IM700 Master Thesis
Unsupervised Identification of the rigid parts of an unknown articulated object

Anna M. Maureder

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Februar 2017

Declaration

Hagenberg, February 28, 2017

Anna M. Maureder

Contents

Preface

Abstract

The proposed work addresses the issue of identifying the rigid parts of an unknown articulated object. Most existing pose estimation methods take advantage of user inputs to estimate the joint positions, e.g. trackers and markers. However, not many completely unsupervised methods exist. A main solution for this situation is proposed by template matching associated with the non-rigid registration of meshes. First, state-of-the-art methods related to the non-rigid-registration are proposed. Subsequently, an approach is developed and tested in 2D as well as in 3D. It is based on the divide and conquer principle to reduce the number of computation steps of iteratively detecting rigid parts in two point clouds, describing the same object in different poses.

Kurzfassung

An dieser Stelle steht eine Zusammenfassung der Arbeit, Umfang max. 1 Seite. ...

Chapter 1

Introduction

Pose and motion estimation of articulated objects that consist of rigid parts linked by joints is an active field of research due to the growing digitalization of day-to-day processes. A case of application the thesis emerged from is the pose capture an object. Thereby, it is essential to know the corresponding rigid parts and joints in two consecutive frames, in order to interpolate the animation between the moved parts.

1.1 Problem statement

A vast majority of existing pose estimation approaches take advantage of labeling for joints and the rigid parts to state basic information about the object to be captured. Often combined with a machine learning approach the results are promising with growing learning phase. However, unsupervised methods that detect the pose of an unknown object, constitute a great potential as the detection of a pose operates completely independent from user input. Among those methods, the non-rigid registration (see section 2.3) is a well-known approach. The input of this algorithm are two or more poses of one articulated object. By merging one *template* pose onto another *query* pose, part correspondences can be made and the articulated object is segmented into its rigid parts. By applying this approach to an unknown input object in two poses, some core questions need to be considered:

- How and in which form is the input data received?
- Which points correspond to each other in two different poses?
- What are the rigid parts of the articulated object?
- Where are the joints linking those rigid parts?
- Which joints/rigid parts correspond to each other in two different poses?

Many challenges have to be overcome emerging from different stages of the pose capture procedure (see section **??**). To name the most crucial ones, digital input data of an articulated object in the real worlds has to be captured. Thereby, input noise depending on the resolution and capture form is a important factor for an successful pose estimation. Furthermore, the ambiguity of body parts is one main problem especially if the articulated object has symmetric body parts which is the case for most living beings. One of the main drawbacks of current methods is the computational expensive procedure to detect rigid parts. As this directly influences the run time there is a great demand for improvements in this area, especially if real-time applications are required.

1.2 Goal

By means of current approaches in this particular field (see chapter 2), my thesis addresses the issue to detect the initial pose of an unknown articulated object given in two poses. Assuming, a 3D or 2D reconstructed model in form of a point cloud is already available, the thesis fully focuses on the segmentation of an articulated object into its rigid parts. The scanning and reconstruction step is therefore passed over. The main goal is then to segment an articulated object M into its unknown number n of rigid parts $\mathcal{P} = \{P_1, \ldots, P_n\}$ and extract all joints m $\mathcal{J} = \{J_1, \ldots, J_m\}$ linking those parts in form of a skeleton structure. The input poses are constituted by two point clouds C_1 and C_2 of M in two different poses. C_1 is thereby used as a *template* to be registered with C_2. The main task is to determine a part assignment P_i and the corresponding transformation T_i for all points of the *template* that aligns them with all points of C_2. The main difficulty is that only a number of unsorted points of C_1 and C_2 are present, no further information, like labeling by the user or PCA (principal component analysis) related variables are available. The only assumption that can be made is that M only consists of rigid parts that can not be deformed or stretched (e.g. the rigid parts of a human) and are linked by joints. Comparing two poses being adopted by the articulated object, the geodesic distance $g(\boldsymbol{p}_i, \boldsymbol{p}_j)$ between two mesh points $\boldsymbol{p}_i(x, y)$ and $\boldsymbol{p}_j(x, y)$ remains constant. Thereby, it is taken advantage of the knowledge that points located on a rigid part P_i have the same transformation T_i .

1.3 Methodology

To answer the questions posed in the beginning, two segmentation approaches are implemented being inspired by State of the Art approaches. The first approach is quite straightforward and linear (see chapter 3) which aims to reduce the computation steps of previous approaches. This is reached by iteratively subdividing C_1 and C_2 with a “divide and conquer” approach. Assumed corresponding sub clusters are verified to match and in this case those clusters are not subdivided any further. This attempt does only depend on the point coordinates of C_1 and C_2 and the orientation of the clusters being compared. In case of many linked parts or too dissimilar transformations between C_1 and C_2 this approach is not reliable as clusters being compared do not correspond to each other. To address the segmentation with focus on articulated objects with a typical skeleton structure (e.g. a human), a feature-based approach (4) is implemented which offers more information about the points of C_1 and C_2 apart from their coordinates. Those support for the initial alignment of the input clusters to find a reliable corresponding rigid part. Proceeding from there, joints can be estimated and all linked rigid parts are detected iteratively. The reference paper for this approach is from Mitra et. al [13]. By showing the outcomes from the two approaches, emerged drawbacks, main difficulties and possible potentials are outlined and discussed. Furthermore, planned future work is proposed to compensate originated difficulties. All those offer a solid base for further improvements in this area.

Chapter 2

State of the Art

Several work has been done regarding pose capture of articulated objects (mostly humans). Many steps are required, many difficulties have to be overcome in order to estimate the joints and skeleton.

Generally, there are two major steps to capture the pose of an object. First, the object has to be digitalized, which is achieved by a 3D scanning and reconstruction approach. The data might be in form of a point cloud or voxels depending on the reconstruction method (see section 2.1). The second major step includes the analysis of the data to recognize body parts and subsequently joints and if required the skeleton. Depending on the chosen approach, which is a segmentation step into rigid parts (see section 2.2), there might be a subdivision into sub steps. Generally, the segmentation is a optimization problem, as many information can be extracted for the segmentation approach.

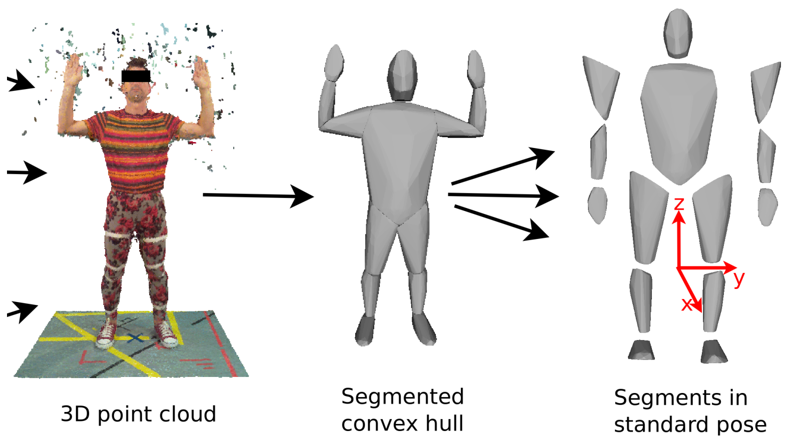


Figure 2.1: General approach to estimate the pose of a real object, including 3D scanning and reconstruction, 3D segmentation to get the joints and subsequently skeleton extraction

2.1 3D Scanning/Reconstruction

Scanning means collecting a real shape as 3D data. Reconstruction means the approach to convert the raw data to a mesh or process the input data.

Voxelization, Shape from Shilouette, Shape from Shading, Depth camera, stereo camera

2.2 3D Segmentation

How it is done: markers, sensors, shape fitting (already known) also real time approaches, due to constraints Cite all papers (Voxelization, ...). There are previous approaches with markers and sensors which will not be treated in this work in detail, as markerless options are taken as focus.

Comparison of rigid and non-rigid[16]

2.2.1 Supervised methods

Many already existing methods don't require markers and sensors but already assume or know the hierarchical structure or the body parts of the object to be captured (see [11], [3], [7] and [12]).

2.2.2 Unsupervised methods

Although the approaches mentioned in section **??** work quite well depending on the application, improvements can be made that are more independent from user inputs. For example the computation of features, ... which leads us to the non-rigid registration 2.3.

2.3 Non-rigid registration

Generally, registration means the alignment of rigid point clouds (see figure **??**). A well-known approach to achieve this, is the iterative closest point (ICP) [4], which requires the input point clouds to be aligned quite similar to avoid a local optimum. After registration, a matching error e is achieved, which states the total euclidean distance between the associated points of the registered point clouds. In case of two non-rigid objects (e.g. a human in different poses which is composed of rigid parts) the ICP won't lead to a satisfying registration as associated parts are transformed differently. In order to register a non-rigid object, a segmentation into its rigid parts is required.

2.3.1 Challenges

There are many challenges regarding the non-rigid registration of point clouds in 2D, as well as in 3D. First off, the input data can be noisy by means of points not belonging to the object. Furthermore, the approach is computationally expensive and time-consuming, as the corresponding body parts of two meshes need to be detected iteratively. Additionally, the inevitable difficulty of finding the global optimum, related to ambiguous body parts, has to be overcome.

2.3.2 Related work

By focusing on approaches computing the segmentation of articulated objects from 3D data, many different ones could be detected. They face similar challenges (see subsection 2.3.1) but solve them in different ways

2.3.3 Correlated Correspondence

A main approach for non-rigid registration is proposed by Anguelov [1] applying the correlated correspondence algorithm [2]. The algorithm takes a *template* Mesh D_0 and other Meshes $D_1, \ldots, D_n$ in different configurations as input. The algorithm then performs a probabilistic framework and Expectation-Maximization (EM) to iterate between finding a decomposition of the *template* into rigid parts and detecting them in the other meshes. Furthermore, a random clustering is applied to facilitate the detection of associated rigid parts. A different approach proposes the recursive detection of body parts by the LRP – "largest rigid part" algorithm [8].

2.3.4 LRP

The LRP algorithm discovers the articulated parts of two objects in different configurations by initially detecting the largest rigid part. This would be the biggest point cluster by applying a single rigid transformation. To reach that, sparse correspondences in combination with RANSAC are implemented. From there, the linking parts are recursively detected by growing clusters from the LRP and reapplying the algorithm. Another approach is achieved by Symmetrization [13], by detecting and aligning the body parts' symmetry axes of an object(see figure **??**). Based on Anguelov [1] and Mitra [13], Chang et al developed a closely related approach [5] [6].

2.3.5 Possible improvements

The proposed approaches achieve convincing results concerning the accuracy of the segmentation and the detection of rigid parts. However, they are all computationally expensive and require a considerable number of computation steps to iteratively detect rigid parts in two associated objects. This reflects on the run time of the algorithm which offers therefore great potential for improvements.

Taking the existing methods as reference (see chapter **??**) a new segmentation approach is developed. Thereby, the main focus is to reduce the computation steps of the correlated correspondence algorithm [2] as well as the LRP algorithm [8]. To fully focus on the segmentation into its rigid part, the 3D reconstruction of the articulated object is assumed to be available.

Notation

M	Input mesh
$\mathcal{P}$	set of rigid parts
$\mathcal{J}$	set of joints
$\mathcal{C}$	set of clusters
$\mathcal{T}$	set of transformations
C_i	cluster
$C_{i,j,\cdots}$	sub cluster with varying depth
p_i	principal axis of C_i
s_i	secondary axis of C_i
θ	orientation of C_i
$\boldsymbol{p}_i(x, y)$	cluster point of C_i
$\boldsymbol{j}_i(x, y)$	joint between two clusters C_i and C_j
$g(\boldsymbol{p}_i, \boldsymbol{p}_j)$	geodesic distance between two cluster points
$d(\boldsymbol{p}_i, \boldsymbol{p}_j)$	euclidean distance between two cluster points
e	matching error between two clusters
e_{avg}	average matching error between two clusters
τ	error threshold
N	Node in a tree
left	left child of N
right	right child of N
$\mathcal{L}$	set of final sub cluster pairs
$L_{i,j}$	sub cluster of $\mathcal{L}$
$\mathcal{U}$	set of unclustered points
$\boldsymbol{u}_i(x, y)$	unclustered point
r	radius
$\vec{n}$	normal vector
H_i	feature histogram of a point p_i
H_μ	mean feature histogram of a cluster C_i

Chapter 3

Linear approach

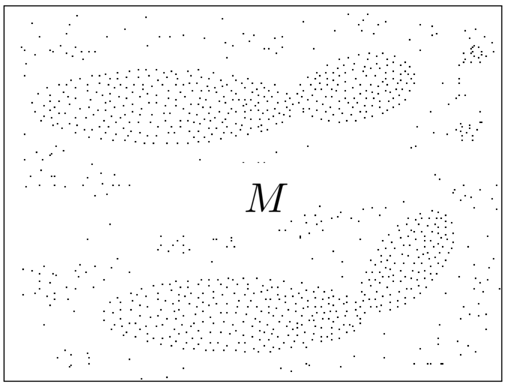
The first approach taken to achieve the goal was a straightforward, linear approach, which subdivides the input into sub clusters until they match. Solely the principal axis of C_1 and C_2 are computed to roughly align the clusters the similarly.

3.1 General functionality

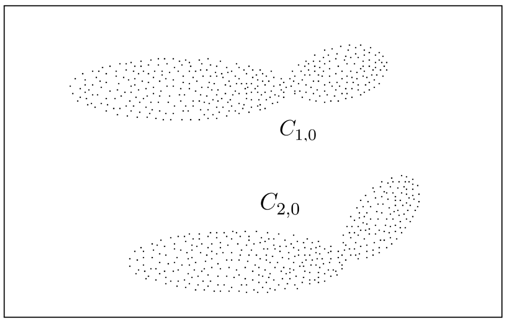
The algorithm starts with two sets of point clouds C_1 and C_2 of an object M in different poses (see figure 3.1). The two point clouds are iteratively subdivided into point clusters $\mathcal{C}_1 = (C_{1,1}, \ldots, C_{1,m})$ and $\mathcal{C}_2 = (C_{2,1}, \ldots, C_{2,m})$. In each iteration step two related sub clusters of C_1 and C_2 are verified to match by applying the ICP (iterative closest point), resulting in a matching error e. In case of $e < \tau$, two clusters are assumed to match. Otherwise, the algorithm is applied recursively and the sub clusters are again subdivided. The algorithm terminates if all resulting sub clusters of C_1 can be matched to all sub clusters of C_2. Subsequently, they are stored depending on their location and then checked to be merged, in case of having divided a rigid part. After that step, the resulting clusters are assigned to rigid parts $\mathcal{P} = \{P_1, \ldots, P_n\}$.

3.1.1 Detecting point clusters

As a first step, all point clusters $\mathcal{C} = \{C_1, \ldots, C_m\}$ are detected by applying region growing on all points of M. A cluster C_i is grown from an unclustered point $\boldsymbol{p}_i(x, y)$. Another point $\boldsymbol{p}_j(x, y)$ is added to the cluster C_i if the euclidean distance between them $d(\boldsymbol{p}_i, \boldsymbol{p}_j)$ is below a predefined threshold τ. This threshold is thereby depending on the resolution and density of M. All points of C_i are then iteratively compared to the remaining unclustered points to allow the cluster to grow. Once, all points of C_i have been treated, another unclustered point is used as a seed. If there are no unclustered points left, the clusters with the highest number of points n are selected as input clusters C_1 and C_2 (see figure 3.1). As a result, the remaining clusters are classified as noise and rejected for further computations.



(a)



(b)

Figure 3.1: Taking a mesh M in two different poses as input (a), removing noise of the input point clouds (b) to achieve the input clusters C_1 and C_2.

3.1.2 Subdividing into clusters

As a next step, the two main clusters C_1 and C_2 are taken as input for further computation steps. If the matching between two clusters does not succeed, they are both subdivided into two sub clusters. Otherwise, no subdividing is done. The whole procedure is repeated recursively for all clusters $\mathcal{C} = (C_{i,1}, \ldots, C_{i,m})$ of C_1 and C_2 until all associated clusters of C_1 match the clusters of C_2.

Divider position

To determine where to divide a cluster, it is taken advantage of the principal component analysis (PCA). As a first step, the orientation θ of C_1 and C_2 are computed by calculating the central moments

$$\mu_{pq}(\mathcal{R}) = \sum_{(u,v)\in\mathcal{R}} (u-\bar{x})^p \cdot (v-\bar{y})^q \tag{3.1}$$

$$\theta(\mathcal{R}) = \frac{1}{2}\tan^{-1}\left(\frac{2\cdot\mu_{11}(\mathcal{R})}{\mu_{20}(\mathcal{R})-\mu_{02}(\mathcal{R})}\right) \tag{3.2}$$

for each cluster. The divider positions are determined by computing the principal axes p_1 and p_2 and subsequently taking the perpendicular secondary axes s_1 and s_2 through the centroids (see figure 3.2). The secondary axes divide C_1 and C_2 into the sub clusters $C_{1,1}$ and $C_{1,2}$ as well as $C_{2,1}$ and $C_{2,2}$.

Declaring the matching condition between two clusters

By applying the ICP and the nearest neighbor approach on two associated clusters C_1 and C_2, a certain matching error e is computed between their cluster points $C_p = \{p_1, \ldots, p_m\}$ and the associated points $C_q = \{q_1, \ldots, q_m\}$. To eliminate the dependency between the matching error and the number of cluster points m, the average error per

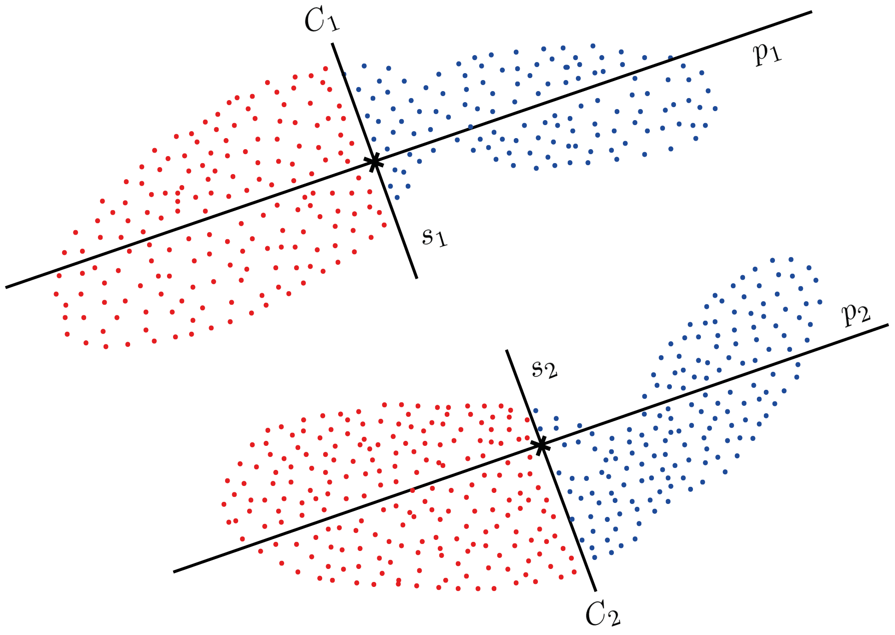


Figure 3.2: Subdividing C_1 and C_2 into two sub clusters by computing the secondary axes s_1 and s_2 perpendicular to p_1 and p_2 through the centroids.

point of C_p and C_q

$$e_{\text{avg}(C_p,C_q)} = \frac{1}{|C_p|} \cdot \sum_{i=0}^{m} \|\boldsymbol{p}_i - \boldsymbol{q}_i\|^2 \tag{3.3}$$

is computed, assuming that the two clusters C_p and C_q contain the same number of cluster points m. In case of varying point amounts, the segmentation needs to be carried out differently that sub clusters contain the same number of points. Alternatively, extra points are not considered in the error amount calculation. To declare when two clusters match, it is quite essential to determine an appropriate threshold τ for the maximum distance $d(p_0, q_0)$ between two associated points p_0 and q_0. In case of being overvalued, clusters are more likely to match which could result in insufficient subdividing. On the other hand, the clusters are difficult to be matched, which will result in further subdividing and the detection of too many rigid parts. The two clusters C_p and C_q are matching, if $e_{avg} < \tau$.

Cluster tree

The subdividing of the clusters C_1 and C_2 is realized by a depth-first approach in a tree. Consequently, C_1 and C_2 represent the root and are subdivided from the left to the right. A node N of the tree contains two related clusters $C_{1,i}$ and $C_{2,i}$, where i defines whether the node is a left ($i = 1$) or right ($i = 2$) node of the parent. In case of further subdividing, a Node *left*, containing the clusters $C_{1,i,1}$ and $C_{2,i,1}$, as well as a Node *right*, containing the clusters $C_{1,i,2}$ and $C_{2,i,2}$ origin. If two associated clusters $C_{1,i,j}$ and $C_{2,i,j}$ in a Node N_i match, no further subdividing is performed. The resulting

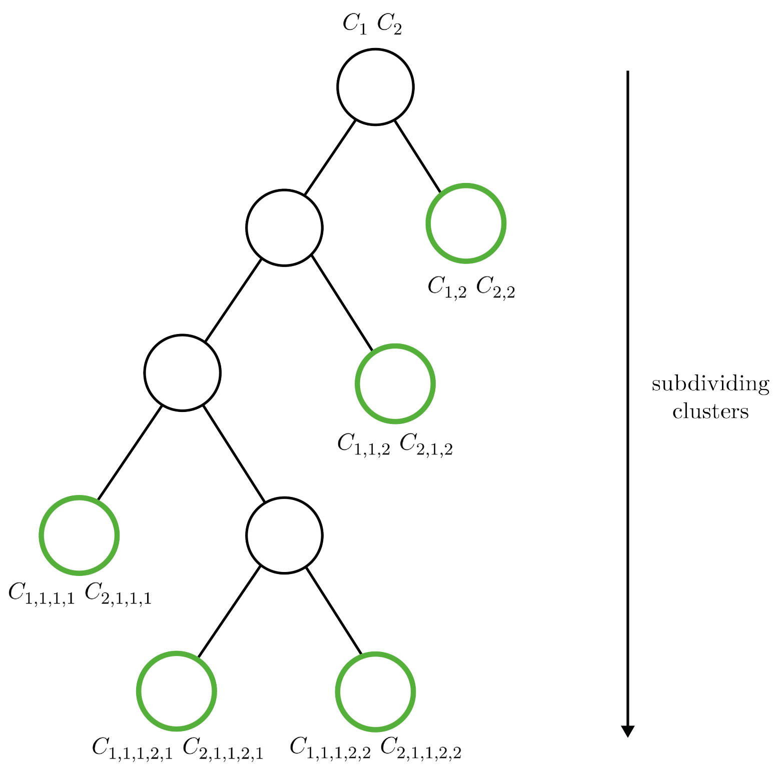

C_1 C_2
$C_{1,2}$ $C_{2,2}$
$C_{1,1,2}$ $C_{2,1,2}$
$C_{1,1,1,1}$ $C_{2,1,1,1}$
$C_{1,1,1,2,1}$ $C_{2,1,1,2,1}$
$C_{1,1,1,2,2}$ $C_{2,1,1,2,2}$
subdividing clusters

Figure 3.3: Subdividing of C_1 and C_2 into matching clusters by a depth-first approach in a tree. The Subdividing terminates if all sub clusters of C_1 and C_2 match when applying the ICP.

leaves of the tree are the final matching sub clusters and stored from left to right in a list $\mathcal{L} = (L_{1,1}, L_{2,1}, \dots, L_{1,m}, L_{2,m})$ (see figure 3.3). By applying the depth-first approach, the neighboring clusters in the list are also neighboring clusters in the main clusters C_1 and C_2. As a result, in the following step the adjacent sub clusters of C_1 or C_2 can be merged (see subsection 3.1.3).

3.1.3 Merging neighboring clusters to rigid parts

x As a next step, adjacent sub clusters from $\mathcal{L}$ are iteratively merged and subsequently verified to still match. This process is required to rejoin, if necessary, detected sub clusters to the rigid parts of the object. This is the case, if a rigid part was subdivided during the subdividing process (see section 3.1.2). The merging initiates with the first clusters in the list $L_{1,i}$, $L_{2,i}$ and its adjacent clusters $L_{1,i+1}, L_{2,i+1}$. If the resulting merged clusters can be matched, the merging proceeds with the adjacent cluster $L_{1,i+2}, C_{2,i+2}$. If not, the merging is not executed and $L_{1,i}$, $L_{2,i}$ are stored in a list of resulting clusters $\mathcal{R}$. The merging procedure then initiates with $L_{1,i+1}, L_{2,i+1}$. The process terminates if all clusters of $\mathcal{L}$ are verified and consequently the clusters of $\mathcal{R}$ are assigned to rigid parts $\mathcal{P} = \{P_1, \dots, P_m\}$ (see figure 3.4).

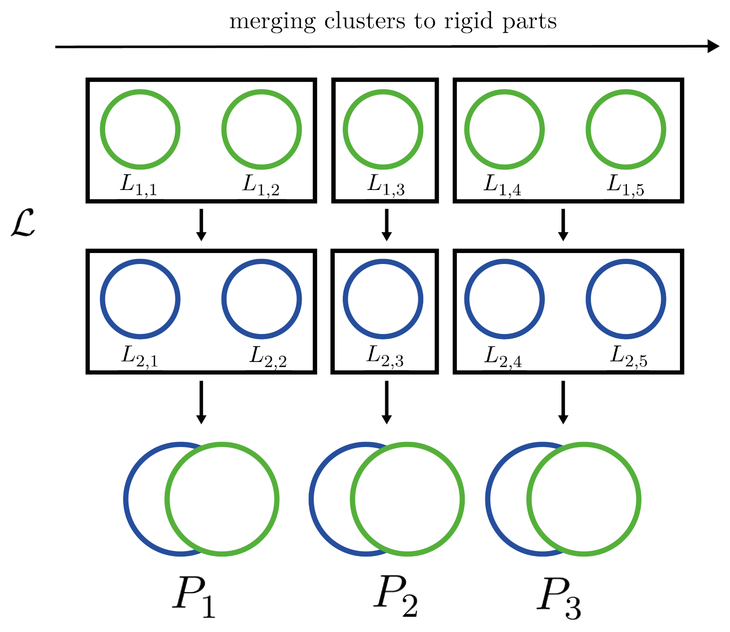


Figure 3.4: Detecting rigid parts of C_1 and C_2 by iteratively merging adjacent clusters of $\mathcal{L}$ that merged clusters still match.

3.1.4 Joint/skeleton estimation

After detecting the rigid parts $\mathcal{P} = \{P_1, \ldots, P_m\}$, they are linked with joints. Their locations are thereby calculated by computing the points of intersection of all principal axis of $\mathcal{P} = \{P_1, \ldots, P_m\}$. However, this calculation assumes that the rigid parts are symmetric, as in the other case the principal axis might not represent the skeleton of an object. For this reason another approach has to be taken into account. Anguelov [1] declares joints as two points of two neighboring rigid parts that undergo the same transformation T. In the current implementation an object point is only allocated to one rigid part P. An improvement of the current situation is therefore to select points that are located near a joint allocated to more neighboring rigid parts and from those selected points the location of a joint is computed.

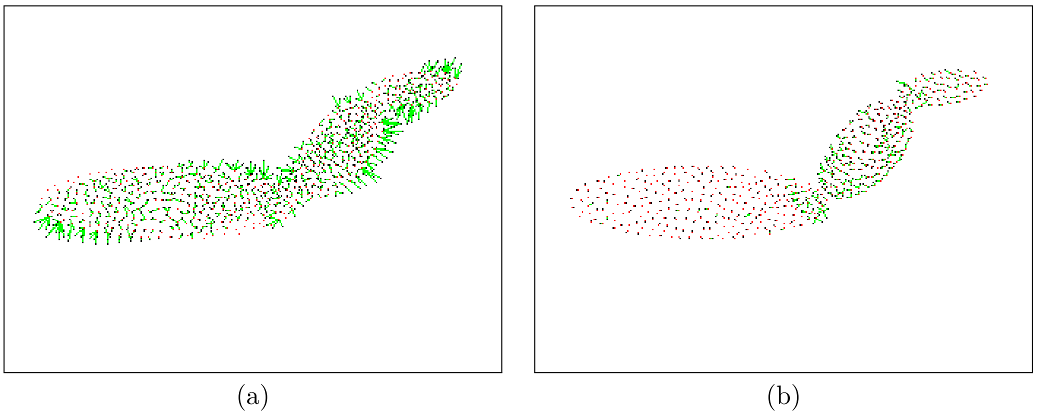


Figure 3.5: Registration of C_1 and C_2 before the segmentation into rigid parts (a) and after the segmentation.

3.2 Implementation

After planning the algorithm, it was initially implemented and for 2D point clouds, in order to be able to focus solely on developing and testing.

3.2.1 Chosen environment

The initial approach was implemented in Java, using ImageJ as processing library. The chosen environment depends on the following factors:

- familiarity and prior experience
- complexity
- available plug-in for image processing

As ImageJ is mainly used for 2D use cases, another implementation would be possible in 3D using PCL in C++. As a result, the attention can be brought to segmentation and visualization in 3D.

3.2.2 Architecture

The initial algorithm was realized with four classes to divide the "divide and conquer" algorithm from the Cluster object as well as the Visualization and the registration process of sub clusters. The main class `Segmentation` solely requires a stack of two point clouds in 2D. As a first step, possible noise is removed from the input mesh M (see algorithm **??**). The algorithm returns the biggest point cluster C_i that is assumed to be an articulated object. A class `Cluster` was implemented to store the cluster's points, its centroid, the orientation θ as well as the principal and secondary axes. For the subdividing, a `ClusterTree` was implemented to simultaneously divide the main clusters C_1 and C_2 into sub clusters (see algorithm 3.2). Each node N contains thereby two associated clusters $C_{1,i}$ and $C_{2,i}$. For the clustering and merging of clusters a `List<Clusters>` was used to dynamically add and remove Clusters. Furthermore, a class `ICP` was created, which takes two clusters to be matched as input, using Procrustes fitting. First, the noise removal and cluster detection is done in the `Cluster` class. As a next step, the detected main clusters m $C_1, \cdots, C_m$ are taken as input with an empty list to get the matching sub clusters of all input clusters. Following, the divide and conquer algorithm (see algorithm 3.2) takes advantage of the ICP class to iteratively register two sub clusters of two main clusters $C_i, \cdots, C_j$. As the segmentation is done depth first the verified sub clusters are stored in the list from left to right. The returned segmented list is taken as input to iteratively merge the sub clusters to detected rigid parts (see algorithm 3.3). As a return we get a list with sub clusters representing the rigid parts of the input clusters. Subsequently, the Visualization class is used, to color the cluster points differently and draw PCA related components, like the axis and interference points of different rigid parts.

3.2.3 ICP

One main part of the algorithm is the modified implementation of the ICP using Procrustes fitting to compute a transformation that detects sparse point correspondences. Thereby, only reciprocal correspondences within a specific distance τ contribute to the calculation. The final point correspondences of C_i and C_j are stored in a `Map<Integer, Integer>` containing the indices of the corresponding points. This is because the storage of points in the form $\boldsymbol{p}_i(x, y)$ would underly a specific transformation, which is applied during the ICP and initial alignment. For further computations using the RANSAC, no transformations are desired. As the main difficulty is the right initial alignment of the actual largest rigid part (torso) the value of the distance threshold τ is chosen generously. As a result, a higher number of false point correspondences is detected which has to be compensated by RANSAC (see subsection 4.3.2).

```
...
for (Map.Entry<Integer, Integer> entry : reference.entrySet()) {
Integer referenceIndex = entry.getKey();
Integer targetIndex = entry.getValue();

if (target.containsKey(targetIndex) && target.get(targetIndex) == referenceIndex) {
referencePoints.add(originalReference.get(referenceIndex));
targetPoints.add(originalTarget.get(targetIndex));
finalAssociations.put(referenceIndex, targetIndex);
}
}
...
```

3.3 Results

At first, the implementation was tested on two point clouds of an articulated object with only two rigid parts. The segmentation results are directly dependent on the matching error threshold τ which can bee seen on table 3.1. The higher the threshold τ, the less clusters and subsequently rigid parts can be detected, as two clusters are more likely to match and are not further subdivided. The lower τ, the more clusters and rigid parts will be detected, as clusters require further subdividing in order to match. Figure 3.7 and figure 3.8 show the clustering and rigid part detection of two simple objects, in regard to their rigid parts linked like a chain. For those objects, with a threshold $\tau = 5$

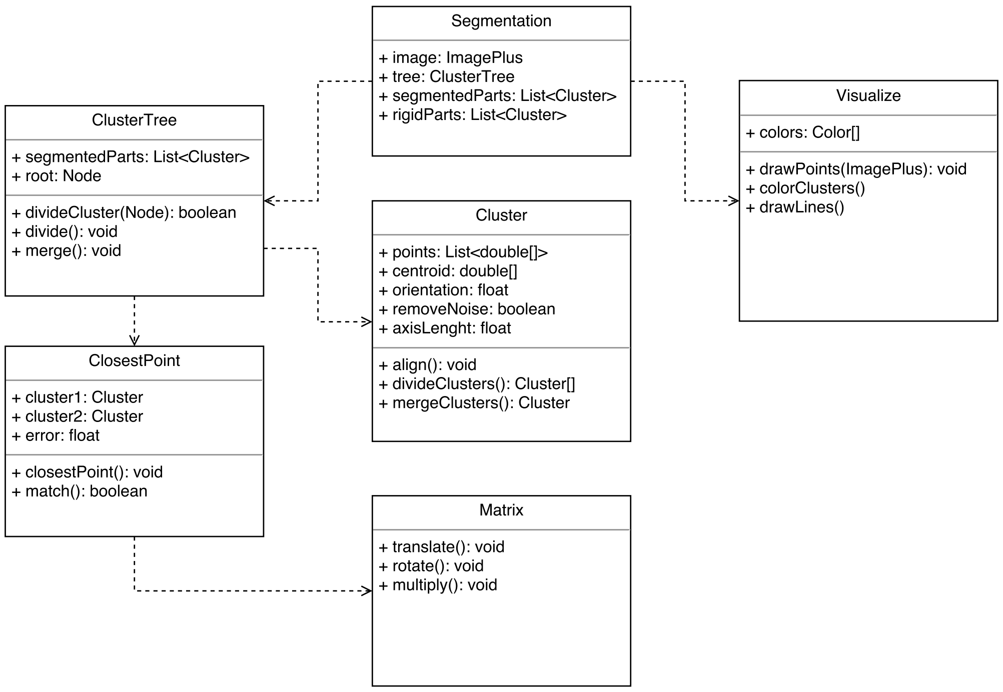


Figure 3.6: UML diagram of the classes related to the implementation of the algorithm.

Rigid parts	τ	detected clusters	detected rigid parts
2	3	21	14
	5	3	2
	7	2	2
3	4	38	31
	6	4	3
	8	3	3
4	6	35	26
	7	3	3
	8	3	3

Table 3.1: Segmentation results

the right number of rigid parts is detected. However, the segmentation position does not correspond to the actual joint of C_1 and C_2. The reason is, that the average error e_{avg} is computed without any weighting of points. Especially points located near a joint need to be treated cautiously. Figure 3.11 shows a more complex object. The segmentation into rigid parts is also apparent, when visualizing the point registration between C_1 and C_2 (see figure 3.9). Two associated points from C_1 and C_2 are thereby linked with green lines. Comparing the registration results before and after a segmentation into rigid parts clearly shows, that the registration of rigid parts with the ICP results in a lower matching error e.

Algorithm 3.1: Noise removal of an input point mesh M in form a set of unclustered points $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$ by region growing. The first point of M is used as seed and grows a cluster $C_{current}$ by iteratively adding neighboring points located inside a threshold τ. Once, all points have been examined, the largest cluster C_{max} is returned and defined as articulated object to be segmented.

```
RemoveNoise($M$)
    $C_{max} \leftarrow ()$
    $C_{current} \leftarrow ()$
    $n \leftarrow sizeOf(M)$
    $m \leftarrow sizeOf(C_{current})$
    while $n > 0$ do
        $c_{current} \leftarrow c_{current} + \boldsymbol{u}_1$
        for $i = 1, \ldots, m$ do
            $M \leftarrow M - C_{current}$
            for $j = 1, \ldots, n$ do
                if $d(\boldsymbol{p}_i, \boldsymbol{u}_j) < \tau$ then
                    $C_{current} \leftarrow C_{current} + \boldsymbol{u}_j$
                end if
            end for
        end for
        $M \leftarrow M - C_{current}$
        if $m > sizeOf(C_{max})$ then
            $C_{max} \leftarrow C_{current}$
        end if
        $C_{current} \leftarrow ()$
    end while
    return $C_{max}$
end
```

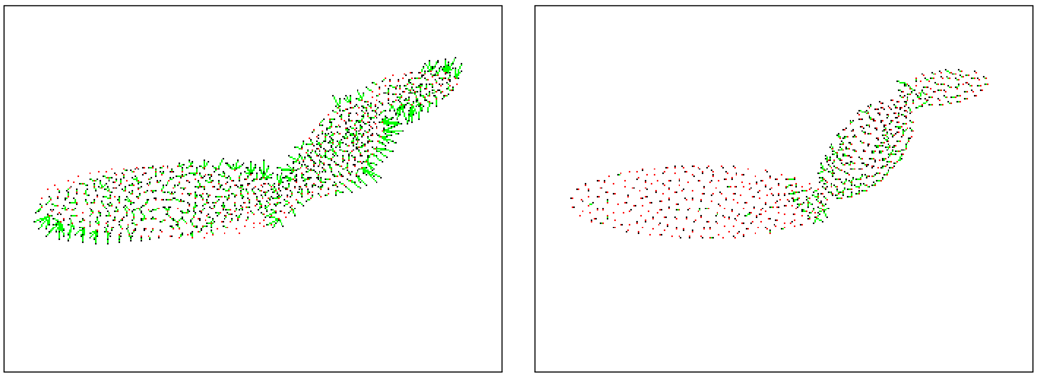

(a) (b)

Figure 3.9: Registration of C_1 and C_2 before the segmentation into rigid parts (a) and after the segmentation (b).

Algorithm 3.2: Recursive subdividing of two clusters C_1 and C_2, in the form of a Node N in a Tree, into matching sub clusters. The ICP is applied on two corresponding clusters in N to verify them to match, in which case they are stored in a list. Otherwise, if the two clusters do not match, they are further subdivided. The list with all matching subclusters is returned once the subdivide algorithm terminates.

```
1:  ClusterTree($C_1$, $C_2$)
2:      $L \leftarrow ()$
3:      $left \leftarrow nil$
4:      $right \leftarrow nil$
5:      $N \leftarrow \langle C_1, C_2, left, right \rangle$
6:      $L \leftarrow$ Subdivide($N$)
7:      $P \leftarrow$ MergeClusters($L$)
8:  end
```

```
1:  Subdivide($N$)
2:      if match($C_1(N)$), $C_2(N)$ then                 ▷ Apply ICP on two clusters
3:          $L \leftarrow L + (N)$
4:      else
5:          $left(N) \leftarrow$ Split($N$)
6:          $right(N) \leftarrow$ Split($N$)      ▷ Split the cluster into a left and right side
7:          Subdivide($left(N)$)
8:          Subdivide($right(N)$)               ▷ Recall the algorithm with sub clusters
9:      end if
10:     return $L$                          ▷ Return all sub clusters after termination.
11: end
```

In case of a more complex object, the simple segmentation algorithm fails. Although varying threshold τ for the matching, there are either too many or few rigid parts detected. When using $\tau = 7$ (see figure 3.10) only 3 clusters and rigid parts can be detected. By decreasing τ to 6 (see Figure 3.11), further subdividing is done, which leads to a too high number of rigid parts and clusters.

Algorithm 3.3: Merging of the sub clusters $L = ((L_{1,1}, L_{2,1}), \ldots, (L_{1,m}, L_{2,m}))$, resulting from algorithm 3.2 in the order of being stored, to rigid parts $\mathcal{P}$. Verify the matching of merged adjacent clusters $L_{i,j}$ and $L_{i,j+1}$ from C_1 and C_2. The merging is continued until no match can be done. In this case the last last merged cluster pairs are stored as rigid parts P_1 and P_2. The algorithm then continues with the next cluster pair in the list and terminates if all pairs have been traversed. The list with all detected rigid parts $\mathcal{P}$ is returned.

```
MergeClusters($L$)
    $P \leftarrow ()$
    $P_1 \leftarrow L_{1,1}$
    $P_2 \leftarrow L_{2,1}$
    $n \leftarrow sizeOf(L)$
    for $i = 2, \ldots, n$ do
        $merge_1 \leftarrow$ Merge($P_1, L_{1,i}$)
        $merge_2 \leftarrow$ Merge($P_2, L_{2,i}$)
        if match($merge_1$, $merge_2$) then        ▷ Apply ICP on two merged clusters
            $P_1 \leftarrow merge_1$
            $P_2 \leftarrow merge_2$        ▷ Continue merging with merged clusters
        else
            $P \leftarrow P + (P_1, P_2)$
            $P_1 \leftarrow L_{1,i}$
            $P_2 \leftarrow L_{2,i}$        ▷ Initiate merging with current clusters
        end if
    end for
    $P \leftarrow P + (merge_1, merge_2)$
    return $P$
end
```

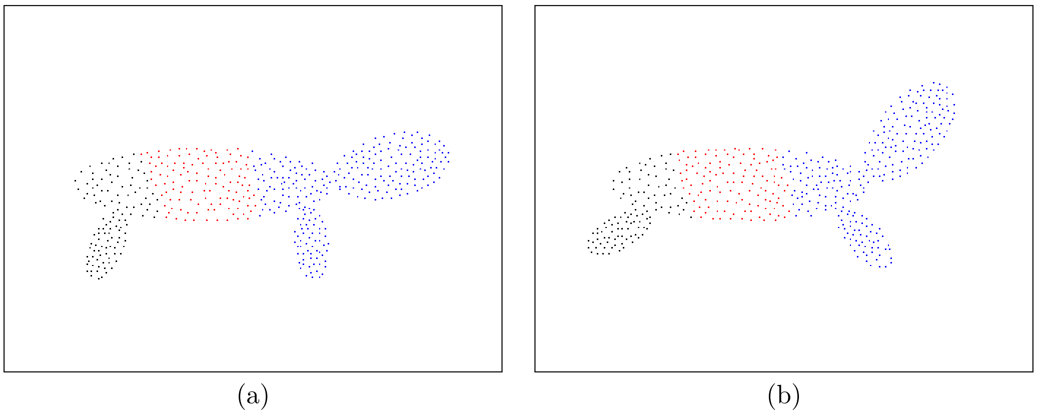


Figure 3.10: Taking a more complex Mesh M in two poses with four rigid parts as an input, with a threshold $\tau = 7$, 3 clusters and rigid parts can be detected in C_1 (a) and C_2 (b).

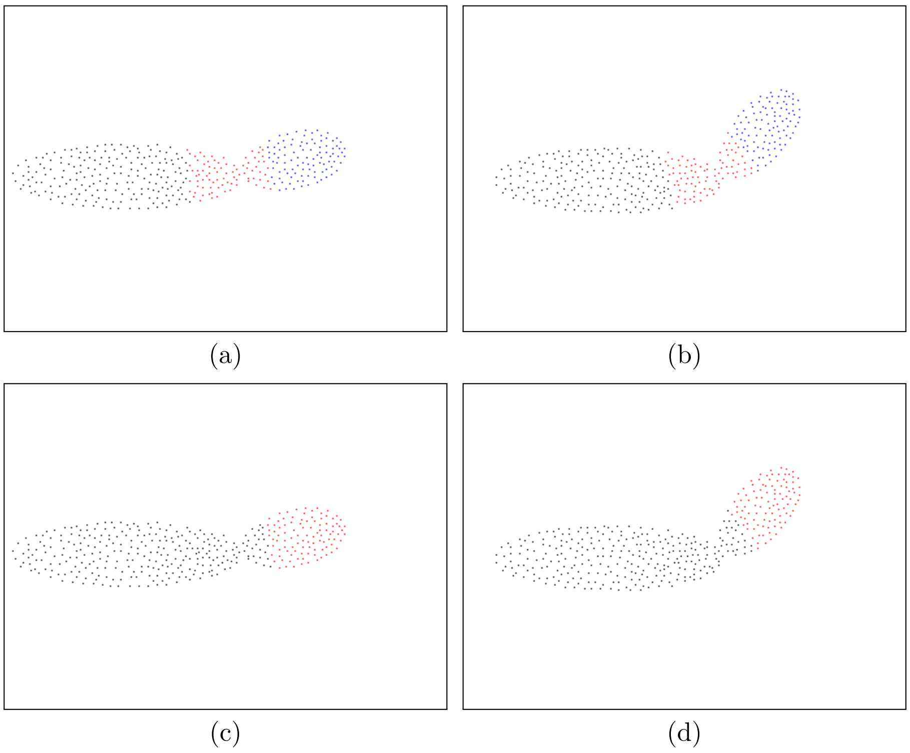


Figure 3.7: Taking a Mesh M in two poses with only two rigid parts as input, with a threshold $\tau = 5$, 3 clusters are detected in C_1 (a) and C_2 (b), which results in 2 rigid parts in C_1 (c) and C_2 (d).

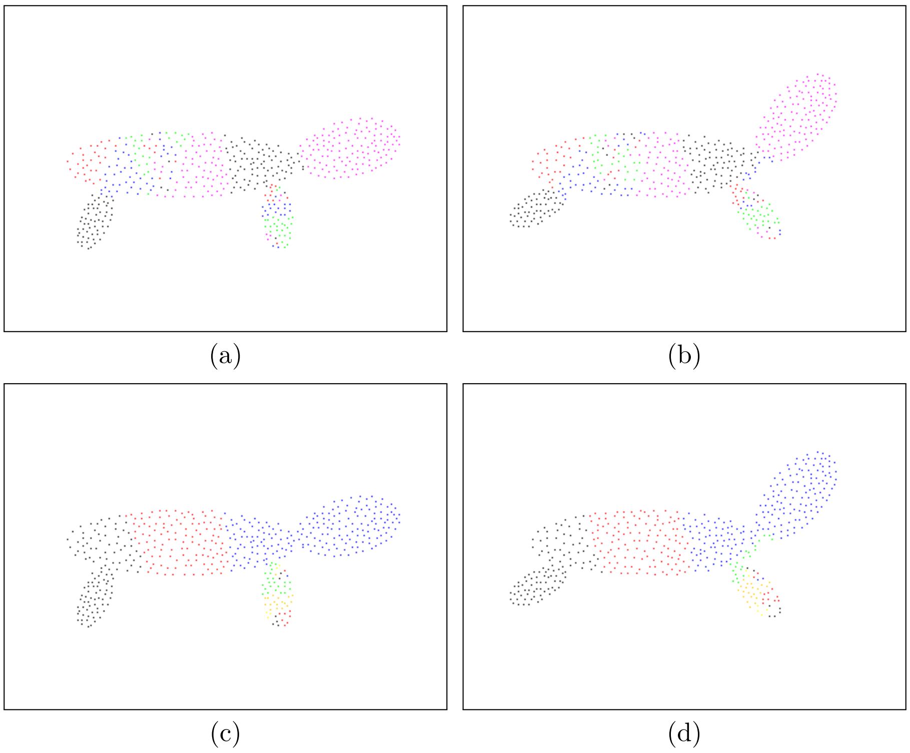


Figure 3.11: Taking a more complex Mesh M in two poses with four rigid parts as an input, with a threshold $\tau = 6$, 35 clusters are detected in C_1 (a) and C_2 (b), which results in 26 rigid parts in C_1 (c) and C_2 (d).

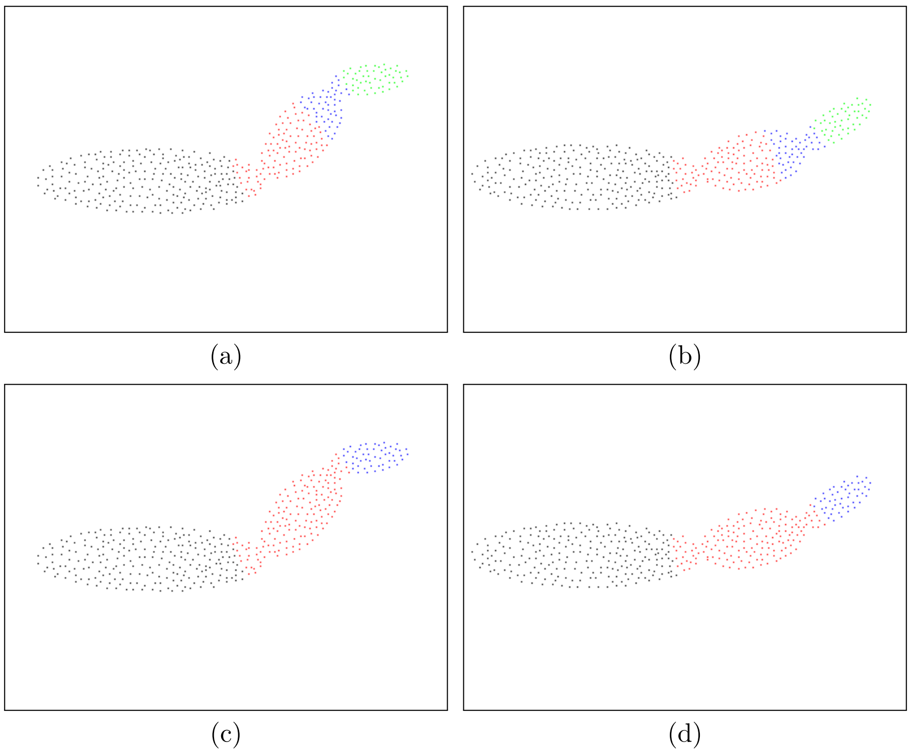


Figure 3.8: Taking a Mesh M in two poses with three rigid parts as an input, with a threshold $\tau = 6$, 4 clusters are detected in C_1 (a) and C_2 (b), which results in 3 rigid parts in C_1 (c) and C_2 (d).

The algorithm generally works for simple objects, in which the rigid parts are linked like a chain. Still, the segmentation location is not accurate, which might be disposed by introducing weights to points located near a joint. For more complex object with a skeleton structure, e.g. a human, where one rigid part is linked to more than two rigid parts, this simple implementation fails. In this case, another approach has to be found, as a skeleton structure is more complex to extract than a simple chain structure.

3.4 Improvements

In case of articulated objects with a skeleton structure, the segmentation algorithm in its simple form fails. The main reasons are the following:

- By recursively subdividing C_1 and C_2 the detected sub clusters might actually count to more than one and being located apart from each other.
- In case of a matching cluster being a part of a rigid part, no further operations are done with the match. The rigid part is anticipated to be generated by merging neighboring sub clusters.
- By further sub dividing clusters, the merging becomes more difficult, as the clusters are scattered next to each other.
- The detection of joints is not exact although the segmentation in the approximate

rigid parts succeeds as sub clusters are good enough to fit, but still there might be better fits with less or more points.

Thus, the initial approach needs to be extended, which solved the issues being mentioned.

3.4.1 Region growing

During the segmentation of an object in two different poses, there is the general case that the divided parts being compared do not contain the same number of points. As this state might come to undesirable matching errors, parts with the same sizes could be generated by region growing. This can be e.g. implemented by starting with the most outside point of two poses and grow regions with the same size of points which are then compared. Furthermore, the detection of multiple clusters can be treated, in case of subdividing clusters. The detected clusters are as a result treated individually.

3.4.2 Matching error

Another improvement is the assurance that during the ICP procedure each point only has one nearest neighbor and also considering the case, that there are not always the same amount of points in two clusters (uneven number of points). By doing so, not all points would contribute to the matching error. Furthermore, weights should be added to the points, especially points located near a joint should be treated cautiously. A cluster could be therefore further subdivided, if a minimum number of points is above the error threshold τ and not the average.

3.4.3 Initial alignment of clusters

The initial "Divide and Conquer" approach is used to detect two matching sub clusters of C_1 and C_2. But unlike previous implementations, the rigid parts are not detected by a following merging step, but by region growing, until the matching error e is above the specified threshold τ. If a rigid part is detected, it is stored and the same procedures initiates with the other points of C_1 and C_2. By doing so, all rigid parts are sequentially detected from one direction to the other.

3.4.4 Segmentation of articulated objects

The most crucial deficit of the proposed algorithm is that is does not work with articulated objects, whose parts are not simple linked as a chain, but a rigid part can have more than two linking rigid parts. An example would be the skeleton of humans and most animals. As a result, the objects are simply too complex to linearly subdivide them. One improvement proposition is thereby the initial alignment of the object, that the largest rigid part is aligned. A similar approach was taken during the LRP algorithm [8]. Then, recursively linked parts of this largest rigid part are detected (see section 4.2).

3.4.5 ICP

Sparse correspondences of two input clusters C_i and C_j were initially achieved by applying the Iterative closest point algorithm. Thereby, the two input clusters are aligned by means of the PCA. Following, corresponding points of C_i and C_j are only considered, if they are *reciprocal* (see subsection 4.2.1). Furthermore, the euclidean distance between two cluster points $d(\boldsymbol{p}_i, \boldsymbol{p}_j)$ must be below a predefined threshold τ. As a consequence, points being located far away from each other do not contribute to the alignment of C_i and C_j and are not stored as correspondence. Those are assumed to be small rigid parts with different transformations.

3.4.6 Main drawbacks

The main drawback of the algorithm represents the first initial alignment of the two poses of the articulated object. Thereby, it is directly dependent on the symmetry of the pose. The less symmetry the higher the changes that the initial alignment on the largest rigid part might fail. The reason is that the ICP expects a good starting alignment. As it is assumed that the major largest rigid part contributes most to the principal axis and the initial alignment, it would work. But in case of an unbalance of the linked parts, the alignment of the LRP might shift in a certain direction and it might not be detected during the ICP. As a result the whole algorithm fails. Furthermore, touching of rigid parts (e.g. the hand touches the leg) constitute difficulties as the region growing would not detect those as potential linked clusters.

Chapter 4

Feature-based approach

To solve the main drawbacks of finding reliable point correspondences between C_1 and C_2 the focus lies on point features for an initial alignment as they provide meaningful information about a point p being compared to its neighbors. The approach to use point features for the non-rigid registration procedure is closely related to Mitra [13]. The main part thereby is the computation of point feature histograms, namely "Fast point feature histograms" (FPFH) [14]. Basically, the computed features describe the curvature between specific points. By computing those features for all points, depending on the curvature, which might be an implied curve for an ellipsoid rigid part, in case of intersections of rigid parts, different features are proposed. Taking only *unique* feature histograms for correspondence estimation which are features considerably deviating from the mean histogram μ of all point feature histograms of a cluster C_i allows to reduce computation time. It is thereby assumed that those unique features are those located between rigid parts.

4.1 Fast point feature histograms

The "Fast Point Feature histograms" (FPFH) algorithm is an improved approach of the ''Persistent Point Feature Histogram for 3D Point Clouds" [15]. It focuses on computing features, depending on a points normal $\vec{n}$ and its k neighbors. The choice of those features are the following:

- rotation- and scale-invariant features
- easy comparison of feature histograms
- straightforward implementation
- approval of approach
- easy adaption from different dimensions (2D and 3D)

By using a histogram the neighborhoods' geometrical properties can be provided in form of the mean surface curvature at a point $\boldsymbol{p}$. As in the unordered list of points no surface and following no point normals are provided, those have to be computed in a initial step (see subsection 4.1.1).

4.1.1 Normal estimation

As a first step, the normals of all unordered points from the input clusters C_1 and C_2 need to be estimated [10]. Those is achieved by taking all points k within a radius r from a point $\boldsymbol{p}$. Subsequently, a least error fit straight line X in the form $ax + by + c = 0$ to all k points is computed by minimizing the squared distances from all points $\boldsymbol{p_i(x_i, y_i)}$

$$\begin{aligned} dist^2(x_i, y_i) &= (ax_i + by_i + c)^2 \\ e &= \sum_{i=1}^{k} dist^2(x_i, y_i) \end{aligned} \tag{4.1}$$

to X. This is done by computing the unknown parameters a, b of X with the covariance matrix

$$\begin{pmatrix} \overline{x^2} - \overline{x} \cdot \overline{x} & \overline{xy} - \overline{x} \cdot \overline{y} \\ \overline{xy} - \overline{y} \cdot \overline{x} & \overline{y^2} - \overline{y} \cdot \overline{y} \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \lambda \cdot \begin{pmatrix} a \\ b \end{pmatrix} \tag{4.2}$$

resulting in two pairs of an eigenvalue and eigenvector (n_1, λ_1) and (n_2, λ_2). The normal of $\boldsymbol{p}$ is computed solving the linear equation for the normal vector $\vec{n_2}$ which is represented by λ_2. The procedure is conducted for all points from C_1 and C_2 (see figure 4.1).

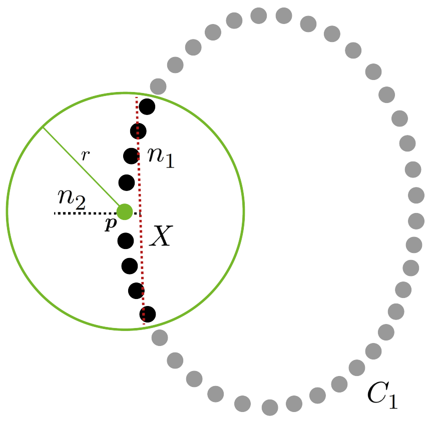


Figure 4.1: Normal estimation for a cluster point $\boldsymbol{p}$ of C_i by a computation of the least squared fitting line X of the neighborhood inside a radius r. Calculation of the eigenvectors $\vec{n_1}$ and $\vec{n_2}$ by the covariance matrix of all k points.

As a next step, it is required that all normals are equally oriented. Basically, all n computed normals are traversed globally, starting with a normal $\vec{n_i}$ of a point p_i. Thereby, it is of particular importance to select a normal that assures consistent orientations between C_1 and C_2. Taking its orientation as parent normal $\vec{n_p}$, the angle δ between $\vec{n_p}$ and all neighboring normals $\vec{n_k}$

$$\delta = \vec{n_p} \cdot \vec{n_k}, \quad \text{for } |\vec{n_p}|, |\vec{n_k}| = 1 \tag{4.3}$$

is computed. In case of $\delta < 0$, $\vec{n_k}$ requires to be flipped 180° ($\vec{n_k} = -\vec{n_k}$). If all k normals have been verified to be oriented in consideration of $\vec{n_p}$ any $\vec{n_k}$ is selected as current $\vec{n_p}$.

The whole algorithm proceeds until all n normals have been verified to correspond with their parent normal n_p. Results of the normal flipping procedure can be seen on figure 4.2.

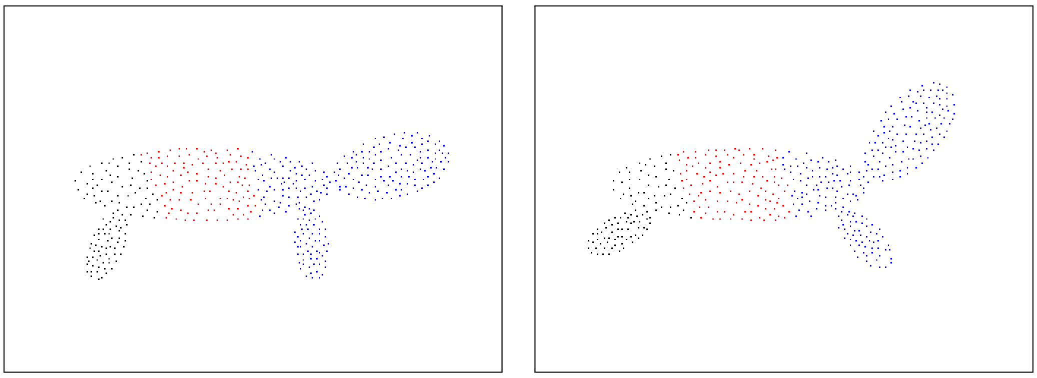

(a) (b)

Figure 4.2: Normal estimation of a Cluster C_1 (a) and a subsequent global traversing of all normals to orient them similar (b).

4.1.2 SPFH and FPFH

The simplified point feature histogram (SPFH) for a point $\boldsymbol{p}$ is then computed by using three geometric features. Between $\boldsymbol{p}$ and each of its k neighbors $\boldsymbol{p_k}$ given a threshold τ, those features are computed – $\boldsymbol{p_i}$ is thereby the point having the smaller angle between its normal and the line connecting the point set, $\boldsymbol{p_j}$ corresponds to the remaining point. Using their normals n_i and n_j a Darboux uvn frame $(u = n_i, v = (p_j - p_i) \times u, w = u \times v)$ is computed. The following angles

$$\begin{aligned} \alpha &= v \cdot n_j \\ \phi &= (u \cdot (p_j - p_i))/\|p_j - p_j\| \\ \theta &= arctan(w \cdot n_j, u \cdot n_j) \end{aligned} \tag{4.4}$$

are computed. In a second step for each point $\boldsymbol{p_i}$ again all n neighbor points given a threshold τ are computed. The simple point histogram SPF of p is then weighted to the final histogram

$$FPFH(\boldsymbol{p}) = SPF(\boldsymbol{p}) + \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{w_i} \cdot SPF(p_i) \tag{4.5}$$

where the weight w_i represents the distance $d(\boldsymbol{p}, \boldsymbol{p}_i)$. The influence region diagram for a Fast Point feature histogram for a query point $\boldsymbol{p_q}$ can be seen on Figure 4.3.

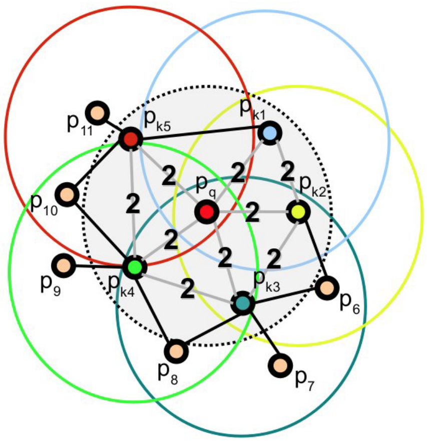


Figure 4.3: The point region for the calculation of the feature histogram for a query point $\boldsymbol{p}_q$. The histogram of $\boldsymbol{p}_q$ and its neighbors (inside the grey circle) is weighted with the further linked neighbors (colored circles) [14].

4.1.3 Creation of feature histograms

The resulting feature values for each point p of C_1 and C_2 in form of three angles between its neighbors k are categorized using a histogram $H\{bin_1, \ldots, bin_b\}$ with $b = q^f$ bins to consider all possible combinations of the feature values. Thereby, q represents the number of intervals and f the number of feature values, in this case 3. After the allocation the associated bin at the index idx

$$idx = \sum_{i=1}^{i \leq f} q(f_i) \cdot 2^{i-1} \tag{4.6}$$

is incremented by 1. The function $q(f)$ returns the interval the feature is allocated to, ranging from 0 to $q-1$. Finally, each bin contains the number of point pairs that are allocated in the specified value interval.As soon as the feature histograms of all points from C_1 and C_2 are computed only salient histograms are taken as comparison for point correspondence between C_1 and C_2 to safe computation time. In order to achieve that the mean of all feature histograms μ of a cluster C_i is computed. Subsequently, the distance of a feature histogram H_i is compared to H_μ and in case of being outside the value $\mu \pm \sigma$ denoted as *unique* and passed to the next step of detecting matching histograms between C_1 and C_2. The standard deviation

$$\sigma = \frac{1}{N} \cdot \sum_{i=1}^{N} (H_i(b) - \overline{H_i})^2 \tag{4.7}$$

is therefore calculated for all histograms H_i with N data entries of all cluster points. The mean histogram as well as the histogram of a not salient point feature can be seen on figure 4.4.

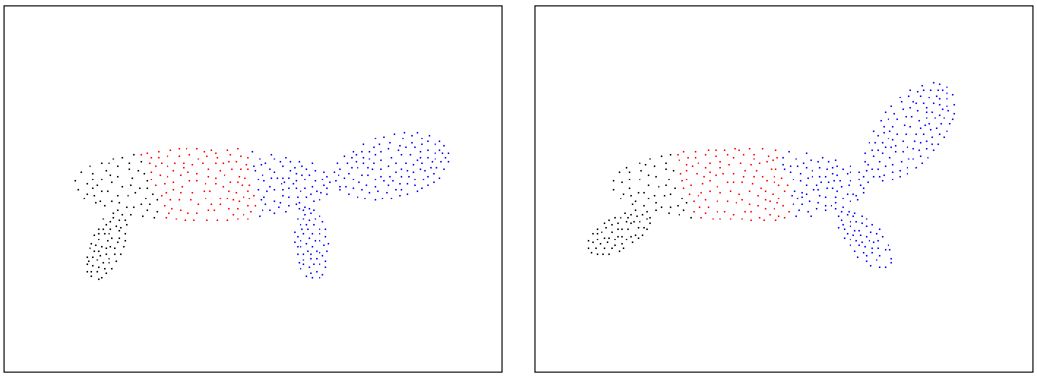

(a) (b)

Figure 4.4: Computing of the μ histogram of C_1 (a) in order to reject frequently emerging feature histograms (b) being inside a value $\mu \pm \theta \cdot \sigma$.

4.1.4 Comparison of histograms

As a next step, all *unique* feature histograms of C_1 are verified to find the closest match from C_2. For that reason, different histogram-similarity criteria have to be determined to measure the similarity between two histograms H_j and H_k with each b bins. In order to detect the best fitting histogram three main criteria mentioned most often in reference papers [17] [9] were selected. As first criterion the squared euclidean distance (L2)

$$\varepsilon(H_j, H_k) = \sum_{i=1}^{b} (H_j(i) - H_k(i))^2 \tag{4.8}$$

is computed. In the reference paper of Mitra et al. [13] the euclidean distance was the only criterian for correspondence computation which generates most of the point correspondences (see section 4.4). Subsequently, the statistical Chi-Square (χ^2) divergence

$$\chi^2(H_j, H_k) = \sum_{i=1}^{b} \frac{(H_j(i) - H_k(i))^2}{(H_j(i) + H_k(i))} \tag{4.9}$$

is examined which achieves better results than the L2 form. Finally, the Kullback-Leibler (KL) divergence

$$\kappa(H_j, H_k) = \sum_{i=1}^{b} (H_j(i) - H_k(i)) \cdot ln \frac{H_j(i)}{H_k(i)} \tag{4.10}$$

is calculated, which is the most computational expensive one due to the usage of ln. As a histogram with q^f bins might contain a lot of zero values, in case of a division or logarithm all zero-values are replaced with the value 1.

4.1.5 Adaptions for 2D

In order to compute those 3D features for 2D points, those can be treaded like 3D points by setting each z-coordinate to 0, by doing so, all calculations can be conducted straight forward. However, one feature would be redundant, as v is the same vector for each point. No considerable deductions could be detected.

4.2 Initial alignment: Largest rigid part

As opposed to the linear approach by subdividing C_1 and C_2 into sub clusters, the feature-based approach is implemented to detect as an initial step the largest rigid part (LRP) of C_1 and C_2. Proceeding from there, all other linked parts are detected by region growing and reapplying the algorithm. This approach has been originally implemented for 3D point clouds by Mitra [13]. For the 2D implementation, the 2D hulls of an articulated object in two different poses are taken as input (see Figure 4.5).

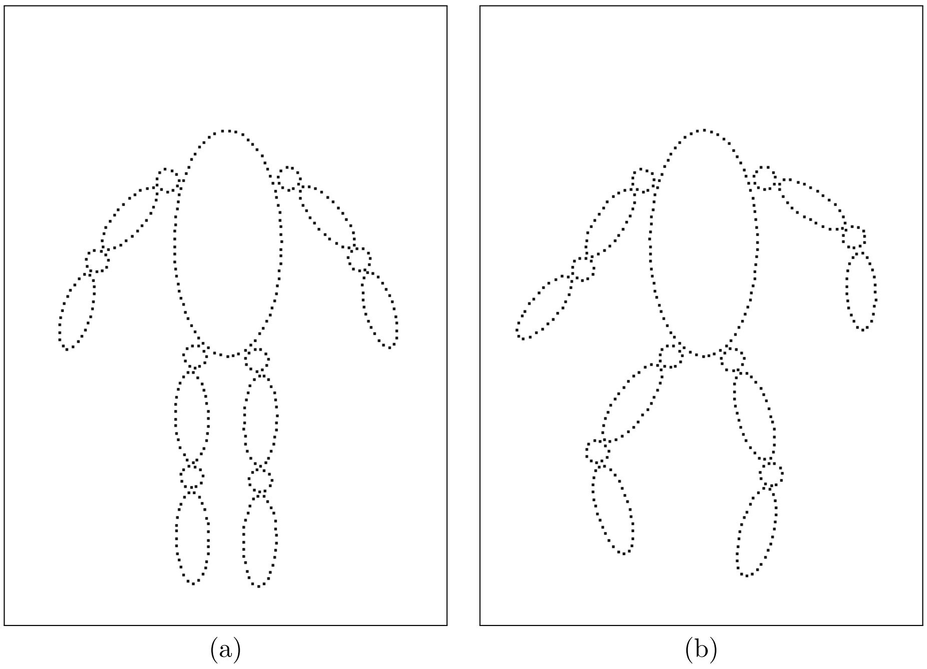


Figure 4.5: Taking an articulated object (puppet) in two different poses C_1 (a) and C_2 (b) in form of a 2D point cloud representing its hull point cloud as input.

4.2.1 Basic functionality

As an initial step, the LRP algorithm attempts to detect the most reliable correspondences between C_1 and C_2. For that, local point descriptors (see section 4.1) are computed. The requirement for a sparse correspondence between two cluster points $\boldsymbol{p}_i(x, y)$ and $\boldsymbol{p}_j(x, y)$ is that they are *reciprocal*, which means that the according point feature histograms are the most similar from each other. Some of the sparse correspondences are assumed to be wrong. Therefore, by applying RANSAC to the point correspondences, a single rigid transformation is aimed for to detect the so-called “largest rigid part” (LRP), which is supported by the largest corresponding point cluster between C_1 and C_2. In case of a human, this would be the torso. Subsequently, all linked rigid parts to the LRP are detected by recursively applying the algorithm on grown clusters from the current LRP.

4.2.2 Implementation steps

In order to re-implement the algorithm in 2D, only minor modifications concerning point coordinates and feature histograms had to be accomplished. The most crucial part of the whole algorithm is the initial alignment of C_1 and C_2 in order to detect the actual largest rigid part of the articulated object. This step is of particular importance, as the subsequent detection of further rigid parts proceeds from there. Therefore, as first step, sparse correspondences between C_1 and C_2 have to be detected (see subsection 4.2.3).

1. The PCA is employed on the input clusters C_1 and C_2 to estimate the normals of all points.
2. Point feature histograms (FPFH) are computed to detect sparse point correspondences between C_1 and C_2. A Point correspondence between C_1 and C_2 needs to be *reciprocal.*
3. The RANSAC approach is applied on those correspondences to detect a T_j that rejects wrong point correspondences. Clusters are detected from all corresponding points by applying region growing.
4. The LRP is assigned to the resulting biggest point cluster.
5. Proceeding with the LRP, unmatched clusters to C_1 and C_2 are seeked by region growing from the LRP. The algorithm is then reapplied on those clusters until all rigid parts have been discovered.

4.2.3 Detection of sparse correspondences

As a first step of the algorithm, the feature histogram for each point of C_1 and C_2 (see section 4.1) is computed. For the detection of sparse correspondences three histogram distances (see section 4.1.4) are taken into account. The most similar histograms are selected as correspondence if they are *reciprocal.* Depending on the chosen distance as criteria, more or less correspondences can be detected, refer to section 4.4 for a detailed comparison. As some of those correspondences might be wrong (see figure 4.6), a RANSAC approach is applied on all correspondences to reject false ones (see subsection 4.2.4).

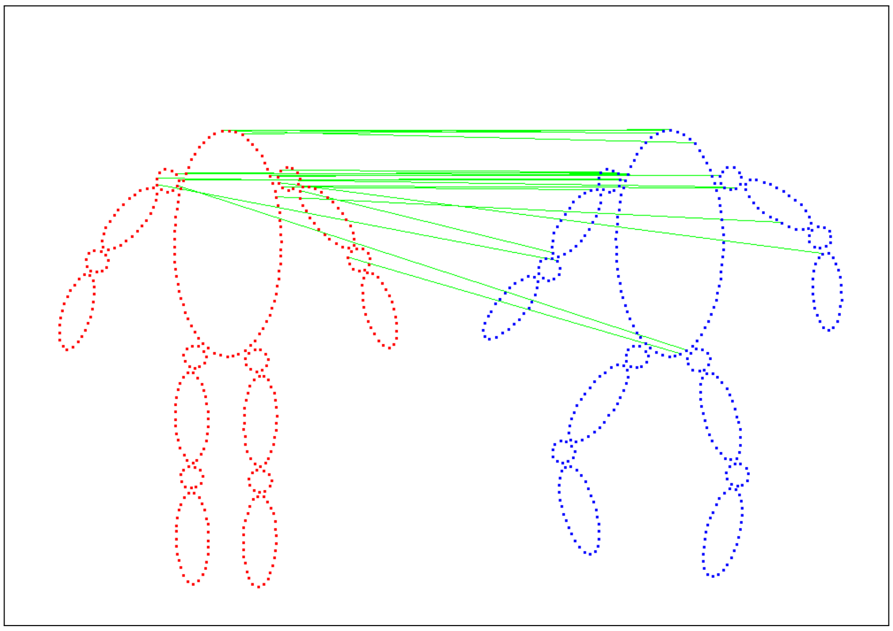


Figure 4.6: Visualization of the point correspondences depending on the most similar, reciprocal feature histograms using the chi-square distance between all points of C_1 (red) and C_2 (blue).

4.2.4 Detection of the largest rigid part

The dense point correspondences from the previous computation step (see subsection 4.2.3) may contain several errors. Therefore, RANSAC is applied as a next step to detect a single rigid transformation T that leads to the biggest overlapping point cluster of C_i and C_j. Thereby, in each iteration, 3 random correspondences are selected and used for the calculation of T which is applied on C_i to be translated on C_j. The number of iterations highly depends on the ratio of right and wrong point correspondences. Based on visual assessments, the number of iterations was set to 1000. Subsequently, clusters are grown from all corresponding points with an euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{p}_j)$ again below a predefined threshold τ. The procedure is applied both on C_i and C_j which results in two rigid parts as output (see figure 4.3.2).

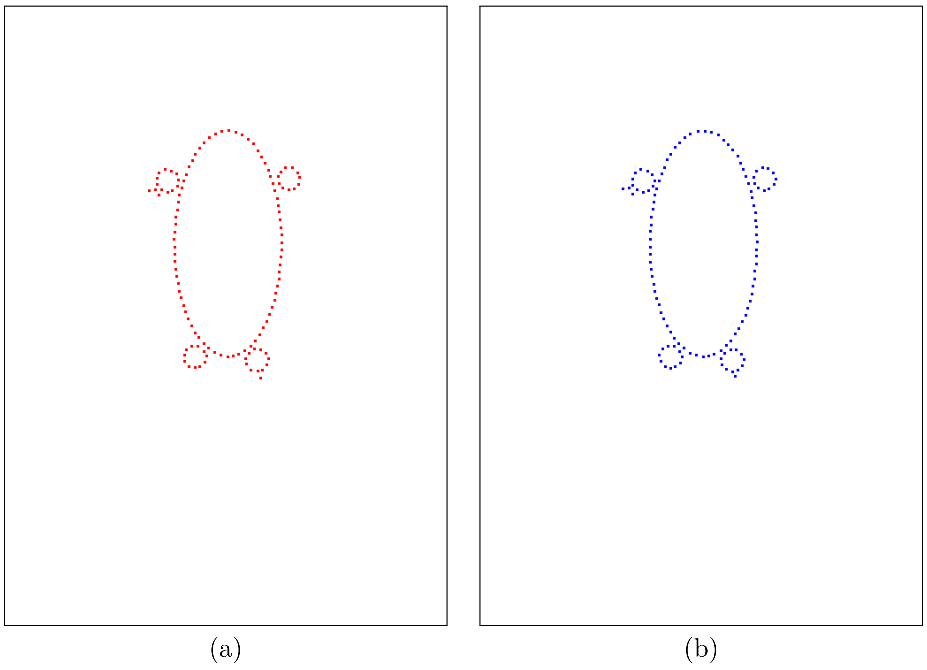


Figure 4.7: Computation of the largest rigid part from C_1 (a) and C_2 (b) by applying RANSAC on the detected point correspondences.

The final transformation T of the RANSAC approach, which leads to the “largest rigid part”, is applied on the reference cluster. This procedure is required in order to similarly align C_1 and C_2 for further computations (see subsection 4.2.5).

4.2.5 Cluster detection by region growing

After successfully detecting a “largest rigid part” P_i and P_j for each input clusters C_i and C_j they are added to a list of rigid parts $\mathcal{P}$. Potential linked rigid parts are detected from region growing of all unclustered points $\mathcal{U} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$. Those comprise all cluster points of C_1 and C_2 excluding already detected largest rigid parts $\mathcal{P}$. The region growing initiates with the first point $\boldsymbol{u}_1$ of the unclustered points $\mathcal{U}$ to form a cluster C_i. Another point of the unclustered points $\boldsymbol{u}_j$ is added to C_i, if the euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{u}_j)$ to any point in C_i is below the threshold τ. If no further unclustered points can be added, the region growing initiates again with the first unclustered point $\boldsymbol{u}_1$ that has not been added by region growing until all points traversed the procedure. The result is a set of clusters $\mathcal{C}$ for each C_i and C_j. Subsequently, a preliminary joint $\boldsymbol{j}_i$ for each output clusters is stored, by detecting the two nearest points of C_i and P_i. The joints are required for following cluster correspondence and joint weights for the ICP (see subsection 4.2.5 and 4.2.6).

Establishment of corresponding clusters

In case of detecting more than one cluster for each C_i and C_j, which might be for example the case for the extremities linked to the torso, it must be verified which clusters correspond to each other. This step is essential, as the algorithm is called recursively (starting from 4.2.3) with two new input clusters. Thereby, the provisional joints $\boldsymbol{j}_i$ are used to associate two clusters of C_i and C_j by detecting the closest joint with the euclidean distance $d(\boldsymbol{j}_i, \boldsymbol{j}_j)$.

4.2.6 Detection of linked rigid parts

In the reference paper the whole algorithm was reapplied on the linked clusters to the already detected rigid parts. However, this approach is not successful in the current 2D application. The reason is that the point features are not meaningful due to the reduced number of points and similar shapes and thus the feature histograms do not lead to sufficient correspondences. In order to iteratively detect further rigid parts linked to already detected ones, the knowledge is used that the rigid parts are transformed by rotating around a joint. For that purpose the estimated joints $\mathcal{J} = j_1, \cdots, j_n$ resulting from 4.2.5 are taken into account. As a first step, two input clusters C_i and C_j are transformed that their joints overlap. Next, the primary axis of C_i and C_j are aligned to guess an initial alignment. Then, C_i is iteratively rotated around its joint ranging in a certain threshold τ from the initial rotation. By computing a least squares error for each iteration, a rotation can be detected that best overlaps the linked rigid parts. Thereby, the preliminary joints are used as weights. As a consequence, a point p_i being located far away from its allocated joint j_n does not contribute as much to the matching error as points located near the joint. The weight w of a cluster point and the total matching error e

$$\begin{aligned} w_i &= \|\boldsymbol{p}_i - \boldsymbol{j}_n\| \\ e &= \sum_{i=1}^{m} \|\boldsymbol{p}_i - \boldsymbol{q}_i\|^2 \cdot \frac{1}{w_i} \end{aligned} \tag{4.11}$$

are achieved by combining the distance to the closest point and to the cluster's joint. For the calculation not only successful correspondences are taken as input but all points, that the error is expressive. The final correspondence equals to the biggest cluster resulting from rotation around the joint $\boldsymbol{j}_i$ with the smallest error e. As it is assumed, that two rigid parts from two different poses are quite similar, the distance threshold τ for detecting the linked part through region growing is considerable small.

4.3 Implementation

For the implementation in Java the individual steps of the largest rigid part algorithm have been split into individual classes for a better overview. Again ImageJ was used as processing library, which only gets a stack with two point clouds as input. The whole algorithm initiates from the `Segmentation` class, where all steps proposed in 4.2 are implemented. Basically, the segmentation proceeds until no unclustered points can be detected or no clusters need to be processed. In the beginnin of the loop already detected rigid parts are removed from the unclustered points (`removeAllLRPs()`). As a next step clusters are detected either for the initial point cloud or in case of linked parts for an LRP. Joints are only estimated for the clusters in case of a detected rigid part which is the case for all iterations except the first one. Then, all clusters are added to the `Stack` by detecting matching ones (`pushMatchingClusters()`). In case of an empty stack, all clusters have been processed and the algorithm terminates. On the other hand, two matching clusters are popped from the Stack and are segmented with the next steps.

```java
while (unclusteredReference.size() > MIN_SIZE || !clusters.isEmpty()) {

  removeAllLRPs();

  referenceClusters = RegionGrowing.detectClusters(unclusteredReference);
  targetClusters = RegionGrowing.detectClusters(unclusteredTarget);

  if (currentLrps != null) {
    detectJoints(currentLrps, referenceClusters, targetClusters);
  }

  if (referenceClusters.size() != 0 && targetClusters.size() != 0) {
    pushMatchingClusters();
  }

  if (clusters.isEmpty()) {
    return;
  }

  currentClusters = clusters.pop();

  if (currentClusters[0].getJoint() == null) {
    FeatureMatching fm = new FeatureMatching(currentClusters[0], currentClusters[1])
      ;
    Map<Integer, Integer> denseCorrespondances = fm.getCorrespondences();

    if (denseCorrespondances.size() < 3) {
    return;
    }

    RANSAC ransac = new RANSAC(currentClusters[0], currentClusters[1],
      denseCorrespondances);
    currentLrps = ransac.getLargestRigidParts();
    unclusteredReference = ransac.getTransformedReferencePoints();
  }

  else {
    PartDetection pd = new PartDetection(currentClusters[0], currentClusters[1]);
    currentLrps = pd.getLinkedParts();
  }

  if (currentLrps[0].getPoints().size() < 5) {
  } else {
  largestRigidParts.add(currentLrps);
  }
}
```

4.3.1 Feature Detection

As a first step, the normals of all points n of C_1 and C_2 are computed. For this matter the class `NormalEstimation` was developed that takes a point p_i with its k neighbors as input. Then, the least fitting error line on those input points is detected (as described in section 4.2.3) and the smallest lambda value λ_2 selected as eigenvalue. By setting the x-value of the normal to 1.0, the y-value can be calculated with the eigenvalue λ_2. The resulting vector represents the normal vector $\vec{n}$ for p_i. In case of being exactly vertically or horizontally, the resulting value for y is either infinite or not a value (Nan), in these cases the normal is set to (0,1) or (1,0). As a last step, the normal is normalized.

```
double eigenvalue = Math.min(lambda1, lambda2);

  covarianceMatrix = new double[][] {
    { a - eigenvalue, b},
    { b, c - eigenvalue}
  };

double[] normal = new double[2];

normal[0] = 1.0;
normal[1] = (eigenvalue * normal[0] - covarianceMatrix[0][0] * normal[0])/
    covarianceMatrix[0][1];

if(Double.isInfinite(normal[1])){
  normal[0] = 0;
  normal[1] = 1;
} else if (Double.isNaN(normal[1])){
normal[1] = 0;
}

double length = Math.sqrt(Math.pow(normal[0], 2) + Math.pow(normal[1], 2));

normal[0] /= length;
normal[1] /= length;
point.setNormal(normal);
```

A `ClusterPoint` was implemented to store the normal n_i for each point and the resulting feature histogram (FPFH) in form of a `Histogram` object containing a `int[]` array. For the calculation of a feature histogram of a point p_i, its k neighbors are taken into account. The `FPFH` class implements various operations for vectors, like a dot or cross product to implement the formula from subsection 4.1.

```
public void featureHistogram(p_i) {
  SPFH = SPFH(p_i);

  for (ClusterPoint p_k : p_i.getNeighborhood()) {
    double weight = p_i.distance(p_k);
    weightedSPFH = weightedSPFH.addHistograms(SPFH(p_k).multiplyHistograms(1.0 /
      weight));
  }

  FPFH = SPFH.addHistograms(weightedSPFH.multiplyHistograms(1.0 / p_i.
    getNeighborhood().size()));

  p_i.setFPFH(FPFH);
}
```

For the comparison of two feature histograms from C_1 and C_2 only unique ones are considered. Point correspondences are calculated by detecting a corresponding point from C_2 for an input point p_i of C_1 applying a distance on the feature histograms (see section 4.2.3). The distance can be chosen by the user.

```java
private int closestPoint(ClusterPoint p_i, List<ClusterPoint> c_2) {
  int closestPoint = -1;
  double distance = Double.MAX_VALUE;
  double distanceNew = 0;

  for (int i = 0; i < c_2.size(); i++) {

    if (Input.distance.equals("Euclidean")) {
      distanceNew = p_i.getFPFH().squaredDistance(c_2.get(i).getFPFH());
    } else if (Input.distance.equals("ChiSquared")) {
      distanceNew = p_i.getFPFH().chiSquare(c_2.get(i).getFPFH());
    } else {
      distanceNew = p_i.getFPFH().kullback(c_2.get(i).getFPFH());
    }

    if (distanceNew < distance) {
    distance = distanceNew;
    closestPoint = i;
    }
  }
  return closestPoint;
}
```

Reciprocal point correspondences between C_1 and C_2 are returned in form of point indices `Map<Integer,Integer>` of the clusters cluster points.

```java
for (Map.Entry<Integer, Integer> entry : reference.entrySet()) {
  Integer referenceIndex = entry.getKey();
  Integer targetIndex = entry.getValue();

  ClusterPoint currentRefPoint = originalReference.get(referenceIndex);
  ClusterPoint currentTargetPoint = originalTarget.get(targetIndex);
...
  if ((reciprocalMatching && target.get(targetIndex) == referenceIndex){

    finalReferencePoints.add(currentRefPoint);
    finalTargetPoints.add(currentTargetPoint);
    finalAssociations.put(referenceIndex, targetIndex);
  }
}
```

4.3.2 RANSAC

The RANSAC algorithm takes the computed dense correspondences between C_i and C_j in form of a `Map<Integer, Integer>` as input. As a first step, three random correspondences are selected from the map to calculate an affine transformation between the three resulting points from each C_i and C_j. The initial orientation and alignment is thereby irrelevant as the transformation T is completely recalculated.

```java
public LargestRigidPart(Cluster c_i, Cluster c_j, Map<Integer, Integer>
    correspondences)
...
points1 = c_i.getPoints();
```

Algorithm 4.1: Computation of the normal and subsequently feature histograms of a cluster point p_i with its k neighbors inside a radius r. The fast point feature histograms FPFH are computed by weighting the SPFH of a p_i and its k neighbors.

```
1:  FPFH($p_i$)
2:      $C_{max} \leftarrow ()$
3:      $C_{current} \leftarrow ()$
4:      $n \leftarrow sizeOf(M)$
5:      $m \leftarrow sizeOf(C_{current})$
6:      while $n > 0$ do
7:          $c_{current} \leftarrow c_{current} + \boldsymbol{u}_1$
8:          for $i = 1, \ldots, m$ do
9:              $M \leftarrow M - C_{current}$
10:             for $j = 1, \ldots, n$ do
11:                 if $d(\boldsymbol{p}_i, \boldsymbol{u}_j) < \tau$ then
12:                     $C_{current} \leftarrow C_{current} + \boldsymbol{u}_j$
13:                 end if
14:             end for
15:         end for
16:         $M \leftarrow M - C_{current}$
17:         if $m > sizeOf(C_{max})$ then
18:             $C_{max} \leftarrow C_{current}$
19:         end if
20:         $C_{current} \leftarrow ()$
21:     end while
22:     return $C_{max}$
23: end
```

```java
points2 = c_j.getPoints();
...
private void getRandomPoints(int num) {
  Integer[] keys = correspondances.keySet().toArray(new Integer[0]);
  Integer[] values = correspondances.values().toArray(new Integer[0]);

  for (int i = 0; i < num; i++) {
    index = (int) (Math.random() * correspondances.size());
    randomPoints1.add(points1.get(keys[index]));
    randomPoints2.add(points2.get(values[index]));
  }
}
...
```

In each iteration on all point correspondences a region growing approach with a threshold τ is conducted. The value is thereby considerably small as a right alignment during any iteration is assumed. The biggest cluster is stored and after all iterations returned as largest rigid part. The whole RANSAC procedure is quite time consuming. It can be reduced by taking a smaller number of correspondences as input which directly affects the required number of iterations until a right match is detected (see 4.4).

4.3.3 Region growing

The region growing algorithm is quite similar to the earlier described algorithm (see algorithm 0). There is also an adaptation, which does not only return the largest cluster, but all clusters above a certain size. The detected clusters are handled in a `Stack<Cluster>` in case of more than one detected clusters. Thereby, all clusters are pushed on the stack. With each recursion two corresponding cluster C_i and C_j are popped from the stack and taken as input for the whole algorithm. If again more clusters are detected they are pushed on the stack and treated before recently added clusters.

4.3.4 Joint rotation

The main part for the detection of linked parts to already detected rigid parts is the rotation of C_i onto C_j. As a first step both clusters are moved to the origin and are similarly aligned.

```
if (c_i.getJoint() != null) {
  referencePoints = Matrix.translate(c_i.getPoints(), -c_i.getJoint().getX(), -c_i.
    getJoint().getY());
  targetPoints = Matrix.translate(c_j.getPoints(), -c_j.getJoint().getX(), -c_j.
    getJoint().getY());
  initialOrientation(c_i.getJoint());
  }
```

As a next step, the reference points are aimed to be rotated onto the target points. This is achieved iteratively by applying a rotation in the direction of an achieved reduced error. By computing an error for each iteration, the assumed biggest overlap is achieved.

```
for (Map.Entry<Integer, Integer> entry : correspondences.entrySet()) {
...
  totalError += currentError / referencePoint.distance(new ClusterPoint(0,0));
...
}
```

To only remain the rigid part and reject correspondences from the final corresponding points that do not belong to the searched part, a region growing algorithm is applied to only keep the largest cluster as successful detected rigid part (see subsection 4.3.3).

4.4 Results

Applying the proposed approach on the 2D data set of an articulated object, following results could be achieved. Alternative results (reducing k neighbors, histogram bins, histogram criteria) could be achieved –> Compare runtime of criteria + number of right correspondences as percentage.

4.4.1 Histogram similarity criteria

The chosen distance between histograms is essential. Euclidean –> most correspondences, more iteration in RANSAC but certainly right more right ones. Chi-Square –> less correspondences, more right ones, less iterations with RANSAC Kullback-Leibler –> few correspondences, LRP might not be detected as too few right ones Not useful in this context, more points required

4.4.2 Main difficulties

The main drawback of the algorithm represents the first initial alignment of the two poses of the articulated object. In case of a failure, the linked parts can not be detected right as they are computed from region growing. By increasing the number of RANSAC iterations, the probability of a faulty initial alignment can be reduced, however this directly affects the runtime and should therefore not be exaggerated. Most certainly, it has to be guaranteed that the number of corresponding points are the same number from C_1 and C_2.

As the input point cloud of an articulated object is in 2D, the imitation of the object's hull is required. As a consequence, the region growing is much more error-prone, as unlikely in 3D, the points of a rigid part have a considerable lower number of neighbors. In case of a few missing cluster points from a rigid part, it will not be fully detected during the region growing due to the gaps which is especially drastically during the RANSAC approach. To counteract this behavior, more and closer hull points could added to each rigid part, that more links for region growing are available. A further main problem is that the algorithm proceeds iteratively from already detected rigid parts. This makes the whole procedure quite unstable and error prone, as one faulty detection leads to an overall unsuccessful segmentation of an articulated object. In the implementation of Mitra enough feature descriptors were available for linked rigid parts, that they could be detected similary as the initial alignment. But for that more data points have to be available which is not the case for this prototype.

Furthermore, touching of rigid parts (e.g. the hand touches the leg) constitute difficulties as the region growing would not detect those as potential linked clusters.

4.5 3D implementation

The next step would be to implement the approach in 3D. A similar implementation was done by Mitra [13] (see section 4.2.1) by using the PCL. Most essential functions like FPFH and subsampling for large point clusters are already provided. A good dataset would be thereby the SCAPE, which offers different poses of a scanned human.

Chapter 5

Conclusion

In the first project phase, intense research has been conducted in order to detect one main issue to focus on in the master project. For that, it was quite essential to form an overall perspective of the state-of-the-art regarding motion and pose estimation. During this process, the field of unsupervised pose estimation frequently arose. By following this direction, the non-rigid registration became a major indicator for possible optimizations. Taking existing methods as reference (see Chapter **??**), an own approach for 2D point clouds was developed, which should reduce the computation steps of detecting the rigid parts of an articulated non-rigid object. Basically, a divide-and-conquer approach was developed, which recursively divides two point-clouds of the same object in different poses into matching clusters. The subdividing was realized with a tree and depth-first traversal to segment the point clouds from the left to the right. As a next step, all neighboring clusters were verified to be merged, in case of having subdivided a rigid part. After the merging the rigid parts of the objects are detected.

5.1 Future work

- Machine learning of feature histograms - 3D implementation –> computation expensive - application puppet pose capture

References

Literature

[1] Dragomir Anguelov et al. “Recovering Articulated Object Models from 3D Range Data”. In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*. UAI ’04. Banff, Canada: AUAI Press, 2004, pp. 18–26 (cit. on pp. 6, 12).

[2] Dragomir Anguelov et al. “The Correlated Correspondence Algorithm for Unsupervised Registration of Nonrigid Surfaces”. In: *Advances in Neural Information Processing Systems 17*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. MIT Press, 2005, pp. 33–40 (cit. on p. 6).

[3] Simon Baker, Takeo Kanade, et al. “Shape-from-silhouette across time part ii: Applications to human modeling and markerless motion tracking”. *International Journal of Computer Vision* 63.3 (2005), pp. 225–245 (cit. on p. 5).

[4] Paul J Besl and Neil D McKay. “Method for registration of 3-D shapes”. In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–607 (cit. on p. 5).

[5] Will Chang and Matthias Zwicker. “Automatic Registration for Articulated Shapes”. *Computer Graphics Forum (Proceedings of Symposium on Geometry Processing 2008)* 27.5 (2008), pp. 1459–1468 (cit. on p. 6).

[6] Will Chang and Matthias Zwicker. “Range Scan Registration Using Reduced Deformable Models”. *Computer Graphics Forum (Proceedings of Eurographics 2009)* 28.2 (2009), pp. 447–456 (cit. on p. 6).

[7] Fernando De Goes, Siome Goldenstein, and Luiz Velho. “A hierarchical segmentation of articulated bodies”. In: *Computer graphics forum*. Vol. 27. 5. Wiley Online Library. 2008, pp. 1349–1356 (cit. on p. 5).

[8] Hao Guo, Dehai Zhu, and Philippos Mordohai. “Correspondence estimation for non-rigid point clouds with automatic part discovery”. *The Visual Computer* 32.12 (2016), pp. 1511–1524 (cit. on pp. 6, 21).

[9] Günter Hetzel et al. “3D object recognition from range images using local feature histograms”. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2001, pp. II–II (cit. on p. 27).

[10] Hugues Hoppe et al. *Surface reconstruction from unorganized points.* Vol. 26. 2. ACM, 1992 (cit. on p. 24).

[11] Marcus A Magnor. “Multi-Layer Skeleton Fitting for Online Human Motion Capture.” In: 2002 (cit. on p. 5).

[12] Brice Michoud et al. “Real-time marker-free motion capture from multiple cameras”. In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on.* IEEE. 2007, pp. 1–7 (cit. on p. 5).

[13] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. “Symmetrization”. *ACM Transactions on Graphics* 26.3 (July 2007) (cit. on pp. 2, 6, 23, 27, 28, 38).

[14] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration”. In: *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on.* IEEE. 2009, pp. 3212–3217 (cit. on pp. 23, 26).

[15] Radu Bogdan Rusu et al. “Persistent point feature histograms for 3D point clouds”. In: *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany.* IOS Press. 2008, pp. 119–128 (cit. on p. 23).

[16] Gary KL Tam et al. “Registration of 3D point clouds and meshes: a survey from rigid to nonrigid”. *IEEE transactions on visualization and computer graphics* 19.7 (2013), pp. 1199–1217 (cit. on p. 5).

[17] Eric Wahl, Ulrich Hillenbrand, and Gerd Hirzinger. “Surflet-pair-relation histograms: a statistical 3D-shape representation for rapid classification”. In: *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on.* IEEE. 2003, pp. 474–481 (cit. on p. 27).