Unsupervised Identification of the Rigid Parts of an Unknown Articulated Object

Anna M. Maureder

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2018

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, June 26, 2018

Anna M. Maureder

Contents

Abstract

The proposed work addresses the issue of identifying the rigid parts of an unknown articulated object to subsequently estimate the joints representing its pose. Most existing pose estimation methods take advantage of user inputs to estimate the joints' positions, for example, markers or an object model. However, methods that operate completely independent of manual user input constitute a great potential. One solution approach for this situation is proposed by the non-rigid registration of an articulated object in different poses, also referred to as template matching. Inspired by unsupervised state-of-the-art pose estimation procedures, two segmentation approaches are implemented. The main goal is thereby to reduce the number of computation steps for segmenting an articulated object into its rigid parts.

Kurzfassung

Die vorliegende Arbeit behandelt die Segmentierung eines unbekannten, artikulierten Objekts in dessen Körperteile, um anschließend die Gelenke und Pose zu beschreiben. Häufig angewandte Methoden basieren auf manuellen Benutzereingaben, welche als Hilfestellung bei der Segmentierung dienen. Dazu zählen etwa auf dem Körper platzierte Marker oder ein vorhandenes Referenzmodel. Hingegen bieten Methoden, die unabhängig von Benutzereingaben fungieren, großes Potential. Ein Lösungsansatz dafür stellt die Registrierung von einem artikulierten Objekt in zwei verschiedenen Posen dar. Inspiriert durch State-of-the-Art Methoden in diesem Bereich werden zwei Segmentierungsalgorithmen implementiert. Die Zielsetzung dabei ist, die Anzahl der Berechnnungsschritte für diese rechenaufwendige Prozedur zu reduzieren.

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
d_i	Divider position of C_i
θ	Orientation of C_i
$\boldsymbol{p}_i(x, y)$	2D cluster point of C_i
$\boldsymbol{p}_i(x, y, z)$	3D cluster point of C_i
$\boldsymbol{c}_i(x, y)$	Centroid of C_i
$\boldsymbol{jt}_i(x, y)$	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	Squared error distance between two clusters
e_{avg}	Average error distance per point between two clusters
τ	Threshold
N	Node in a tree
left	Left child of N
right	Right child of N
$\mathcal{L}$	Set of matching sub cluster pairs
$L_{i,j}$	Sub cluster of $\mathcal{L}$
LRP	Largest rigid part P of a cluster C_i
$\mathcal{C}_U$	Set of unmatched clusters
C_U	Cluster with no allocation to a rigid part P
$\mathcal{U}$	Set of unclustered points
$\boldsymbol{u}_i(x, y)$	Unclustered point
r	Radius

$\boldsymbol{n}_i$	Normal vector of a point $\boldsymbol{p}_i$
H_i	Feature histogram of a point $\boldsymbol{p}_i$
H_μ	Mean feature histogram of a cluster C_i

Chapter 1

Introduction

Pose and motion estimation of articulated objects is a fundamental task in Computer Vision due to the progressive digitalization of day-to-day processes. A variety of practical applications exist, such as activity recognition, video surveillance and human-computer interfaces. An application the thesis emerged from is the pose capture of a real-world articulated object used as input for a digital animation process. By detecting the object's associated rigid parts and joints in two consecutive poses, the animation between transformed rigid parts can be determined and applied on a digital character.

1.1 Problem Statement

Generally, pose estimation of an articulated object can be described as a segmentation problem, as the individual rigid parts and joints linking them are desired. The vast majority of existing pose estimation approaches take advantage of assumed object models in 3D, manually-labeled joints and rigid parts to determine basic information about an object's pose. Often combined with a machine-learning approach, the results are promising after a completed training phase. However, methods that detect the pose of an unknown object are great possibilities, as the pose estimation operates completely independently of user input. Among those methods, the non-rigid registration (see Section 2.1) is a well-known approach. The input of this algorithm comes from two or more poses of one articulated object. By merging one *template* pose with another *query* pose, part correspondences can be determined and the articulated object is segmented into its rigid parts. By applying this approach to an unknown input consisting of an object in two poses, five core questions are formulated to be considered:

- In which form is the input data for pose estimation received?
- Which data points correspond to one another in two different poses?
- To which rigid part can a data point be assigned to?
- How can the joints, linking detected rigid parts, be estimated?
- Which joints/rigid parts correspond to one another in two different poses?

Many challenges emerging from different stages of the pose estimation procedure have to be overcome (see Section 2.2). First of all, digital input data of an articulated object in the real world has to be captured. Thereby, input noise emerging from low resolu-

tion scanning technologies is an essential factor which has to be considered for pose estimation. Furthermore, the ambiguity of body parts poses a significant difficulty, especially if the articulated object is composed of symmetric body parts. One of the main drawbacks of current methods is the computational-expensive procedure to detect rigid parts. The root of this problem originates from the detection of correct point correspondences between two input meshes. As this directly influences the run time, there is a great demand for improvements, particularly if real-time applications are required.

1.2 Goal

By means of current approaches in this particular field (see Chapter 2), this thesis addresses the issue of detecting the initial pose of an unknown articulated object given in two poses. The focus lies thereby on reducing the number of computation steps of the segmentation procedure. A detailed overview of the used notations can be found in the Chapter before. The main goal can be formulated as segmenting an articulated object M into its unknown number n of rigid parts $\mathcal{P} = \{P_1, \ldots, P_n\}$ and extracting all m joints $\mathcal{J} = \{\boldsymbol{jt}_1, \ldots, \boldsymbol{jt}_m\}$ linking those parts. The input poses are represented by two point clouds C_1 and C_2 of M in two different poses being composed of 2D points in the form of $\boldsymbol{p}_i(x, y)$. It is assumed that the digitalized poses are already available, to fully focus on the segmentation. C_1 is 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 set of unsorted points of C_1 and C_2 is present. No further information, such as manual labeling by the user or an object model as indicator for the number of rigid parts, are available. The only assumption that can be made is that M only consists of rigid parts that can not be deformed or stretched. 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. It is also taken advantage of the knowledge that points located on a rigid part P_i undergo the same transformation T_i.

1.3 Methodology

To accomplish the proposed goal, an analysis of state-of-the-art pose estimation approaches is conducted in Chapter 2. Additionally, the concept of surface registration is presented (see Section 2.1) to provide necessary background knowledge on object segmentation. Consequently, two segmentation approaches are implemented taking unsupervised methods (see Section 2.4) into consideration. Although the referred pose estimation approaches are performed on 3D data sets, it was a conscious decision to conduct the implementation on 2D point clouds. The main advantages include fewer degrees of freedom of the data and a possible pose estimation in the absence of 3D reconstructed data. Furthermore, attention can be brought to an implementation of potential improvements.

The first approach is a straightforward and linear method (see Chapter 3) that aims to drastically reduce the computation steps of previous approaches. This is achieved by iteratively subdividing C_1 and C_2 with a “divide and conquer” approach. Corresponding

sub clusters are verified to match; in negative cases they are further subdivided. This attempt only depends on all point coordinates of C_1 and C_2 and the orientation θ of the clusters. In the case of many linked parts or too dissimilar transformations between C_1 and C_2 this approach is not reliable, as clusters being compared are not actually corresponding to one another. To address the segmentation with a focus on articulated objects with a typical skeleton structure (e.g. a human), a feature-based approach (see Chapter 4) is implemented. In this case, additional descriptors are extracted for all points of C_1 and C_2. These assist the initial alignment of the input clusters to detect a reliable corresponding rigid part. Proceeding from there, joints can be estimated and all linked rigid parts are detected iteratively. The main reference paper for this approach is from Guo et al. [11]. The main contribution of the proposed feature-based segmentation approach is the reduction of computation steps considering the recursive detection of linked rigid parts. In contrary to [11], the feature computation and matching is only applied for the first alignment. Then, motion information about the rigid parts is taken into account to stepwise align them in two poses. By demonstrating the outcome from the proposed approaches (see Section 4.4), evident drawbacks, main difficulties and possibilities are outlined. Furthermore, planned future work is proposed to compensate originated difficulties (see Chapter 5). All of these things offer a solid base for further improvements in this area.

Chapter 2

State-of-the-Art Pose Estimation

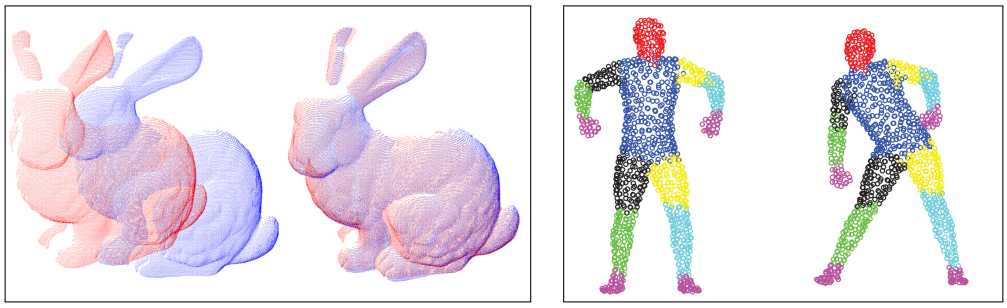
To answer the posed questions (see Chapter 1), attention during research was directed towards pose estimation of articulated objects. The majority of the selected reference papers focus on the pose extraction of a human body. Depending on the approach, different steps are required; these can generally be subdivided into the digitalization of the object to be captured (see Section 2.2.1) and its segmentation into rigid parts (see Section 2.2.2). Regardless of the chosen approach, many difficulties must be overcome, in order to estimate the joints and pose. A key role of the pose estimation method is the registration of surfaces (see Section 2.1). It is commonly referred to as an optimization problem, as the goal is to obtain the best possible outcome.

2.1 Surface Registration

Generally, registration in Computer Vision and Computer Graphics refers to the alignment of overlapping parts of two or more digital data sets [28]. The most essential component is the detection of point correspondences between two surfaces to be registered, often supported by RANSAC [9]. One main application is the alignment of two or more incomplete range scans of an object from different view points to obtain a complete model. Further applications are symmetry detection, subpart identification and articulation of non-rigid objects. For this reason, a vast number of pose estimation and skeleton extraction approaches rely on a successful registration of the digitalized objects.

2.1.1 Functionality

In general, a discriminatory approach between rigid and non-rigid registration can be used. In the former case, it is assumed that two surfaces are related by a rigid transformation, which can be seen in Figure 2.1 (a). A well-known approach for a rigid registration is the *Iterative Closest Point* (ICP) [5]. It requires a similar initial position of two shapes to avoid a local optimum. For this purpose it is often taken advantage of the *Principal Component Analysis* (PCA) [33] of shapes. With each iteration step the point correspondences between two input objects are updated by selecting the closest points. In the next step, the rigid transformation between two shapes is recalculated considering the detected correspondences. A matching error e is achieved, which states



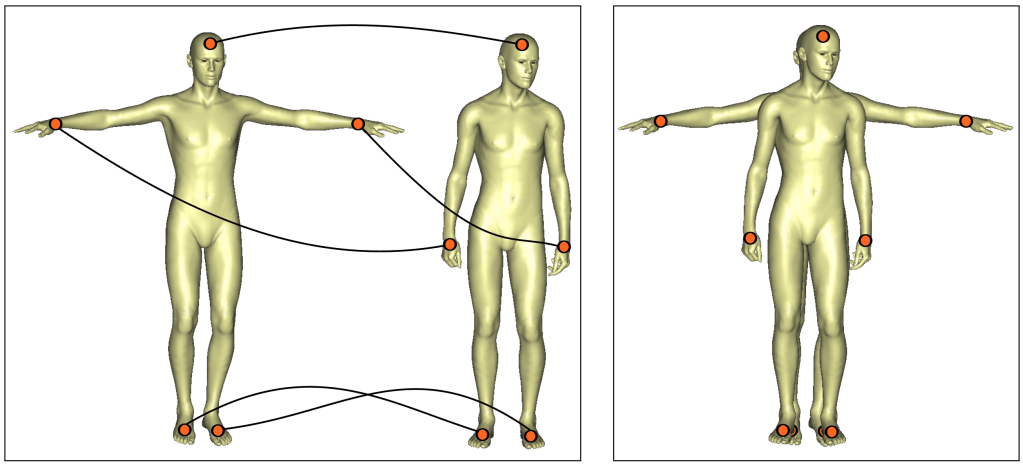
(a) (b)

Figure 2.1: Rigid registration of two similar objects (a) [21] and non-rigid registration of an object [10] in two different poses composed of several rigid parts (b).

the total Euclidean distance between the associated points of the registered shapes. The algorithm terminates after a predefined number of iterations or if a specified value for e is achieved. In the case of two non-rigid surfaces composed of multiple rigid parts (e.g. a human), a rigid registration will not lead to a convincing result, because the individual rigid parts may undergo different rigid transformations. In this case, a non-rigid registration is required to perform a segmentation into rigid parts, which can be seen in Figure 2.1 (b).

2.1.2 Difficulties

Some factors complicate the accomplishment of a visual successful registration. Main difficulties are noisy data and outliers that can often arise from low resolution scans. Furthermore, there might only be a limited amount of overlapping data, which is the case for multiple incomplete scans of an object from different view points. Self occlusion and variations from initial poses are factors that also complicate a successful registration. In the case of a non-rigid registration, the main difficulty is the establishment of point correspondences between two poses, because several transformations of the rigid parts must be considered (see Figure 2.2). Specific constraints can be set to reduce the correspondence space. These include, for example, the computation of point features, which are quantities that add additional information besides its coordinates to a point. By applying feature matching between two shapes, possible point correspondences can be detected. Saliences also play a major role in correspondence detection; these are represented by points that are considerably different than neighboring points. By placing markers on the object or assuming topology, the allocation of points to rigid parts can be simplified. Also, the prior information of articulation places a valuable contribution, as no deformations have to be considered. One main drawback of the non-rigid registration is that it is expensive and time-consuming to compute, due to the fact that the corresponding body parts of two poses must be detected iteratively. Additionally, the inevitable difficulty of detecting the global optimum related to ambiguous body parts is present.



(a) (b)

Figure 2.2: The main difficulty of the non-rigid registration is the detection of point correspondences between two input shapes (a). It is the basis for an alignment of one object in different poses (b) [31].

2.1.3 Optimization

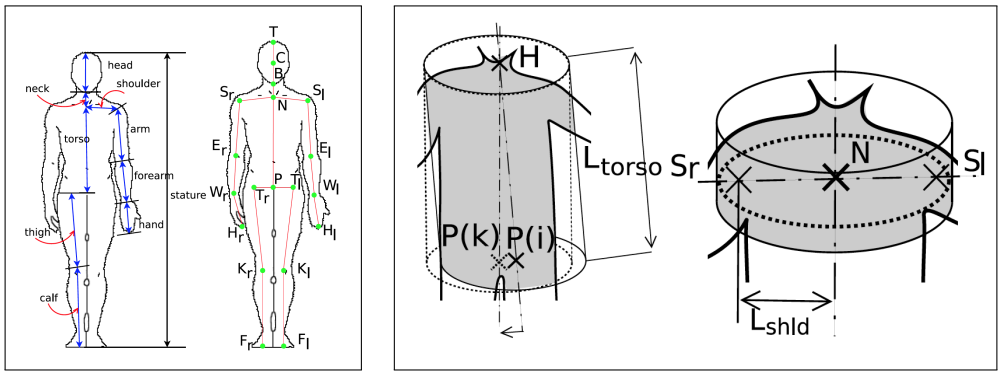
Several optimization approaches are proposed to overcome certain of the stated difficulties. One established method, regarding the detection of correspondences, is the *Expectation-Maximization* (EM), which alternates between two steps: detecting correspondences and optimizing the transformation. To avoid local minima a global optimization can be pursued, for example, by forming a decision tree. Furthermore, stochastic optimization takes into account statistics and probabilities. Popular related methods apply voting and belief propagation, such as *Markov Random Fields* (MRF).

2.2 Pose Estimation of articulated Objects

To successfully estimate the pose of an articulated object, generally two main steps must be performed: the digitalization of the object and the segmentation of the digital model into its rigid parts. Although 3D scanners are an approved method for digitizing real world objects, reconstruction from 2D images are gaining importance. As this thesis focuses on the segmentation of the input data into rigid parts, the digitalization of the object will not be covered in detail.

2.2.1 Digitalization of the Object

As a first step, the object to be captured must be digitalized for the subsequent segmentation step. Through scanning, the real shape is collected as either 2D or 3D data. The raw data in the form of 2D images, video streams, or even point clouds in 2D or 3D can be directly used for pose estimation. Multiple commercial 3D scanners are available; they strongly vary in precision and resolution depending on the cost. Usually, a subsequent reconstruction step is performed that converts the raw data into a mesh. This



(a) (b)

Figure 2.3: Estimating the pose of an articulated object with an object model, providing prior information about the rigid parts (a). By fitting shapes to a digitized model, the joints can be determined (b) [18].

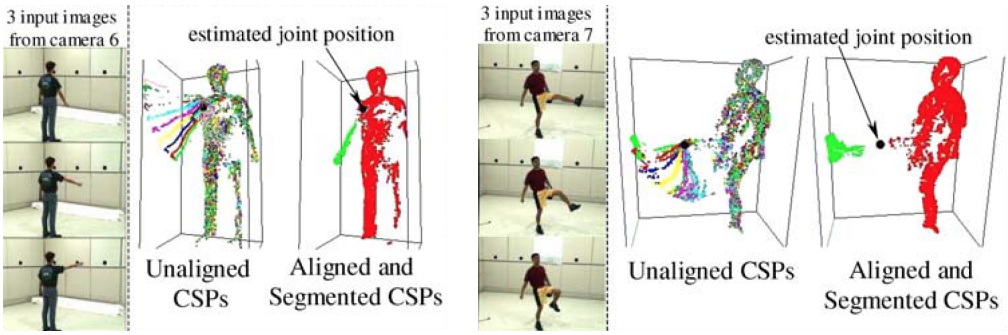
procedure frequently uses the registration of multiple scans from different view points. RGB-D sensors have become more significant for 3D reconstruction [10], because they are easily accessible and inexpensive. Furthermore, 3D reconstruction from images and videos is frequently made use of because no expensive 3D scanners are required. These methods include *Shape from Silhouette* [4, 26], *Shape from Shading* [15] as well as *Shape from Motion* [17]. Ramakrishna [22] performs a 3D human pose reconstruction by manually placing 2D landmarks on single images. Certain approaches skip the digitization step and take a 3D mesh from a modeling software as input.

2.2.2 Segmentation

The crucial Computer Vision task of the pose estimation is the segmentation of the digitized object into its rigid parts, in order to estimate the joints and pose. The main idea is to allocate each data point of an input object to a rigid part. For this process, any prior information in addition to the input data is indispensable. Regarding the non-rigid registration, the same object in another pose is required, which is usually referred to as a *template*. Generally, the segmentation approaches are classified into supervised (see Section 2.3) and unsupervised methods (see Section 2.4), whereby the former depends on manual user input.

2.3 Supervised Methods

Supervised methods for pose estimation greatly simplify the segmentation procedure, as certain assumptions of the object can be made. Examples include the placement of markers on the real object or the digitized model in order to label its joints, linking the rigid parts (see [20]). Furthermore, the usage of an *object model* is frequently employed to obtain prior knowledge about the number of rigid parts, and possibly the shape of an object (see [20, 29, 35]). Michoud et al. [18] propose a shape fitting approach with prior knowledge of the object to be captured (see Figure 2.3). Other approaches use



(a) (b)

Figure 2.4: Detection of joints of an object by sequentially moving the rigid parts one by one (a), (b) [4].

the motion information of known point correspondences from image sequences (see [14, 34]). The approach proposed by Baker et al. [4] sequentially estimates each joint by the person being captured moving one body part at a time. By extracting and registering the resulting CSPs (*Colored Surface Points*), a joint can be estimated (see Figure 2.4). Further approaches utilize training data and machine learning to estimate the pose of a human (see [25]). The approach by Lifshitz et al. [16] uses a *Convolutional Neural Network* (CNN) to train a key point detector on 2D images.

2.4 Unsupervised Methods

Although the supervised pose estimation approaches from Section 2.3 result in promising results, they depend on manual user input to be specified before the segmentation. Thus, they are either time consuming, inconvenient or restrict themselves to specific articulated objects (e.g. humans). Therefore, unsupervised methods are proposed that work independently from user inputs. These are applicable for unknown articulated objects without having prior knowledge about the topology. The proposed methods only require an articulated object in two different configurations as input. As those assumptions conform to the goal of this thesis, all reference papers are listed under related work (see Section 2.4.1).

2.4.1 Related Work

A main approach for non-rigid registration is proposed by Anguelov [1] that applies the *Correlated Correspondence Algorithm* [3]. The algorithm takes a *template* mesh of an object M_0 and any number n of meshes of the same object $M_1, \ldots, M_n$ in different configurations as input. The algorithm then performs a *Markov Network* with loopy belief propagation and Expectation-Maximization to iterate between finding a decomposition of the *template* into rigid parts, and detecting them in the other meshes. Thereby, it takes advantage of the PCA and ICP. Furthermore, a random clustering is applied to facilitate the detection of associated rigid parts (see Figure 2.5). Another approach proposes the recursive detection of body parts by the *Largest Rigid Part Algorithm*

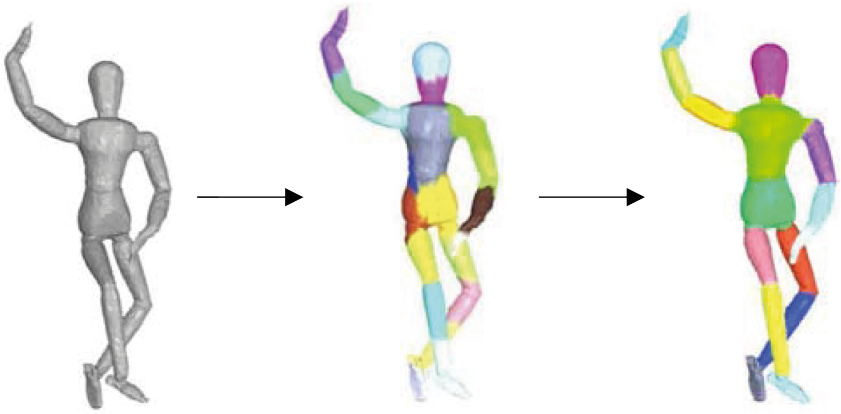


Figure 2.5: Segmentation of a template mesh M into its rigid parts by applying random clustering and a probabilistic framework to iteratively detect associating rigid parts in another mesh [1].

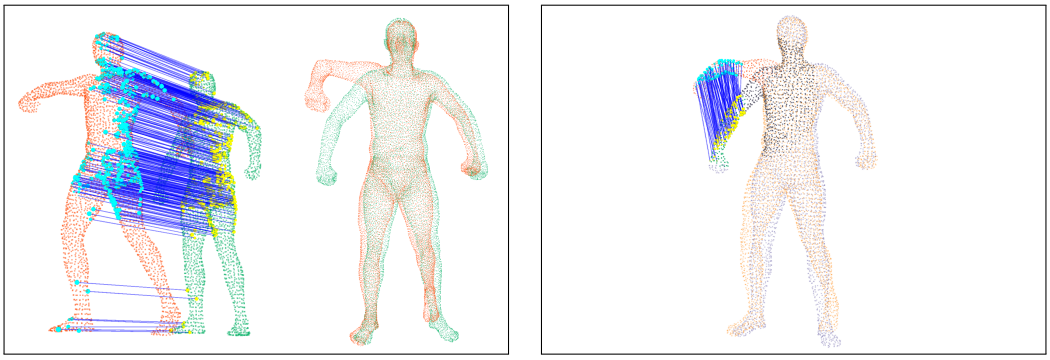


Figure 2.6: Detecting the largest rigid part of an object by detecting point correspondences by feature matching and applying RANSAC (a). Linked parts to the detected LRP are detected by region growing and reapplying the algorithm (b) [11].

(LRP) [11]. It discovers the rigid parts of an object in different configurations by initially detecting the largest rigid part. This is represented as the biggest overlapping point clusters between two poses by applying a single rigid transformation. To achieve this, sparse correspondences are detected by performing feature matching in combination with RANSAC. Proceeding from a detected LRP, linked rigid parts are recursively searched by growing clusters from all remaining unclustered points and reapplying the algorithm (see Figure 2.6).

Chang et al. [6] developed an approach for the alignment of articulated shapes from range scans with missing data in different poses. They sampled the motion of corresponding points by matching feature descriptors of both shapes. The generated

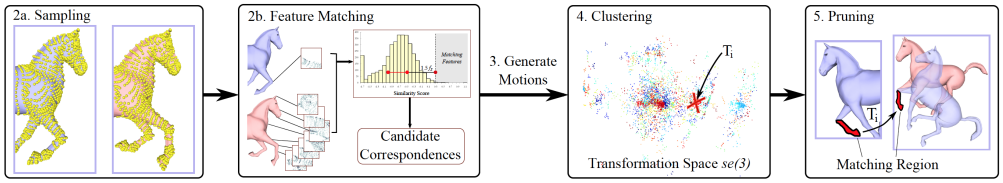


Figure 2.7: Alignment of articulated objects in the form of incomplete range scan by feature matching and clustering of generated motion [6].

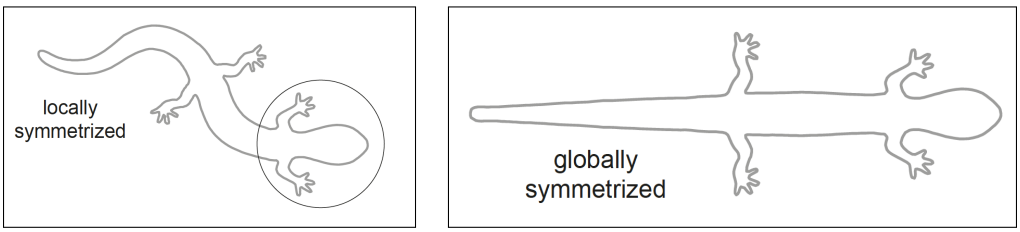


Figure 2.8: Detection of the rigid parts of an object by local (a) and global (b) Symmetrization [19].

motion in the form of transformations is clustered to detect the most optimal shape alignment (see Figure 2.7). A related approach by Chang et al. [7] proposed skinning weights to the motion. Additionally, Expectation-Maximization is applied to update those weights and the calculated transformation.

Another segmentation approach relies on Symmetrization [19], by detecting and aligning the body parts' symmetry axes (see Figure 2.8). De Goes et al. [8] detect consistent segments from a single pose by computing the diffusion distance of a surface. Another unsupervised method poses the probability-based registration of an articulated human body in the form of voxels [27]. By fitting splines to the data, the input object can be segmented into its rigid parts.

2.4.2 Main Drawbacks

The proposed approaches achieve convincing results concerning the accuracy of the segmentation and the detection of rigid parts. However, they all require a considerable number of computation steps to iteratively detect point correspondences and subsequent rigid parts from an object in two poses. This reflects on the run time of the algorithm, which offers therefore great potential for improvements (e.g. to allow pose estimation in real-time). Using the existing methods as a reference (see Section 2.4.1), two segmentation approaches are proposed. Thereby, the main focus is to reduce the number of computation steps of the *Correlated Correspondence Algorithm* [3] and the *LRP Algorithm* [11]. To fully focus on the segmentation of an articulated object into its rigid parts, the data input in the form of a 2D point cloud is assumed to be available.

Chapter 3

Linear Approach

Analyzing the reference papers' [1, 11] main drawbacks concerning the unsupervised segmentation, the first implemented approach aims to considerably reduce the number of computation steps. Unlike previous approaches, point correspondences are not sought in order to segment the articulated object. Instead, a segmentation is performed, in order to find point correspondences. The approach operates in a straightforward way, as it linearly subdivides an articulated object M in two different configurations (referred to as C_1 and C_2) into sub clusters. The subdividing of these clusters proceeds until all sub clusters can be matched (see Section 3.2). Thereby, it is taken advantage of the PCA to associate sub clusters of C_1 and C_2.

3.1 Cluster Detection by Region Growing

First, the resolution of M is estimated, in order to have an indicator for the specific threshold values. Thus, ten random points are selected from M and the distance to their closest points are calculated. Next, the median value is taken as the resolution to avoid distortion in the case of outliers. Then, the initial goal is to remove possible noise and outliers to proceed the segmentation approach with the two clusters C_1 and C_2. This is achieved by applying region growing on all points of M. A cluster C_i is grown from an unclustered point $\boldsymbol{u}_i(x, y)$. Another point $\boldsymbol{u}_j(x, y)$ is added to the cluster C_i if the Euclidean distance between them $d(\boldsymbol{u}_i, \boldsymbol{u}_j)$ is below a predefined threshold τ. As a next step, all points of C_i are iteratively compared to the remaining unclustered points, in order to allow the cluster to grow further. Once, all points of C_i have been treated and no further points are added to the cluster, any unclustered point is used as a seed to grow another cluster C_j. If there are no unclustered points left, the cluster with the highest number of points n is selected as an input cluster for the segmentation algorithm. The remaining clusters are classified as noise and rejected for further computations. The region growing is performed for both configurations of M in order to proceed to the segmentation with the clusters C_1 and C_2 (see Figure 3.1).

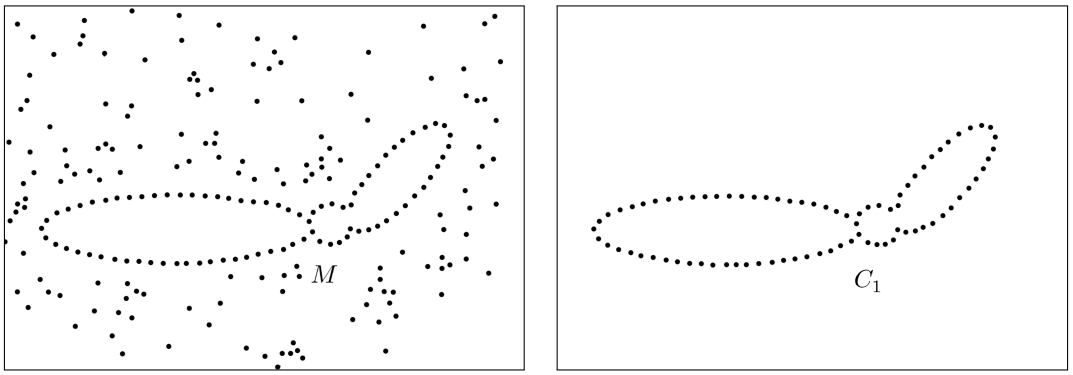


Figure 3.1: Taking an articulated mesh M in the form of a point cloud as input (a), noise is removed to achieve the input clusters C_1 (b).

3.2 Subdividing into Clusters

The first step of the subdividing process is the computation of the principal axes p_1 and p_2 of C_1 and C_2. They are required to offer a similar divider position base for the subdividing procedure. For that, the orientation θ of all clusters, that is

$$\theta(\mathcal{C}) = \frac{1}{2}\tan^{-1}\left(\frac{2 \cdot \mu_{11}(\mathcal{C})}{\mu_{20}(\mathcal{C}) - \mu_{02}(\mathcal{C})}\right), \tag{3.1}$$

is calculated by computing their central moments

$$\mu_{pq}(\mathcal{C}) = \sum_{(x,y)\in\mathcal{C}} (x - \bar{x})^p \cdot (y - \bar{y})^q. \tag{3.2}$$

The divider positions d are then determined by computing the perpendicular secondary axes s_1 and s_2 to p_1 and p_2 through the centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. 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 $C_{2,2}$. The association of clusters between C_1 and C_2 is determined by the sub clusters that are located to the left or right of s_1 and s_2 (see Figure 3.2). In each iteration step two related sub clusters are then verified to match. By applying the ICP on two associated clusters $C_p = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m\}$ and $C_q = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m\}$, a certain matching error e is computed between their associated cluster points. To eliminate the dependency between the matching error and the number of cluster points m, the average error per point of C_p and C_q, that is

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

is calculated. This is assuming that the two clusters C_p and C_q contain the same number of cluster points m. In the case of varying point amounts, excessive points are not considered in the error amount calculation. Two clusters C_p and C_q are stated to match, if $e_{\text{avg}} < \tau$. It is quite essential to determine an appropriate threshold τ, taking the resolution of the input data into account. In the case of being overvalued, clusters are

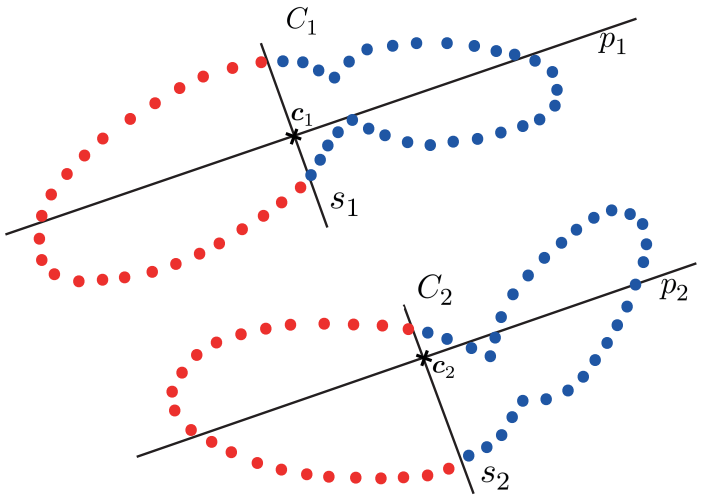


Figure 3.2: Subdividing C_1 and C_2 into two sub clusters by computing the secondary axes s_1 and s_2 perpendicular to p_1 and p_2 through the centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Associated clusters are visualized in the same color: red for both left clusters and blue for both right clusters.

more likely to match, potentially resulting in insufficient subdividing. On the contrary, it becomes increasingly unlikely that clusters match. This will result in further subdividing and the detection of too many rigid parts. If the matching between two clusters does not succeed, they are both further subdivided into two sub clusters. 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 sub clusters of C_1 match the sub clusters of C_2. These are then stored in a list $\mathcal{L}$ sorted by their actual location resulting from a cluster tree (see Section 3.5.5).

3.3 Merging Sub Clusters to Rigid Parts

In the next step, adjacent sub clusters from $\mathcal{L}$ are iteratively merged and subsequently verified to ensure they still match. This process is required to rejoin, if necessary, segmented sub clusters to the rigid parts of the articulated object (see Figure 3.3). This is the case, if a rigid part was subdivided during the previous subdividing step (see Section 3.2). The merging begins with the first set of associated sub clusters in the list $(L_{1,i},\ L_{2,i})$ and their adjacent sub clusters $(L_{1,i+1}, L_{2,i+1})$. If the resulting merged clusters can be matched in terms of the matching error e_{avg}, the merging proceeds with the following cluster set $L_{1,i+2}, L_{2,i+2}$. If not, the merging is not executed and $L_{1,i}$, $L_{2,i}$ are stored in a list of resulting rigid parts $\mathcal{P}$. The merging procedure then initiates with $L_{1,i+1}, L_{2,i+1}$. The process terminates if all clusters of $\mathcal{L}$ are processed and consequently all sub clusters are assigned to rigid parts $\mathcal{P} = \{P_1, \ldots, P_m\}$.

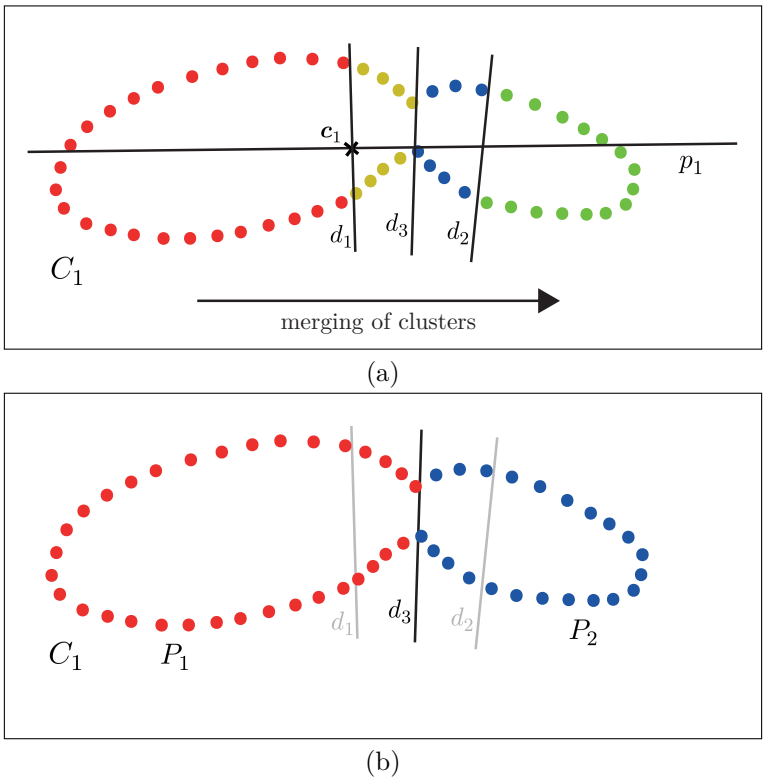


Figure 3.3: Merging of neighboring sub clusters from C_1 stored into $\mathcal{L}$ (a) into matching rigid parts $\mathcal{P} = \{P_1, \ldots, P_m\}$ (b).

3.4 Joint Estimation

After detecting the rigid parts $\mathcal{P} = \{P_1, \ldots, P_m\}$, the joints linking them are estimated. As an initial approach, the points of intersection between all principal axes of $\mathcal{P} = \{P_1, \ldots, P_m\}$ are computed, which are determined to represent the joints. However, this calculation assumes that the rigid parts are symmetric; as in the other case the principal axes may not represent the skeleton of a rigid part. For this reason, another approach must be taken into account. Anguelov [1] declares a joint $\boldsymbol{jt}(x, y)$ as a point belonging to two neighboring rigid parts P_i and P_j that undergoes the same transformation $T_i(\boldsymbol{jt}) = T_j(\boldsymbol{jt})$. In the current implementation a cluster point is only allocated to one rigid part P. An improvement of the current situation is therefore to select a desired number of closest points between neighboring rigid parts and calculate the average point representing the joint $\boldsymbol{jt}$ (see Figure 3.4).

3.5 Implementation

In order to primarily focus on a potential optimization of current segmentation approaches, the proposed algorithm is implemented in 2D. The input for the segmentation is a 2D point cloud of an articulated object. Similar to 3D point clouds, it is represented by its surface, which is described by its hull.

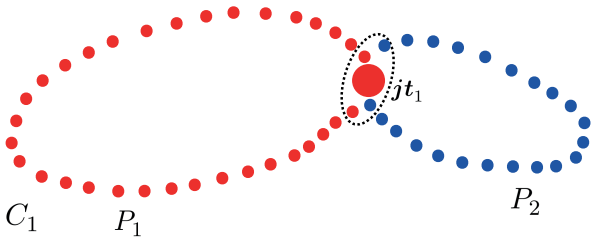


Figure 3.4: Estimation of the joint $\boldsymbol{jt}_1$ located between the detected rigid parts P_1 and P_2 of C_1 by selecting the four closest points and calculating an average point.

3.5.1 Chosen Environment

Java is chosen for the programming environment, using ImageJ[1] as an image processing library. The environment depends on the following factors:

- familiarity and prior experience,
- complexity,
- availability of plug-ins for image processing.

As ImageJ is mainly used for 2D-use cases, another implementation would be possible in 3D using PCL in C++ (see Section 4.5). As a result, attention could be brought to 3D segmentation and visualization of articulated objects.

3.5.2 Overview

The algorithm was split into several java classes to separate the individual steps within the algorithm from one other. The starting point of the algorithm is represented by the class `Segmentation`. As input it solely requires a stack of two 2D images indicating the point clouds of a mesh M in two different poses. As a first step, all potential cluster points are detected by iterating over the 2D images. An unclustered point $\boldsymbol{u}_i(x, y)$ is determined by a pixel colored in black, and is stored as a `ClusterPoint` object with its image coordinates. Next, possible noise is removed from the input meshes, represented by the detected cluster points. The result is two point clusters C_1 and C_2 which represent the articulated object in two poses. A class `Cluster` was implemented to store a cluster C_i with all of its points, its centroid $\boldsymbol{c}_i$, its resolution, the orientation θ as well as the principal and secondary axes p_i and s_i. For the simultaneous subdividing of the clusters C_1 and C_2 into sub clusters(see Section 3.2), a `ClusterTree` class was implemented (see Algorithm 3.2). Each node N contains thereby two associated clusters $C_{1,i}$ and $C_{2,i}$. The `Registration` class is applied on two associated clusters from a node N, which registers them by taking advantage of the ICP and Procrustes fitting. The recursive subdiving approach returns a list of matching sub clusters that are subsequently merged to rigid parts (see Algorithm 3.3). The `Visualization` class is responsible for displaying the sub clusters and rigid parts in different colors, and drawing PCA related components,

[1] http://imagej.net

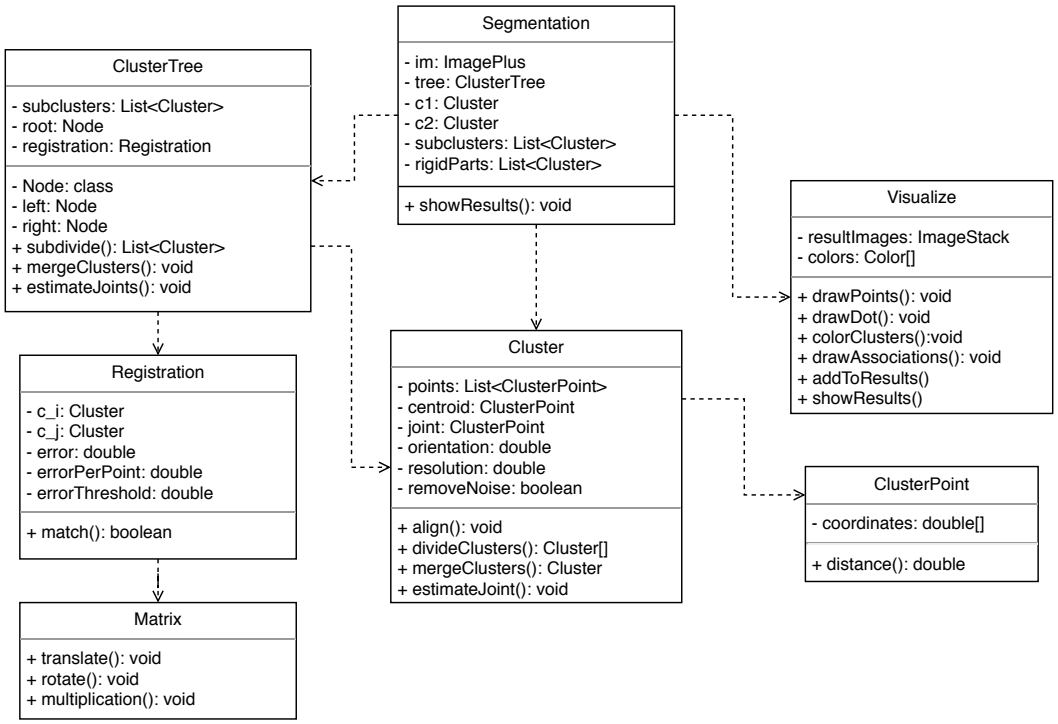


Figure 3.5: UML diagram of the classes related to the implementation of the linear segmentation approach.

such as the axes and joints of the rigid parts. The `Matrix` class provides operations for performing transformations on clusters. An overview of the architecture can be seen in Figure 3.5.

3.5.3 Region Growing

One main algorithm to remove outliers is the region growing from unclustered points given as input from a mesh M (see Algorithm 3.1).

3.5.4 PCA

One main factor, when subdividing a cluster C_i, is the computation of its secondary axis, which represents the divider d. To simplify the segmentation of C_i into a left and right sub cluster, it is horizontally aligned. This is achieved using a rotation Matrix with the negative orientation of the cluster (see Section 3.2). As a first step, the cluster is translated so that the desired rotation point (which is the centroid of the cluster) is located at the origin of the coordinates. After applying a rotation matrix with the negative orientation as angle on the cluster's points, it is translated back to the initial position:

```
public List<ClusterPoint> alignAxis(double orientation) {
  points = Matrix.translate(points, -centroid.getX(), -centroid.getY());
  points = Matrix.rotate(points, orientation);
  points = Matrix.translate(points, centroid.getX(), centroid.getY());
```

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

RemoveNoise(M)
Input: $M = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$
Returns: the largest cluster $C_{\max} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m\}$
$C_{\max} \leftarrow ()$
$C_{\text{current}} \leftarrow ()$
$n \leftarrow \lfloor M \rfloor$
$m \leftarrow \lfloor C_{\text{current}} \rfloor$
while $n > 0$ **do**
$C_{\text{current}} \leftarrow C_{\text{current}} + \boldsymbol{u}_1$
for $i = 1, \ldots, m$ **do**
$M \leftarrow M - C_{\text{current}}$
for $j = 1, \ldots, n$ **do**
if $d(\boldsymbol{p}_i, \boldsymbol{u}_j) < \tau$ **then**
$C_{\text{current}} \leftarrow C_{\text{current}} + \boldsymbol{u}_j$
end if
end for
end for
$M \leftarrow M - C_{\text{current}}$
if $m > \lfloor C_{\max} \rfloor$ **then**
$C_{\max} \leftarrow C_{\text{current}}$
end if
$C_{\text{current}} \leftarrow ()$
end while
return $C_{\max}$
end

```
    orientation = 0;

    return points;
  }
```

As a consequence, the secondary axis going through the cluster's centroid is vertically aligned. For subdivision the x-coordinate of each point from the cluster to be divided is compared to the centroid's x-coordinate:

```
  for (ClusterPoint point : points) {
    if (point.getX() <= centroid.getX()) {
      left.add(point);
    } else {
      right.add(point);
    }
  }
```

3.5.5 Cluster Tree

The subdivision of the clusters C_1 and C_2 is realized with a depth-first approach in a tree, storing associated clusters in its nodes. 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. The actual decision maker of the algorithm is the registration (see Section 3.5.6) of two associated clusters of a node. The resulting error e_{avg} decides whether two clusters match, and subsequently require further subdivision:

```
public List<Cluster[]> subdivide(Node node) {
  registration = new Registration(node.cluster[0], node.cluster[1]);

  if (!registration.match()) {
    split(node);
    subdivide(node.left);
    subdivide(node.right);
  } else {
    subclusters.add(node.clusters);
  }
  return subclusters;
}
```

In the 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 $C_{2,i,2}$ originate. If two associated sub clusters $C_{1,i,j}$ and $C_{2,i,j}$ in a Node N match, no further subdividing is performed. The resulting leaves of the tree are the final matching sub cluster sets and are stored in a list $\mathcal{L} = (L_{1,1}, L_{2,1}, \ldots, L_{1,m}, L_{2,m})$ from left to right (see Algorithm 3.2). By applying a depth-first approach, the matching clusters stored in the list are actual neighboring clusters in C_1 and C_2. As a result, the adjacent sub clusters from $\mathcal{L}$ can be verified to be merged (see Algorithm 3.3). Figure 3.6 illustrates the subdividing step in the form of a `ClusterTree` and the merging of the resulting leaves.

3.5.6 Registration Procedure

The `Registration` class aims to apply a rigid transformation on the reference points from cluster C_i which results in the lowest error distance e to the target points of the input cluster C_j. For an initial alignment, C_i and C_j are similarly oriented and translated so that their centroids $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ overlap. Then, iteratively the best possible alignment of C_i and C_j is aimed for. As a first step, the sorted associated target points for the reference points are computed. This is achieved by seeking the closest neighbor for each reference point $\boldsymbol{p}_i$ in terms of the smallest Euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{p}_j)$ to a target point $\boldsymbol{p}_j$. As a next step, *Procrustes fitting* [30] is applied to the reference points and its associated target points. *Procrustes fitting* tries to enforce a rigid transformation between two corresponding, sorted point lists which results in a squared error distance e. In the case of a smaller error than from previous iterations the reference and associated points are stored as final transformations and target points. Additionally, the smallest error is updated. By applying the calculated transformation from Procrustes fitting, the iterative process initiates with updated reference point coordinates. It terminates, if the two clusters finally match or the maximum number of iterations is reached. This

Algorithm 3.2: Recursive subdividing of two clusters C_1 and C_2, in the form of a Node N in a `ClusterTree`, into matching sub clusters. The registration is performed on two corresponding clusters in N to verify them to match, in which case they are stored in a list of matching sub clusters. Otherwise, if the two clusters do not match, they are further subdivided. The list with all matching sub clusters $\mathcal{L}$ is returned once the subdivide algorithm terminates.

1: ClusterTree(C_1, C_2)
Input: the clusters C_1, C_2
Returns: the list of sub clusters $\mathcal{L}$
2: $\mathcal{L} \leftarrow ()$
3: $left \leftarrow nil$
4: $right \leftarrow nil$
5: $N \leftarrow \langle C_1, C_2, left, right \rangle$
6: $\mathcal{L} \leftarrow$ Subdivide(N)
7: $P \leftarrow$ MergeClusters(L)
8: **end**

9: Subdivide(N)
10: **if** match$(C_1(N), C_2(N))$ **then** ▷ apply registration on two clusters
11: $\mathcal{L} \leftarrow \mathcal{L} + N$
12: **else**
13: $left(N) \leftarrow$ Split(N)
14: $right(N) \leftarrow$ Split(N) ▷ split the cluster into a left and right sub cluster
15: Subdivide$(left(N))$
16: Subdivide$(right(N))$ ▷ recall the algorithm with the sub clusters
17: **end if**
18: **return** $\mathcal{L}$ ▷ return all matching sub clusters after termination
19: **end**

is required, in order to avoid an infinite loop that would occur if two clusters do not match. The iterative algorithm can be seen in the following code snippet:

```
while (!match() && iterations < MAX_ITERATIONS) {
  association = getAssociation(referencePoints, targetPoints);

  pro.fit(referencePoints, association);
  tmp_error = pro.getError();

  if (tmp_error < error) {
    error = tmp_error;
    finalTargetPoints = association;
    finalReferencePoints = referencePoints;
  }

  ClusterPoint c = calculateCentroid(finalReferencePoints);

  // apply the calculated transformation from procrustes fitting to the reference points

  referencePoints = Matrix.translate(referencePoints, -c.getX(), -c.getY());
```

Algorithm 3.3: Merging of the sub clusters $\mathcal{L} = ((L_{1,1}, L_{2,1}), \ldots, (L_{1,m}, L_{2,m}))$, resulting from Algorithm 3.2 in the order of being stored, to the rigid parts $\mathcal{P}$. Merged adjacent clusters $L_{i,j}$ and $L_{i,j+1}$ from C_1 and C_2 are then verified to still match. The merging proceeds until clusters do not match anymore. In this case, the 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.

```
1:  MergeClusters(L)
    Input: the sub clusters L = ((L_{1,1}, L_{2,1}), ..., (L_{1,m}, L_{2,m}))
    Returns: the rigid parts P
2:      P ← ()
3:      P_1 ← L_{1,1}
4:      P_2 ← L_{2,1}
5:      n ← ⌊L⌋
6:      for i = 2, ..., n do
7:          m_1 ← Merge(P_1, L_{1,i})
8:          m_2 ← Merge(P_2, L_{2,i})
9:          if match(m_1, m_2) then          ▷ apply ICP on two merged clusters
10:             P_1 ← m_1
11:             P_2 ← m_2                    ▷ continue merging with merged clusters
12:         else
13:             P ← P + (P_1, P_2)
14:             P_1 ← L_{1,i}
15:             P_2 ← L_{2,i}                ▷ proceed merging with current clusters
16:         end if
17:     end for
18:     P ← P + (m_1, m_2)
19:     return P
20: end
```

```
    referencePoints = Matrix.rotate(referencePoints, Math.acos(pro.getR().getEntry(0,
      0)));
    referencePoints = Matrix.translate(referencePoints, c.getX() + pro.getT().getEntry
      (0), c.getY() + pro.getT().getEntry(1));

    iterations++;
  }
```

The average error per point e_{avg} is calculated by dividing the total squared error e by the total number of reference points. In the case of $e_{\text{avg}<\tau}$ two clusters are determined to match:

```
  public boolean match() {
    double errorPerPoint = error / amountPoints;

    if (errorPerPoint < errorThreshold) {
      return true;
    }
```

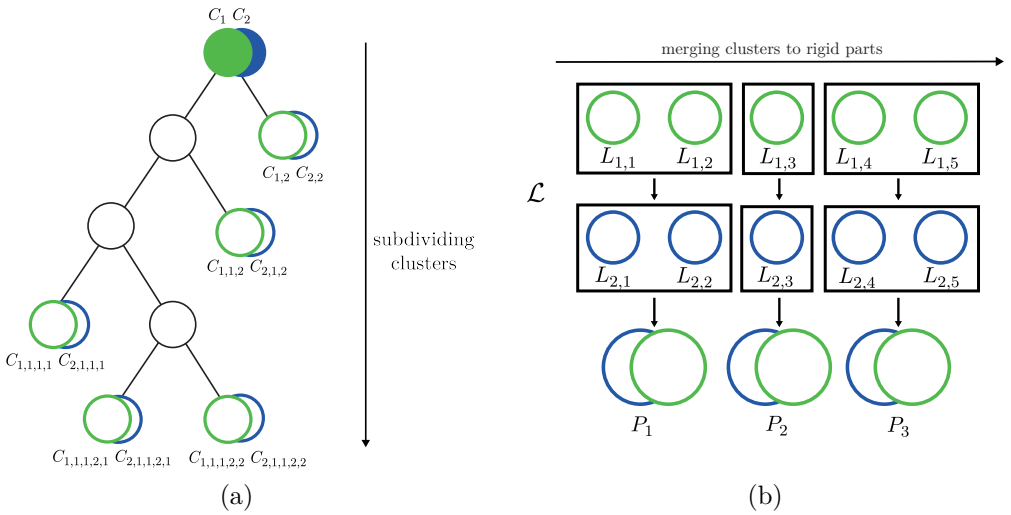


Figure 3.6: Subdividing of C_1 and C_2 into matching clusters by a depth-first approach in a tree. The subdividing terminates if all sub clusters of C_1 and C_2 match after registering them (a). Detection of the rigid parts of C_1 and C_2 by iteratively merging adjacent sub clusters of $\mathcal{L}$ that they still match (b).

```
    return false;
}
```

3.6 Results

At first, the implemented algorithm was applied on two point clouds of an articulated object composed of two rigid parts. The estimated resolution accounts to 7.0 for C_1 and C_2. 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 fewer clusters and subsequently rigid parts can be detected. The reason is that two clusters are more likely to match and are not further subdivided. The lower, the more clusters and rigid parts will be detected, as clusters require further subdividing in order to match. Figure 3.7 shows the segmentation results with an error threshold $\tau = 4.0$ in which case the correct 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 for this is, that the average error e_{avg} is calculated without any weighting of points. Points located near a joint must be treated particularly cautiously. By decreasing the threshold τ to a value of 3.5, the joint estimation can be considerably improved. However, not the right number of rigid parts is detected. In the case of an articulated object, composed of three rigid parts linked like a chain, similar results can be achieved (see Figure 3.8).

As the main goal is to segment a human-like articulated object into its rigid parts, a more complex mesh was taken as input. Contrary to the previous input objects, a rigid part is linked to multiple other rigid parts, such as the extremities of a human to the torso. Figure 3.9 shows the segmentation results with a threshold τ of 8.0. In this case, only three rigid parts can be detected. By decreasing τ to 7.0 (see Figure 3.10), further

Rigid parts	τ	detected clusters	detected rigid parts
2	3.0	6	4
	3.5	6	3
	4.0	6	2
3	5.0	13	7
	6.0	8	5
	7.0	6	3
5	7.0	41	29
	7.5	15	6
	8.0	15	3

Table 3.1: Segmentation results with varying values for τ.

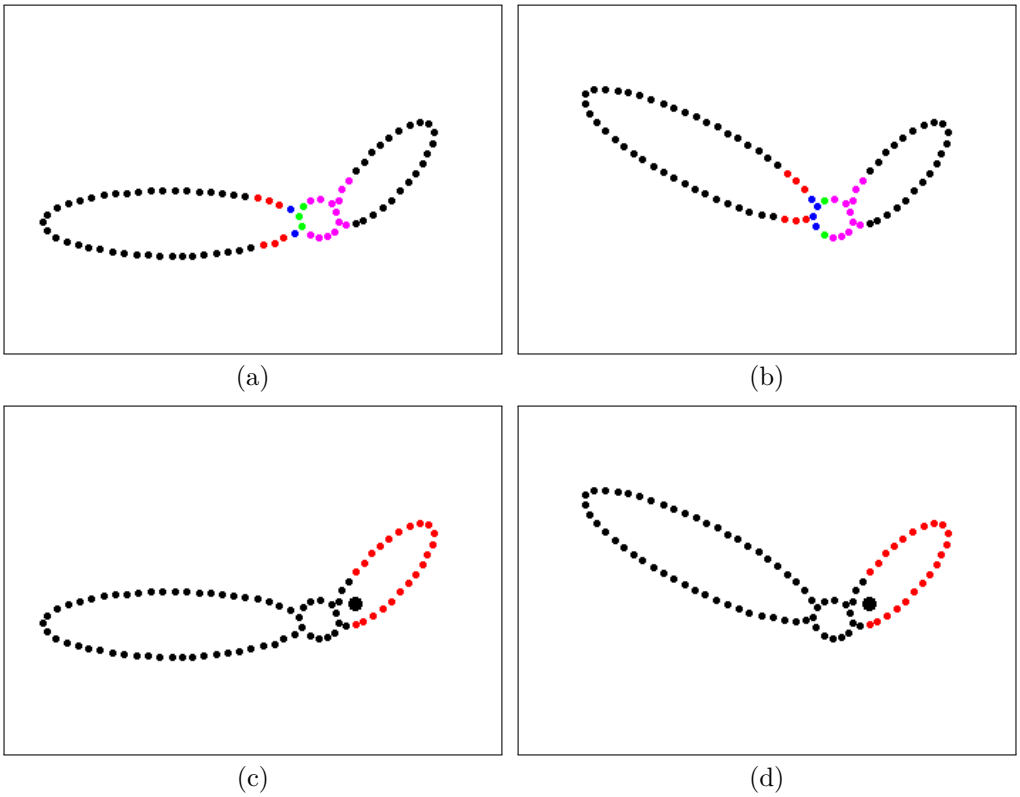


Figure 3.7: Taking a Mesh M in two poses with only two rigid parts as input, with a threshold $\tau = 4.0$, six clusters are detected in C_1 (a) and C_2 (b), which result in two rigid parts (c) and (d).

subdividing is performed, which leads to six rigid parts. However, the points from a rigid part may be located far away from one another, which leads to joints being placed in desolated point areas (see Figure 3.10). The reason is that the segmentation is solely determined by the secondary axis of the object that does not consider the coherence of logical related points (see Section 3.7 for details).

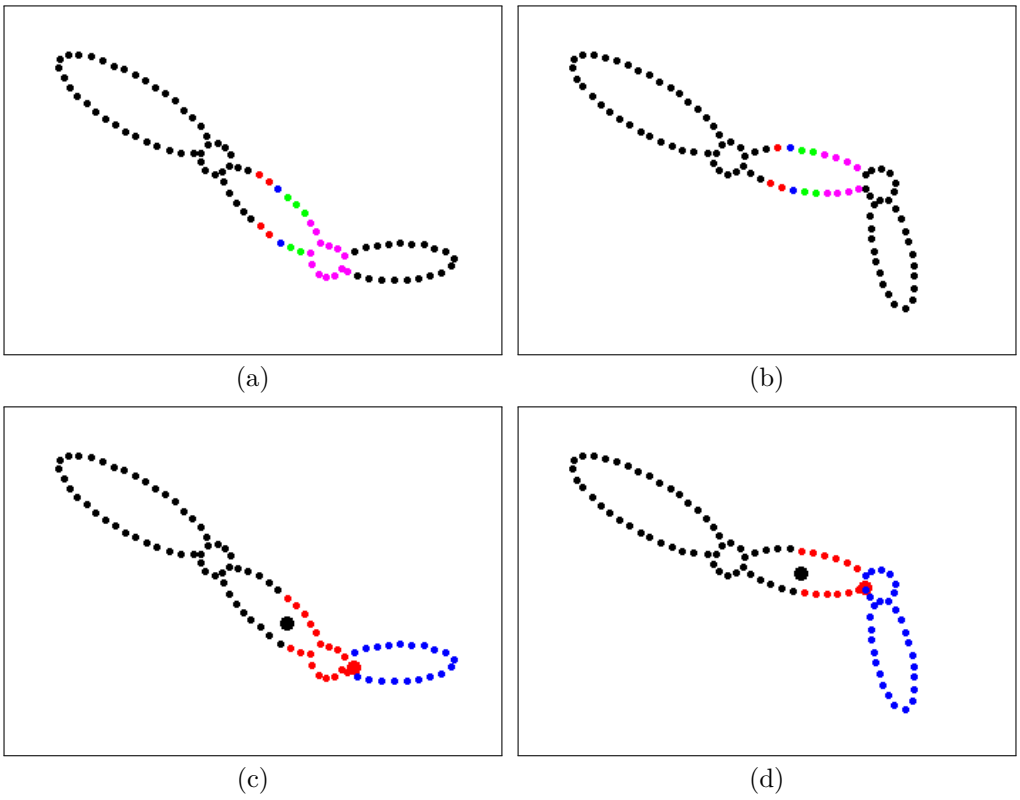


Figure 3.8: Taking a Mesh M in two poses with three rigid parts as an input, with a threshold $\tau = 7.0$, six clusters are detected in C_1 (a) and C_2 (b), which results in three rigid parts (c) and (d).

3.7 Possible Improvements

In the case of more complex objects with a skeletal structure, where one rigid part is linked to more than two rigid parts, the segmentation algorithm in its simple form fails. Although varying thresholds τ are used for the registration, there are either too many or too few rigid parts detected. The main issues can be summarized as the following:

1. By linearly subdividing a cluster C_i along its secondary axis, more than two sub clusters can originate. In this case, a sub cluster might consist of points located far apart from one other, which distorts the segmentation into rigid parts.
2. Clusters being compared might not contain the same number or not even the same corresponding points, because dividing only builds on the secondary axis as divider. In the case of considerable transformation differences of corresponding rigid parts, the position of the computed secondary axis and subsequently the sub clusters, may considerably differ from one another.
3. The segmentation procedure results only into approximated rigid parts. The reason for this is that sub clusters might match even if some points considerably contribute to a higher average matching error. Points with a notable low error will compensate these outliers and as a result, a successful match may be detected.

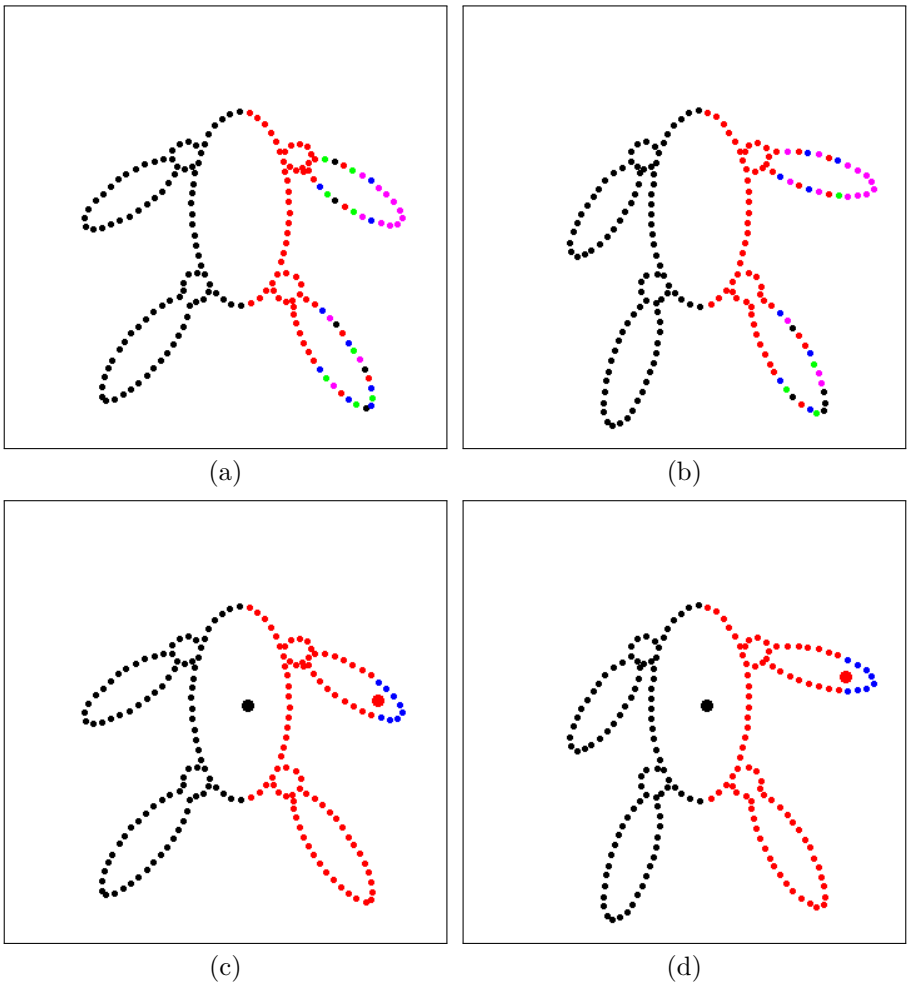
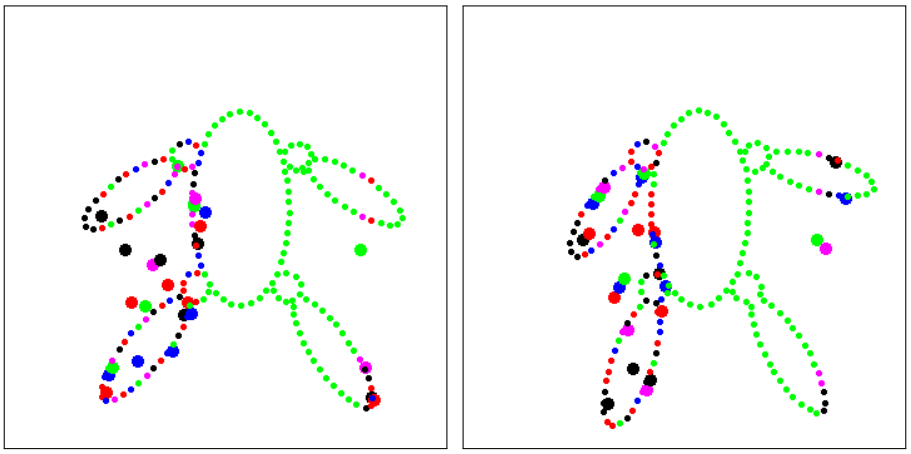


Figure 3.9: Taking a more complex Mesh M in two poses with five rigid parts as an input, with a threshold $\tau = 8.0$, 15 clusters are detected in C_1 (a) and C_2 (b), which results in three rigid parts (c) and (d).

Still, there might be a better segmentation position with fewer points.

4. In the case of detecting two matching clusters that are a part of a rigid part, no further operations to detect the actual rigid part they belong to are directly conducted. It is anticipated to be detected in a later step by merging neighboring sub clusters.
5. By further sub dividing sub clusters, merging becomes more difficult, because the clusters are scattered next to each other. This makes it difficult to guarantee that stored adjacent clusters are associated in two different poses.

Since the aforementioned issues are responsible for the unsuccessful results for articulated objects, the initial approach needs to be extended.



(a) (b)

Figure 3.10: Joint estimation for a mesh M composed of five rigid parts, with a threshold $\tau = 7.0$, 29 rigid parts are detected in C_1 (a) and C_2 (b). The scattering of points belonging to the same rigid part leads to a joint estimation in desolated point areas.

3.7.1 Assuring corresponding, similar Clusters

During the segmentation of an object in two different poses, there is the chance that the divided parts being compared do not contain the same number or points, or that they even contain additional points that correspond to another rigid part (see issues 1 and 2 from Section 3.7). Since this state might result in undesirable matching errors, parts with the same sizes could be generated by region growing. This could be implemented by starting the clustering with two points that are the farthest from the centroids of two input clusters. By growing regions with the same number of points, presumably similar clusters are compared. Additionally, the detection of multiple clusters is considered and treated individually.

3.7.2 Matching Error

The current matching error is determined by the average error per point e_{avg} where no weighting of points is considered. Consequently, considerable close point correspondences might compensate for points that contribute to a high error as they do not belong to the same rigid part. The goal should be to focus solely on the best point correspondences as an indicator where the segmentation must take place, namely in the area of higher distances between corresponding points. By introducing weights to points, the error e would be more expressive as instead of compensating points with a high error distance, a further segmentation will be conducted. Furthermore, by locating those outliers, the segmentation could be conducted by selecting the best matching points instead of linearly subdividing a cluster (see issues 3 and 4 from Section 3.7).

3.7.3 Initial Alignment of the largest Rigid Part

The most crucial deficit of the proposed algorithm is that it does not result in a successful segmentation regarding complex articulated objects, whose rigid parts are not composed like a chain. Instead, a rigid part may be linked to multiple rigid parts. As a result, the objects are too complex to be linearly subdivided. One suggestion for improvement in regards to the initial alignment of the object, is the detection of a reliable corresponding largest rigid part. Then, recursively linked parts of this largest rigid part may be detected. A similar approach was taken during the recursive algorithm from Guo et al. [11] (see Section 4.2). The results are directly dependent on the extent of the transformations of the two articulated clusters C_1 and C_2. The more different the transformations are, the higher the chances are that the initial alignment of the largest rigid part will fail. This is because the ICP expects a good starting alignment. It is assumed that the largest rigid part contributes most to the principal axis and the initial alignment. But in the case of a considerable unbalance of the other rigid 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, if only the points' locations and the cluster axes (see Section 3.8) are considered.

3.8 Outcome

Although many optimizations are proposed, the desired segmentation results for complex objects will not be achieved. However, the implementation of this linear approach allowed me to acquire a significant amount of knowledge regarding the segmentation of non-rigid objects into their rigid parts. The most relevant observation is that relying only on the coordinates of points is not sufficient for the segmentation of complex articulated objects. The reason is that with an increasing number of rigid parts, the detection of correct point correspondences between C_1 and C_2 becomes increasingly difficult. Additional point descriptors are required to detect reliable point correspondences between C_1 and C_2. The focus at this point is shifted to a feature-based segmentation approach (see Chapter 4).

Chapter 4

Feature-Based Approach

To solve the main difficulty of detecting reliable point correspondences between C_1 and C_2, the focus on the second approach is on point features. They are introduced because they provide additional, meaningful information about a point $\boldsymbol{p}_i$ besides its coordinates. For this reason, point feature histograms, namely *Fast Point Feature Histograms* (FPFH) [23] are calculated for all points of C_1 and C_2. A histogram of a point $\boldsymbol{p}_i$ describes the curvature of a surface defined by itself and its k neighbors. By detecting similar point feature histograms between C_1 and C_2 (referred to as feature matching), point correspondences are acquired. These are then used for an initial alignment of C_1 and C_2 so that their largest rigid parts overlap. Proceeding from the largest rigid part, linked rigid parts can be detected. This approach is closely related to Guo et al. [11].

4.1 Fast Point Feature Histograms

The *Fast Point Feature Histograms Algorithm* is an improved approach of the *Persistent Point Feature Histogram for 3D Point Clouds* [24] in terms of computation time. It focuses on computing a feature histogram for a point $\boldsymbol{p}_i$ by comparing its normal $\boldsymbol{n}_i$ to the normals of all k neighbors. The choice of these point features are the following:

- rotation- and scale-invariant,
- easy comparison of feature histograms,
- approval of the approach,
- straightforward adaption of different dimensions (2D and 3D).

The data input for this algorithm is a list of n unordered points given as $\boldsymbol{p}_i(x, y, z)$. It only provides information about the points' coordinates in 3D space. As a result, no surface is provided for the computation of feature histograms. Therefore, in an initial step the neighborhood and normals of all points is computed (see Section 4.1.1).

4.1.1 Normal Estimation

First, the normals of all unordered points from the input clusters C_1 and C_2 are estimated, following the approach of Hoppe et al. [13]. The normal $\boldsymbol{n}_i$ of a point $\boldsymbol{p}_i$ can be calculated by considering all of the neighboring points k within a radius r of $\boldsymbol{p}_i$. Subsequently, a least squared fitting line X in the form $ax + by + c = 0$ to all selected

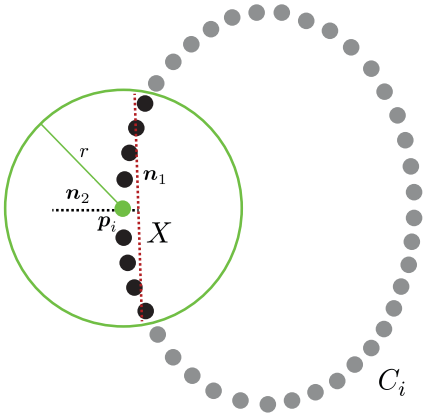


Figure 4.1: Normal estimation of a cluster point $\boldsymbol{p}_i$ of C_i by calculating the least squared fitting line X to the neighborhood inside a radius r. Calculation of the eigenvector $\boldsymbol{n}_2$ by forming the covariance matrix of $\boldsymbol{p}_i$ and all k points.

points including $\boldsymbol{p_i}$ is computed. This is achieved by minimizing the squared distances

$$dist^2(x_i, y_i) = (ax_i + by_i + c)^2 \tag{4.1}$$

from all points to X, that sum up to an error

$$e = \sum_{i=1}^{k} dist^2(x_i, y_i). \tag{4.2}$$

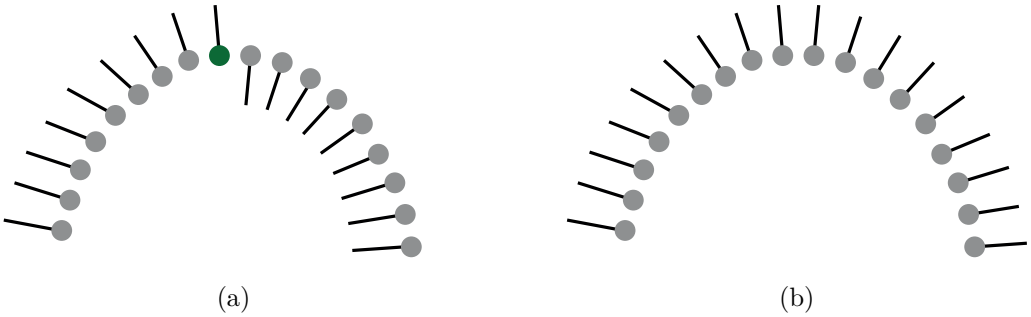
The unknown parameters a, b of X are calculated with the covariance matrix of all points, that is

$$\begin{pmatrix} \overline{x^2} - \overline{x} \cdot \overline{x} & \overline{xy} - \overline{x} \cdot \overline{y} \\ \overline{xy} - \overline{y} \cdot \overline{x} & \overline{y^2} - \overline{y} \cdot \overline{y} \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \lambda \cdot \begin{pmatrix} a \\ b \end{pmatrix}, \tag{4.3}$$

resulting in two pairs of an eigenvalue and eigenvector $(\boldsymbol{n}_1, \lambda_1)$ and $(\boldsymbol{n}_2, \lambda_2)$. The normal of $\boldsymbol{p}_i$ is calculated solving the linear equation for the normal vector $\boldsymbol{n}_2$ which is represented by λ_2. This procedure is conducted for all points of C_1 and C_2 (see Figure 4.1). Next, the computed normals are globally oriented. All n normals are traversed, starting with a normal $\boldsymbol{n}_i$ of any point $\boldsymbol{p}_i$. 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 $\boldsymbol{n}_p$, the angle δ between $\boldsymbol{n}_p$ and all neighboring point normals $\boldsymbol{n}_k$, that is

$$\delta = \boldsymbol{n}_p \cdot \boldsymbol{n}_k, \tag{4.4}$$

is calculated, expecting that $|\boldsymbol{n}_p| = |\boldsymbol{n}_k| = 1$. In the case of $\delta < 0$, $\boldsymbol{n}_k$ must be flipped 180°, that $\boldsymbol{n}_k = -\boldsymbol{n}_k$. If all k normals have been verified to be oriented in consideration of $\boldsymbol{n}_p$, any normal $\boldsymbol{n}_k$ is selected as current $\boldsymbol{n}_p$. The whole algorithm proceeds until all n normals have been verified to correspond with their parent normal $\boldsymbol{n}_p$ (see Figure 4.2).



(a) (b)

Figure 4.2: Global flipping of all point normals (black lines) of a surface considering a parent point (green) (a) to similarly orient them (b).

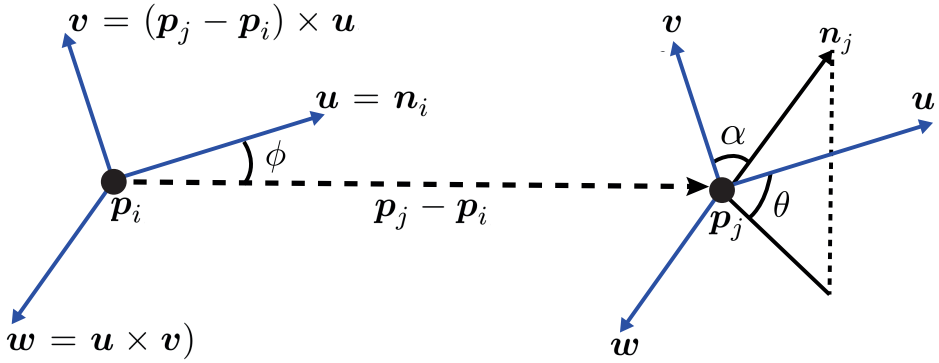


Figure 4.3: Calculation of the Darboux frame $\boldsymbol{uvw}$, in order to obtain the features between a point set described by three angles.

4.1.2 SPFH and FPFH

As an intermediate step before computing the FPFH, the *Simplified Point Feature Histogram* (SPFH) of a point $\boldsymbol{p}_i$ is computed. This is achieved by calculating three geometric features between $\boldsymbol{p}_i$ and each of its k neighbors $\boldsymbol{p}_k$. For the following calculations, $\boldsymbol{p}_i$ represents the point that has the smallest angle between its normal and the vector formed by two compared points; $\boldsymbol{p}_j$ corresponds to the other point. Using their normals $\boldsymbol{n}_i$ and $\boldsymbol{n}_j$, a Darboux $\boldsymbol{uvw}$ frame $(\boldsymbol{u} = \boldsymbol{n}_i, \boldsymbol{v} = (\boldsymbol{p}_j - \boldsymbol{p}_i) \times \boldsymbol{u}, \boldsymbol{w} = \boldsymbol{u} \times \boldsymbol{v})$ is calculated, which describes the coordinate system of a point on a surface (see Figure 4.3). Consequently, the following three angles

$$\begin{aligned} \alpha &= \boldsymbol{v} \cdot \boldsymbol{n}_j, \\ \phi &= \boldsymbol{u} \cdot (\boldsymbol{p}_j - \boldsymbol{p}_i) \cdot \frac{1}{\|\boldsymbol{p}_j - \boldsymbol{p}_i\|}, \\ \theta &= \arctan(\boldsymbol{w} \cdot \boldsymbol{n}_j, \boldsymbol{u} \cdot \boldsymbol{n}_j), \end{aligned} \tag{4.5}$$

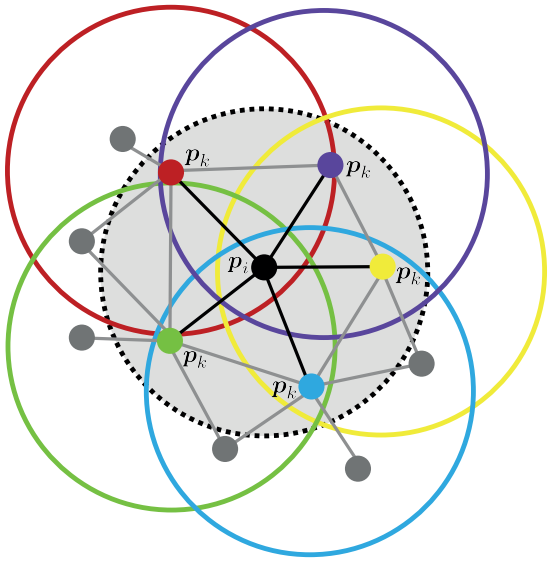


Figure 4.4: The point region for the calculation of the feature histogram H_i for a point $\boldsymbol{p}_i$. The SPFH of $\boldsymbol{p}_i$ and its k neighbors (inside the grey circle) is weighted with the SPFHs of all k neighbors (grey points inside the colored circles) [23].

are calculated and then categorized into a histogram (see Section 4.1.3). The SPFH of $\boldsymbol{p}_i$ is then weighted to its FPFH, that is

$$\text{FPFH}(\boldsymbol{p}_i) = \text{SPFH}(\boldsymbol{p}_i) + \frac{1}{n} \cdot \sum_{k=1}^{n} \frac{1}{w_k} \cdot \text{SPFH}(\boldsymbol{p}_k), \tag{4.6}$$

by computing the SPFH for all of its k neighbors. The weight w_k represents the Euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{p}_k)$. The influence region diagram for a *Fast Point Feature Histogram* of a point $\boldsymbol{p}_i$ can be seen in Figure 4.4.

4.1.3 Feature Histograms

The resulting feature values for each point $\boldsymbol{p}$ of C_1 and C_2 in the form of three angles between its k neighbors are categorized using a histogram $H = \{b_1, \ldots, b_m\}$ containing $m = q^n$ bins to consider all possible combinations of the three feature values. Thereby, q represents the number of intervals that a feature value can be categorized into. The number of feature values is represented by n. Taking into account three feature values calculated between a point set ($\boldsymbol{p}_i$,$\boldsymbol{p}_j$) the bin value at the index

$$idx = \sum_{i=1}^{n} q(f_i) \cdot 2^{i-1} \tag{4.7}$$

from H_i is incremented by 1. The function $q(f_i)$ returns the interval, the specific feature value 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 used for the detection of point correspondence between C_1 and C_2, in order to reduce the correspondence space. To achieve this, the mean of all feature histograms μ of a cluster C_i is calculated. Subsequently, the distance of a feature histogram H_i is compared to H_μ and in the case of being outside the value $\mu \pm \sigma$, it is denoted as *unique* and passed to the next step of detecting matching histograms between C_1 and C_2. The standard deviation, that is

$$\sigma = \frac{1}{N} \cdot \sum_{b=1}^{N} (H_i(b) - \overline{H_i})^2, \tag{4.8}$$

is therefore calculated for all histograms H_i with N data entries of all cluster points.

4.1.4 Feature Matching

Next, all *unique* feature histograms of C_1 are aiming for detecting the most similar feature histograms from C_2. For this reason, different histogram-similarity criteria are determined, in order to measure the similarity between two histograms H_i and H_j with b bins each. In order to detect the best fitting histogram, three main criteria that are most frequently applied for this specific use case [12, 32] are attempted. As per the first criterion, the squared Euclidean distance (L2), that is

$$\varepsilon(H_i, H_j) = \sum_{n=1}^{b} \|H_i(n) - H_j(n)\|^2, \tag{4.9}$$

is calculated. In [11] this distance was selected as the only similarity criterion between feature histograms. It generates more point correspondences (see Section 4.4) than the other two criteria and operates straightforward. Furthermore, the statistical Chi-Square (χ^2) divergence, that is

$$\chi^2(H_i, H_j) = \sum_{n=1}^{b} \frac{\|H_i(n) - H_j(n)\|^2}{\|H_i(n) + H_j(n)\|}, \tag{4.10}$$

is examined which achieves considerably fewer point correspondences than the L2 form. Finally, the Kullback-Leibler (KL) divergence, that is

$$\kappa(H_i, H_j) = \sum_{n=1}^{b} \|H_i(n) - H_j(n)\| \cdot \ln \frac{H_i(n)}{H_j(n)}, \tag{4.11}$$

is calculated, which is the most computationally expensive one due to the application of ln. Due to the fact that a histogram with q^n bins may contain many zero values, in the case of a division or logarithmic operation all zero-values are replaced with the value 1.

4.1.5 Adaptions for 2D

In order to compute the proposed feature histograms for 2D points, all points are treated like 3D points. This is achieved by setting each z-coordinate to 0 to allow for all calculations originally targeting 3D points. 2D point clouds are generally composed of fewer

points than 3D point clouds, which results in fewer neighboring points for both normal estimation and feature computation. As a result, the computation time is considerably reduced. Since the number of point correspondences is also reduced, all histograms are considered during the feature histogram matching, not just salient ones.

4.2 LRP Algorithm

The approach proposed by Guo et al. [11] aims to detect the so-called *Largest Rigid Part* (LRP) of two poses of an object, in order to have a solid basis for a successful segmentation into the rigid parts. A similar procedure is sought in the proposed feature-based approach.

4.2.1 Basic Functionality

As an initial step, the *LRP Algorithm* attempts to detect the most reliable correspondences between two clusters C_1 and C_2. For this, 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*, meaning that two corresponding point feature histograms are the most similar to one another. Some of the sparse correspondences are assumed to be wrong. Therefore, RANSAC is applied to the point correspondences, in order to aim for a single rigid transformation to detect the LRP, which is supported by the largest overlapping point cluster between C_1 and C_2. In the 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. The most crucial component of the 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, because the subsequent detection of further rigid parts proceeds from there.

4.2.2 Input Data

For the proposed 2D implementation, the 2D hulls of an articulated object in two different poses are taken as input (see Figure 4.6). Therefore, an articulated wooden puppet was taken as reference model (see Figure 4.5). The resulting data sets in the form of 2D points were traced manually, as some essential factors needed to be considered for successful segmentation results. The application of an automatic scanning approach would lead to further challenges, such as noise and corner detection, which is not the focus of this thesis. To support the point cohesiveness of a rigid part after a transformation or segmentation, joint-likely spheres were placed between rigid parts; these are also used as rotation points. For the computation, they are treated like they belong to a rigid part and no prior information about any joints are extracted. To obtain an indicator for the adjustment of thresholds (e.g. the maximum distance to the closest point), the resolution of the input clusters is computed. By applying the approach from Section 3.1, the estimated resolutions account to 9.0 for C_1 and 8.9 for C_2.

[1] https://www.aliexpress.com/item/8-12-inch-Joints-wood-Wooden-mannequin-toy-wooden-puppet-wooden-manikin-Home-Decoration-Model-Painting/32809065479.html

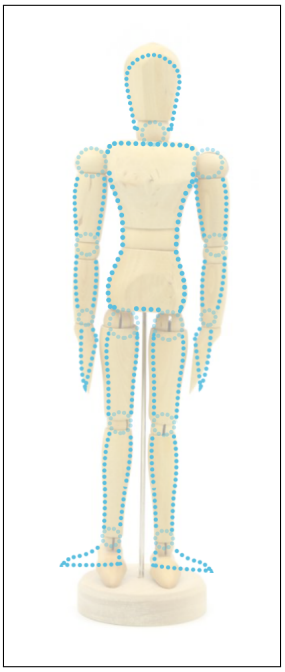
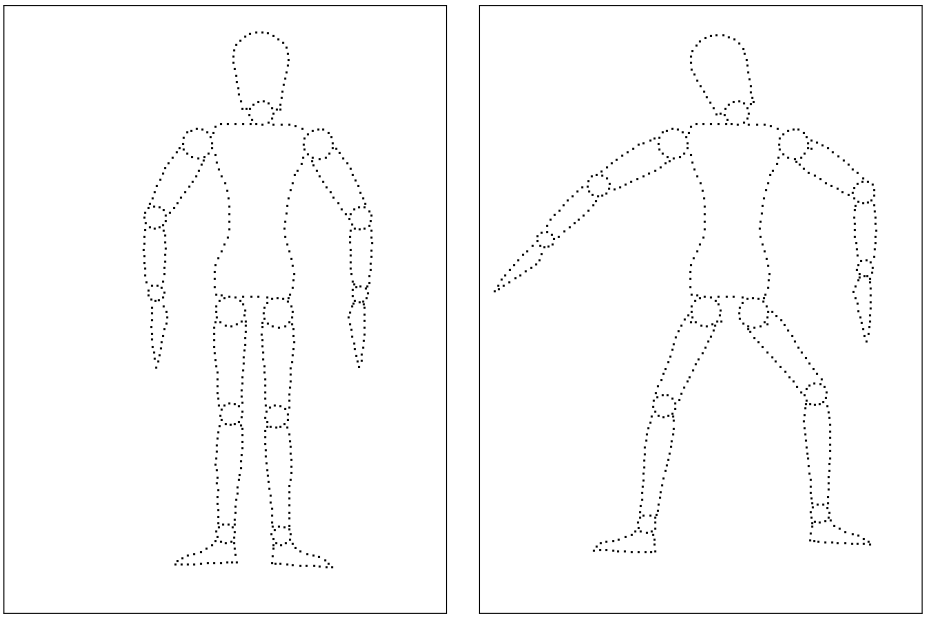


Figure 4.5: As input model an image of an articulated puppet[1]was traced in order to obtain 2D points describing its hull.



(a) (b)

Figure 4.6: Taking an articulated object (puppet) in two different poses C_1 (a) and C_2 (b) in the form of a 2D point cloud representing its hull as input.

4.2.3 Implementation Steps

In order to re-implement the proposed *LRP Algorithm* in 2D, only minor modifications concerning point coordinates and feature histograms had to be accomplished (see Section 4.1.5). Concerning the detection of linked rigid parts, an approach with fewer computation steps was implemented. Generally, the implementation can be split into six main parts:

1. The PCA is applied on the input clusters C_1 and C_2 to estimate the normals of all points.
2. *Fast Point Feature Histograms* (FPFH) are computed for all points of C_1 and C_2, in order to compute sparse point correspondences by feature matching that require to be *reciprocal*.
3. The RANSAC approach is applied to these correspondences to detect a rigid transformation T that aligns the largest rigid parts of C_1 and C_2. In each iteration, clusters are detected from overlapping points. The LRP is assigned to the largest overlapping point cluster.
4. All remaining unmatched clusters $\mathcal{C}_U$ from C_1 and C_2 are taken as input to detect linked rigid parts to the current LRP.
5. Joints are estimated between the current LRP and all unmatched clusters $\mathcal{C}_U$ linked to the LRP. These are required for associating two clusters from C_1 and C_2.
6. Linked rigid parts are detected by overlapping two corresponding clusters' joints and rotating one cluster onto the other one. Certain constraints and a weighted error are taken into account.

4.2.4 Detection of sparse Correspondences

At first, the normals as well as the feature histograms for all points of C_1 and C_2 (see Section 4.1) are computed. Results of the normal flipping procedure can be seen in Figure 4.7. It can be observed that the flipping of normals does not globally succeed. This is because of the corner positions of the 2D hull. Considering two normals surrounding a corner point, they are expected to face each other. However, during the normal flipping they are similarly oriented, although the corner point between them should counteract this behavior. As a consequence, the global flipping of the normals, depending on the given data set, does not lead to a better ratio of right point correspondences. For this reason, the normal flipping is passed over in the 2D implementation. For the detection of sparse correspondences between C_1 and C_2 three histogram distances (see Section 4.1.4) are considered. Depending on the distance chosen as the criterion and number of the regarded neighboring points for the feature computation, more or fewer correspondences may be detected; refer to Section 4.4 for a detailed comparison. The mean histogram of all histograms from C_1, as well as a *unique* histogram can be seen in Figure 4.8, considering ten points for the feature histogram computation. Figure 4.9 represents the point correspondences between C_1 and C_2 resulting from feature matching. It is evident that points located near a corner are more likely to detect a reciprocal point correspondence than points located on smooth surfaces. The reason for this is that these features are determined to be *unique*, which makes them more distinguishable. On the

contrary, all points located on a smooth surface have similar feature histograms. As some of these correspondences are assumed to be wrong, a RANSAC approach is applied on all correspondences, in order to reject the false ones (see Section 4.2.5).

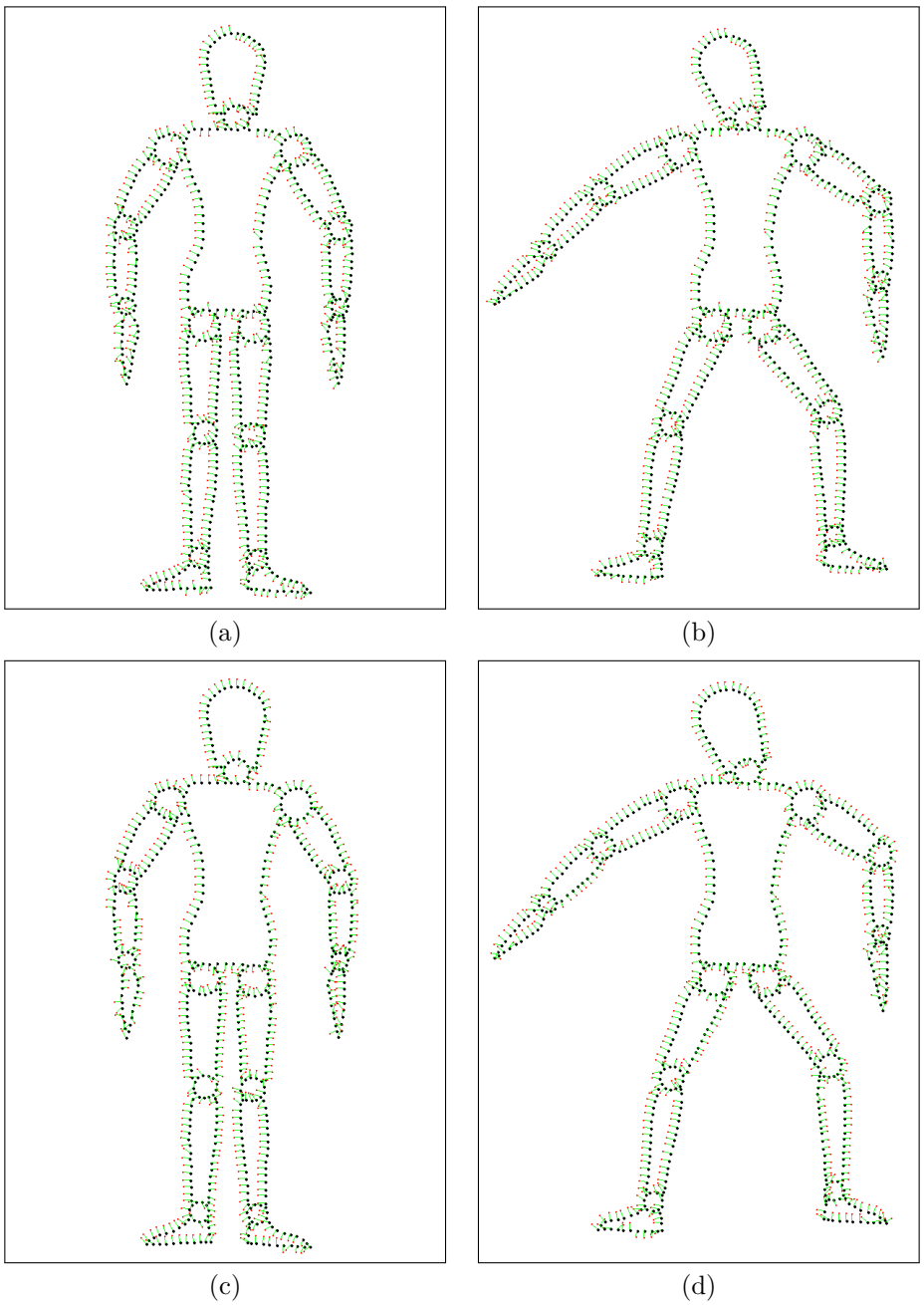
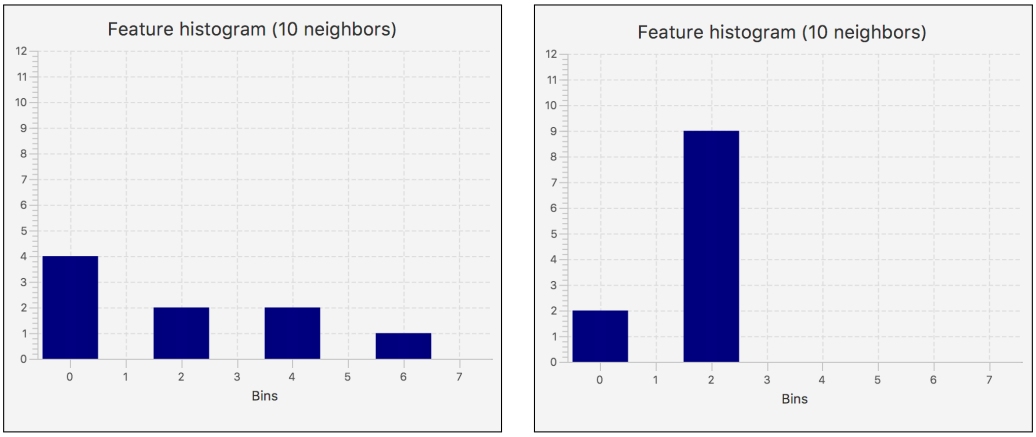


Figure 4.7: Normal estimation of two clusters C_1 (a) and C_2 (b). All normals are oriented similarly by traversing all normals globally (c) and (d).



(a) (b)

Figure 4.8: Computing the μ histogram of C_1 to represent frequently arising feature histograms (a). A *unique* histogram is stated to considerably deviate from the μ histogram (b).

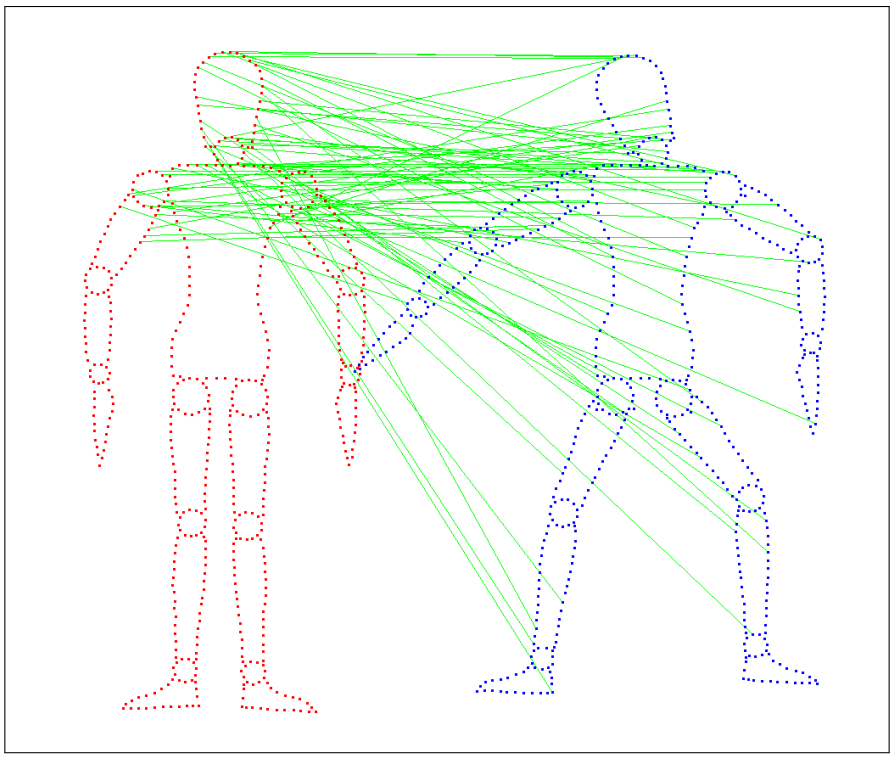


Figure 4.9: Visualization of the point correspondences established from reciprocal feature matching between all points of C_1 (red) and C_2 (blue). As distance criteria the L2 distance with a feature neighborhood of ten points is applied.

4.2.5 Detection of the largest Rigid Part

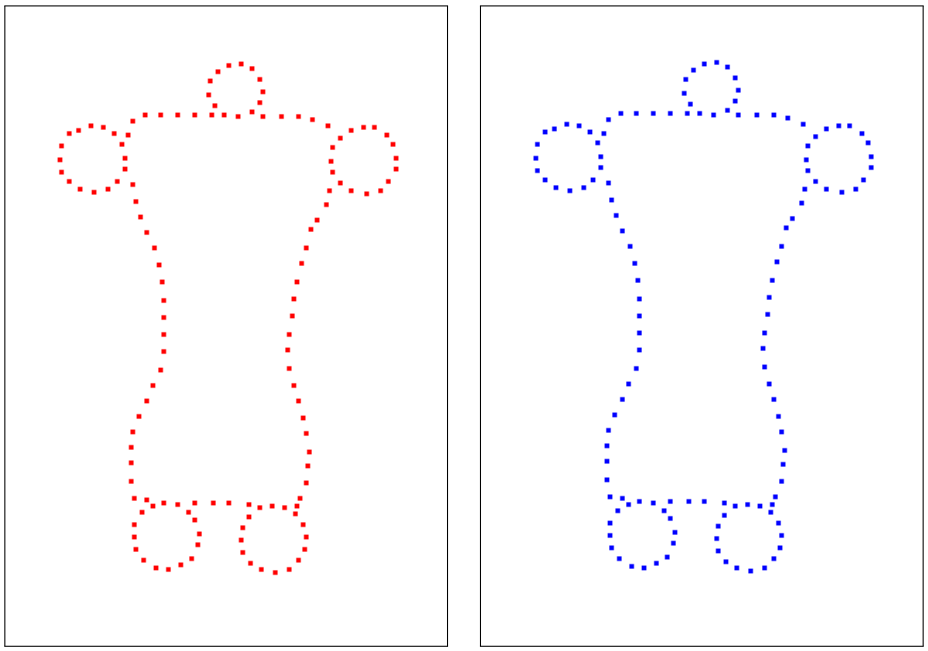
The dense point correspondences from the previous computation step (see Section 4.2.4) may contain several errors. Therefore, RANSAC is applied, in order to detect a single rigid transformation T that leads to the biggest overlapping point cluster of C_i and C_j. Thereby, in each iteration, two random point correspondences are selected and used for the calculation of T, which is applied on C_i to be translated onto C_j. The number of required iterations for a successful alignment of the largest rigid parts highly depends on the ratio of correct and incorrect point correspondences. Based on visual assessments and the calculated probability of choosing two correct correspondences from the actual LRP (torso), the required number of RANSAC iterations can be estimated. In the case of the L2 distance, approximately 20% of the point correspondences are assumed to be correct and located on the torso (see Figure 4.9). By calculating the probability that almost assures (99%) the desired alignment of C_1 and C_2, that is

$$\begin{aligned} 1 - (n \cdot q^n) &= 0.99, \\ 1 - (n \cdot 0.8^n) &= 0.99, \\ n &= 247.857, \end{aligned} \tag{4.12}$$

the number of iterations n can be set to 250. In each iteration, clusters are grown from all overlapping points with an Euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{p}_j)$ (again below a predefined threshold τ). The procedure is applied to both C_i and C_j which results in two rigid parts as output representing the largest overlapping clusters (see Figure 4.10). The final transformation T of the RANSAC approach, which leads to the largest overlapping clusters, is applied to the reference cluster C_1. This procedure is required in order to similarly align C_1 and C_2 for further computations (see Section 4.2.6).

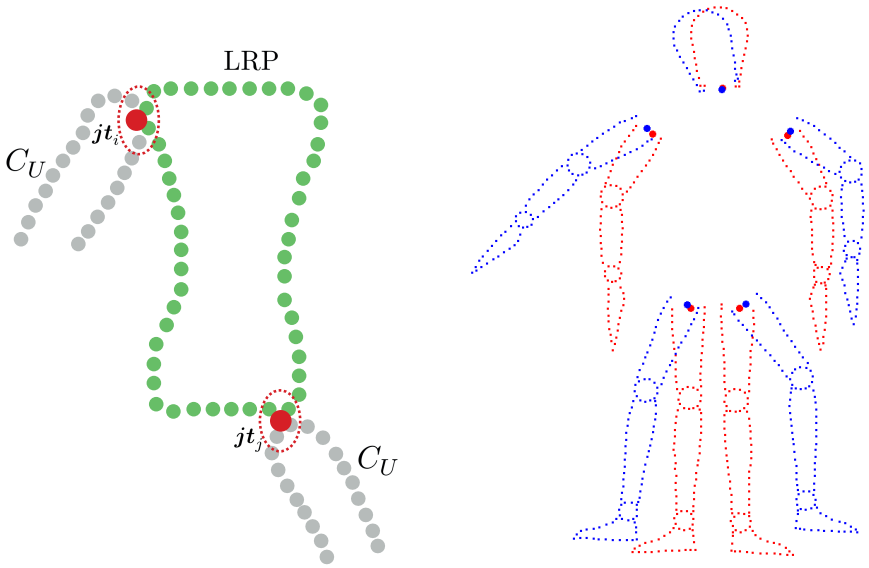
4.2.6 Cluster Detection by Region Growing

After successfully detecting a LRP for the input clusters C_i and C_j, they are added to a list of rigid parts $\mathcal{P}$. Potentially linked rigid parts are detected from region growing of all non-clustered points $\mathcal{U} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$. These comprise all cluster points of C_1 and C_2 excluding already detected rigid parts $\mathcal{P}$. An adapted region growing procedure from Section 3.5.3 is applied, with the previously detected LRP as seed. This approach not only returns the largest cluster, but all clusters above a certain size. The result is a set of clusters $\mathcal{C}_\mathcal{U}$ for each C_i and C_j comprised by unclustered points. In the case of detecting more than one cluster resulting from region growing, the clusters that correspond to one another in two different poses must be associated. This might be, for example, the case for the extremities linked to the torso. This association step is essential, as the detection of linked parts to the previously detected LRP assumes two clusters representing similar rigid parts. Thereby, the joint $\boldsymbol{jt}$ is estimated for a cluster C_U (see Figure 4.11). The joints between two rigid parts are estimated, by taking the two closest points between them, and calculating the average point. Two clusters with reciprocal closest joints, represented by the Euclidean distance $d(\boldsymbol{jt}_i, \boldsymbol{jt}_j)$, are assumed to correspond. The joints are further used for the detection of linked rigid parts (see Section 4.2.7).



(a) (b)

Figure 4.10: Alignment of the largest overlapping clusters from C_1 (a) and C_2 (b) by performing 250 RANSAC iterations on the detected point correspondences from feature matching. The resulting clusters are the LRPs.



(a) (b)

Figure 4.11: Estimation of a joint $\boldsymbol{jt}_i$ and $\boldsymbol{jt}_j$ between a detected LRP and two linked clusters C_U for correspondence and part detection (a). Resulting joints for all grown clusters C_U from the initially detected LRP (b).

4.2.7 Detection of linked Rigid Parts

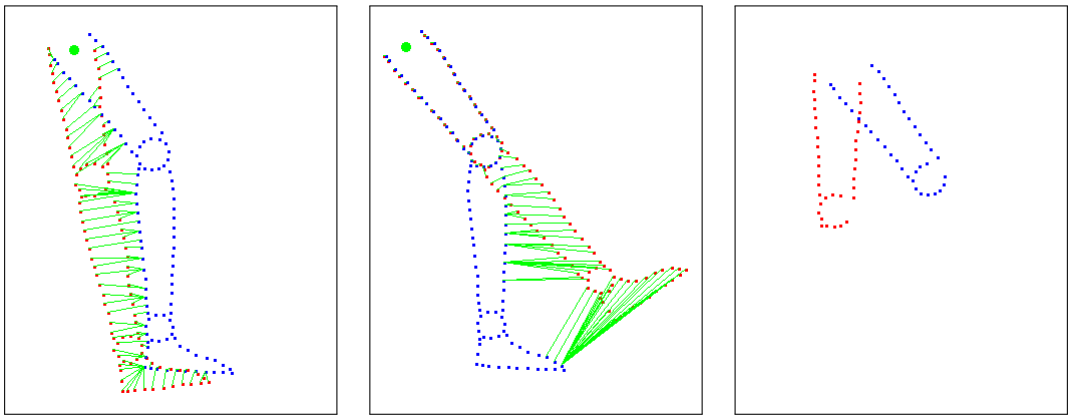
In [11], the feature detection and histogram matching in combination with RANSAC was reapplied to the linked clusters to the already detected LRP. However, this approach has not been re-implemented in 2D for the detection of further rigid parts. The proposed approach manages to operate less computational-expensive and time-consuming as RANSAC as well as feature computation and matching are not required. Thereby, the knowledge that rigid parts are transformed by rotating around a joint is used. For that purpose, the estimated joints $\mathcal{J} = \{\boldsymbol{jt}_1, \cdots, \boldsymbol{jt}_n\}$ resulting from the approach of Section 4.2.6 are taken into account. First, two input clusters C_i and C_j are transformed so that their joints overlap. Next, the principal axes of C_i and C_j are aligned to guess an initial alignment that should further reduce the computation steps. Then, C_i is iteratively rotated around its joint into the direction of an decreasing least squared error e in order to overlap with C_j. By iteratively updating e after each rotational step, the most ideal overlapping position between the linked rigid parts can be detected. Thereby, the joints are used as weights for the error calculation. A point $\boldsymbol{p}_i$ being located far away from its allocated joint $\boldsymbol{jt}_j$ does not contribute as much to the matching error as points located near the joint. A weight w is calculated by comparing the Euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{jt}_j)$ to the maximal distance $d_{\max}$ between $\boldsymbol{jt}_j$ and the furthest allocated point, that is

$$w_i = \begin{cases} \|\boldsymbol{p}_i - \boldsymbol{jt}_j\| \cdot \frac{1}{d_{\max}} & \text{if } w_i > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

In case of a weight below 0.5, which indicates that a point is far located from a joint, the weight is determined to be 0. Subsequently, a total matching error

$$e = \sum_{i=1}^{m} \|\boldsymbol{p}_i - \boldsymbol{q}_i\|^2 \cdot (1 - w_i)^2, \tag{4.14}$$

is calculated by combining the distance of a point of C_i to its closest point of C_j and the joint weight w. The error is squared, in order to further weaken the influence of cluster points that are located far away from the joint. The final detected rigid parts are represented by the biggest clusters, resulting from a rotation with the lowest error e. Lastly, all points with a closest point below a certain threshold τ are taken as input for region growing to reject possible overlapping points from other rigid parts. It is assumed, that two linked rigid parts are almost similarly aligned, therefore the distance threshold τ is considerable small. The largest cluster is finally returned as detected linked part and taken as input for the current LRP (see Figure 4.12). The whole approach proceeds with this LRP to detect further linked rigid parts. If no further linked parts exist, another unmatched cluster C_U with no allocation to a rigid part P is segmented by joint rotation. With varying distance thresholds τ for the selection of closest points or region growing, either target points may be skipped, or unnecessary points from another rigid part are added. Those circumstances lead to difficulties for the detection of linked rigid parts. Detailed results about this appearance are discussed in Section 4.4.



(a) (b) (c)

Figure 4.12: The joints $\boldsymbol{jt}_i$ and $\boldsymbol{jt}_j$ (green dots) of the reference cluster C_i (red) and the target cluster C_j (blue) are overlapped and the principal axis are oriented similarly. The associated target points for C_i are computed (a). Stepwise rotation of C_i onto C_j in the direction of a decreasing matching error e until its minimal value is achieved (b). The largest overlapping cluster of C_i and C_j given a threshold τ is assigned to the current LRP (c).

4.3 Implementation

For the implementation in Java the individual steps of the *Largest Rigid Part Algorithm* have been split into individual classes; this is to give it a better overview (see Figure 4.13). Again ImageJ was used as a processing library.

4.3.1 Iterative Segmentation

The iterative segmentation algorithm is performed in the `Segmentation` class, in which all steps proposed in Section 4.2 are implemented (see Algorithm 4.1). The segmentation will proceed until no unclustered points of C_1 and C_2 remain. In the first step of the iterative segmentation approach, rigid parts that have already been detected are removed from the unclustered points (`removeAllLRPs()`). Next, clusters are detected either for an initial input, or in the case of an already existing LRP, to detect its linked rigid parts. Joints are only estimated if there is a previously detected LRP; this is the case for all iterations except for the first one. Then, all matching clusters are pushed on a `Stack` (`pushMatchingClusters()`) to be processed in the form of a "last in - first out" process. In the case of an empty stack, all clusters have been processed and the algorithm terminates:

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

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

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

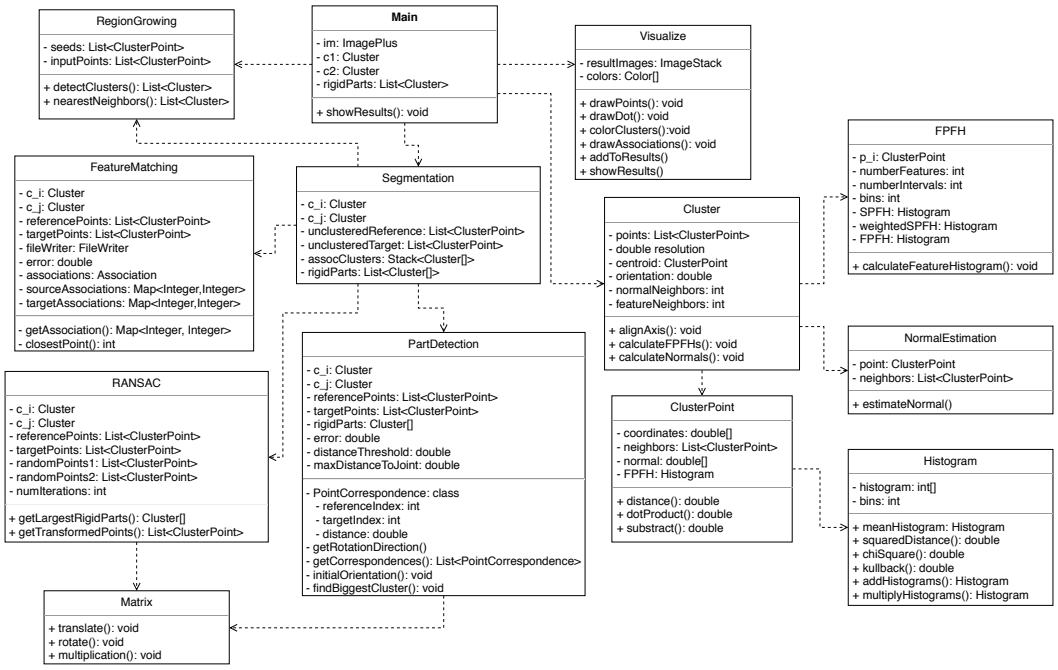


Figure 4.13: UML diagram of the classes related to the implementation of the feature-based segmentation approach.

```
    }

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

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

If unmatched clusters C_U still exist, two corresponding clusters are popped from the Stack. In the absence of allocated joints, no LRP has been detected yet. Therefore, the `FeatureMatching` class (see Section 4.3.2) is applied to two corresponding clusters. In combination with RANSAC, an initial LRP can be detected. On the other hand, two LRPs are obtained by joint rotation of two corresponding clusters onto one another. In this case, the `PartDetection` class is used to detect linked rigid parts from the previously detected LRPs:

```
    {
    ...
    currentClusters = clusters.pop();

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

```
      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();
    }

    largestRigidParts.add(currentLrps);
  }
```

4.3.2 Feature Matching

For the calculation of point normals required for the feature histograms, the class `NormalEstimation` is developed. It takes a point $\boldsymbol{p}_i$ with its k neighbors as input. Then, the least fitting line on these input points is detected (as described in Section 4.1.1), and the smallest lambda value λ_2 is selected. By setting the x-value of the normal to 1.0, the y-value can be calculated by inserting λ_2 into the Equation 4.3. The resulting vector represents the normal vector $\boldsymbol{n}_i$ for $\boldsymbol{p}_i$. If it is oriented exactly vertically or horizontally the resulting y-value is either *infinite* or `NaN`. In these cases, the normal is either set to (0,1) or (1,0). Lastly, 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` class was implemented to store the normal $\boldsymbol{n}_i$ for each point and the resulting feature histogram (FPFH). This is stored in the form of a `Histogram` object containing an `int[]` array for categorizing the feature values. For the calculation

Algorithm 4.1: Iterative segmentation algorithm to acquire the corresponding rigid parts $\mathcal{P}$ from the unclustered points $\mathcal{U}_r$ of the reference cluster C_1 and $\mathcal{U}_t$ of the target cluster C_2. In each iteration, clusters that have not been assigned to a rigid part P are detected by region growing. Corresponding clusters C_U are stored in a Stack C, in order to process one cluster pair at a time by a *LIFO* procedure. For the initial iteration, feature matching is applied on two clusters C_U. The detected correspondences c are taken as input for RANSAC to detect the *LRPs* of two corresponding clusters C_U. For all further iterations, the *LRP* is detected by joint rotation of two corresponding clusters C_U.

```
1:  SEGMENTATION(C_1, C_2)
    Input: the reference and target cluster C_1 and C_2
    Returns: the rigid parts P of the input clusters
2:      U_r ← C_1
3:      U_t ← C_2
4:      C ← ()
5:      m ← ⌊C⌋
6:      n ← ⌊U_r⌋
7:      P ← ()
8:      LRP_r ← ()
9:      LRP_t ← ()
10:     while n > 0 ∨ m > 0 do
11:         U_r ← U_r − LRP_r
12:         U_t ← U_t − LRP_t        ▷ remove LRPs from all unclustered points U_r and U_t
13:         r ← clusters(U_r)
14:         t ← clusters(U_t)
15:         C_U ← MATCH(r, t)
16:         C ← push(C_U)                ▷ push corresponding cluster C_U on the stack C
17:         if LRP ≠ null then
18:             JOINTESTIMATION(C)
19:         end if
20:         current ← pop(C)                    ▷ pop one cluster set from the stack C
21:         if joint(current) = null then
22:             c ← FEATUREMATCHING(current)
23:             LRP ← RANSAC(c)
24:         else
25:             LRP ← PARTDETECTION(current)
26:         end if
27:         P ← P + LRP
28:     end while
29:     return P
30: end
```

of a feature histogram of a point $\boldsymbol{p}_i$, its k neighbors are considered. The `FPFH` class implements various operations for vectors, such as the dot or cross product to implement the equation from Section 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);
}
```

The three feature values of a point $\boldsymbol{p}_i$ result from the computation of three angles between a point set $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$:

```
double feature1 = dot(v, n_j);
double feature2 = dot(u, p_j.subtract(p_i)) / p_j.distance(p_i);
double feature3 = Math.atan2(dot(w, n_j), dot(u, n_j));
```

By setting a minimum and maximum value a feature value can adopt, as well as the desired number of intervals, a feature is classified into a certain interval. The resulting interval is used to calculate the index in the histogram to be incremented:

```
private int getInterval(double feature, double min, double max) {
  feature -= 0.001; // avoid an out of range interval in case of feature = max
  double range = (max - min) / numberIntervals;
  return (int) ((feature - min) / range);
}
```

Subsequently, point correspondences are detected by computing the distances between all feature histograms of C_1 and C_2 (see Section 4.2.4). For that, the closest point algorithm is configured to use the distance between histograms instead of the Euclidean distance between points. Reciprocal point correspondences between C_1 and C_2 are returned in form of point indices `Map<Integer,Integer>` of the clusters:

```
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);
  }
}
```

4.3.3 RANSAC

The RANSAC algorithm takes the computed sparse correspondences between two clusters in the form of a `Map<Integer, Integer>` as input. In each iteration, two random point correspondences are selected from the map to calculate an affine transformation a between the selected points from each C_i and C_j. The reference points from C_i are filled into a transformation matrix TM. The assumed target points from C_j represent the vector b. By utilizing the `DecompositionSolver` class, the affine transformation, represented by the vector a, that is

$$\underbrace{\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{pmatrix}}_{TM} \cdot \underbrace{\begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \end{pmatrix}}_{b}, \tag{4.15}$$

can be computed:

```
double[][] TM = fillTransformMatrix(referencePoints);

double[] b = new double[] {
  targetPoints.get(0).getX(), targetPoints.get(0).getY(),
  targetPoints.get(1).getX(), targetPoints.get(1).getY()
};

DecompositionSolver solver = new SingularValueDecomposition(MatrixUtils.
    createRealMatrix(TM)).getSolver();

RealVector a = solver.solve(MatrixUtils.createRealVector(b));
```

In the end of an iteration, a region growing approach with a threshold τ is conducted on all overlapping points, resulting from the applied affine transformation a on the reference points. The value is thereby considerable small, because a right alignment during any iteration is assumed. The biggest clusters are stored and, after termination of the RANSAC procedure, returned as the LRPs. The whole procedure is considerably time consuming. The total run-time can be reduced by taking a smaller number of correspondences as input, which directly affects the required number of iterations until the right match is detected.

4.3.4 Joint Rotation

A main approach required for the detection of linked rigid parts to an already detected LRP, is the rotation around the estimated joints of corresponding clusters. First, the reference points from C_i and the target points from C_j are translated to the origin and similarly aligned:

```
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());
```

Next, the reference points are aimed to be rotated onto the target points. This is achieved iteratively by applying a stepwise rotation in the direction of an reduced matching error e, which is computed beforehand. By terminating at the point where the weighted total error increases again, the assumed biggest overlap is achieved:

```
for (PointCorrespondence pointCorrespondence : pointCorrespondences) {
  ClusterPoint referencePoint = referencePoints.get(pointCorrespondence.
    referenceIndex);

  double distanceToJoint = referencePoint.distance(new ClusterPoint(0, 0));
  double currentError = pointCorrespondence.distance;
  double weight = (1 - distanceToJoint / maxDistanceToJoint);
  weight = weight < 0.5 ? 0.0 : weight;
  totalError += currentError * Math.pow(weight, 2);
}
```

To only remain the rigid part and reject corresponding points that do not belong to the desired rigid part, a region growing algorithm is applied. As a result, only the largest cluster is kept as successful detected rigid part (see Algorithm 3.1).

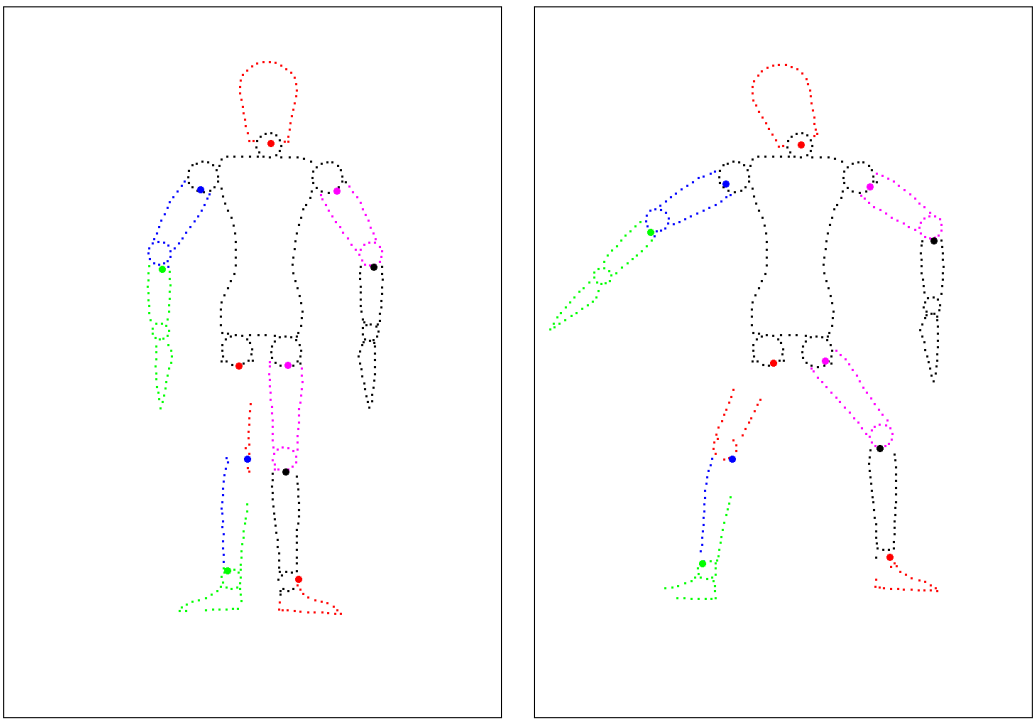
4.4 Results

Applying the proposed approach on the 2D data set of an articulated object (see Section 4.2.2), different segmentation results could be achieved with adjusted parameters (see Table 4.1). These are based on the estimated resolution of 8.01 and 8.2 for C_1 and C_2. The results of iteration 1 show that almost all rigid parts and joints could be detected correctly. The segmentation errors result from the imprecise joint estimation of corresponding rigid parts (see Figure 4.15). Consequently, it is assured that the computed closest two points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ of a cluster C_U to the LRP are not neighboring points by comparing the Euclidean distance $d(\boldsymbol{p}_i, \boldsymbol{p}_j)$ to a threshold τ. In a positive case, the third closest point $\boldsymbol{p}_k$ to the LRP is selected instead of the second closest point. By doing so, improved joint estimation results could be achieved. Another occurring

Parameter	*Description*
FN	Number of neighbors for the feature histogram
NN	Number of neighbors for the normal computation
RANSAC	Closest point distance for RANSAC
JR	Closest point distance for joint rotation
W	Minimum joint weight of a point to influence the error
DC	Distance criterion for feature matching

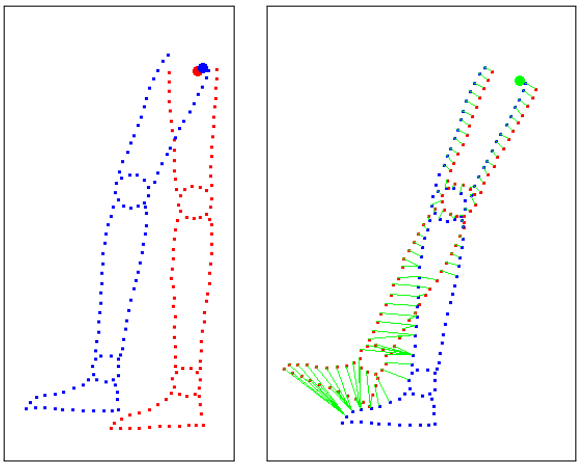
Iteration	*FN*	*NN*	τ *RANSAC*	τ *JR*	*W*	*DC*
1	6	2	3.0	6.0	0.5	Euclidean
2	6	2	3.0	7.0	0.5	Euclidean
3	6	2	3.0	6.0	0.3	Euclidean
4	4	2	3.0	6.0	0.3	Euclidean
5	10	2	3.0	7.0	0.3	Chi-Square
6	6	2	3.0	6.0	0.3	Kullback-Leibler

Table 4.1: Segmentation parameters to be adjusted for different segmentation results.



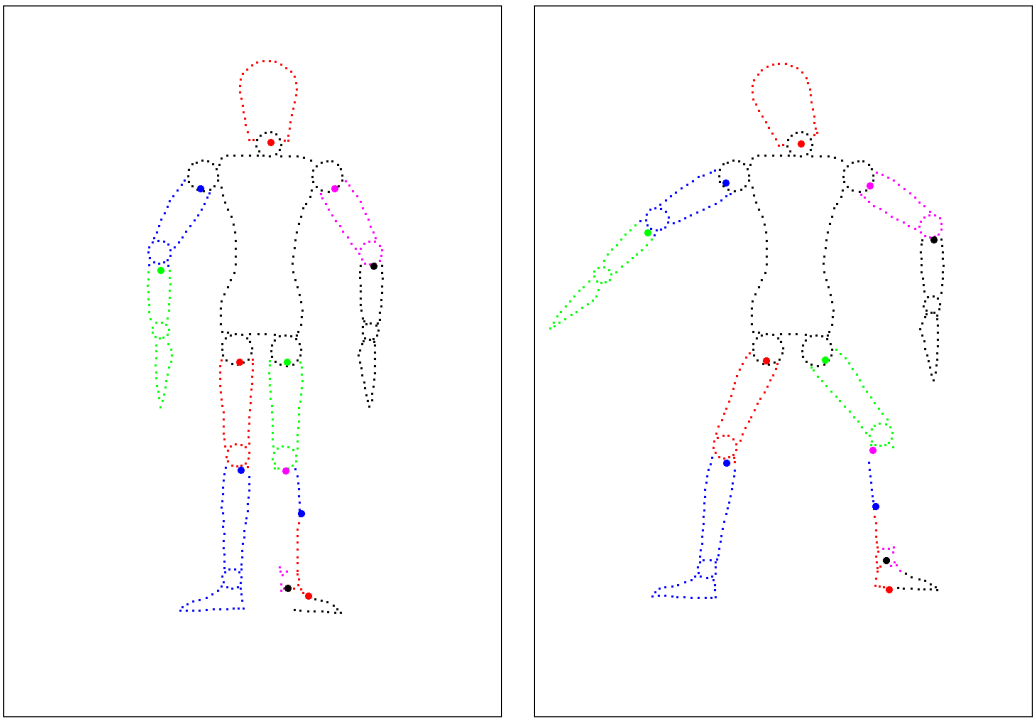
(a) (b)

Figure 4.14: Segmentation results for C_1 (a) and C_2 (b) with the parameters of iteration 1 (see Table 4.1).



(a) (b)

Figure 4.15: An imprecise estimation of joints of corresponding clusters (a) leads to a considerable point offsets during the rotation of the reference cluster (red) onto the target cluster (blue) (b).



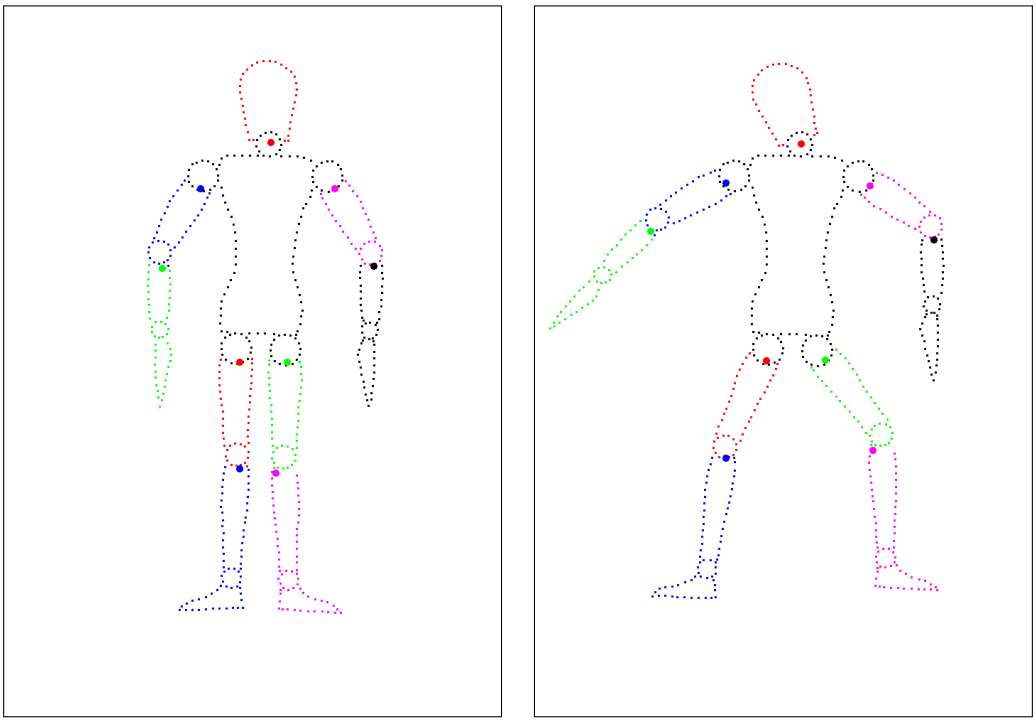
(a) (b)

Figure 4.16: Segmentation results for C_1 (a) and C_2 (b) with the parameters of iteration 2 (see Table 4.1). By improving the joint computation and adjusting the parameter JR, a more precise segmentation compared to iteration 1 could be achieved.

problem was the incorrect detection of rigid parts with no further linked parts (see 4.16). The reason is that only the closest 50% of the points to the joint are considered for the error calculation. As a result, no precise overlapping between C_1 and C_2 could be obtained. To solve the problem of imprecise overlapping of two rigid parts resulting from joint rotation, the weight W was decreased to 0.3. Consequently, more points located far away from the joint contribute to the total matching error. The resulting segmentation into rigid parts is considered as successful (see Figure 4.17). In order to focus on the main goal of reducing the number of computations steps, improvements could be done for the feature matching and RANSAC approach. To achieve that, different histogram similarity criteria are applied for the feature matching (see Section 4.4.1). The increase of histogram bins by providing a higher number of intervals q did not lead to an improved alignment. Therefore the minimum number of two intervals q was retained.

4.4.1 Histogram Distances for Feature Matching

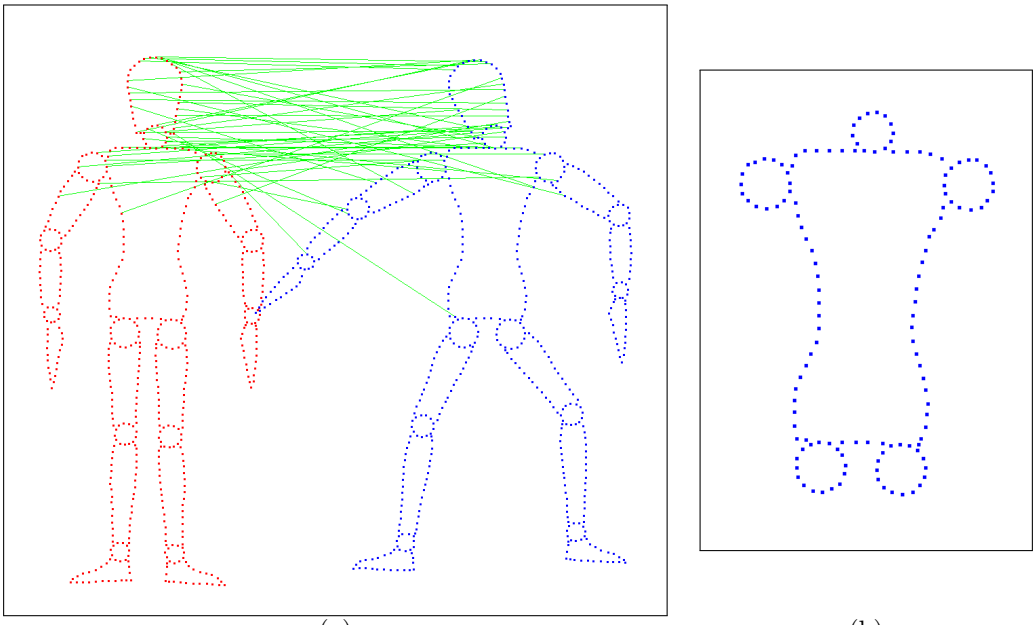
Three different distances are considered as histogram similarity criteria: the Euclidean, the Chi-Square and the Kullback-Leibler distance. Compared to the other two histogram distances, the Euclidean distance results in the highest number of point correspondences. This leads to a higher number of RANSAC iterations. On the other hand, also the number of correct correspondences is higher. Thus, the overall probability of detecting the



(a) (b)

Figure 4.17: Segmentation results for C_1 (a) and C_2 (b) with the parameters of iteration 3 (see Table 4.1). By adjusting the weight parameter W, a successful segmentation into all rigid parts could be achieved.

LRP is considerable high. A successful initial alignment can be achieved by only considering four neighbors for the feature computation ($FN = 4$), and two neighbors for the normal estimation ($NN = 2$) (see Figure 4.18). The number of RANSAC iterations is set to 250. In the case of applying the Chi-Square distance, fewer correspondences are detected in comparison to the Euclidean distance. However, the ratio of correct correspondences is too low for an successful detection of the LRP (see Figure 4.19). Consequently, the Chi-Square distance is determined as not useful for this segmentation approach. Finally, the Kullback-Leibler distance was applied with an enlarged feature neighborhood ($FN = 6$). There are fewer point correspondences detected in comparison to the Euclidean distance. However, the ratio of correct correspondences is higher. As a consequence, the number of RANSAC iterations can be reduced to 100 for a successful alignment (see Figure 4.20). In a further step, the Euclidean distance and the Kullback-Leibler distance are compared in terms of their runtime, as they both acquire successful alignment results of the actual LRP. The Euclidean distance requires fewer neighboring points for the computation of feature histograms. In contrary, the Kullback-Leibler distance has a higher ratio of correct point correspondences. By comparing the run time, the Kullback-Leibler represents the fastest choice with 8.0731 seconds for the segmentation approach, followed by the Euclidean distance with 8.8683 seconds. Completing the segmentation approach several times, the run time may considerably vary, depending on background calculations, such as the visualization of segmentation results. As the



(a) (b)

Figure 4.18: Applying the Euclidean distance on the feature histograms of C_1 and C_2 with the parameters of iteration 4 (see Table 4.1), in order to detect point correspondences (a). The initial LRP can be detected successfully witb 250 RANSAC iterations (b).

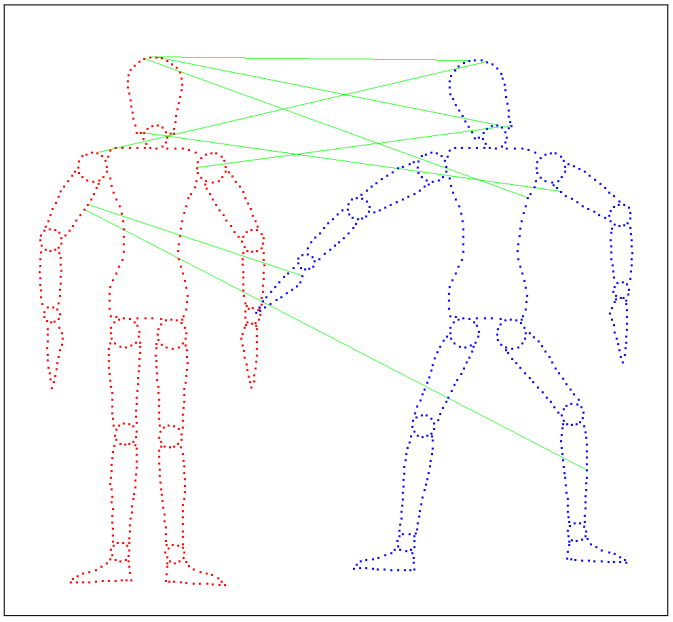
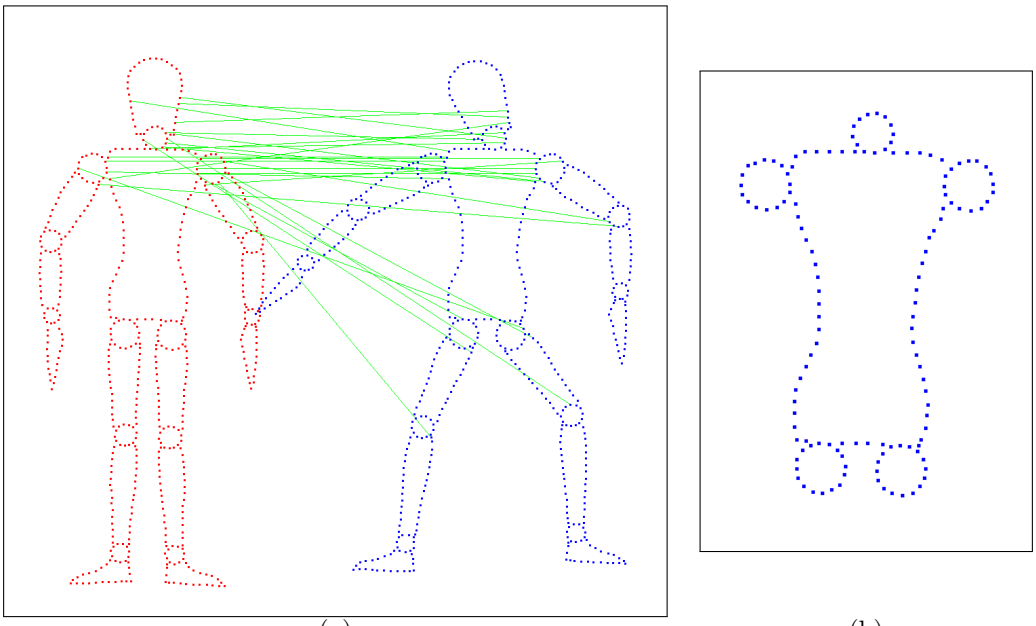


Figure 4.19: Applying the Chi-Square distance on the feature histograms of C_1 and C_2 with the parameters of iteration 5 (see Table 4.1), in order to detect point correspondences.



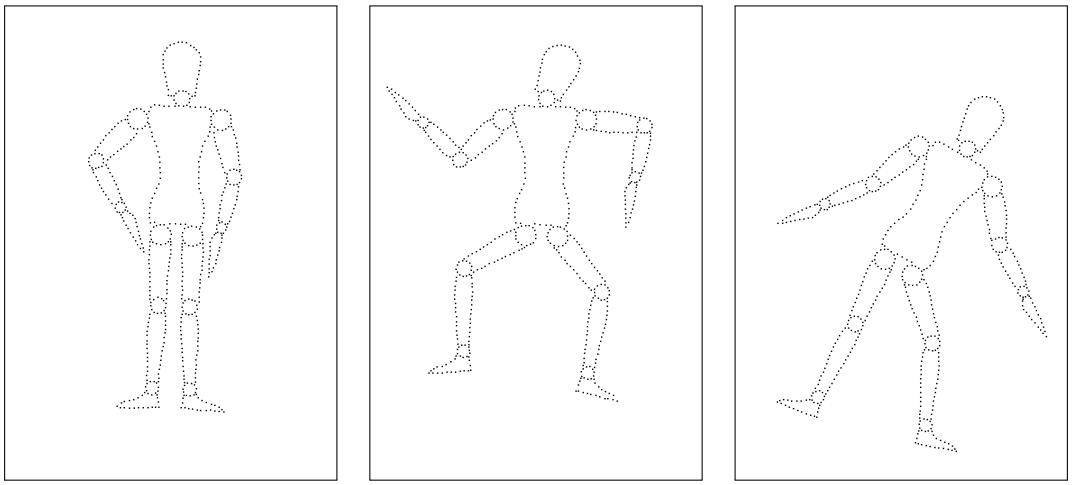
(a) (b)

Figure 4.20: Applying the Kullback-Leibler distance on the feature histograms of C_1 and C_2 with the parameters of iteration 6 (see Table 4.1), in order to detect point correspondences (a). The initial LRP can be detected successfully witb 100 RANSAC iterations (b).

goal of this thesis is the reduction of the number of computation steps, the run time is not further considered. Summarizing, the proposed segmentation approach successfully detects the rigid parts of an articulated object whose poses are not considerable different from one another. Therefore, the approach was applied on further poses in order to face the difficulties of considerable different transformations, orientations and partial occlusion (see Figure 4.21). In the case of these input poses, the segmentation approach fails. The reason is that there are not sufficient correct point correspondences detected between two poses (see Figure 4.22), which results from ambiguous body parts. As a result, during the RANSAC iterations the LRP is not detected.

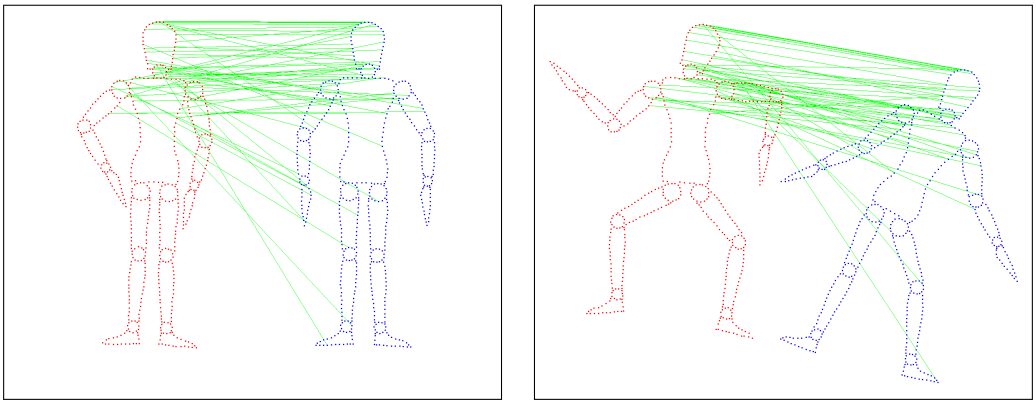
4.4.2 Main Drawbacks

The main drawback of the proposed segmentation algorithm regards the first initial alignment of C_1 and C_2 to detect the actual LRP of the articulated object. In the case of a failure, the linked rigid parts can not be detected, as they are directly dependent on a successful initial alignment. By increasing the number of RANSAC iterations, the probability of a correct initial alignment is increased; however, it also directly affects the runtime and should therefore not be exaggerated. A major factor for the successful detection of a LRP is the input data in the form of a 2D hull of the object. As the object's surface is imitated by 2D points, the region growing is much more error-prone. The reason for this is that unlike with 3D, the points of a rigid part in 2D have a considerably lower number of neighbors. If there are a few missing points from a rigid part, it will not be fully detected during the region growing due to the resulting point



(a) (b) (c)

Figure 4.21: The segmentation approach is applied on three more complex poses that face partial occlusion (a), considerable different transformations of rigid parts (b) and orientations (c). In these case the initial detection of the LRP fails which leads to an unsuccessful segmentation.



(a) (b)

Figure 4.22: The point correspondences, applying the Euclidean distance, between more complex poses, facing partial occlusion (a) and considerably different transformed rigid parts and orientation (b).

offset. This is especially drastically during the RANSAC approach, as it results in the initial LRP. To counteract this behavior, more and closer hull points may be added to the input mesh. A further main problem is that the algorithm proceeds iteratively from already detected rigid parts. This makes the whole procedure notably unstable and error-prone, because one incorrect rigid part detection could lead to an overall unsuccessful segmentation. Furthermore, occlusion of rigid parts (e.g. the hand touches the leg) poses difficulties, as the region growing would detect those movements as potentially linked clusters. The difficulty of ambiguous rigid parts results from the simple 2D data set

which is the main reason that the proposed algorithm fails for most of the data sets. Most of the discussed drawbacks concerning the input data can be overcome by an implementation in 3D (see Section 4.5).

4.5 3D Implementation

The next step would be to transfer the optimized 2D implementation into 3D to reduce the number of computation steps from [11] (see Section 4.2.1). It can be implemented by using the *Point Cloud Library* [2] which operates on 3D point clouds. Certainly, many new challenges must be overcome due to the three-dimensional space. Straightforward rotation of linked rigid parts around their estimated joints will pose a major difficulty in 3D, since multiple rotation axes are available. In return, it is noticeable that PCL offers many required functions by default. For example, both *Fast Point Feature Histograms* completed with the normal estimation and sub sampling of dense point clouds are already provided. Especially, the feature histogram computation can be conducted with a few lines, by computing all point normals:

```
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normalEstimation;
pcl::PointCloud<pcl::Normal>::Ptr referenceNormals(new pcl::PointCloud<pcl::Normal>)
    ;

normalEstimation.setInputCloud(referenceCloud);
normalEstimation.setKSearch(5);
normalEstimation.compute(*referenceNormals);
```

As a next step, those computed normals are taken as input for the computation of the feature histograms of the point clouds:

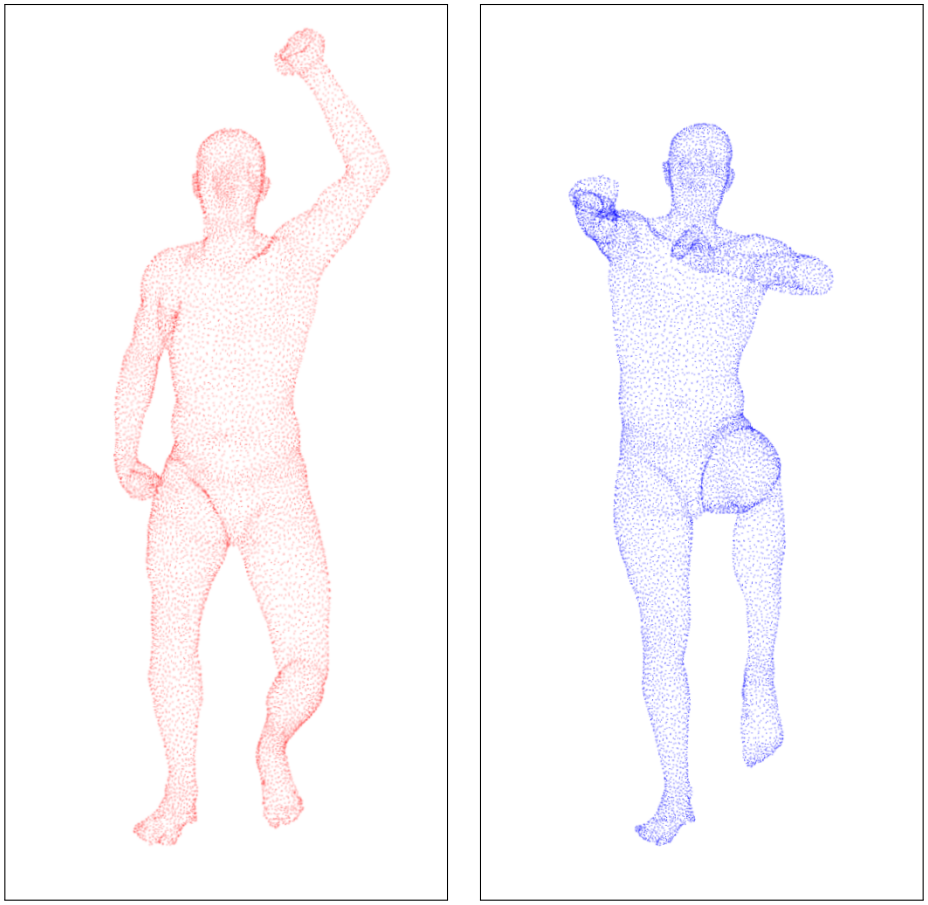
```
pcl::FPFHEstimation<pcl::PointXYZ, pcl::Normal> fpfhEstimation;

pcl::PointCloud<pcl::FPFHSignature33>::Ptr referenceFeatures(new pcl::PointCloud<pcl
    ::FPFHSignature33>);

fpfhEstimation.setKSearch(5);
fpfhEstimation.setInputCloud(referenceCloud);
fpfhEstimation.setInputNormals(referenceNormals);
fpfhEstimation.compute(*referenceFeatures);
```

An appropriate dataset for the 3D segmentation constitutes the SCAPE data set [2], which provides scans of a human in different poses (see Figure 4.23).

[2] http://pointclouds.org



(a) (b)

Figure 4.23: Taking two 3D poses of an articulated object as input (a), (b), the segmentation approach can be implemented in 3D.

Chapter 5

Conclusion

In the first project phase, intense research was conducted regarding pose estimation to detect a main issue to focus on for this thesis. It was essential to form an overall perspective on the state-of-the-art methods of pose estimation of articulated objects. During this process the field of unsupervised pose estimation of unknown objects was frequently referred to because its related approaches perform completely independently of user input. Therefore, the non-rigid registration became a major indicator for possible optimizations. Taking existing, unsupervised methods as a reference (see Section 2.4), two approaches for the segmentation of an articulated object into its rigid parts were developed. Thereby, the main goal of this thesis became the reduction of the number of computation steps in the segmentation procedure. The first approach was a linear, straightforward, divide-and-conquer procedure. It recursively divides two 2D point clouds of the same object in different configurations into matching sub clusters (see Chapter 3). The subdivision was performed with a cluster tree and depth-first traversal to segment the point clouds from one side to the other. Next, all neighboring clusters were verified to be merged, in case they represented a subdivided rigid part. After merging, the rigid parts of the objects may be obtained. The second approach to be implemented tried to compensate for the drawbacks of the linear approach, specifically the segmentation of articulated objects with a skeletal structure (e.g. a human) (see Chapter 4). This approach uses both feature matching [23] and RANSAC for the initial alignment of the two point clouds to detect the *Largest Rigid Part* – LRP. Due to the motion constraints of rigid parts by their joints, linked rigid parts to the LRP could be detected. The reference paper for this approach is represented by Guo et al. [11].

5.1 Achieved Results

The linear approach aimed for a drastic decrease in the number of computation steps required for the segmentation of an articulated object. To do this, the segmentation relied only on the point coordinates of the input clusters C_1 and C_2 and the initial orientation of clusters. The proximate rigid parts could be detected for simple objects, composed of a few rigid parts that are linked like a chain. However, the results regarding points located near an actual joint were not precise. In the case of more complex objects,

the approach in its simple form failed. Different optimization possibilities were proposed to counteract most of the issues that emerged. Eventually, it became evident that a more advanced approach would be required for the segmentation of a complex, articulated object.

Next, a feature-based approach was developed to overcome the main difficulty of detecting reliable point correspondences between C_1 and C_2. Feature matching in combination with RANSAC was applied for the initial alignment and the detection of the actual rigid part. The resulting LRP could be detected successfully. This approach contributes to the subsequent detection of further linked rigid parts to the LRP. It utilizes the constrained translation of rigid parts that result from their estimated joints. Unlike the approach from [11], with this approach only a stepwise rotation around the joint is required to detect a rigid part. For the successful detection of an actual linked rigid part to the LRP, joint weights were introduced. Thus, there are considerably fewer steps needed in comparison to both the recursive application of the feature matching and RANSAC approaches. By adjusting the different thresholds used in the implementation, the correct number of joints and rigid parts could be detected in two different poses. Furthermore, a successful correspondence of rigid parts and joints could be achieved for a specific data set. Subsequently, the results are satisfying, if not necessarily precise. By testing further poses of the same articulated object, no successful segmentation could be achieved. This behavior can be explained by the input data in the form of a 2D hull of the object which was created manually for this specific 2D-use case. Especially the ambiguous body parts of the input mesh pose a major difficulty for the detection of correct point correspondences.

Overall, the greatest achievement is the knowledge gained about pose estimation of articulated objects regarding segmentation and surface registration. Since the implementation of basic algorithms required for the two proposed segmentation approaches was conducted from scratch, a deep understanding of concepts such as feature matching was acquired. Additionally, the newly acquired familiarization with scientific papers is noteworthy.

5.2 Main Difficulties and Drawbacks

A key difficulty at the beginning of the research phase was the vast amount of references about pose estimation. It was difficult to uncover one key aspect to focus on for this thesis. A general overview had to be created to familiarize myself with the relevant material. To gain a deep understanding of the reference approaches, the implementation was conducted in 2D to fully focus on possible segmentation optimizations. When describing a 3D object in 2D, important information about its 3D pose is lost. Thus, it is impossible to gain the actual pose without being confronted with ambiguous poses. For this reason no transformations towards the z-axis were assumed for the implementation of the two approaches. Due to the limited amount of available test data, the manual creation of 2D point clouds of an articulated object in different configurations was also a time-consuming procedure. Since the success of segmentation approaches are directly dependent on appropriate data sets, multiple datasets were manually created. Certain requirements concerning the input data had to be fulfilled such as achieving a certain density of data points to make it less error-prone for region growing.

A drawback of the linear approach is that it can only achieve successful results if a simple object with a rigid part only linked to a maximum of two other parts is given as input. For the segmentation of complex objects it generally fails and is therefore not useful for the desired goal.

The main drawback of the proposed feature-based approach is that it is dependent on a successful initial alignment of two different poses. If this first main step fails, the pose of the articulated object cannot be successfully extracted. Another drawback is the application of RANSAC, as too many iterations are required for a correct alignment of C_1 and C_2. The main deficit is the 2D test data which is not ideal for a precise segmentation into rigid parts. However, this condition can be overcome by conducting additional improvements (see Section 5.3).

5.3 Future Work

To overcome the difficulty of manually creating test data form articulated objects in 2D, the object's hull could be directly computed from the silhouette of the real-world object. It would result in a high number of points that could be used for the segmentation step. The next step would be the implementation of the algorithm in 3D, based on the implementation of Guo et al. The focus would be thereby to implement an optimized segmentation into rigid parts to reduce the number of computation steps (similar to the approach in 2D). Another interesting possibility would be to take the computed pose of an object as input for a machine-learning approach. It could result in a collection of different template shapes. With a growing learning phase, a considerably accelerated pose estimation mechanism could be created. Another objective would be to integrate an optimized segmentation approach into an actual pose capture application to transfer the pose resulting from the joints of a real object onto a digital character.

Appendix A

CD Contents

The enclosed CD includes the thesis in digital form and additionally all used reference images, results and illustrations. Furthermore, the source code of the two segmentation approaches with all dependencies is included.

Path: /

Thesis.pdf	Thesis in digital form
SourceCode.zip	Source Code of the Java Implementation

Path: /images/illustrations

*.ai	Original Adobe Illustrator-Files
*.pdf	Original pdf images

Path: /images/results

*.png	Original pixel images

Path: /images/references

*.png	Original pixel images

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*. AUAI Press. 2004, pp. 18–26 (cit. on pp. 8, 9, 11, 14).

[2] Dragomir Anguelov et al. “SCAPE: shape completion and animation of people”. In: *ACM Transactions on Graphics (TOG)*. Vol. 24. 3. ACM. 2005, pp. 408–416 (cit. on p. 53).

[3] 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 pp. 8, 10).

[4] 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 pp. 7, 8).

[5] Paul Besl and Neil 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. 4).

[6] Will Chang and Matthias Zwicker. “Automatic Registration for Articulated Shapes”. In: *Proceedings of the Computer Graphics Forum (Proceedings of Symposium on Geometry Processing 2008)*. Vol. 27. 5. Copenhagen, Denmark, 2008, pp. 1459–1468 (cit. on pp. 9, 10).

[7] Will Chang and Matthias Zwicker. “Range Scan Registration Using Reduced Deformable Models”. In: *Proceedings of the Computer Graphics Forum (Proceedings of Eurographics 2009)*. Vol. 28. 2. Wiley Online Library. 2009, pp. 447–456 (cit. on p. 10).

[8] Fernando De Goes, Siome Goldenstein, and Luiz Velho. “A hierarchical segmentation of articulated bodies”. In: *Proceedings of the Computer Graphics Forum*. Vol. 27. 5. Wiley Online Library. 2008, pp. 1349–1356 (cit. on p. 10).

[9] Martin Fischler and Robert Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Readings in Computer Vision*. Elsevier, 1987, pp. 726–740 (cit. on p. 4).

[10] Song Ge and Guoliang Fan. “Articulated Non-Rigid Point Set Registration for Human Pose Estimation from 3D Sensors”. *Sensors* 15.7 (2015), pp. 15218–15245 (cit. on pp. 5, 7).

[11] 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. 3, 9–11, 26, 27, 31, 32, 39, 53, 55, 56).

[12] Günter Hetzel et al. “3D object recognition from range images using local feature histograms”. In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition, 2001.* Vol. 2. IEEE. 2001, pp. II–II (cit. on p. 31).

[13] Hugues Hoppe et al. “Surface Reconstruction from Unorganized Points”. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’92. New York, NY, USA: ACM, 1992, pp. 71–78 (cit. on p. 27).

[14] Emre Kalafatlar and Yucel Yemez. “3d articulated shape segmentation using motion information”. In: *Proceedings of the 20th International Conference on Pattern Recognition (ICPR), 2010*. IEEE. 2010, pp. 3595–3598 (cit. on p. 8).

[15] Fabian Langguth et al. “Shading-aware multi-view stereo”. In: *Proceedings of the European Conference on Computer Vision*. Springer. 2016, pp. 469–485 (cit. on p. 7).

[16] Ita Lifshitz, Ethan Fetaya, and Shimon Ullman. “Human pose estimation using deep consensus voting”. In: *Proceedings of the European Conference on Computer Vision*. Springer. 2016, pp. 246–260 (cit. on p. 8).

[17] Lu Lou et al. “Accurate multi-view stereo 3D reconstruction for cost-effective plant phenotyping”. In: *Proceedings of the International Conference Image Analysis and Recognition*. Springer. 2014, pp. 349–356 (cit. on p. 7).

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

[19] Niloy Mitra, Leonidas Guibas, and Mark Pauly. “Symmetrization”. In: *Proceedings of the ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 63 (cit. on p. 10).

[20] Greg Mori and Jitendra Malik. “Estimating human body configurations using shape context matching”. In: *Proceedings of the European Conference on Computer Vision*. Springer. 2002, pp. 666–680 (cit. on p. 7).

[21] Li Peng and Wang Jian. “Improved algorithm for point cloud registration based on fast point feature histograms”. *Journal of Applied Remote Sensing* 10.5 (2016), pp. 10–23 (cit. on p. 5).

[22] Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. “Reconstructing 3d human pose from 2d image landmarks”. In: *Proceedings of the European Conference on Computer Vision*. Springer. 2012, pp. 573–586 (cit. on p. 7).

[23] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration”. In: *Proceedings of the International Conference on Robotics and Automation, 2009. ICRA'09.* IEEE. 2009, pp. 3212–3217 (cit. on pp. 27, 30, 55).

[24] Radu Bogdan Rusu et al. “Persistent point feature histograms for 3D point clouds”. In: *Proceedings of the 10th International Conference for Intelligent Autonomous Systems (IAS-10), Baden-Baden, Germany.* IOS Press. 2008, pp. 119–128 (cit. on p. 27).

[25] Avinash Sharma, Etienne Von Lavante, and Radu Horaud. “Learning shape segmentation using constrained spectral clustering and probabilistic label transfer”. In: *Proceedings of the European Conference on Computer Vision.* Springer. 2010, pp. 743–756 (cit. on p. 8).

[26] Aravind Sundaresan and Rama Chellappa. “Markerless motion capture using multiple cameras”. In: *Proceedings of the Computer Vision for Interactive and Intelligent Environment, 2005.* IEEE. 2005, pp. 15–26 (cit. on p. 7).

[27] Aravind Sundaresan and Rama Chellappa. “Segmentation and probabilistic registration of articulated body models”. In: *Proceedings of the 18th International Conference on Pattern Recognition, 2006. ICPR 2006.* Vol. 2. IEEE. 2006, pp. 92–96 (cit. on p. 10).

[28] Gary 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. 4).

[29] Yuandong Tian, Lawrence Zitnick, and Srinivasa Narasimhan. “Exploring the spatial hierarchy of mixture models for human pose estimation”. In: *Proceedings of the European Conference on Computer Vision.* Springer. 2012, pp. 256–269 (cit. on p. 7).

[30] Shinji Umeyama. “Least-squares estimation of transformation parameters between two point patterns”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.4 (1991), pp. 376–380 (cit. on p. 18).

[31] Oliver Van Kaick et al. “A survey on shape correspondence”. In: *In Proceedings of the Computer Graphics Forum.* Vol. 30. 6. Wiley Online Library. 2011, pp. 1681–1707 (cit. on p. 6).

[32] Eric Wahl, Ulrich Hillenbrand, and Gerd Hirzinger. “Surflet-pair-relation histograms: a statistical 3D-shape representation for rapid classification”. In: *Proceedings of the 4th International Conference on 3-D Digital Imaging and Modeling, 2003.* IEEE. 2003, pp. 474–481 (cit. on p. 31).

[33] Svante Wold, Kim Esbensen, and Paul Geladi. “Principal Component Analysis”. *Chemometrics and Intelligent Laboratory Systems* 2.1-3 (1987), pp. 37–52 (cit. on p. 4).

[34] Stefanie Wuhrer and Alan Brunton. “Segmenting animated objects into near-rigid components”. *The Visual Computer* 26.2 (2010), pp. 147–155 (cit. on p. 8).

[35] Bangpeng Yao and Li Fei-Fei. “Action recognition with exemplar based 2.5 d graph matching”. In: *Proceedings of the European Conference on Computer Vision*. Springer. 2012, pp. 173–186 (cit. on p. 7).