UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Object Detection and 3D Pose Estimation with Point Pair Features

Bachelorarbeit
zur Erlangung des Grades
BACHELOR OF SCIENCE
im Studiengang Informatik

vorgelegt von

Kevin Fischer Rios

Betreuer: Daniel Müller, M. Sc., Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Daniel Müller, M. Sc., Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im September 2022

Kurzfassung

Diese Bachelorarbeit erforscht eine Methode zur 3D-Objekterkennung und Posenschätzung, basierend auf dem Punkte-Paare-Eigenschaften-Verfahren (PPE) von Drost et. al. [Dro+10]. Die Methoden der Posenschätzung haben sich in den letzten Jahre zwar deutlich verbessert, stellen jedoch weiterhin ein zentrales Problem im Bereich der Computervisualistik dar. Im Rahmen dieser Arbeit wurde ein Programm implementiert, welches Punktewolkenszenen als Ausgangspunkt erhält und daraus eine Objekterkennung und Posenschätzung durchführt. Das Programm deckt alle Schritte eines Objekterkennungsprogramm ab, indem es 3D-Modelle von Objekten verarbeitet, um deren PPE zu extrahieren. Diese Eigenschaften werden gruppiert und in einer Tabelle gespeichert. Anhand des Auswahlverfahrens, bei dem die Übereinstimmung der Eigenschaften überprüft wird, können potenzielle Posen des Objekts ermittelt werden. Die Posen mit der größten Übereinstimmung werden miteinander verglichen, um ähnliche Posen zu gruppieren. Die Gruppen mit der höchsten Übereinstimmung werden erneut überprüft, sodass am Ende nur eine Pose ausgewählt wird. Das Programm wurde anhand von Real– und Simulationsdaten Daten getestet. Die erhaltenen Ergebnisse wurden anschließend analysiert und evaluiert.

Abstract

This thesis explores a 3D object detection and pose estimation approach based on the point pair features method presented by Drost et. al. [Dro+10]. While pose estimation methods have shown good improvements, they still remain a crucial problem on the computer vision field. In this work, we implemented a program that takes point cloud scenes as input and returns the detected object with their estimated pose. The program fully covers an object detection pipeline by processing 3D models during an offline phase, extracting their point pair features and creating a global descriptor out of them. During an online phase, the same features are extracted from a point cloud scene and are matched to the model features. After the voting scheme, potential poses of the object are retrieved. The poses end being clustered together and post-processed to finally deliver a result. The program was tested using simulated and real data. We evaluate these tests and present the final results, by discussing the achieved accuracy of the detections and the estimated poses.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den September 27, 2022

Contents

1 Introduction **9**
1.1 Motivation 9

2 Related Work **11**
2.1 Learning-based methods 11
2.2 Template-based methods 11
2.3 Feature-based methods 12

3 Method **15**
3.1 Point Pair Features 16
3.2 Model Features Extraction 17
3.2.1 Point Sampling with Triangle Interpolation 17
3.2.2 Poisson Disk Sampling 18
3.2.3 PPF extraction 20
3.3 Detection phase 22
3.3.1 Normal estimation 22
3.3.2 Scene downsampling 23
3.3.3 Pose retrieval 24
3.3.4 Feature matching 25
3.3.5 Pose voting 26
3.3.6 Clustering 27
3.3.7 Global Hypothesis Verification 28
3.3.8 ICP registration refinement 32

4 Experimental Results and Evaluation **35**
4.1 Simulated Experiments 35
4.2 Experiments on Real Data 36
4.3 Accuracy 37
4.4 Reliability 40
4.5 Runtime 41
4.6 Sampling density 41

4.7 Implementation 42

5 Conclusion **43**
5.1 Summary 43
5.2 Future Work 44

Chapter 1

Introduction

The detection of objects has been an essential topic in the area of computer vision for the last few years. Machine learning methods can already localize and classify objects on images in real-time by using bounding boxes i.e., Regions of Interest (RoI), or segmentation masks, showing very satisfying results that can be retrieved in real-time. But very challenging problems remain, such as the accurate retrieval of 3D object poses, also called 6DOF (Degrees Of Freedom) poses, which are essential for object interaction tasks in the robotic field. Some of the problems that pose estimation could solve, are manipulation of objects or grasping tasks executed by robots. The use of robots systems has increased in the last years as part of repetitive tasks in the industry, but the interest on their application in more complex situations is starting to grow more. For example, if a robot wants to grab any item, lets assume something with a handle, the certain object should not only be detected on an image, but real coordinates about where the object lies must be known. By retrieving the pose of the object using a full 3D model representation, the robot would know where the object is. Thanks to a specific orientation of the 3D model the robot can know the pose of the handle too, even if the handle is not visible. This type of tasks come with requisites that are not very easy to satisfy, such as low computational costs, efficiency, accuracy and real-time application.

1.1 Motivation

As part of this thesis, an object detection algorithm has been implemented, with the goal of estimating accurate 3D poses and retrieving the full shape of the found instances. Principally, we will be working with 3D models and depth images, where meaningful features can be extracted and matched. The program bases its method mostly on Drost et al. [Dro+10] and other improvements to this approach [Hin+17; BI15; Vid+18]. We also propose and implement some changes to existing approaches for the verification of hypotheses.

During the coming chapters, this thesis will assess related work and explain all steps used to implement the object detection algorithm. Finally, we evaluate the program, discussing its accuracy, runtime, efficiency and what could be improved.

Chapter 2

Related Work

There are many different approaches that detect objects and estimate their 3D poses. The most known approaches are divided into two main categories. One category is applied to depth images and the other one is applied to intensity and color images. Before good depth images retrieved from sensors started delivering good results, 2D-image-based approaches where more popular, but in the recent years more approaches are applied to depth maps, as the quality and resolution of these have increased. Also, many approaches that were initially applied to 2D-images have been adapted to work with depth data too. The categories mentioned before can be divided into the next subcategories.

2.1 Learning-based methods

Machine learning approaches have gained a lot of popularity the last years because of their good performances and accurate results in different areas. These methods are divided into two phases. The training, which takes place during an offline phase and requires a large amount of data and time. And an online phase, where the detection would be made by using a model created and trained during the offline phase. There has been a large interest on using convolutional neural networks on both RGB images and depth images, but they have shown limitations on efficiency. Other negative aspects are the expensive training times and acquisition of large training data.

2.2 Template-based methods

Another type of approach is the retrieval of templates by rendering the 3D model of an object from different view points. These templates are normally acquired from synthetic images by rendering 3D models, as the templates have to cover all possible views. The

pose is retrieved when a template matches a part of the input scene. The matching is normally done after computing scores of the templates on the input scene, where the highest score is the matched template. This approach presents fast performances and accurate results even for objects with non-descriptive textures, but also some limitations when it comes to occlusion and clutter.

2.3 Feature-based methods

Feature-based methods are well known to handle occlusion and clutter much better then template-based methods. In this type of methods, features are extracted from the input scene and are matched against the features of the models. Normally, feature-based methods can be divided into two steps. The first step retrieves an initial pose by matching the features globally and the second step is used to refine the retrieved pose by using local features.

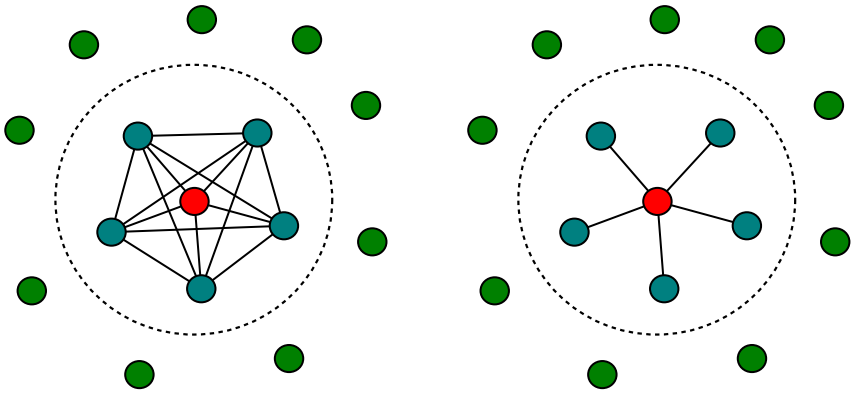
The point pair feature method explored in this work happens to take the advantages of both feature types, local and global, by computing local features of the models and creating a global descriptor where the local descriptors are stored. This approach was inspired by a similar type of feature, the surflet-pair feature, presented by Rusu et. al. [Rus+08] with *Point Feature Histogram* (PHF). The surflet-pair is a description of the local surface of an oriented point on a 3D space. In a set of points P, for each point $p \in P$, the point and all its neighbours in a certain radius are connected. Every connected pair of points (p_i, p_j) and their corresponding normals $(n_i n_j)$ are processed, where $i \neq j$ and the angle between n_i and the line connecting the pairs is smaller then the one of n_j. The pair forms a Darboux frame, which is described as a moving frame on a surface and is calculated as follows:

$$u = n_i,\ v = p_j - p_i,\ w = u \times v \tag{2.1}$$

Using these values we compute the following features of the pair:

$$f_1 = v \cdot n_j \tag{2.2}$$
$$f_2 = \|p_j - p_i\| \tag{2.3}$$
$$f_3 = u \cdot \frac{p_j - p_i}{f_2} \tag{2.4}$$
$$f_4 = arctan(w \cdot n_j, u \cdot n_j) \tag{2.5}$$

Each feature belongs to a bin, whose index can be calculated with its value. The indexes form a key for a 4-dimensional Histogram bin, where each match is registered increasing its score by one. The features with the most votes are selected and used in machine learning methods for classification.



(a) PFH **(b)** FPFH

Figure 2.1: Comparison of the pair retrieval between PFH and FPFH. The red point is the point in turn, the blue points are the neighbours inside the radius and the green points are the points outside the radius. Inspired by [RBB09].

This procedure was later improved by Rusu. et. al in [RBB09], presenting the Fast Point Feature Histogram (FPFH). The FPFH gets rid of the f_1 feature, as tests showed that the robustness did not notably decrease. The number of pairs that are collected for each point p are reduced, since the computational complexity was too high. To reduce the number of pairs, the Simplified Point Feature (SPF) was proposed. The difference is that it only connects the point p being processed with its neighbours. While some pairs are lost, the descriptive information is enough to keep delivering satisfying results. A representation of the pairs can be seen in Fig. 2.1. Finally, the FPFH can be calculated as follows:

$$FPFH(p) = SPF(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_i}SPF(p_i), \tag{2.6}$$

where p_i is a neighbour point and w_i is the distance between p and p_i.

Chapter 3

Method

Different approaches for object recognition match searched objects to global features found on images. This has been proven not to be very accurate and it also leads to very slow and inefficient computations. Local features have shown to be more appropriate for this kind of tasks, resulting on faster and more exact detections. Locally based methods that use 3D data for the detections need dense information for the representation of the local features, which are very sensible to background noise and can cause high computation times and .

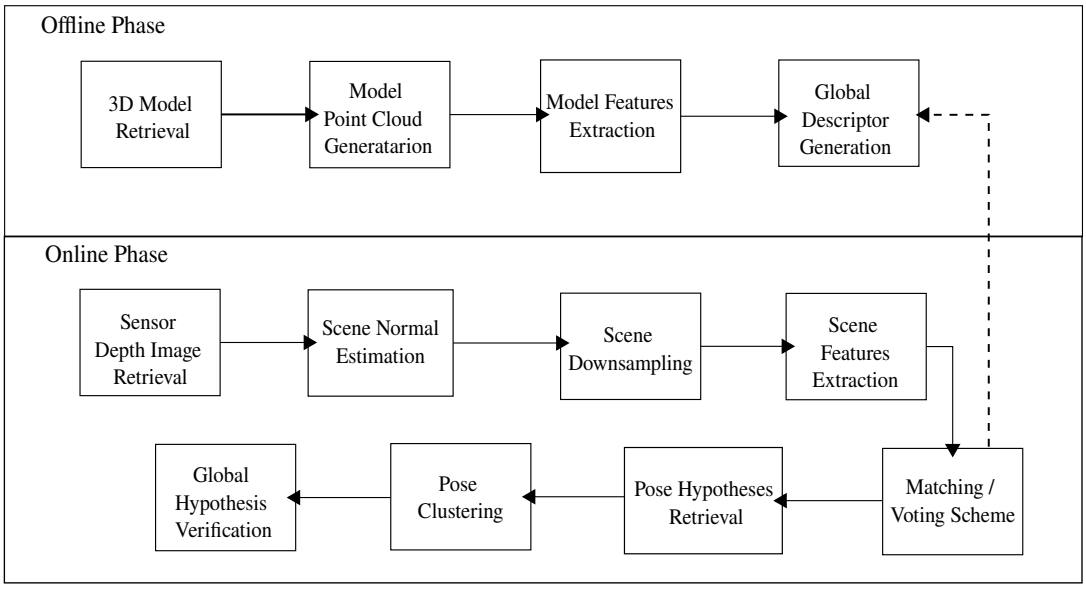


Figure 3.1: Pipeline for object detection and pose estimation divided into two sections. In the offline phase, the features are extracted from the model. In the online phase, the features extracted from the scene are matched to the model features to finally retrieve the model pose in the scene. Inspired by [Dro+10].

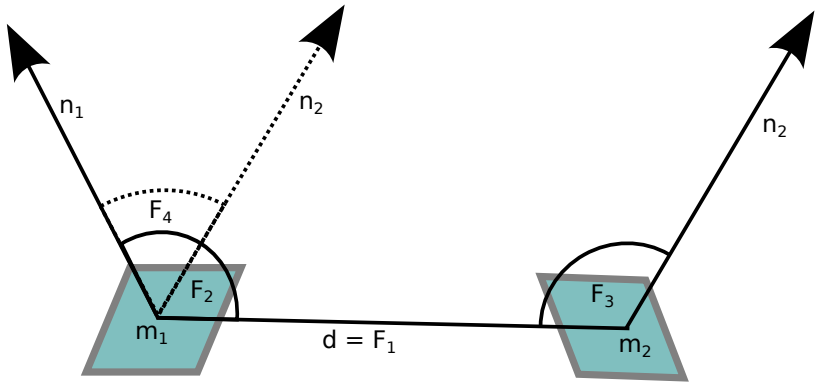


Figure 3.2: Representation of a point pair feature F, where two points m_1 and m_2 have normals n_1 and n_2 respectively. The normal orientations represent the surface where the points lie. The feature F contains the distance between both points (F_1), the two angles (F_2, F_3) between F_1 and the normals and the angle between both normals (F_4).

The approach used for the object detection program is principally based on Drost´s [Dro+10] method, which presents a local descriptor called point pair feature and is extracted from only two points obtained in the 3D data. For more efficient computations and better results, improvements presented in other works [Hin+17; BI15; Vid+18] on the method are explored and integrated into the program.

3.1 Point Pair Features

A point pair feature can be extracted from a pair of oriented points. Suppose m_1 and m_2 are points in a three-dimensional space, n_1 and n_2 are their corresponding normals which represent their surface orientation and d is the vector between both points ($m_1 - m_2$). Finally, the definition of the feature F is the following:

$$F(m_1, m_2) = (\|d\|_2, \angle(n_1, d), \angle(n_2, d), \angle(n_1, n_2))^\top, \tag{3.1}$$

where $\angle(a, b); a, b \in \mathbb{R}^3$ is the angle between vectors a and b.

The point pair feature has the advantage of being very descriptive. In addition, the local descriptor is invariant against rigid motions, which means that the feature can be retrieved from any transformation the 3D model could have. Another trait of this feature is that by matching and aligning two point pair features together, a potential pose can be retrieved. On the other side, it highly depends on the resolution of the retrieved 3D data, as depth errors and very noisy surfaces will directly affect the results of the feature.

Figure 3.3: Some of the mesh models from the YCB-Dataset used for simulated scenes. [Cal+15; Cal+]

3.2 Model Features Extraction

To estimate a pose of an object, the point pair features of the searched object need to be known. For this, a representation of the object as a 3D Model is necessary. In this case, the program was adapted to work with polygon mesh models. Examples can be seen in Fig. 3.3 and Fig. 3.4.

This section will focus on the offline phase of the pipeline where the model data is processed to then be used during the online phase, i.e. the object detection phase. First we show how the sampling of points from a mesh model works. Afterwards, we will explore how to extract these features from the models, following with the creation of a global descriptor, where all the useful point pair features of the model are stored and grouped.

3.2.1 Point Sampling with Triangle Interpolation

The first step to get a good point cloud representation of a 3D model would be by generating points on the triangle faces of the mesh.

To distribute all the points over the mesh, a list of $\mathbf{n}$ random indices of the triangle faces is needed, $\mathbf{n}$ being the size of the point cloud to be generated. But to do this with an uniform distribution, the quantity of points on a face has to depend on the area of the triangle. For this, the indices are weighted, so smaller triangles have a smaller probability. If this is not done, it is very probable that small and big triangles end with a similar quantity of generated points, and hence resulting on a bad distribution. [Por17]

Figure 3.4: Some of the mesh models used in real scenes.

The next step is to generate a random point on the surface of the triangle. To solve this problem Barycentric coordinates are applied as follows:

1. A linear combination of the triangle vertices and three numbers is used. These numbers are defined as u, v, w and $u + v + w \leq 1$.

2. For u and v two random numbers are generated. If $u + v > 1$, then the values are set as $u = 1 - v$ and $v = 1 - u$ to ensure that the definition applies.

3. Finally set $w = 1 - (u + v)$.

The vertices of the triangle are defined as x_1, x_2 and x_3. To generate a point p on the triangle surface compute the following:

$$p = x_1 u + x_2 v + x_3 w \tag{3.2}$$

The same approach can be applied for the computation of the point normals using the vertices normals. Examples of generated model point clouds can be seen in Fig. 3.5.

3.2.2 Poisson Disk Sampling

While a point sampling technique has already been explored, the Poisson Disk Sampling approach [Yuk15] offers a much cleaner distribution of the points, this is also called blue noise distribution [CCS12]. This technique checks when points are too close to each other and only generates points that are satisfactorily far from other samples, but

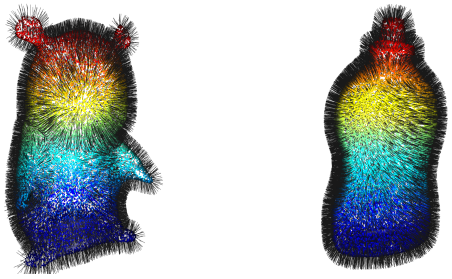


Figure 3.5: Visualization of the generated point clouds with normals using triangle interpolation.

a) b) c)

Figure 3.6: a) is a demonstration of the point cloud generated by sampling the points only with triangle interpolation as seen in (3.2.1). b) follows the same approach as in a), but more points are sampled. For c), the sample elimination was applied on b) to form a Poisson Disk Sample set, which shows a much better distribution of the points in comparison with a).

it comes with high computational costs and is more complex. As an alternative, there is a method that eliminates samples in a given radius of each point and forms a Poisson Disk Sample set out of it. First, we set values of the radius and the number of remaining samples n. In [Hin+17], a maximum of 650 points are told to be enough to have accurate and fast detections, as more points drastically increase the detection times, so we decide to apply the same number in here. The distance of the radius d_{dist} depends on the model diameter and is set as

$$d_{\text{dist}} = \tau_d \cdot diameter(M) \tag{3.3}$$

where $\tau_d = 0.05$. Similar values are also used in [Hin+17; BI15]. Having these values, we create a 3D spatial locator of the given samples using a k-dimensional tree for a faster search of the points. Then all samples are weighted depending on their distance to their neighbours on the given radius r. This would mean that, the closer the neighbours are

inside the radius, the bigger the weight gets. Finally, the point with the biggest weight is eliminated and the weights of its neighbours are updated. The process repeats itself until the size of the remaining samples equals n or until there are no points left to remove. An example of a Poisson Disk Sample can be seen in Fig. 3.6.

3.2.3 PPF extraction

To extract the values of the point pair feature from a pair of points as in 3.1, we have to calculate the distance between the points m_1, m_2 by computing the norm of $d = m_2 - m_1$. To compute an angle $\alpha \in [0, \pi]$ between two normalized vectors v_1, v_2, calculate the magnitude $y = \|v_1 \times v_2\|$ and the dot product $x = v_1 \cdot v_2$. The magnitude of a cross product and the dot product can be interpreted as:

$$\|a \times b\| = \|a\|\|b\||sin(\theta)| \tag{3.4}$$

$$a \cdot b = \|a\|\|b\||cos(\theta)| \tag{3.5}$$

These properties and the law of tangents show that

$$\frac{y}{x} = \frac{sin(\theta)}{cos(\theta)} = tan(\theta) \tag{3.6}$$

therefore the angle θ can be retrieved with the inverse trigonometric function $arctan(\frac{sin(y)}{cos(x)})$. To fulfill the definition of α, the function $atan2(y, x)$ returns an angle within the interval of $[0, \pi]$, where π is the mathematical constant $\pi \approx 3.1416$. The next function shows how this can be done in practice:

Listing 3.1: PPF extraction

```
void extract_ppf(const Eigen::Vector3d& m1,
                 const Eigen::Vector3d& m2,
                 const Eigen::Vector3d& n1,
                 const Eigen::Vector3d& n2,
                 Eigen::Array4d& ppf)
{
    //Calculate vector between points m1 and m2.
    Eigen::Vector3d d(m2 - m1);
    //Get distance from d
    const double dn = d.norm();
    //Normalize vector
    d.normalize();

    //Calculate angles between normals and vector d.
    const double a1 = atan2(d.cross(n1).norm(), d.dot(n1));
    const double a2 = atan2(d.cross(n2).norm(), d.dot(n2));
    const double a3 = atan2(n1.cross(n2).norm(), n1.dot(n2));
```

```
18
19      //Store features in ppf
20      ppf << dn, a1, a2, a3;
21  }
```

The point pair features are stored in a look-up table after their extraction. This table works as a global descriptor and it groups the features in bins for a faster search. The index of each bin is the combination of the four values in a point pair feature. Each feature value f_i is discretized as

$$id_i = \left\lfloor \frac{f_i}{s_i} \right\rfloor \tag{3.7}$$

where s_i is the step between bins. For the distance value f_1 the step s_1 is set equal to d_{dist} as defined in (3.2.2). For the angle values f_2, f_3 and f_4 the corresponding steps s_i are set to $d_{\text{angle}} = 2\pi/n_{\text{angle}}$, where n_{angle} is set to 30. With these parameters the global descriptor has a size of $30^3 \times 20$ bins. Finally, in each bin is a list of the point pairs with the same discretized feature.

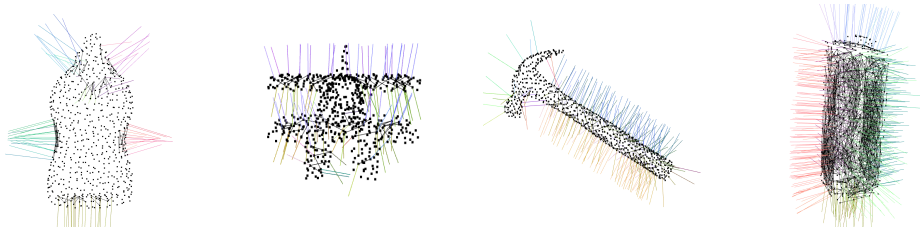
As already mentioned, noisy data is very inconvenient when working with local features. To try to minimize the errors caused by this, we are going to store the point pairs into the neighbour bins too. This idea is presented first in [Hin+17], where pairs are stored in all possible neighbour bins. In this case, there is a total of $(3^4) - 1 = 80$ neighbours for each bin, as the number of neighbours increases exponentially for every dimension that is added to the global descriptor (4 features = 4 dimensions). But in [Lef19], the author argues that this not only increases the total number of pairs stored, but increases the running time too. To overcome this problem, in [Lef19] is proposed to compute the quantization error for each feature of each pair and therefore only store the pair on the neighbour bins that are the most probable to be matched during the detection phase. The quantization error is calculated for each feature as

$$e_q = \frac{f_i}{s_i} - id_i \tag{3.8}$$

Then we can determine if the feature should be stored in one of both neighbours as follows:

$$g(e_q) = \begin{cases} -1, & e_q < \frac{1}{3} \\ 1, & e_q > 1 - \frac{1}{3} \\ 0, & \text{else} \end{cases} \tag{3.9}$$

If the result is zero, we do not store the pair in any neighbour for this feature, otherwise we store the pair in $id_i + g(e_q)$. This results in max. $2^4 = 16$ neighbours were the pair is stored. We choose to follow this approach, as the results remain accurate and helps maintain the detection times low. An example of the features extracted from models can be seen in Fig. 3.7



a) b) c) d)

Figure 3.7: Visualization of bins from the global descriptors. Each image shows all point pair features that belong to the same bin. The colored lines are the normals and the black lines connect the pairs.

3.3 Detection phase

This section will explain how to find an object on a point cloud using a global descriptor created during the offline phase. For a faster detection time, the program has to know beforehand what object is going to be searched on the point cloud, since the downsampling of the scene is dependent of the model diameter.

3.3.1 Normal estimation

The first step during the detection process is to calculate the normals of the point cloud before it gets downsampled, that way the normals are more accurate. To estimate the normal of a point, we need to estimate the normal of a plane tangent of the surface. As explained in [Rus10], the plane tangent is represented by a point x and a normal n. The value of x and n have to be calcuated in such a way, so that a distance to a point p_i is defined as $d_i = (p_i - x) \cdot n$ and $d_i = 0$. This is done by taking

$$x = \frac{1}{k} \cdot \sum_{i=1}^{k} p_i, \tag{3.10}$$

as the centroid of the points taken into consideration. The value of n can be found through the eigenvalues and eigenvectors of the covariance matrix

$$\mathcal{C} = \frac{1}{k} \cdot \sum_{i=1}^{k} (p_i - x) \cdot (p_i - x)^\top, \mathcal{C} \cdot v_j = \lambda_j \cdot v_j, j \in \{0, 1, 2\} \tag{3.11}$$

By applying Principal Component Analysis, if $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$, then the eigenvector v_0 corresponding to the smallest eigenvalue λ_0, would be the best estimation for the normal n. The point density in the point cloud is very important, since the normals will be more accurate if the points being considerate are as near as possible to the point

being processed. This is why the normals estimation takes place before reducing the samples in the scene, otherwise the normals would not be precise. But this is not a major problem, because the computational cost for estimating normals on a point cloud frame is low and does not affect the runtime.

Another point to be made, is that normals can be oriented the wrong way. This problem has multiple different solution, but since we only need to orient them towards the camera viewpoint v_p, we prove that this is satisfied:

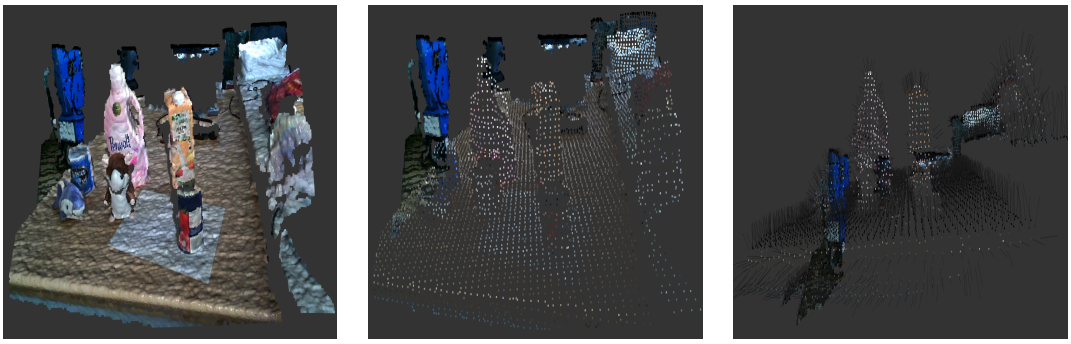
$$n_i \cdot (v_p - p_i) > 0 \tag{3.12}$$

If this is not satisfied, the normal is flipped by applying $n_i = (-1) \cdot n_i$.

3.3.2 Scene downsampling

At this step, the scene point cloud is dense, meaning it still has to be reduced to allow detections to be fast, since processing all possible pairs of points without a reduction would have high computational costs. To avoid this losing too much descriptive information we use the same value as the sample elimination of the model point cloud in (3.2.2), where the points have a minimum distance of τ_d. By using the same distance τ_d for the size of the bins where the point pair features are stored and for the scene downsampling, we allow to efficiently eliminate samples without loosing a lot of information about the scene.

The sample elimination explored in (3.2.2) is not very useful during the online phase considering it is not fast enough for the online phase. For that reason we use another approach for downsampling the scene by using a voxelized grid. A 3D voxel grid is a grid composed by 3D boxes that are put over the point cloud, where each box has a size



a) b) c)

Figure 3.8: The pre-processed scene point cloud can be seen in image a). After downsampling the scene by applying voxelization, the point cloud results as in b). The computed normals can be seen in c).

of τ_d for each direction (length, width and height). Then, the point which is nearest to the centroid of all points in the voxel remains with its normal unchanged. An example of a downsampled seen can be seen in Fig.3.8.

3.3.3 Pose retrieval

After pre-processing the point cloud scene, the detection process can start. To understand how a potential pose of the model scene is retrieved from the scene, we define a reference point s_r and a point s_i as the first and second point of a pair sampled from the scene.

Assuming the point s_r is a point on the object, there should be a point m_r on the model that corresponds to it. The first step to retrieve the pose of the model on the scene would be by aligning both points and their normals. This can be done by bringing both points s_r and m_r to the origin and aligning their normals with the x-axis applying a rotation matrix R and a translation vector t. This can be done for any point p_r and its normal n_r, where p_r can be a scene point s_r or a model point m_r.

$$R = R_y(\theta_y) \cdot R_z(\theta_z) \tag{3.13}$$

$$t = -1 \cdot (R \cdot p_r) \tag{3.14}$$

To get the angles θ_y and θ_z for the rotation over the axes, we project the normal on the the x-z-plane and compute the angle difference to the x-axis

$$\theta_z = -1 \cdot atan2(n_r(z), n_r(x)), \tag{3.15}$$

and rotate the normal over the z-axis by θ_z

$$n_r = R_z(\theta_z) \cdot n_r \tag{3.16}$$

Finally we project the rotated normal on the y-z-plane and compute the angle difference to the x-axis again

$$\theta_y = atan2(n_r(y), n_r(x)) \tag{3.17}$$

Having s_r and m_r and their respective normals aligned, we can now assume that a point pair (s_r, s_i) lays on the object and corresponds to the model point pair (m_r, m_i). To align both pairs, we only need to rotate the model pair over the x-axis by an angle α, where α is the difference angle between both pairs. For efficiency reasons, we calculate α in two different parts $\alpha = \alpha_m - \alpha_s$, where α_m and α_s are the angles differences to the y-axis for the model and the scene respectively. This way, we can store α_m in the global descriptor during the offline phase and save time during the online phase. In order to do this, we transform the second point of the pair

$$p_i = R \cdot p_i + t, \tag{3.18}$$

then project it on the x-y-plane and compute their angle difference to the y-axis (3.19).

$$\alpha = atan2(p_i(y), p_i(z)) \tag{3.19}$$

The transformation matrix that transforms the model to its pose on the scene can be computed as follows:

$$T_{pose} = T_s^{-1} \cdot R_x(\alpha) \cdot T_m, \tag{3.20}$$

where α is the difference between the model alpha α_m and the scene alpha α_s:

$$\alpha = \alpha_m - \alpha_s. \tag{3.21}$$

The transformation matrices are created from their corresponding R and t

$$T = \begin{pmatrix} R_{0,0} & R_{0,1} & R_{0,2} & t_0 \\ R_{0,1} & R_{1,1} & R_{1,2} & t_1 \\ R_{0,2} & R_{1,2} & R_{2,2} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.22}$$

All transformation matrices T_m are stored during the offline phase, while T_s can be computed for all reference points before starting the voting scheme.

3.3.4 Feature matching

To retrieve potential poses are, we need to create a voting table that stores the scores of the poses computed after the matching and alignment of two features. Such a table is created for each reference point s_r. The reference points can be all points in the scene or just a fraction of them. By taking a fraction, the computation times are lower but the accuracy decreases too. In our program, we decide to choose all points in the scene as reference points. For each s_r we need to compare every possible point pair feature $F(s_r, s_i)$ (3.1) with the stored pairs in the global descriptor, where s_i is a also a point in the scene but not necessary a reference point.

The process starts choosing the first reference point s_r from the input scene. To get all possible pairs, we use a kd-tree to find all neighbours of s_r in a radius size of the model diameter. All point that are outside of the radius are ignored, as they would form pairs of points that could not be on the model and would not have any impact on the voting. For every reference point s_r, an accumulator is created to work as a register table to track the votes that are made. This accumulator is two dimensional, one dimension N_m is defined by the number of model points $|M|$ and the other dimension N_{angle} is defined by the number of rotation angles N_{angle}. The number of rotation angles is set to $N_{\text{angle}} = 30$.
For every point pair (s_r, s_i), we extract the point pair feature values as seen in (3.2.3). These values have to fulfill some criteria to be considered descriptive enough.

Criterias:
The angle value a_1 between the normals has to...

1. ...be bigger then $\pi/6$, as long the other angles a_2 and a_3 satisfy $\pi/7 < a_i < \pi/8$, $i \in \{2, 3\}$.

2. ...be smaller then $(\pi - d_{\text{angle}})$.

The criteria 1 is applied since we do not want to consider pairs of points that lie on the same planar surface. Otherwise, points that are near each other will have a major impact on the voting and will leave more discriminative features out. It can also be considered to apply this criteria only for points that are near, but results have shown that applying it for all distances reduces computation time and does not affect the accuracy. The criteria 2 helps to eliminate unprobable angle values between normals.

The features that have fulfilled the criteria are allowed to retrieve the model pairs that have a similar feature by computing the key for the bin they belong to in the global descriptor. Each one of this pairs (m_r, α_m) can now have a vote on the accumulator. We do not need to store any information about the second point m_i on the pair, since the angle difference a_m to the y-axis is stored and represents m_i.

3.3.5 Pose voting

For each pair (s_r, s_i) to make a vote, we use the index of m_r and the index value of α (3.21) as coordinates in the accumulator. In order to do this, we discretize α the same way as in (3.2.3). In the accumulator, the coordinates for the dimension N_{angle} are defined as $\alpha_i \in [0; 29]$, where each coordinate α_i is an angle step d_{angle} in the interval $[-\pi; \pi]$. After (3.21), it can be assumed that $\alpha \in [-2\pi; 2\pi]$, therefore we need to make sure that $\alpha \in [-\pi; \pi]$ with the next case distinction:

$$f(\alpha) = \begin{cases} \alpha + 2\pi, & \alpha < -\pi \\ \alpha - 2\pi, & \alpha > \pi \\ \alpha, & \text{else} \end{cases} \tag{3.23}$$

Finally we can add π to $f(\alpha)$ to get a coordinate a_i after the discretization.

Having the coordinates for the accumulator, the vote can be registered by adding one point to the box. All boxes in the accumulator have at the beginning a score of zero.

After all votes have been registered for every pair of every matched bin, the coordinates of the box with the most highest score are retrieved. Through these coordinates we can now retrieve a potential pose of the model in the scene, by using the transformation matrices T_s of s_r and T_m of m_r stored during the offline phase and the α angle. To retrieve α, we take the coordinate of the current pose with the highest score and multiply it by the angle step d_{angle} (3.2.3) and subtract π from it. Now to compute the pose, we just make use of the equation defined in (3.20).

3.3.6 Clustering

When all potential poses for each reference point s_r have been retrieved, the poses have to be clustered into groups. Poses that have similar rotations and translations are clustered together. The rotations can be compared with their relative angle difference and the translations by their euclidean distance, so that small differences under a certain threshold belong to the same cluster.

The clustering begins by sorting out all poses from the highest score to the lowest. A kd-tree is created using the translation vectors t to accelerate the process, since the list of hypotheses to process contains a large number of poses.

Starting with the pose with the highest score, we iterate over all poses by searching all other poses within a certain radius $(2 \cdot d_{\text{dist}})$. If any of these poses is the centroid of a cluster, the rotations are compared. In order to do this, we need to compute the smallest angle between both rotations. To achieve this, we compute the rotation matrix R_{smallest} from the pose rotation R_{pose} to the centroid rotation R_{centroid}.

$$R_{\text{smallest}} = R_{\text{pose}} \cdot R_{\text{centroid}}^{-1} \tag{3.24}$$

Having a rotation matrix 3×3 allows to retrieve the smallest angle of rotation by calculating its trace $tr(R)$.

$$tr(R) = R_{11} + R_{22} + R_{33} \tag{3.25}$$

The trace is no other than the sum of the diagonal elements in the matrix. The trace can then be interpreted as

$$tr(R) = 1 + 2cos(\theta), \tag{3.26}$$

where θ is the relative angle of the rotation matrix. Consequently, we can retrieve the angle as follows:

$$\theta = arccos(\frac{tr(R_{\text{smallest}}) - 1}{2}) \tag{3.27}$$

The pose can be grouped to the cluster if $\theta < 2 \cdot d_{\text{angle}}$. The pose is added to every cluster that satisfies the criteria. If the pose cannot be added to any cluster, a new cluster is created with the current pose as its centroid.

Many cluster scores get too many votes because of bias caused by repetitive geometric structures. It would be better, if the clustering would only add poses extracted from different local descriptors and not from multiple similar features. This is why in [Hin+17] is proposed to only accept new poses in the cluster, only if the model point m_r that belongs to this pose has not been considered through other poses that had already been added to the cluster. This can be easily done by creating a bit array with a size of

$|M|$. If the bit belonging to the index of the model point is zero, then the pose is added to the cluster and the bit is set to 1, otherwise the pose is ignored.

When all poses have been clustered, we sort them by their score, which is the determined by the sum of the scores of the poses in the cluster. Having all poses clustered and sorted, we can normally expect to get a very good pose estimation in one of the firsts clusters on the list.

3.3.7 Global Hypothesis Verification

For the final step, we need to verify which pose is the most probable to have detected an object accurately and estimated its pose correctly, as it is still possible to get false positives very high on the list. To achieve a good reliable verification, we use a similar approach as proposed by Vidal et. al. [Vid+18], which computes a score that should estimate which pose is the best aligned to the scene, and hence the most probable to be a true positive. For a hypothesis h_i, where h_i is one of the poses gotten from the clusters with the largest number of votes, calculate its score by counting how many model points are valid. For a point to be valid, the distance to the closest point in the scene has to be under a certain threshold, where this threshold is set to $\frac{d_{\text{dist}}}{2}$. The valid points are then used to compute a fitting score.

Vidal et. al. [Vid+18] presents another verification step to try and eliminate false positives from the hypotheses by rendering the model and scene. The pixels of the model can be sorted into three groups, valid, occluded and not-valid, which generate another score. Additionaly, they [Vid+18] compare areas with the silhouette of the model with estimated edges in the scene.

For our program, we decide to skip the fitting score and classify the points into the three groups directly. Instead of rendering the model, we transform the model point cloud to the desired pose h_i and remove all the self-occluded points. To select a point as valid, we propose to not only check if a point is close to the scene, but also check if the points surface are similar by computing the angle difference between both normals as in (3.2.3). We set the threshold for the angle difference to be relaxed (24 degrees), since normals that lie on the silhouette of the model on the scene are not very precise. Points that are close enough to the scene and have a similar surface are considered valid. Points that do not have a similar surface but are close enough to the scene are directly considered inconsistent and hence not-valid. We remove the points occluded by the scene from the remaining points, the points that are not occluded are added to the not-valid points. We do this for the first n hypotheses in the list, where in our case n is set to 10.

To eliminate all occluded points, self-occluded or occluded by the scene, we look into the approach presented in [KTB07]. The paper introduces the *Hidden Point Removal* operator, which has the following properties:

1. "Correctness: in the limit, as the density $\rho \to 0$, a point p_i on S should be marked visible by the operator, if and only if it is indeed visible."

2. "The operator should handle oblique surfaces, while avoiding to compute the surface normals locally."

3. "The asymptotic complexity and the running time should be reasonable, even in software." [KTB07]

The first step for the creation of such operator is the application of a *spherical flipping* of the point cloud, also referred as an inversion. For this, we center a 3-Dimensional sphere at the origin. In this case, we also place the camera view position at the origin, which is normally the real position of the camera C. If the camera is positioned somewhere else, the point cloud has to be transformed correspondingly. The sphere has to have a radius r big enough to cover the whole point cloud. To apply the inversion for all $p_i \in P$, where P is the point cloud being processed, we use the next function:

$$h(p_i) = p_i + 2(r - \|p_i\|)\frac{p_i}{\|p_i\|} \tag{3.28}$$

The resulting point cloud after the transformation is denoted as $\hat{P}$. An example of an inversion can be seen in Fig. 3.9.

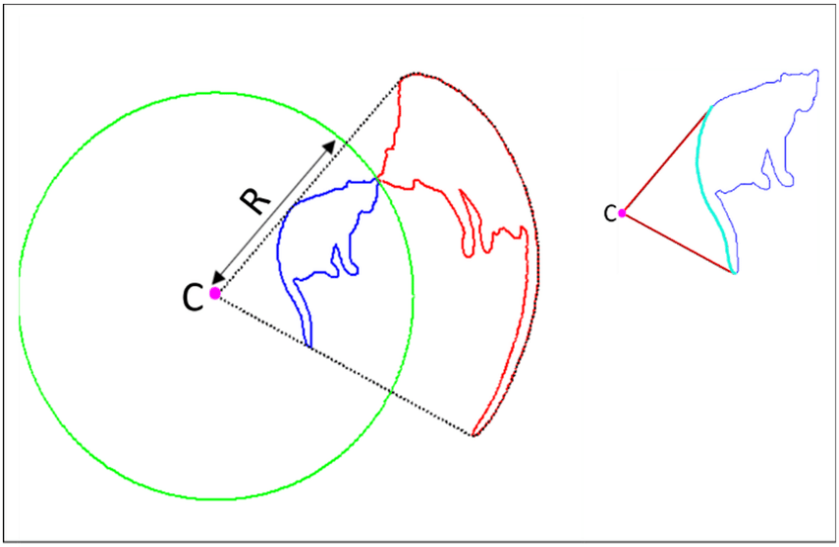


Figure 3.9: Example of how the inversion of the *HPR* operator works. In practice the radius is set to be much bigger. Image taken from [Gra+19].

The following step is the construction of a convex hull, which is the smallest convex set that contains all points. In a convex set, every pair of points that belongs to the set forms a single line that lies completely inside the set. To form a convex hull we take a look into *The Quickhull algorithm for convex hulls* proposed in [BDH96]. The algorithm starts by creating a hull using four points from the set of points that is given. The initial points are not allowed to be on the same hyperplane. For each face in the hull, a set of outside points is created, where all the points above a certain face belong to this set. The algorithm starts iterating over the faces, where for each face on the hull, the point in the outside set that is the furthest away from the face is selected. Then all the neighbour faces that are visible for the selected point are stored on a set V, forming a set of horizon ridges in H. For every ridge, a new face is created using the selected point. For the new face, create a new outside set of points. This is done, until all points are inside the hull or belong to the hull. An example of the convex hull over a model can be seen in Fig. 3.10.

The set of points $\hat{P} \cup C$ is given to construct the convex hull. Since C belongs to the set, all points that are not visible (hidden) before the inversion cannot lie on the convex hull. All points that lie on the convex hull are assigned as visible points. This approach works very good on dense point clouds as on point clouds with low density. An example of the application of the HPR operator can be seen in Fig. 3.11.

The original model point cloud must be transformed to its corresponding pose before applying the *HPR* operator on the current hypotheses h_i. The transformed point cloud and a radius r are given to the operator. Is recommended that the radius for the spherical flipping has a very high value, as this returns much better results. In our case, we use the distance of the translation vector used for the transformation of the model and multiply it by hundred.

After the operator returns the set of visible points S of the model, we count how many points have a valid correspondence to the scene. Valid points will belong to the point set S_m. All valid points can then be removed from the current model point cloud. The points that are left over are merged with the scene point cloud creating a new set of points $(M \cup S \setminus S_m)$. The *HPR* operator is applied for a second time to remove all points that are occluded by the scene. The occluded points belong to the point set S_o. Finally, the points of the model point cloud that were not removed are not valid. The non-valid points belong to the point set S_n.

The score can be computed for the remaining hypotheses as follows:

$$score = (1 - \frac{|S_o|}{|S|}) \cdot (\frac{|S_v|}{|S| - |S_o|}) \tag{3.29}$$

Very occluded detections ($|S_o| > \frac{|S|}{4}$) and poses with low scores ($score < 0.4$) are not considered candidates. This means if no pose has a good enough score then no detection is returned. This may lead to true positives also being rejected.

Algorithm 1 Quickhull algorithm on a set of 3D points [BDH96]

Require: set of points P

```
create hull H with 4 points from P
for each face F in H do
    for each unassigned point p in P do
        if p is above F then
            assign p to F´s outside set
        end if
    end for
end for
for each face F with a non-empty outside set do
    select the furthest point p of F´s outside set and add it to hull H
    initialize the visible set V to F
    add all visible faces from p to V
    for each unvisited neighbour face N do
        if p is above N then
            add N to V
        end if
    end for
    the boundary formed by V is the set of horizon ridges in H
    for each ridge R in H do
        create a new face from R and p
        link the new face to its neighbours
    end for
    for each new face F´ do
        for each unassigned point q in an outside set of a face in V do
            if q is above F´ then
                assign q to F´s outside set
            end if
        end for
    end for
    delete the faces in V
end for
```

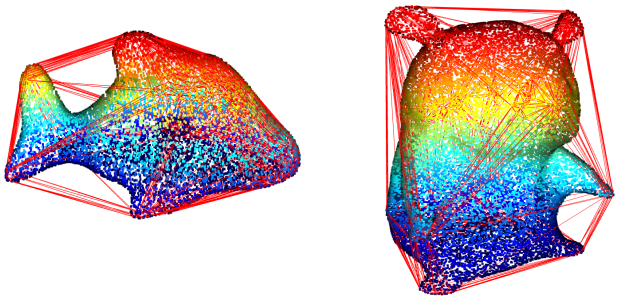
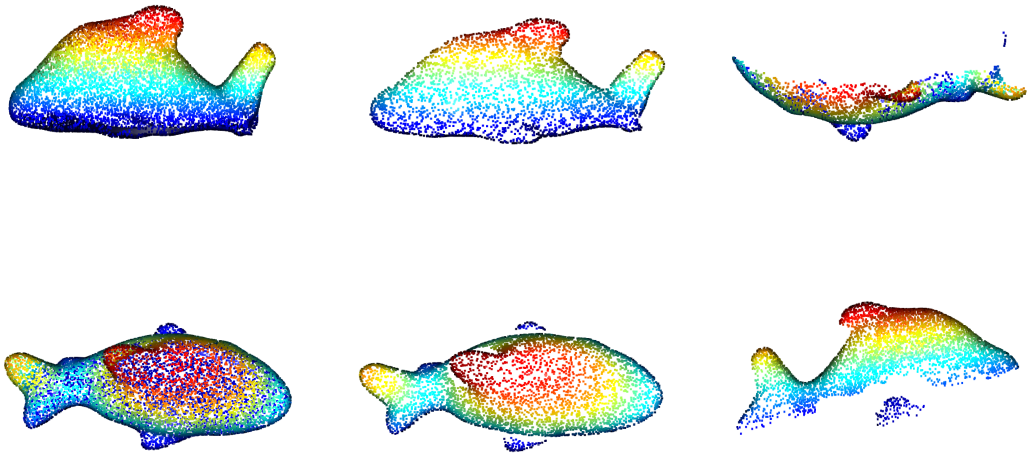


Figure 3.10: Visualization of convex hull over point clouds without inversion.



Camera view before HPR Camera view after HPR Removed hidden side

Figure 3.11: Visualization of HPR results over point clouds.

3.3.8 ICP registration refinement

The result after the pose clustering is already very accurate, but it is still possible to refine the poses with the help of the Iterative Closest Point approach.

The goal for this approach is to minimize the distance between two sets of points by transforming one set that brings it closer to the other. The algorithm can be explained as an iteration over the next steps:

1. Get the correspondence set $C = (s, m)$, where S is the target point cloud(scene) and M is the source point cloud (model).

2. Compute the transformation matrix T between one set to the other with by using the correspondence set by minimizing the error function $E(T)$.

3. Update the source cloud M with the transformation T.

$$E(T) = \sum_{i=1}^{|M|} \|s_i - Tm_i\| \tag{3.30}$$

The process can stop after a certain number of iterations or until a certain error threshold has been achieved. The algorithm is normally applied having a good initial pose of the source cloud, as this reduces the computational time by a lot if it iterates until convergence. For more understanding of this algorithm refer to [BM92].

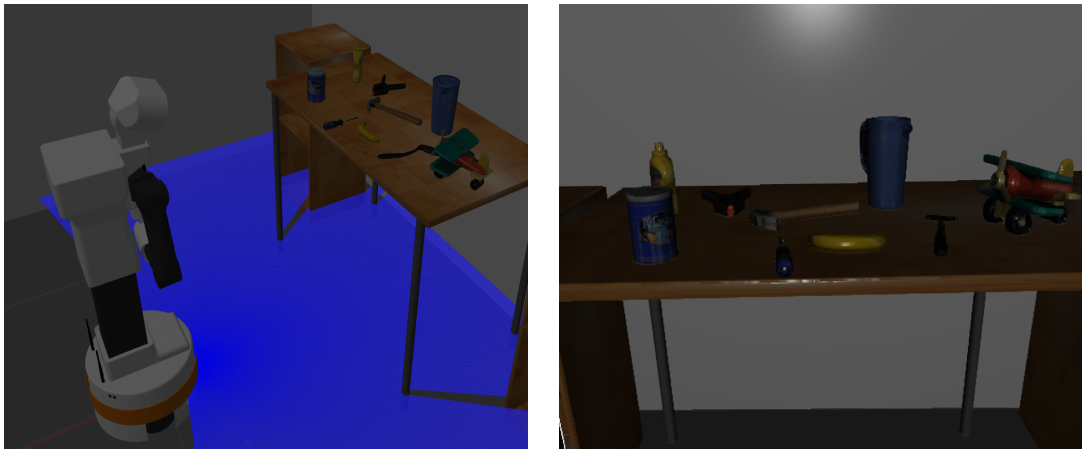
Chapter 4

Experimental Results and Evaluation

The evaluation of the program was done on simulated i.e. synthetic and real data. We chose to use different model datasets for the two categories, some model from the YCB-Dataset (Fig.3.3) [Cal+15; Cal+] were used on the synthetic experiments, while for the real scene other available reconstructed 3D models were used (Fig.3.4). Some of the detections acquired in these evaluation can be seen in Fig.4.7.

4.1 Simulated Experiments

The simulated data was retrieved by using the *Gazebo Simulation* environment [Sim]. The simulation was prepared with different objects sparsed over a room. To retrieve the point clouds, we simulated a robot system that would retrieve multiple depth images



Gazebo Simulation Point cloud

Figure 4.1: Simulation used to retrieved the point clouds.

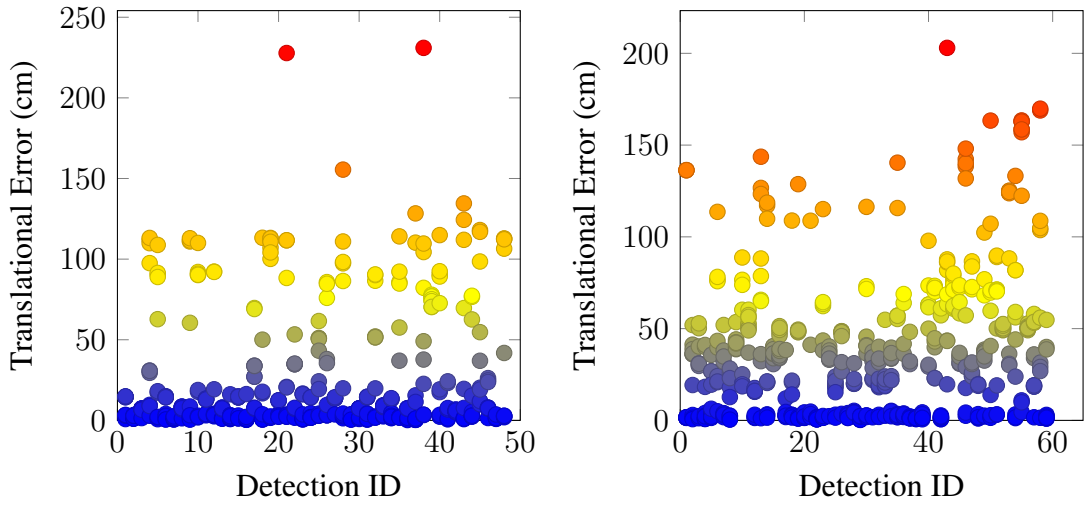
from different views and angles. The point clouds that are retrieved from the environment are very clean and dense, as in comparison with real data, were the point clouds are normally disturbed by noise. Real images present a density decrease on areas that are further away from the camera, while the simulated point clouds are still very dense on distant areas. This is very important to mention, since it will make an impact on the results of the detections and pose estimations between simulated and real data.

To be able to compare the results of the detections, we stored the ground truth poses from the objects at the moment of the point cloud generation, as we can easily retrieve their positions and orientations by getting their poses stored in the simulation.

4.2 Experiments on Real Data

For the generation of real data, we placed the chosen objects in different positions and generated scene point clouds from different points of view. We also tested the program by using point clouds retrieved with different types of sensors, as they may vary on resolution and noise.

To compare the results of the program on the real data, we retrieve the ground truth pose of the objects by manually placing the 3D models on the point cloud so they overlap with the visible surface of the object on the scene. Then the resulting pose of the objects could be stored as their ground truth.

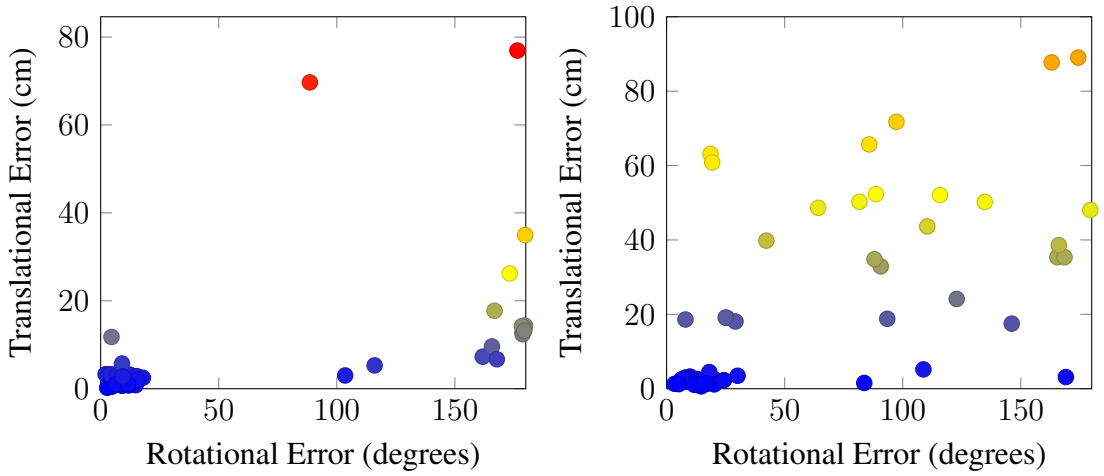


(a) Simulated data **(b)** Real data

Figure 4.2: Representation of the top 10 poses for every detection after clustering.

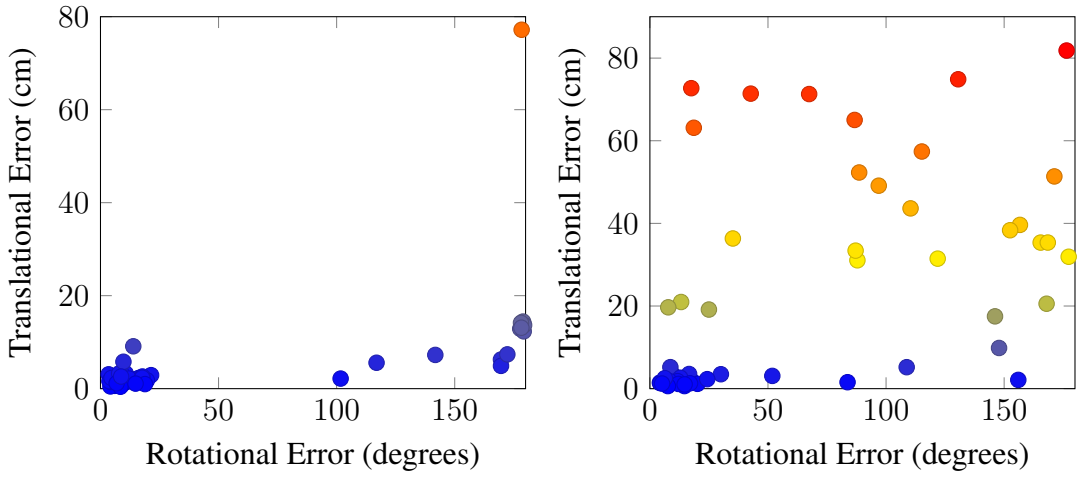
4.3 Accuracy

To analyse the accuracy of the results, we divided the acquired data into different categories. First we have have the results as seen in Fig. 4.3, where the best poses of each detection after clustering were retrieved. The results show the majority of the poses being relative near to the ground truth.



(a) Simulated data **(b)** Real data

Figure 4.3: Representation of the best pose for every detection after clustering.



(a) Simulated data **(b)** Real data

Figure 4.4: Representation of the pose with the best score (GHV) for every detection after clustering.

The results shown in Fig.4.4 demonstrate the poses returned after the Global Hypothesis Verification (GHV) 3.3.7. The implemented GHV returns less false positives in comparison with taking the pose with the most votes. This improvement is more notable in the simulated experiments, but the experiments on real data also show an improvement.

Comparing the results of the experiments with simulated and real data, the simulated experiments delivered more accurate detections in the top 10 poses as the experiments with real data. This probably happens mostly due to the lack of noise in the simulated data. If future depth sensors are able to retrieve dense clouds with not much noise affecting them, approaches like this one could become even much accurate and reliable. The results for the poses with the most votes and the poses chosen by the GHV are very similar between both type of experiments.

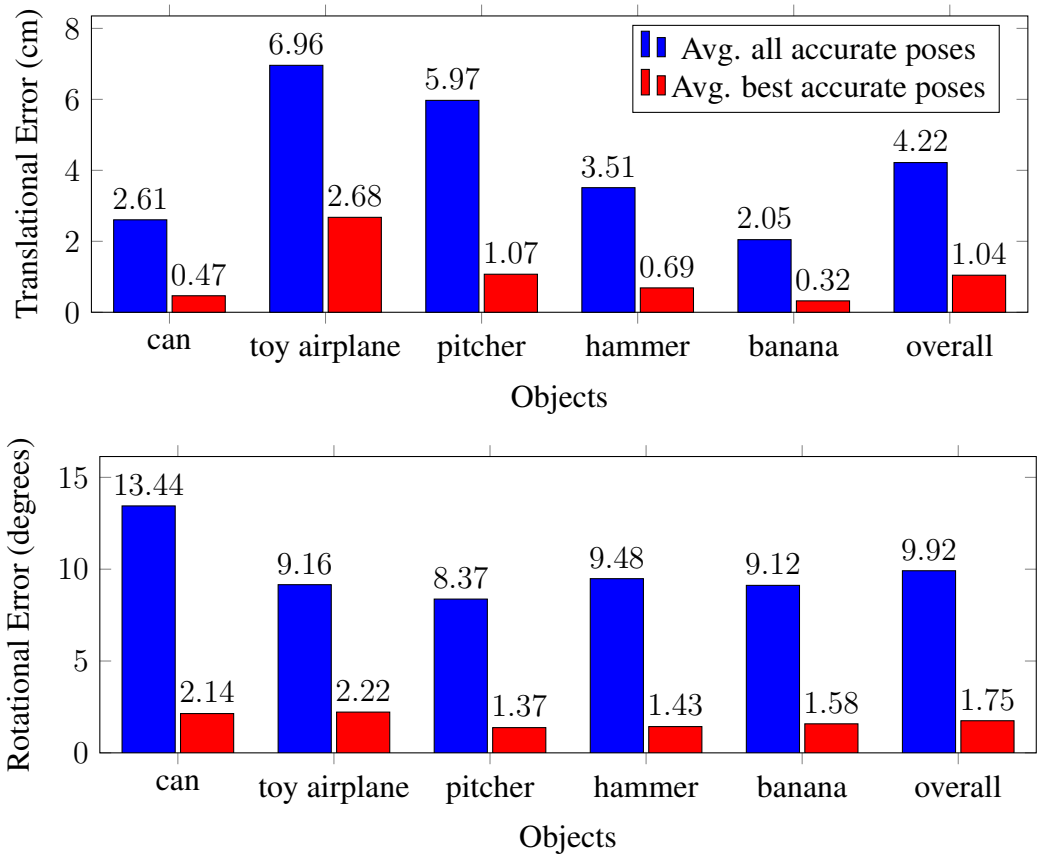


Figure 4.5: Pose estimation error average of all accurate poses and best poses from all top 10 lists returned after clustering from simulated data.

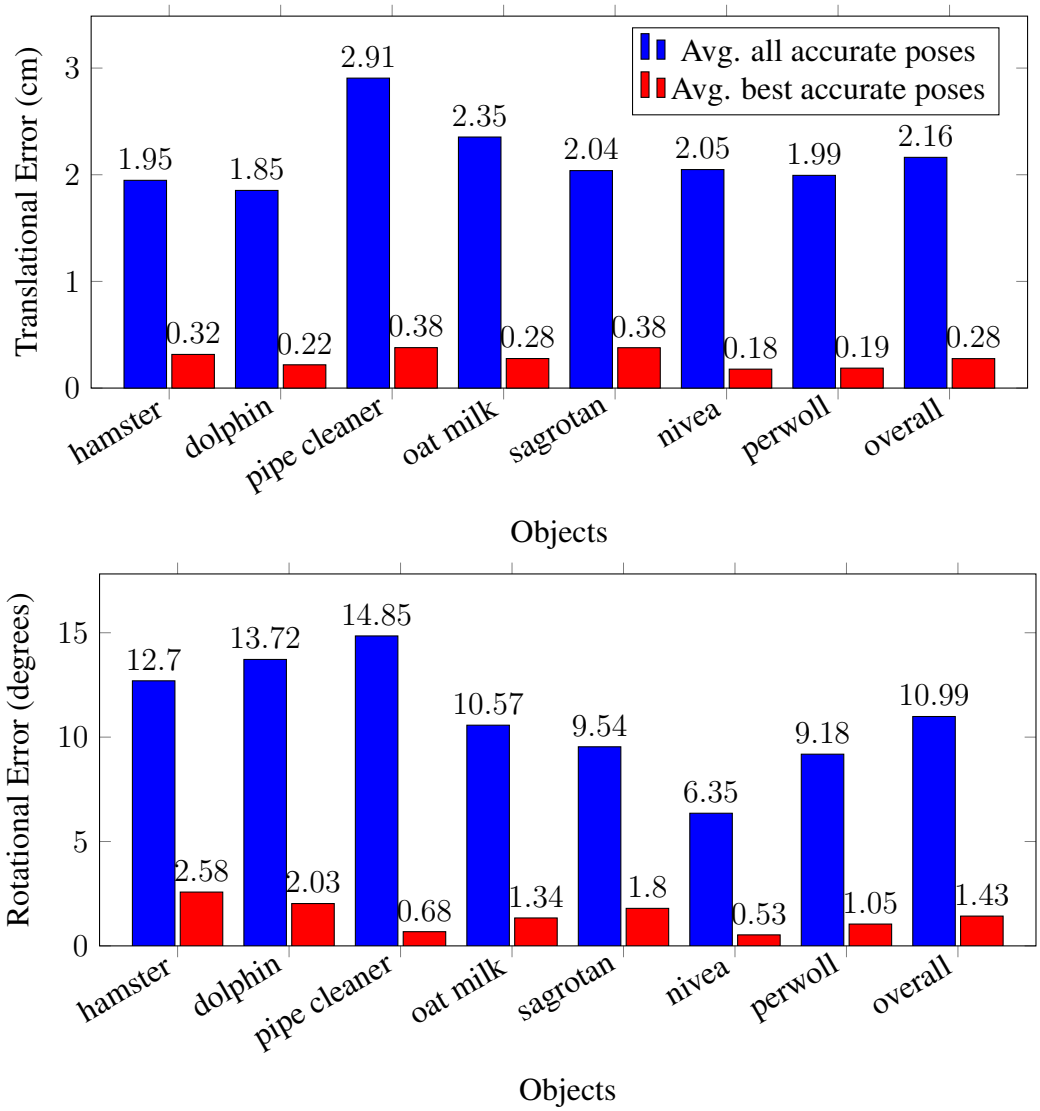


Figure 4.6: Pose estimation error average of all accurate poses and best poses from all top 10 lists returned after clustering from real data.

In Fig. 4.2, we can observe the 10 poses with the most votes for every detection made, meaning no GHV was yet applied. It can be observed, that the majority of detections have multiple accurate positions and very few of the detections return no accurate poses at all.

For the pose estimation evaluation, we compare the translational and rotational errors of the resulting pose estimations for each object (4.5, 4.6). These results are obtained from the accurate poses listed in the top 10, where the average overall is already good, but the average for all the best poses are very near to the ground truth. It is im-

portant to mention that some objects used in the experiments are very symmetrical and therefore many of their estimated poses may have a small translational error and an inaccurate orientation (i.e. can, oat milk). We evaluated these poses as accurate, for all the poses that had this type of error, as it is very difficult to estimate the ground truth orientation of the model.

Table 4.1: Detection accuracy on simulated data

	Accuracy		
Object	Top 1	GHV	Top 10[1]
can	9.09 %	18.18 %	72.73 %
toy airplane	90.91 %	100.00 %	100.00 %
pitcher	50.00 %	60.00 %	100.00 %
hammer	85.71 %	57.14 %	100.00 %
banana	100.00 %	100.00 %	100.00 %
Overall	64.58 %	67.00 %	93.75 %

Table 4.2: Detection accuracy on real data

	Accuracy		
Object	Top 1	GHV	Top 10[1]
hamster	71.43 %	85.71 %	85.71 %
dolphin	71.43 %	57.14 %	71.43 %
pipe cleaner	25.00 %	37.50 %	100.00 %
oat milk	38.46 %	38.46 %	76.92 %
sagrotan	87.50 %	75.00 %	100.00 %
nivea	60.00 %	60.00 %	80.00 %
perwoll	64.00 %	64.00 %	87.50 %
Overall	59.69 %	70.66 %	85.94 %

4.4 Reliability

The goal of the program is to deliver accurate and reliable pose estimations of detected objects, but the results do not fully achieve this. We tested two different ways to get accurate poses and avoid returning false positives as much as possible. One way is by taking the pose with the most votes and the other by computing a score with the use of a Global Hypothesis Verification (3.3.7). Both ways show a low result on returning an

[1] This column refers to acquiring at least one accurate pose on the top 10 lists.

accurate detection. This means that for the current status, the program cannot stop false positives to be returned on a reliable way. To try to achieve a better selection of true positives, future work should focus on implementing other GHV methods that may be more appropriate and reliable for this approach.

On the other hand, the estimated poses on accurate detection's are very satisfying and accurate, with very low distance errors compared with the ground truths. This means that by successfully removing false positives, the program could achieve very reliable and accurate pose estimations. The experiments were also able to detect some objects under occlusion.

4.5 Runtime

The detection times can vary depending on the size of the object that we want to detect, since the downsampling of the input scenes takes the diameter of the object as an argument for the size of its voxels. This means that for smaller objects the detections will take more time compared to bigger objects.

The detections made during the evaluation had times between 1.0s and 9.0s. This computation times are not very useful for tasks were the detections have to be made in real time e.g., object tracking tasks. There exist many extensions on how to improve the current program to work faster and more efficient, so that detections can be made under 1.0**s**.

4.6 Sampling density

The initial density of the point clouds is crucial to achieve accurate estimation of the normals since they are essential for the used approach. As already mentioned, we considered using different sensors that provide different resolutions and quality. For this, different resolutions between 512 x 424 and 1280 × 720 were used. Depth images with less resolution were also tested, but the results were not good. These resolutions proved to provide good detections and pose estimations. For the approach in general, the point clouds are downsampled, meaning the density is mainly important for normals to be estimated accurately. There is other advantage for denser point clouds, which relies on detections that lie further away from the camera. This can be seen on the results of the simulated experiments, since we placed objects that lied further away of the view point compared to the average distance from the real experiments and still delivered better results. In the experiments were real depth images were used, areas that lie further away are less dense and therefore exist some limits on the distance of an object to the camera view, meaning smaller objects will need to be closer to be detected accurately.

4.7 Implementation

The algorithm was implemented in C++ and Python. All experiments were run on a 1.80GHz Intel Core i7-8550U CPU with 8 cores and 16GB RAM. To improve the runtime during the detection phase, the extraction of point pair features and the pose retrieval were implemented to be run on multiple threads. Some of the functions used in the implementation were taken from the PCL [RC11] and Open3D [ZPK18] libraries.

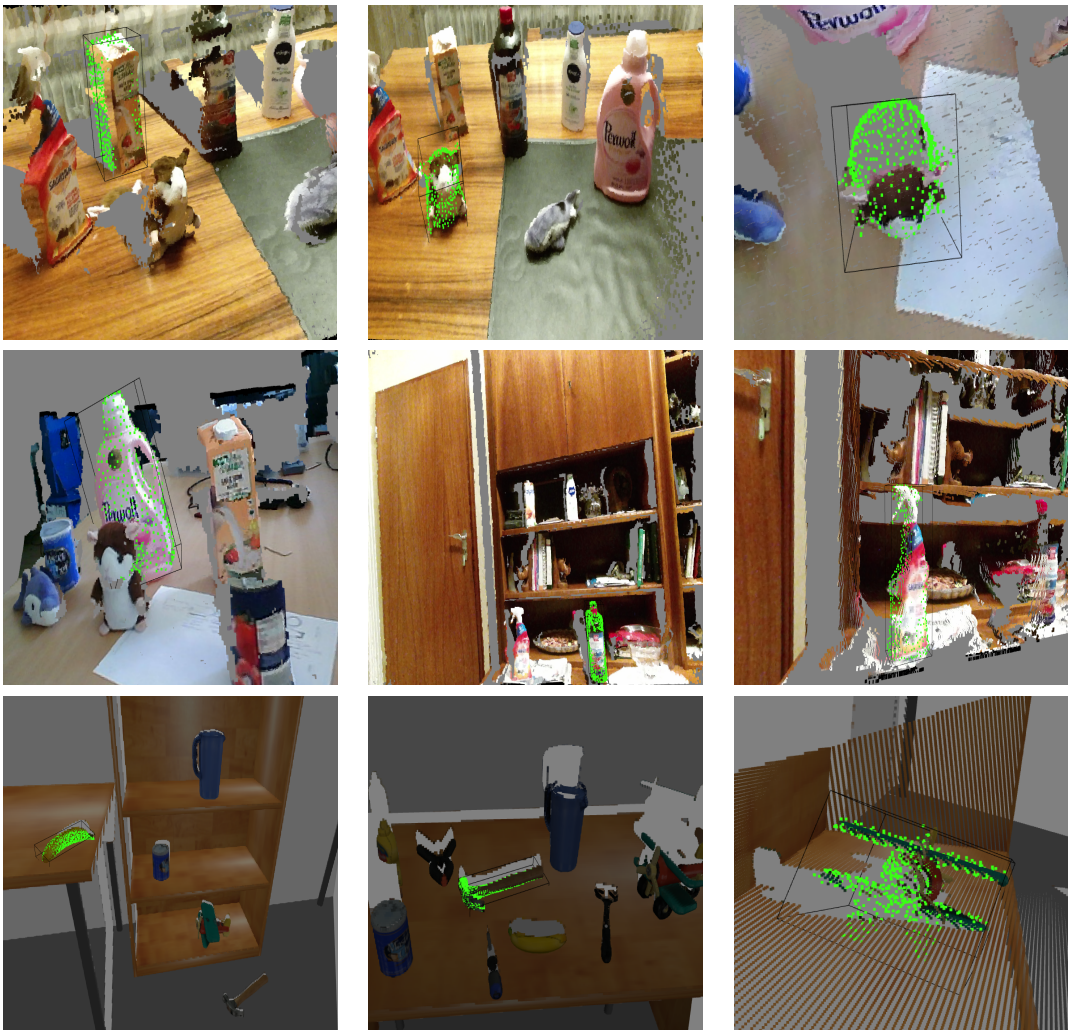


Figure 4.7: Detections and pose estimations delivered by the program. The detected poses are shown as the green colored points that form the model point clouds. The detected objects are surrounded by a bounding box.

Chapter 5

Conclusion

5.1 Summary

This bachelor thesis implemented a program with the goal to detect objects and estimate their 3D pose. The implementation was principally based on the approach presented by [Dro+10]. This approach uses local descriptors called point pair features which are extracted from point clouds generated from 3D models. These features are deserialized and quantized so they can be grouped into bins creating a global descriptor of the model. During an online phase, input scenes are pre-processed by estimating their normals and being downsampled. In the next step the features are extracted from the scene and matched to the model features stored in the global descriptor. This allows to retrieve potential poses of the object in the scene by aligning the point pairs and their normals. The poses get a score by voting for them after every match. Poses with the most votes are stored in a sorted list to then be grouped into clusters increasing their score. Finally, the best clusters are selected. To eliminate potential false positives, a Global Hypothesis Verification is applied giving each pose a score. This score determines which pose is the most probable to be a true positive and selects it as the final pose for the detection.

The program was evaluated by using two different approaches, one approach was retrieving synthetic point clouds on a simulation environment and the other using depth sensors to retrieve real data. The evaluation showed a much better result on the synthetic data compared to the experiments with real data. This is due to the lack of noise on the simulated data and its better resolution i.e. sampling density. The experiments also showed bad results on detection accuracy, as the elimination of false positives using our verification approach was not very successful. On the other hand, we could prove that accurate detections were present in the majority of the top 10 poses gotten from the clustering. These accurate detections also included good pose estimations, presenting very low translational and rotational errors.

5.2 Future Work

Other works [Dro+10; Hin+17; BI15; Vid+18] have shown better results in accuracy and runtime by using similar approaches as this one, meaning that the approach has more potential than the results we got. The implemented program could be improved in many different ways, starting from implementing a more reliable verification approach for the pose hypotheses and improving the runtime.

One of the main traits of this approach is being neural network free. But having very accurate pose estimations as its strength, we could improve the detection step by adding a neural network for object detection to the pipeline. Some classifiers as e.g., YOLO[1], are very accurate and reliable nowadays and could improve the results of this approach drastically, since the size of the point cloud would be reduced making the detections more accurate and the computation times much faster. This would reduce the probability of false negatives being selected, as it would only search on areas with a high probability of the object being there.

The current voting scheme increases the score of a pose by one for any local feature matching. This could be changed by applying weights to the point pair features depending on how repetitive and descriptive they are. The weights could also be different for each model and could be processed during the offline phase when creating the global descriptor. The program could be accelerated by adapting parts of the code to work on a GPU.

[1] https://pjreddie.com/darknet/yolo/

List of Tables

4.1 Detection accuracy on simulated data 40
4.2 Detection accuracy on real data 40

List of Figures

2.1 PFH and FPFH 13

3.1 Pipeline 15
3.2 Point Pair Feature 16
3.3 YCB-Models 17
3.4 Real models 18
3.5 Model point clouds 19
3.6 Model downsampling with PDS 19
3.7 PPF´s bins visualization 22
3.8 Scene downsampling 23
3.9 Hidden Point Removal: Spherical Flipping 29
3.10 Convex Hull 32
3.11 Hidden Point Removal 32

4.1 Simulation 35
4.2 Top 10 poses per detection 36
4.3 Best pose per detection 37
4.4 Best score (GHV) per detection 37
4.5 Simulation data - Pose error avg. 38
4.6 Real data - Pose error avg. 39
4.7 Detections 42

Bibliography

Veröffentlichte Literatur

[BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. "The Quickhull Algorithm for Convex Hulls". In: *ACM Trans. Math. Softw.* 22.4 (Dec. 1996), pp. 469–483. ISSN: 0098-3500. DOI: 10.1145/235815.235821. URL: https://doi.org/10.1145/235815.235821.

[BI15] Tolga Birdal and Slobodan Ilic. "Point Pair Features Based Object Detection and Pose Estimation Revisited". In: *2015 International Conference on 3D Vision*. 2015, pp. 527–535. DOI: 10.1109/3DV.2015.65.

[BM92] P.J. Besl and Neil D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791.

[Cal+] Berk Calli et al. *YCB-Dataset*. Accessed: 2022-09-20. URL: https://www.ycbbenchmarks.com/.

[Cal+15] Berk Calli et al. "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set". In: *IEEE Robotics & Automation Magazine* 22.3 (2015), pp. 36–52. DOI: 10.1109/MRA.2015.2448951.

[CCS12] Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes". In: *IEEE Transaction on Visualization and Computer Graphics* 18.6 (2012). http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.34, pp. 914–924. URL: http://vcg.isti.cnr.it/Publications/2012/CCS12.

[Dro+10] Bertram Drost et al. "Model Globally, Match Locally: Efficient and Robust 3D Object Recognition". In: July 2010, pp. 998–1005. DOI: 10.1109/CVPR.2010.5540108.

[Gra+19] Teresa Gracchi et al. "Optimizing Wireless Sensor Network Installations by Visibility Analysis on 3D Point Clouds". In: *ISPRS International Journal of Geo-Information* 8 (Oct. 2019), p. 460. DOI: 10.3390/ijgi8100460.

[Hin+17] Stefan Hinterstoisser et al. “Going Further with Point Pair Features”. In: *CoRR* abs/1711.04061 (2017). arXiv: 1711.04061. URL: http://arxiv.org/abs/1711.04061.

[KTB07] Sagi Katz, Ayellet Tal, and Ronen Basri. “Direct visibility of point sets”. In: vol. 26. July 2007. DOI: 10.1145/1275808.1276407.

[Lef19] Annette Lef. “CAD-based Pose Estimation - Algorithm Investigation”. In: 2019. URL: https://liu.diva-portal.org/smash/get/diva2:1330419/FULLTEXT01.pdf.

[Por17] Jamie Portsmouth. “Efficient barycentric point sampling on meshes”. In: *CoRR* abs/1708.07559 (2017). arXiv: 1708.07559. URL: http://arxiv.org/abs/1708.07559.

[RBB09] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[RC11] Radu Bogdan Rusu and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.

[Rus+08] Radu Bogdan Rusu et al. “Learning informative point classes for the acquisition of object model maps”. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. 2008, pp. 643–650. DOI: 10.1109/ICARCV.2008.4795593.

[Rus10] Radu Rusu. “Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments”. In: *KI - Künstliche Intelligenz* 24 (Nov. 2010), pp. 45–50. DOI: 10.1007/s13218-010-0059-6.

[Sim] Gazebo Sim. *Gazebo Sim*. Accessed: 2022-09-20. URL: https://gazebosim.org/home.

[Vid+18] Joel Vidal et al. “A Method for 6D Pose Estimation of Free-Form Rigid Objects Using Point Pair Features on Range Data”. In: *Sensors* 18 (Aug. 2018), p. 2678. DOI: 10.3390/s18082678.

[Yuk15] Cem Yuksel. “Sample Elimination for Generating Poisson Disk Sample Sets”. In: *Computer Graphics Forum* 34 (May 2015), pp. 45–50. DOI: 10.1111/cgf.12538.

[ZPK18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: *CoRR* abs/1801.09847 (2018). arXiv: 1801.09847. URL: http://arxiv.org/abs/1801.09847.