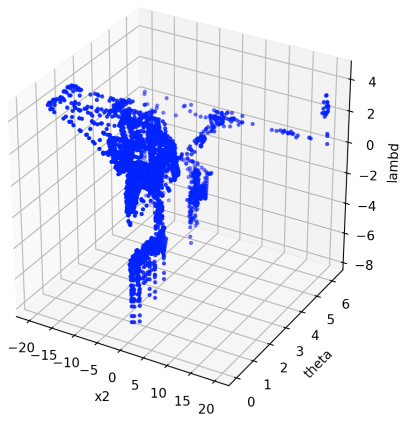
(a) (x_1, x_2, θ)



(b) (x_1, x_2, λ)



(c) (x_1, θ, λ)



(d) (x_2, θ, λ)

RANSAC A k-dimensional linear space generates a k-orbit in point cloud. Using the distance metric $\|\cdot\|_E$, we describe the algorithm as following:

Algorithm 1 RANSAC

```
Require: Set $T \subset \mathfrak{sim}(2)$, dimension $k$, distance coefficients $\alpha$, $\beta$, $\gamma$, threshold $s$, number of draws $n$
1: best_generator ← None
2: best_score ← None
3: for $i \in \{1, ..., n\}$ do
4:     choose randomly k points in $T$ as generators
5:     estimate how many points in $T$ are in the subspace ($dist < s$)
6:     if score > best_score then
7:         update best_generator and best_score
8:     end if
9: end for
```

Mean Shift To locate the mode of the set T, we perform a Mean shift algorithm with the help of a guassian kernel. The algorithm is described as below:

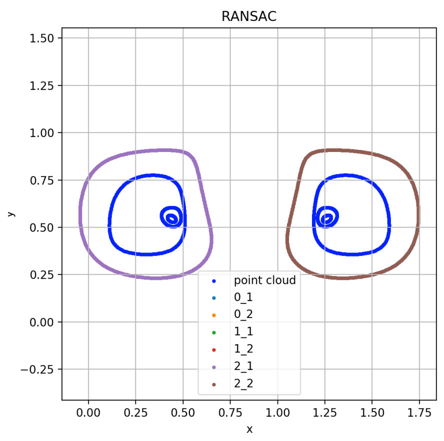
Algorithm 2 Mean Shift Algorithm

```
Require: Set $T \subset \mathfrak{sim}(2)$, kernel variance $\sigma$, distance coefficients $\alpha$, $\beta$, $\gamma$, threshold $s$, maximum iterator $n$
1: choose randomly a point in $T$ as current center
2: for $i \in \{1, ..., n\}$ do
3:     calculate the distance from the center to all other points
4:     apply a gaussian kernel on the distance vector
5:     update the center
6:     if $\|c_{new} - c\| \leq s$ then
7:         break
8:     end if
9: end for
```

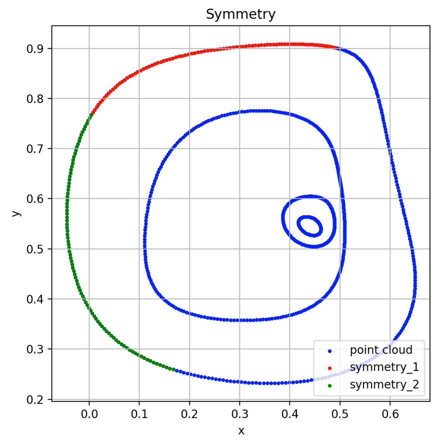
Region Growing Given a transformation T from p_i to p_j, the region growing algorithm helps us to find the two patches corresponding to p_i and p_j, respectively. The algorithm is described as below:

Algorithm 3 Region Growing Algorithm

```
Require: Point set $\tilde{P}$, $p_i \in \tilde{P}$, $p_j \in \tilde{P}$, distance coefficients $\alpha$, $\beta$, $\gamma$, threshold $s$
1: Initialize two sets $n_i = \{p_i\}$, $n_j = \{p_j\}$
2: while alignment error between $n_i$ and $n_j$ is smaller than $s$ do
3:     add the left neighbor and right neighbor of $n_i$ in $n_i$
4:     add the left neighbor and right neighbor of $n_j$ in $n_j$
5:     if there are repeated points in $n_i$ and $n_j$ then
6:         break
7:     end if
8: end while
9: return $n_i$ and $n_j$
```



(a) RANSAC



(b) Mean Shift

3.2 3D Implementation

The C++ version is robust and efficient, and it can deal with more general cases. CGAL is used to calculate local features and Eigen is used to perform matrix operations. Here we only explain the parts which are different from the 2D version.

I/O The algorithm takes as input a *xyz* file and save it in a std::vector<Point>.

Local Features The normals and curvatures are computed by the CGAL component: Estimation of Local Differential Properties of Point-Sampled Surfaces. It estimates the differential properties up to order 2 for every point in the point cloud, which indicate the two principal directions and the corresponding curvature values. Then the normals are computed by taking the cross product of two principal curvature vectors.

Clustering An elements in SIM(3) and the corresponding one in $\mathfrak{sim}(3)$ has following relationship [2]:

$$T = \begin{pmatrix} R_{11} & R_{12} & R_{13} & v_1 \\ R_{21} & R_{22} & R_{23} & v_2 \\ R_{31} & R_{32} & R_{33} & v_3 \\ 0 & 0 & 0 & s \end{pmatrix} \rightarrow \log(T) = \begin{pmatrix} 0 & -\omega_3 & \omega_2 & x_1 \\ \omega_3 & 0 & -\omega_1 & x_2 \\ -\omega_2 & \omega_1 & 0 & x_3 \\ 0 & 0 & 0 & \lambda \end{pmatrix}$$

with the logarithm map being:

$$\begin{cases} \lambda = \log s \\ \theta = \arccos\left(\frac{tr(R)-1}{2}\right) \\ [\omega]_\times = \dfrac{(R-R^T)\theta}{\sin\theta} \\ V = I_3 + \dfrac{1-\cos\theta}{\theta^2}[\omega]_\times + \dfrac{\theta - \sin\theta}{\theta^3}[\omega]_\times^2 \\ (x_1, x_2, x_3)^T = V^{-1}(v_1, v_2, v_3)^T \end{cases}$$

4 Experiment

4.1 Parameter Study

Radius r This parameter decides the quality of the estimation of local features. It is influenced by the sampling method and the spacing of the point cloud. Thus it should be chosen carefully for each point cloud. One solution to have a data-adaptive r is to calculate the average spacing between p_i and its 6 nearest neighbors of the point cloud, noted d_i, and then take $r_i = 2d_i$.

Number of points n The computational cost of the following algorithm grows quadratically with n thus a proper choose is demanding. Normally it should be between 100 and 500.

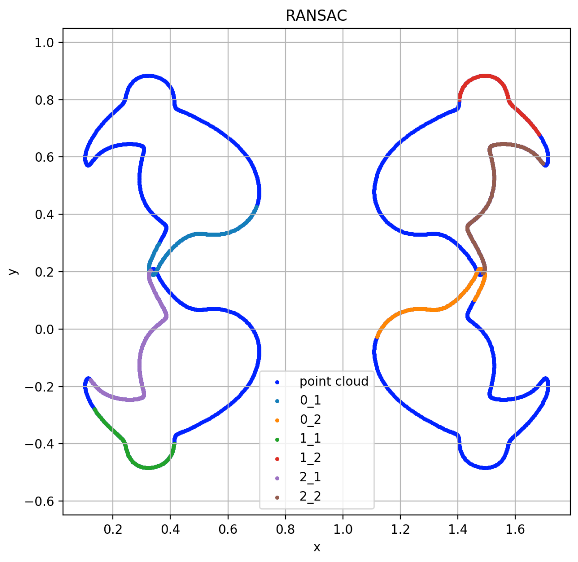
Threshold for pairing s_1 This parameter is used to eliminate transformations with high alignment error. It improves the result in the existence of noise. In the experiment, it is fixed to be 0.0005.

Threshold for RANSAC s_2 This parameter is used when we calculate the score for a subspace. All points whose distance from the subspace are less than s_2 are counted. Unlike performing RANSAC in natural euclidean space, the choice of s_2 is not evident. The algorithm is not too sensitive to s_2 thus one may fix it to 0.005.

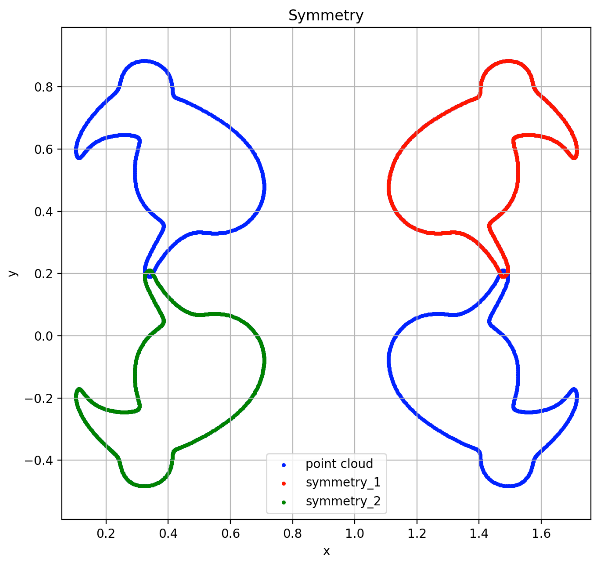
Variance of gaussian kernel σ In the Mean-shift algorithm, σ is used to decide how smooth the gram matrix is. The experiments show that σ has a significant impact on the algorithm and its choice is not evident. Thus the article proposed one way to find a data-adaptive σ. We estimate the average pairwise distance $\overline{d}$ by Monte-Carlo method and then σ is chosen in the range $[0.05\overline{d}, 0.1\overline{d}]$

4.2 Some Examples

Bird From the results we can observe that all the three pairs of patches found by RANSAC form an orbit respectively, which is exactly what we expect. In term of the symmetry result, Mean shift also finds one of the significant pairs.



(a) Bird - RANSAC



(b) Bird - Mean Shift

Blob The RANSAC finds three pairs of orbits in same region thus the other two one are covered by the one shown in the figure. The Mean shift finds the reflection perfectly.

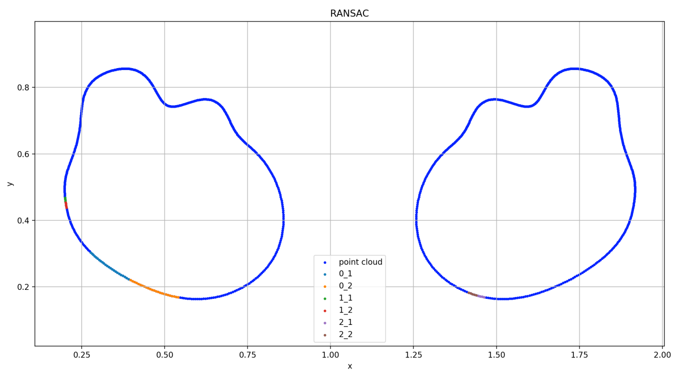


Figure 7: Blob - RANSAC

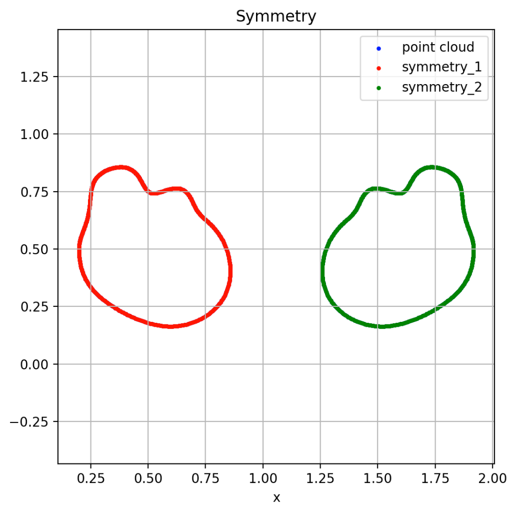


Figure 8: Blob - RANSAC

5 Perspective

In this section we analyse the advantages and the disadvantages of the algorithm.

5.1 Advantages

Unified framework The algorithm proposed an unified framework to detect both local and global symmetries and orbits. Compared to other voting-based methods, they avoid mapping the transformations to humain-designed spaces based on which symmetries are looked for. Thus it simplifies the detection step.

Coordinate frame independent clustering The algorithm proposed to embed all transformations in a Lie algebra space and they invent a coordiante frame independent metric to compute the distances. It helps to reduce manual tuning everytime when we apply it on a new point cloud.

5.2 Disadvantages

High computational cost Given that the transformations are computed pairwisely, the computation cost grows quickly with the number of the points. In the article, 250 points uniformly sampled from the raw point cloud are used. However we should realize that it is hard to get an uniform sampling, what's more, 250 points are far from enough to represent a complicated point cloud. As a result, local symmetries may be discarded.

Parameter tuning The parameters have a great influence on the final result of the algorithm. However the choice is not evident, especially some of them are related. A good visualization tool can help us to make a better decision, but it is still time consuming.

Lackness of capacity in complicated scene The estimated transformations computed by a pair of points are not stable especially in existence of noise. When the scene is complicated, the estimations are biased, which will cause the failure of the algorithm.

Subjective evaluation criteria For symmetry detecting tasks, we don't have a common criteria to evaluate if the result is good or not. Till now, most of the works evaluate their results by visualization, which is not convincing.

Limited sampling method The reduced point cloud $\tilde{P}$ is supposed to be uniformly sampled from the whole point cloud, which may not be feasible in some cases.

5.3 Future works

Transformations based on segmentation In stead of estimate transformations based on pairs of points, we can perform at first a segmentation on the point cloud. Based on the obtained super-points, we can estimate transformations and perform following algorithms. This operation will save computational energy and give a more reliable result.

Deep Learning Methods With the big success of deep learning methods used in 3D vision, one can also try to combine the proposed algorithm with deep neural networks. For instance, instead of using the transformations, we can learn the features for all pairs of points and predict if there exists a symmetry. However a big dataset is demanding.

6 Conclusion

In this project we studied the article [7]. Most of the proposed algorithms are analysed and implemented in Python and C++. While the Python version allows us visualize all steps and perform the algorithm on 2D point cloud, the C++ version allows us perform the algorithm on a large 3D point cloud. In general, the algorithm is well established but it is still far from being used in industry in terms of the capacity and speed.

References

[1] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on pattern analysis and machine intelligence*, 16(6):641–647, 1994.

[2] Jose-Luis Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3, 2010.

[3] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

[4] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[5] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[6] Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 560–568. ACM, 2006.

[7] Zeyun Shi, Pierre Alliez, Mathieu Desbrun, Hujun Bao, and Jin Huang. Symmetry and orbit detection via lie-algebra voting. In *Computer Graphics Forum*, volume 35, pages 217–227. Wiley Online Library, 2016.

[8] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.13 edition, 2018.