TECHNISCHE UNIVERSITÄT WIEN

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Pose Estimation of Deformable Objects

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

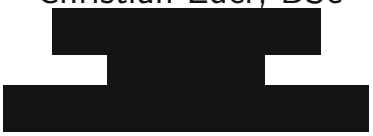
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. M. Vincze
MSc Dr.techn. S. Thalhammer

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by

Christian Eder, BSc



St. Leonhard am Forst, January 2024

Vision for Robotics Group
A-1040 Wien, Gußhausstr. 27, Internet: https://www.acin.tuwien.ac.at

Preamble

Diese Diplomarbeit wurde als Abschlussarbeit für das Masterstudium Energie- und Automatisierungstechnik verfasst. Ich möchte mich an dieser Stelle bei allen bedanken, die mich in diesem Studium unterstützt und dazu beigetragen haben, dass ich am Ende offiziell den Titel Dipl.-Ing. führen darf.

Mein Dank gilt besonders der Vision for Robotics Gruppe am Institut für Automatisierungs- und Regelungstechnik unter der Leitung von Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze, die mir in den letzten vier Jahren die Möglichkeit gegeben haben, im Labor einen großen Wissensbereich aufzubauen. Für seine Unterstützung während der Arbeit möchte ich mich zudem bei meinem Betreuer MSc Dr.techn. S. Thalhammer bedanken, der mir alle Freiheiten für die selbständige Umsetzung geben hat, aber trotzdem immer erreichbar war.

Ebenfalls möchte ich mich bei meinen Studienkollegen bedanken, besonders bei denen, die ich mittlerweile auch außerhalb der Universität zu meinen engsten Freundeskreis zählen darf. Unsere abendlichen Diskussionen auf der Fachschaft Elektrotechnik werden mir sehr fehlen. Ohne euch wäre dieser Erfolg nie möglich gewesen. DANKE!

Ein ganz besonderer Dank natürlich gilt meiner Familie, die mich immer unterstützt und mir das Gefühl gegeben hat, dass ich auf dem richtigen Weg bin. Zu guter Letzt möchte ich mich bei Rebecca bedanken, weil sie mich motiviert und mir den Raum zum Arbeiten gegeben hat.

Christian Eder, BSc

St. Leonhard am Forst, January 2024

Abstract

This study introduces a technique for estimating the pose of deformable objects. In particular, everyday objects are considered, which can then be further processed by a robot. Deformations, e.g. due to daily use or to save space during waste disposal, mainly affect the appearance, which significantly impairs the performance of many pose estimators.

The focus of this work is to train a modern pose estimator with deformed versions of an object and to analyze the results. The data of the "Deformed Object Dataset (DOD)", which was developed for this work, serves as the basis. In the dataset, the 3D models of various everyday objects such as toothpaste, juice bags, cans, pastry or chips packaging are deformed by an algorithm. Images and the corresponding masks are calculated from mixed scenes with known and unknown objects and used as training data.

Pix2Pose is used as a pose estimator because it is suitable for textureless objects and also works with UV prediction for estimation. These form the basis for the training process. The correct prediction of the UV coordinates enables a precise estimation of the pose.

In the final experiments, it was found that an accurate estimation of the pose is not easily possible. A constant deviation in the UV coordinates has a direct effect on the translation and rotation error in the pose. In addition, it was found that there are stable and unstable regions, so a more accurate estimate is quite possible with more advanced methods.

Kurzzusammenfassung

In dieser Arbeit wird ein System vorgestellt, dass es erlaubt die Pose von deformierbaren Objekten im Raum zu schätzen. Es wird besonders auf Alltagsgegenstände eingegangen, die dann von einem Roboter, zum Beispiel durch Greifen, weiterverarbeitet werden können. Verformungen, z.B. durch täglichen Gebrauch oder aus Platzspargründen bei der Abfallentsorgungen, beeinflussen vor allem die äußere Erscheinung, wodurch die Leistungen vieler Posenschätzer erheblich beeinträchtigt werden.

Der Fokus dieser Arbeit liegt darauf einen Posenschätzer nach aktuellem Stand der Technik mit deformierten Versionen eines Objektes zu trainieren und die Ergebnisse zu analysieren. Als Grundlage dienen die Daten des „Deformed Object Dataset (DOD)“, welches für diese Arbeit entwickelt wurde. Im Datensatz werden die 3D-Modelle diverser Alltagsgegenstände wie Zahnpasta, Tetrapaks, Dosen, Teig- oder Chipsverpackungen durch einen Algorithmus deformiert. Aus gemischten Szenen mit bekannten und unbekannten Objekten werden Bilder und die zugehörigen Masken berechnet, die als Trainingsdaten genutzt werden.

Als Posenschätzer wird Pix2Pose verwendet, weil er einerseits für texturlose Objekte geeignet ist und anderseits mit UV-Koordinaten für die Schätzung arbeitet. Diese bilden die Grundlage des Trainingsprozesses. Eine korrekte Vorhersagung der UV-Koordinaten ermöglicht eine präzise Schätzung der Pose.

In den abschließenden Experimenten wurde festgestellt, dass eine genaue Schätzung der Pose nicht einfach möglich ist. Eine konstante Abweichung in den UV-Koordinaten wirkt sich direkt als Verschiebungs- und Drehfehler in der Pose aus. Zusätzlich wurde festgestellt, dass es stabile und instabile Bereiche gibt, wodurch eine präzisere Schätzung durch fortgeschrittenere Methoden durchaus möglich ist.

Contents

List of Figures

List of Tables

1 Introduction

Object pose estimation is an important task for a robotic system to understand the given scene. Based on this information, higher level tasks such as object manipulation are performed. In most cases, certain knowledge about the object model is used. The known 3D model in combination with RGB or RGB-D data enables suitable results. The task becomes even more difficult when the model no longer matches the real world. This can be due to physical deformations or texture changes.

1.1 Motivation

In real-life scenarios, objects often deviate from their modeled 3D shape, e.g. due to pressure, deformation, breakage or similar factors. Such damage is typically caused by external influences during transportation, handling or disposal. As the level of automation increases, it becomes increasingly important to understand all aspects of the objects and their environment. This understanding goes beyond the basic object and includes all modified forms that enable manipulation by machines.
Although it is possible to capture 3D models of different deformations and treat each of them separately, this approach involves considerable effort. An object can take on countless different shapes, making simulation programs a more convenient option. The continuous development in this field, combined with the available computing power, offers immense potential.

The issue of computing power is both an advantage and a challenge. On the one hand, it facilitates the manipulation of objects and enables multiple images of the same object to be created quickly. On the other hand, computing power is often limited in industrial applications. Although industrial computers are getting better and better, their costs are also rising. In the industrial sector, a balanced price/performance ratio is particularly important, so a compromise must be found. The optimal solution is to use high computing power in the learning phase and minimize it in the recognition phase.

1.2 Problem Statement

Differences between the object model and the real object lead to a significant performance degradation in pose estimation systems. This starts with object recognition and ends with the pose estimator itself. Figure 1.1 shows this fact. Classical pose estimation algorithms try to fit the object model to the given scene, but fail if the object in the

scene differs from the original model. Recording different representations of the same object is time consuming. Modern simulation tools offer the possibility to manipulate objects and create different versions in a short time.
In this work, a state-of-the-art algorithm for estimating the object position is trained with synthetic deformed models. The results are then tested for plausibility, performance and future potential.

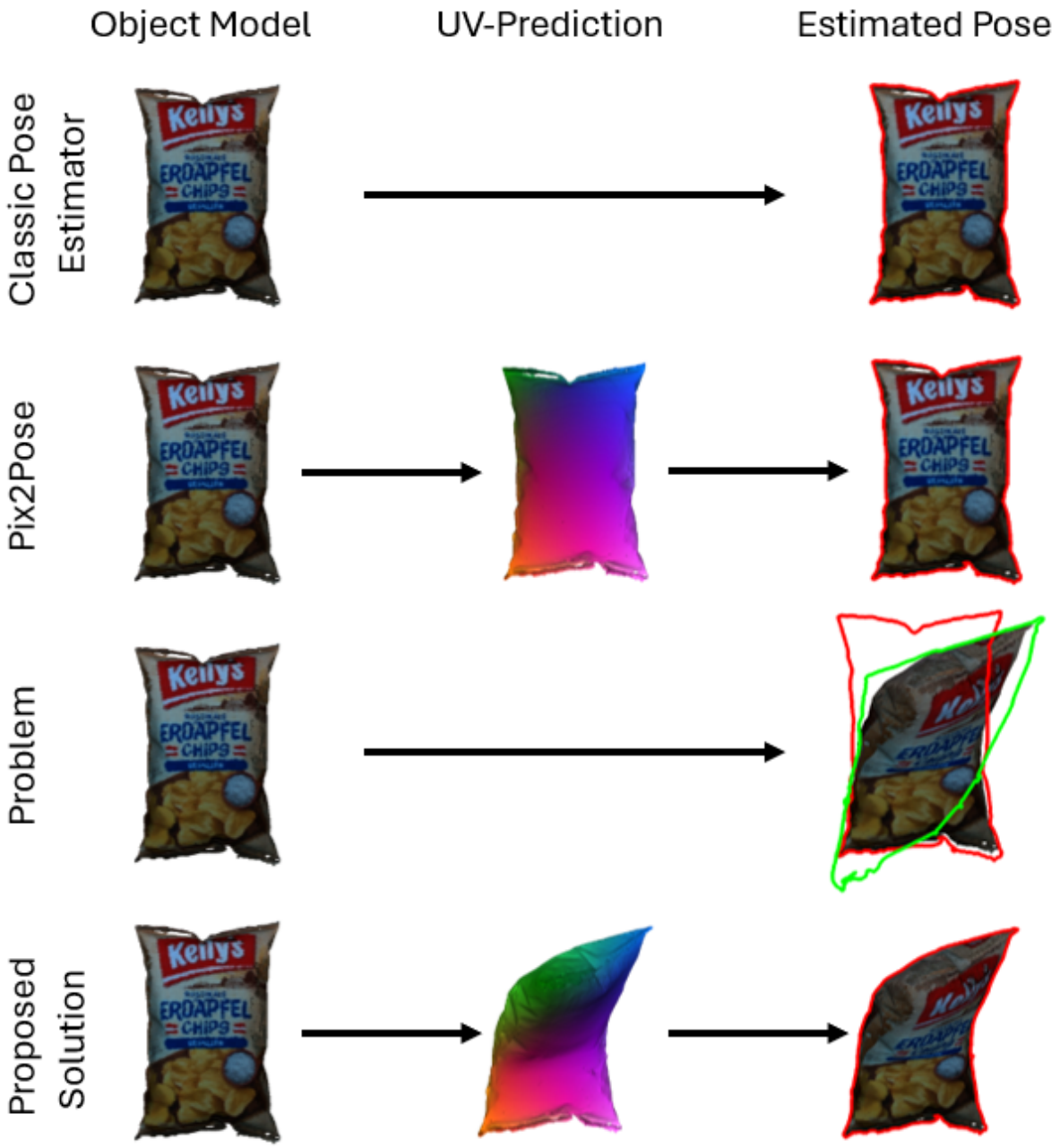


Figure 1.1: Proposed solution of the thesis

1.3 Contribution

The contribution of this work is divided into three important parts. First, the creation of a dataset with deformable objects. This requires the implementation of a method for synthetic object deformation. The second part is the training of a state-of-the-art pose estimator with the prepared dataset. And last but not least, the evaluation of the resulting data. The parts are explained in more detail below:

Deformable Object Dataset (DOD): The focus of this part of the work is on a small dataset for deformable objects. Seven representative household objects are used for this purpose. The objects differ according to various criteria such as shape (boxes, tubes, packs, cans), material, deformability and use. First of all, a high-resolution reconstruction of the objects is required. Therefore, many images of the object are used. Since the object model is the fundamental part of the project, it is important that the reconstruction meets the quality criteria. To ensure usability and comparability in the later phase, the dataset corresponds to the BOP challenge structure. During the deformation process, synthetically generated deformed versions of the object model are created using 3D model software. These correspond to the real world in terms of material properties, volume stability and texture changes. As many different versions of the object deformation are required, an automated process with a certain degree of randomness is implemented. At least the objects of the dataset are placed in certain scenes with some randomly selected distraction objects. The information from the scene is used to generate images and masks. Together with the models, this forms the Deformable Object Dataset.

Pose Estimation: In this work, a state-of-the-art pose estimator called Pix2Pose is trained on the given Deformable Object Dataset. Pix2Pose offers several advantages. First, it can handle textureless objects and second, it uses UV predictions for pose estimation. The UV prediction is particularly helpful. During deformation, the UV coordinates of the models are stored, which correspond directly to the base model. This information is later used as input for training. The aim is to predict the UV coordinates and use this information to determine the pose of the object.

Evaluation: The results of Pix2Pose are evaluated in the experiments. The predicted pose and the rendered pose are compared. The metric used is the translation and rotation error. When training the pose estimator, the mean loss is analyzed on the basis of the variable augmentation probability value. The results show a direct correlation between the UV prediction error and the translation and rotation error. The deviation caused by the UV prediction error was too high for useful results. The calculated heat map shows that there are stable and unstable areas that offer future potential.

1.4 Chapter Organization

Chapter 2 deals with different methods for estimating the object pose. It begins with a brief introduction to object pose estimation systems. Then the common methods are explained. Additionally, the BOP challenge is explained to compare these methods. Finally, a focus is placed on deformable objects and the generation of synthetic data. In Chapter 3 the common software tools are explained. There is also a part about the created Deformable Object Dataset, where the idea about the selected objects is shown. Chapter 4 shows the implementation of the system. First the object reconstruction part, followed by the deformation process. Then the training of the object detector and the pose estimator Pix2Pose is explained. Chapter 5 presents the evaluation process with the synthetic dataset. First, the evaluation metric is explained and then the results are discussed. The plausibility of the results is also discussed. In Chapter 6 the whole system is analyzed. The thesis ends with a discussion of future work.

2 Related Work

A pose estimator is a system that can recognize and estimate the 6D pose of an object based on RGB or RGB-D data, usually for rigid objects with a unique texture. As the performance of the algorithms improves, research is being conducted on pose estimation for objects without texture and deformable objects. To support this work, datasets are needed that are small samples from the real world, either captured or generated in a virtual environment. This chapter focuses on recent work in the areas of pose estimation (2.1), deformable objects (2.2) and synthetic data generation (2.3).

2.1 Pose Estimator

The purpose of pose estimation is to calculate the 6D pose of an object in a particular scene. An example can be seen in Figure 2.1. 6D stands for the combination of 3D translation and 3D rotation. There are different methods to calculate the pose of an object based on 2D image data. The aim is to create a link between 2D and 3D. This section deals with three approaches to pose estimation (2.1.1) and the benchmarking of pose estimation algorithms (2.1.2).

2.1.1 Pose Estimation Methods

There are several common methods for estimating posture. In this section, the three most commonly used methods are explained in detail. First, there are direct pose algorithms that search for the center of the region of interest (ROI) and then compute a 3D displacement between this point and the camera coordinate system. Another approach is a combination of keypoint algorithms and Perspective-n-Point (PnP) algorithms. The idea is to use the camera intrinsic and calculated keypoints to predict the pose of the camera. The last method uses additional data in the form of UV predictions to estimate the object pose.

Direct Pose: Direct pose algorithms such as PoseCNN [2], Deep Model-Based 6D Pose Refinement in RGB [3] and Multi-Task Deep Networks for Depth-Based 6D Object Pose and Joint Registration in Crowd Scenarios [4] attempt to fit a 3D model of the object directly into the image. The 3D model must be available for these approaches. First, a recognition algorithm such as Mask-RCNN [5], YOLO [6] (latest version YOLOv7 [7]), SSD [8], Faster-RCNN [9], RetinaNet [10] or FCOS [11] is used to identify the region of interest (ROI) in the image. PoseCNN, for example, searches for the center of the object. The 3D displacement between the center coordinate system and the camera coordinate

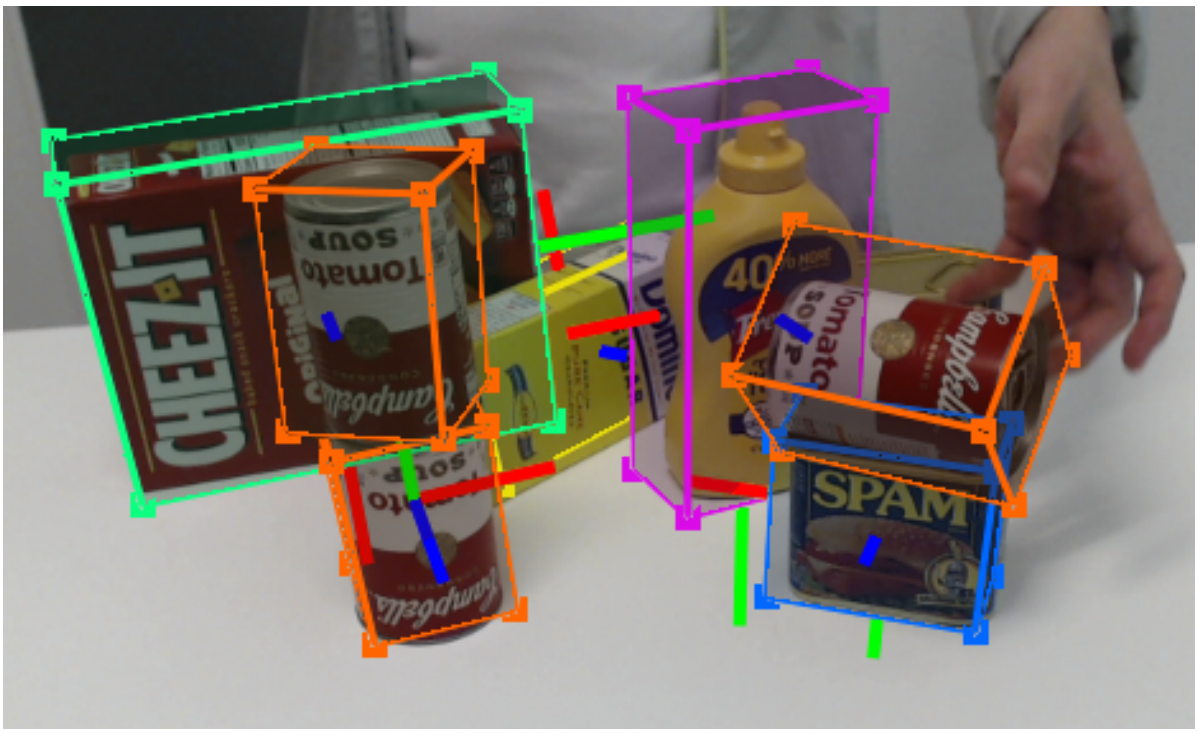


Figure 2.1: Pose estimation with bounding box and coordinate system [1]

system is then calculated. For this purpose, a Convolutional Neural Network (CNN) is usually trained to predict the geometric 2D-3D correspondence. The scale of the object is also taken into account in the calculation. The PoseCNN then adjusts the rotation. For this, the texture of the object is used to calculate the visual features of the 3D model and the image. In this step, the image is limited to the bounding box around the object, which originates from the recognition phase. Now a prediction is calculated for each pair of features. The final prediction is determined based on a loss function and represents the rotation. CNNs are often used for prediction. The other algorithms work in a similar way. For example, Deep Model-Based 6D Pose Refinement in RGB uses the object contours instead of the center point for prediction, but generally follows the same principle.

Keypoint: The latest keypoint algorithms, such as COPE [12], Keypoint-graph-driven learning framework for object pose estimation [13], BB8 [14], PVNet [15], Segmentation-driven Pose [16] and Seamless Single Shot 6D [17], use robust keypoint detection in combination with a variation of the Perspective-n-Point (PnP) algorithm. This process is divided into two stages: First, the keypoints are identified, and then the 6 DOF pose of the camera is predicted, which requires intrinsic parameters of the camera. The algorithms differ in the way they locate and represent the keypoints, which can be done using coordinates, heatmaps or vector field-based representations, and in the CNNs used to account for different lighting conditions, occlusions and powers. Despite the differences, the basic concept remains the same.

UV correspondences: Algorithms such as EPOS [18], CDPN [19], Pix2Pose [20], GDR-Net [21], SO-Pose [22], Perspective Flow Aggragation [23], SurfEmb [24], OSOP [25] and Neural Correspondence Fields [26] use additional data from UV coordinates to make the pose estimator more reliable and improve its performance. This is particularly useful for objects without texture, as seen in Pix2Pose. The UV coordinates are a local representation of the 3D object point in a 2D image. GDR-Net uses intermediate geometric representations based on a dense correspondence as keypoints, which are then adjusted in a second stage using a PnP algorithm and RANSAC [27] to obtain the best prediction. According to the BOP challenge (2.1.2), this type of algorithm currently provides the best results on multiple datasets.

2.1.2 BOP-Challenge

In 2019, the first open BOP challenge [28] was conducted with the aim of comparing different current methods for estimating the pose of objects. The process started with the detection or segmentation of the objects, followed by the estimation of the 6D pose. The ranking provides an overview of the test image format, the performance score for each dataset and the time required. The two test image formats are RGB and RGB-D, the latter containing depth information in addition to color. In 2022, the BOP Challenge [29] was expanded to include a separate challenge for the recognition and segmentation task. The current top algorithms in the competition are based on the Geometry-Guided Direct Regression Network (GDR-Net) [21], with various versions occupying the first five places. This is followed by Perspective Flow Aggregation (PFA) [23], RCVPose 3D [30], ZebraPose [31] and SurfEmb [24].

As already mentioned, datasets help to compare different algorithms. One of the first large-scale open source datasets was ImageNet [32]. Other suitable datasets are Microsoft Common Objects in Context (MS COCO) [33], Yale-CMU-Berkley (YCB) [34] and Large Vocabulary Instance Segmentation (LVIS) [35]. The BOP project [36] offers the possibility to benchmark certain algorithms on the given datasets. Over time, more and more different datasets will be added. At the moment 13 are included and an overview can be seen in Table 2.1.

2.2 Deformable Objects

Most robotic tasks such as object recognition, pose estimation and object manipulation focus mainly on rigid objects. The reason for this is that deformable objects present a number of special challenges. They must be able to handle different shapes and material properties such as elasticity and plasticity [47]. When a rigid object is gripped by a gripper, its shape changes only slightly. However, a deformable object can change drastically depending on the force of the gripper. Gripping an empty cardboard box, for example, is different from gripping a block of wood.

dataset	objects	special feature
LM (Linemod) [37]	15	texture-less
LM-O (Linemod-Occuleded) [38]	15	texture-less, various level of occlusion, addition to LM
T-LESS [39]	30	exhibit symmetries and mutual similarities in shape and/or size, no significant texture or discriminative color
ITODD (MVTec ITODD) [40]	28	realistic industrial setups
HB (HomebrewedDB) [41]	33	13 scenes with varying complexity
HOPE (NVIDIA Household Objects for Pose Estimation) [42]	28	50 scenes from 10 household/office environments, up to 5 lighting variations per scene
YCB-V (YCB-Video) [43]	21	in 92 videos
RU-APC (Rutgers APC) [44]	14	textured products from Amazon Picking Challenge
IC-MI [45]	6	two texture-less and four textured household
IC-BIN [46]	2	two objects from IC-MI, which appear in multiple locations with heavy occlusion in a bin-picking scenario
TUD-L (TUD Light) [36]	3	moving objects under eight lighting condition
TYO-L (Toyota Light) [36]	21	table-top setup, with four different table cloths and five different lighting conditions

Table 2.1: Overview BOP Challenge datasets

A major challenge is to create a model of a deformable object. Generally, a discrete representation is chosen, e.g. a mesh, a skeleton, a deformable template, landmarks, particles or point clouds. All of these methods use specific keypoints to identify specific parts. For example, a mesh consists of vertices, edges and faces [48]. The face is surrounded by edges that are connected to the vertices. Figure 4.2 explains the terms vertex, edge and face. First, the mesh is calculated for the rigid object or for the initial state of the object. Then the vertices are linked to the key points. If the key points move, the vertices, edges and faces also move. The vertices only change their position, the edges can change their scale and orientation, and the face can change its shape, scale, orientation and rotation. This information can be used to compare the original and resulting mesh and create heat maps. Another approach is the use of skeletons. The basic elements of the skeleton are bones connected by joints that allow flexion. When a bone moves, the entire skeleton moves. The deformation itself is the interesting part. It can be modeled with the following approach [47]. At the beginning, the object is in its initial state S_0:

$$\mathcal{S}_0 = \left\{ \mathbf{p}_i^0 = \left\{ x_i^0, y_i^0, z_i^0 \right\} \in \mathbb{R}^{n=3}, i \in N \right\} \tag{2.1}$$

The number of points that define the shape is denoted by N. If an external force is applied to the object, the points will move to a new position $\mathbf{p}^{\text{new}}$. Examples of external forces include gravity, manipulators, and contact with rigid objects. The deformation can be expressed as the difference between the initial position and the resulting position $\mathbf{u} = \mathbf{p}^{\text{new}} - \mathbf{p}^0$, which is a displacement vector field. The stress tensor σ and the strain tensor ϵ are then defined. The stress tensor indicates the force per unit area that acts on the shape of the object, and is calculated for each point from $\mathcal{S}_0$ using Hooke's law $\sigma = \mathbf{E}\epsilon$. This requires the strain tensor, which provides an overview of the deformation relative to the initial object. The strain tensor is calculated as $\epsilon = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)$ with $\nabla\mathbf{u}$ using the displacement vector field, shown in Equation 2.2. The advantage of this formula $\sigma = \mathbf{E}\epsilon$ is that it works only for small displacements, and there is a linear relationship between stress and strain on the elasticity tensor $\mathbf{E}$.

$$\nabla\mathbf{u} = \begin{pmatrix} \partial u_x/\partial x & \partial u_x/\partial y & \partial u_x/\partial z \\ \partial u_y/\partial x & \partial u_y/\partial y & \partial u_y/\partial z \\ \partial u_z/\partial x & \partial u_z/\partial y & \partial u_y/\partial z \end{pmatrix} \tag{2.2}$$

The deformation of the particle i over time is determined by Newton's second law. The position of the particle i at time t is denoted by $\mathbf{p}_i^t$. With Equation 2.3 and Equation 2.4 the velocity $\mathbf{v}^t \in \mathbb{R}^3$ and the acceleration $\mathbf{a}_i^t \in \mathbb{R}^3$ are calculated by taking the first-order derivative with the mass $m_i \in \mathbb{R}^3$ and the external force $\mathbf{f}_{\text{ext}\ i}^t \in \mathbb{R}^3$.

$$\mathbf{v}_i^t = \dot{\mathbf{p}}_i^t, \quad \mathbf{a}_i^t = \dot{\mathbf{v}}_i^t, \quad m_i\mathbf{a}_i^t = \mathbf{f}_{\text{ext}_i^t}^t \tag{2.3}$$

$$\dot{\mathbf{p}}_i^t = \frac{\left(\mathbf{p}_i^{t+\Delta t} - \mathbf{p}_i^t\right)}{\Delta t} \quad \dot{\mathbf{v}}_i^t = \frac{\left(\mathbf{v}_i^{t+\Delta t} - \mathbf{v}_i^t\right)}{\Delta t} \tag{2.4}$$

Next, with explicit Euler integration in Equation 2.5, the velocity $\mathbf{v}_i^t$ and position $\mathbf{p}_i^t$ of point i at time t are calculated.

$$\begin{aligned} \mathbf{p}_i^{t+\Delta t} &= \mathbf{p}_i^t + \mathbf{v}_i^t \Delta t, \\ \mathbf{v}_i^{t+\Delta t} &= \mathbf{v}^t + \frac{1}{m} \mathbf{f}_{\text{ext } i} \Delta t, \end{aligned} \tag{2.5}$$

This is only a simple solution for representing the deformation. There are many other approaches that are more stable against unrealistic deformation behavior. One example would be the representation in the form of a mass-spring system or model-based with neural networks.

2.3 Synthetic Data

The real world is the most comprehensive dataset we have. There is an infinite number of objects and scenes. Unfortunately, it is impossible to capture all this data. Taking a photo of the same object with 1000 different views would take a very long time, especially if the object is moving. As an alternative, synthetic data generation is used to generate data. [49]. This information is then used by computer simulations or algorithms to gain a better understanding of the real world. Synthetic data is not only artificial, it should also reflect the real world. This is done using mathematical or statistical methods that have been learned from real datasets. For neural networks, the motto "more is more" applies. Large, well-labeled datasets are required for training in order to obtain a result that comes close to the real world. In addition, generating synthetic data is less time-consuming and less expensive than manually labeling a dataset. Comparing the amount of data between a hand-labeled dataset and a GPU-powered server-engine dataset is an unfair scenario.
Synthetic data can be generated using three different approaches [50]. The first is a stochastic process that generates random data based on the structure of the real world. This is the optimal method when the content is not as important as the structure, such as in a stress test of a system with a large amount of data. The second method is a rule-based process that is guided by specific rules defined by humans. For example, a deformation task may be limited to a range of 20-120 degrees, as opposed to a stochastic process that can randomly choose any angle between 0 and 360 degrees. This method can lead to distortions in the synthetic data as the generation process can be steered in a certain direction. Also, the rules may not be able to keep up with the changing real data, leading to a deviation in the results. The last method is based on deep learning, where a machine learning model attempts to estimate values based on its own training data from the real world. This is the most realistic method, but also the most time-consuming and expensive, as it requires a large real-world dataset to achieve accurate results. This method has opened up a new field of research, with AI systems such as OpenAI ChatGPT [51], Google LaMDA [52], and DeepMind Sparrow [53] being able to produce and learn very realistic data. However, this has also raised concerns about data privacy, copyright, and security.

3 System Components

This chapter deals with the hardware and software components of the project. Metashape is used to reconstruct dataset objects (3.1) from RGB images, and the Open3D library is used for post-processing. Blenderproc, a Blender plug-in, is then used to calculate deformed versions of the objects (3.2). Pix2Pose (3.3) is trained with the resulting object variants. At last the evaluation is done with the synthetically generated data.

3.1 Objects of Deformable Object Dataset

As described in Chapter 2, a dataset is used to compare different algorithms or systems. Currently, there is no dataset for deformable objects. One of the main advantages of a dataset should be its reusability, which is why the Deformable Object Dataset (DOD) was designed similarly to the BOP challenge examples. This allows other algorithms to perform the benchmark test without additional effort. The dataset comprises seven objects with four degrees of deformation and 240 views each. The Tables 3.1 - 3.7 show the objects chips, juice, paste, pringles, shampoo, teabox and pastry. All of these objects are classic household items that have typical shapes such as boxes, cones, packets and cylinders. Other criteria for these objects are rigidity, surface texture and stability of volume. Pringles, for example, have a much higher rigidity than pastry, but both objects have a high volume stability. The teabox, on the other hand, can be easily deformed. In their untouched state, the bag of potato chips and the juice represent an almost rigid object that can only be deformed slightly. However, when they are opened, they become unstable and can be deformed. When assembling the objects, attention was also paid to the surface finish. Shiny and transparent materials were avoided as they can pose a problem for object reconstruction and pose estimation. The pastry is the only object with a small amount of transparency.

3.2 Software packages

This section deals with the software packages used. The advantages and disadvantages as well as the features of the individual packages are examined. First, Metashape (3.2.1) is used to reconstruct the object model. Open3D (3.2.2) is used to post-process the models. Finally, BlenderProc (3.2.3) is used in combination with Blender to perform the deformation of the object models.

Deform 1	Deform 2	Deform 3	Deform 4
			

Table 3.1: Chips reconstruction, front/back view

Deform 1	Deform 2	Deform 3	Deform 4
			

Table 3.2: Juice reconstruction, front/back view

Deform 1	Deform 2	Deform 3	Deform 4
			

Table 3.3: Paste reconstruction, front/back view

Deform 1	Deform 2	Deform 3	Deform 4
			

Table 3.4: Pringles reconstruction, front/back view

Deform 1	Deform 2	Deform 3	Deform 4
			

Table 3.5: Shampoo reconstruction, front/back view

Deform 1	Deform 2	Deform 3	Deform 4
			

Table 3.6: Teabox reconstruction, front/back view

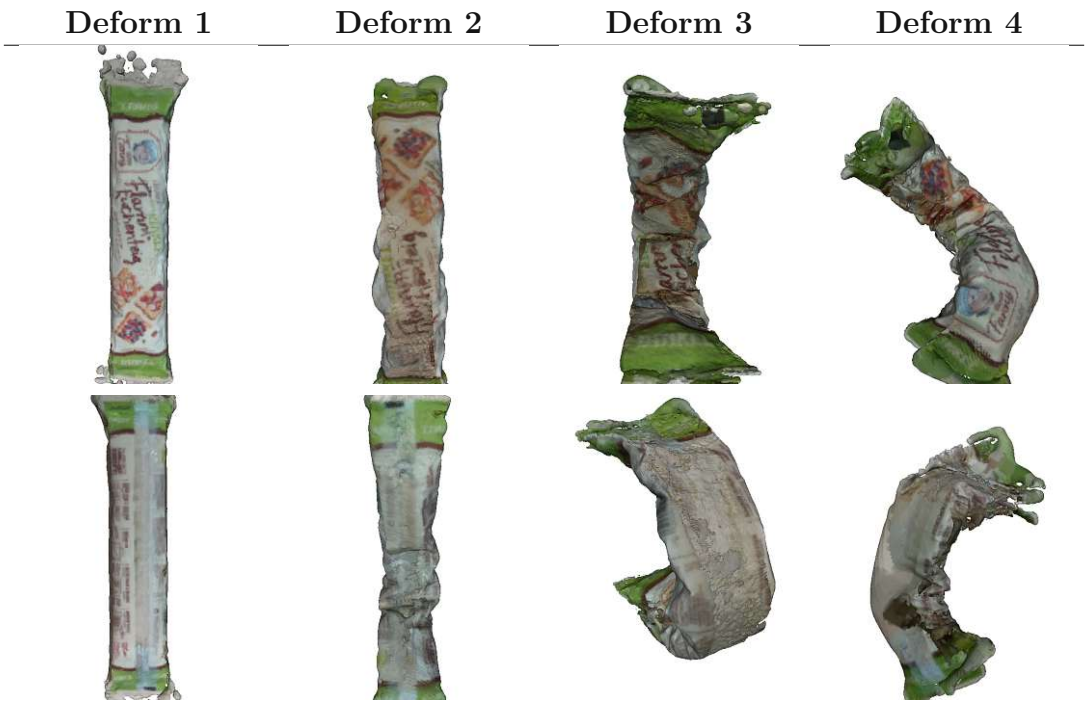


Table 3.7: Pastry reconstruction, front/back view

3.2.1 Metashape

Agisoft's Metashape software enables the creation of 3D object models from 2D images. Photogrammetric processing of the images is used to generate a point cloud, which is achieved by calculating feature points in each image and then mapping them to tie points. It is recommended to use at least one masked image to filter out unimportant points. The tie points are then used to align the images using aerotriangulation (AT) and bundle block adjustment (BBA). The point cloud is then used to reconstruct the surface using camera position and image information as well as dense matching. The mesh can then be exported in various file formats (.ply, .obj, .stl) for further use. The main advantage of Metashape is that it can create a highly detailed 3D model of an object with just a few masks and images from different views. The downside is that the software is not free, although Agisoft offers a 30-day free trial. If reconstructing objects is a recurring task, it may be necessary to switch to an open source alternative.

3.2.2 Open3D

Open3D[54] is an open source Python library that is mainly used for processing 3D data. It also supports C++. This library provides structures, visualization, processing algorithms, machine learning support, scene reconstruction, surface alignment and GPU acceleration. It is particularly useful for 3D data processing algorithms such as layer segmentation, filtering, scaling and surface reconstruction. Open3D is also an excellent

tool for visualizing the results of the deform pipeline steps. It enables fast visual debugging with minimal effort. Open3D supports all common file extensions. The main advantage of Open3D is the fast and uncomplicated manipulation and visualization of 3D data. The package is well structured and easy to understand. The disadvantages are the limited processing algorithms and the inconsistencies between versions. As the package is still under development, the method calls and parameters may change. This can be a problem if the version is not specified, especially when using Docker environments.

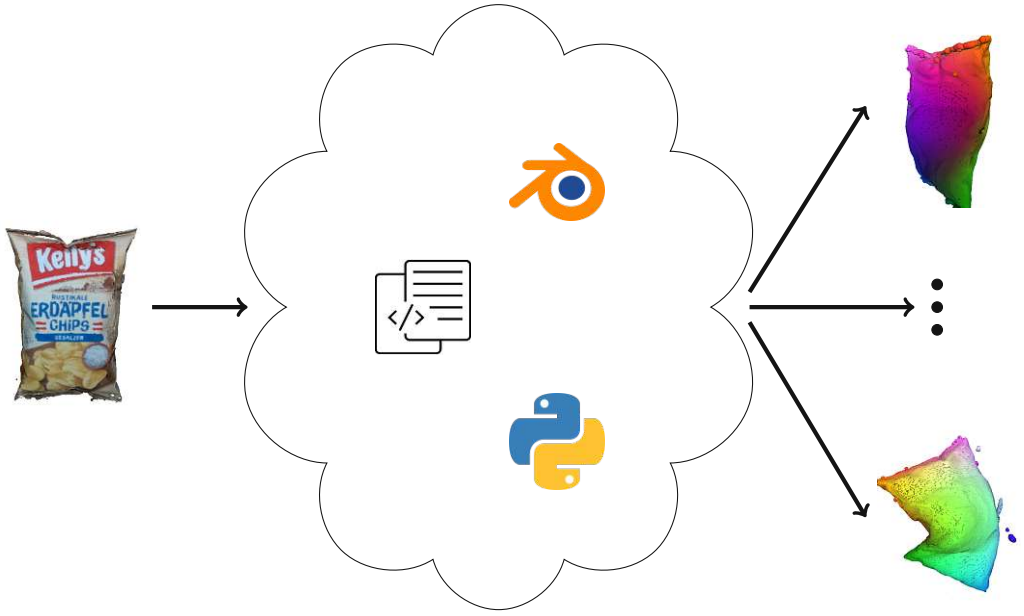


Figure 3.1: Concept of the deforming pipeline

3.2.3 BlenderProc

BlenderProc [55] is an add-on program for Blender, a free and open source software for 3D objects. It was developed by the German Aerospace Center for photorealistic rendering. Blender is capable of modeling, texturing, manipulating, simulating, rendering and much more, making it a powerful tool for creating anything from simple 3D printed objects to a video game or animated film. BlenderProc is used to control Blender with Python commands. This allows users to load, edit and save objects as well as control the entire scene, including various objects, cameras, light sources and more. This requires a different approach than the typical Blender project, which is more like a clicking adventure with hundreds of options and parameters that can be seen in the visualization. BlenderProc does not offer such a powerful interface, which makes debugging difficult and slow. However, it has the great advantage that a sequential process, such as a deformation pipeline, can be applied to many different objects with different values. Once the pipeline is coded, it can be used multiple times. BlenderProc's debugging window saves all commands, which, combined with the documentation, allows the user

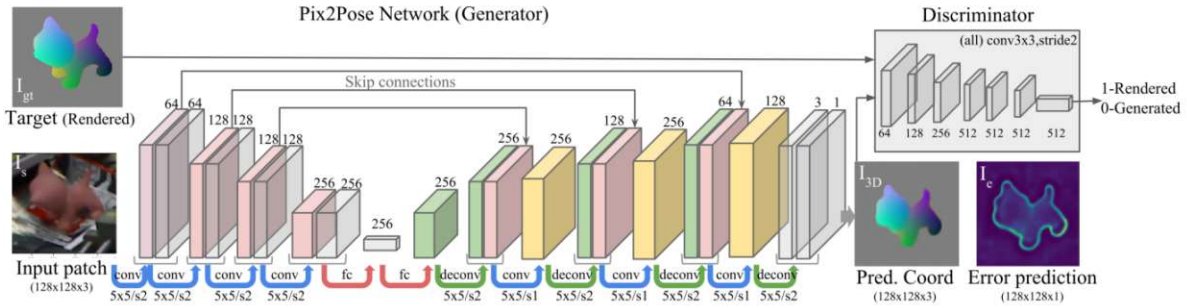


Figure 3.2: Pix2Pose architecture [20]

to write code in a relatively short time, even without prior knowledge. In addition, the functionality of Blender can be used with all Python libraries, such as Open3D. Figure 3.1 shows the general concept of the deformation process. The object as input is processed with a combination of Blender, Blenderproc and the respective algorithm. The result is various deformed objects that are saved in the desired format.

3.3 Pose Estimator - Pix2Pose

The following session explains the process of basic pose estimation. Pix2Pose [20] is a framework that regresses 3D coordinates pixel by pixel between an RGB image and a 3D model. This 3D model is textureless, which means that all information is stored in the RGB-colored UV prediction image. To clarify: The RGB information of the pixels does not represent the texture of the object. Figure 3.2 shows the architecture of the framework. A modern object detector is used to calculate the bounding box around the object, which is then cut out and results in an image I_s. This image is used as input to the Pix2Pose system, and the output is the 3D coordinates of each pixel I_{3D}, which are normalized. Moreover, the estimated error for each prediction $I_e = G(I_s)$ is a function of the Pix2Pose network G, which can be interpreted as the confidence value for each pixel. The relationship between XYZ and RGB is used to map an XYZ coordinate to an RGB coordinate, since both use three values for representation. For training, the ground truth image I_{gt} is generated by loading the textureless 3D model into the ground truth scene and coloring it with the UV prediction. The network consists of multiple convolutional layers, deconvolutional layers and fully connected layers.

The Pix2Pose network generates an output that is used to calculate the position of the object. This process is illustrated in Figure 3.3. First, the width, height and center of the bounding box are used to crop the relevant part of the image with an input size of 128 x 128 px. If the bounding box is not square, the width and height are set to the same size. To take masking into account, the resulting size is multiplied by a factor of 1.5. A two-stage prediction is then carried out. In the first stage, the bounding box is aligned with the center of the object, which may vary slightly depending on the base

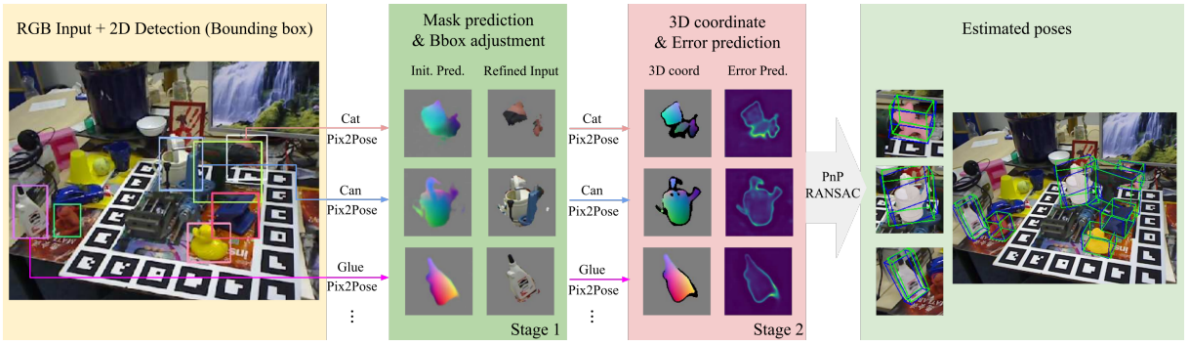


Figure 3.3: Pix2Pose pose estimation process [20]

object detector. In the second stage, the same mesh is used to create a final prediction using the refined inputs from the first stage. This is achieved using a PnP algorithm in combination with a RANSAC algorithm. The fact that Pix2Pose uses textureless models instead of texture models makes the algorithm fast.

3.4 Robot Sasha

Figure 3.4 shows a Toyota HSR (Human Support Robot) used for robotic experiments. This robot is designed to assist people in their daily lives, such as cleaning, carrying objects and other manipulation tasks. As can be seen in the illustration, the robot is modeled on the human body, with a head containing several cameras, a display and a microphone. The body is cylindrical and mounted on a movable base that provides stability and flexibility. The base enables omnidirectional movement, making it virtually impossible to tip over. The robot arm is equipped with 5 Degrees of Freedom (DoF), which is a limitation as the base must reach all points, making movement planning difficult. At the end of the arm is a gripper for handling objects, which also contains a suction cup.

The robot is equipped with several camera systems located in the head and on the end effector, as well as a Lidar system located above the base. These are the primary means of collecting data about the environment. For safety, the robot has bumper sensors on the base, a magnetic sensor on the underside of the base and an emergency stop switch on the back. The magnetic sensor is used to detect if the robot crosses its power cable. The robot main camera system is the Xtion PRO LIVE RGB-D camera. This camera uses various technologies, such as infrared sensors, adaptive depth detection, color image recognition and audio streaming, to capture RGB and depth images and create a point cloud. To do this, the camera emits a known signal pattern, which is then captured by a CMOS sensor. The difference between the detected signal and the known pattern, combined with the geometry of the camera systems, produces the point cloud.

The robot's battery system enables an autonomous operating time of up to two hours. A power cable or docking station is available for charging. In addition, the robot can be controlled manually by the user with a Playstation controller. This simplifies daily work.

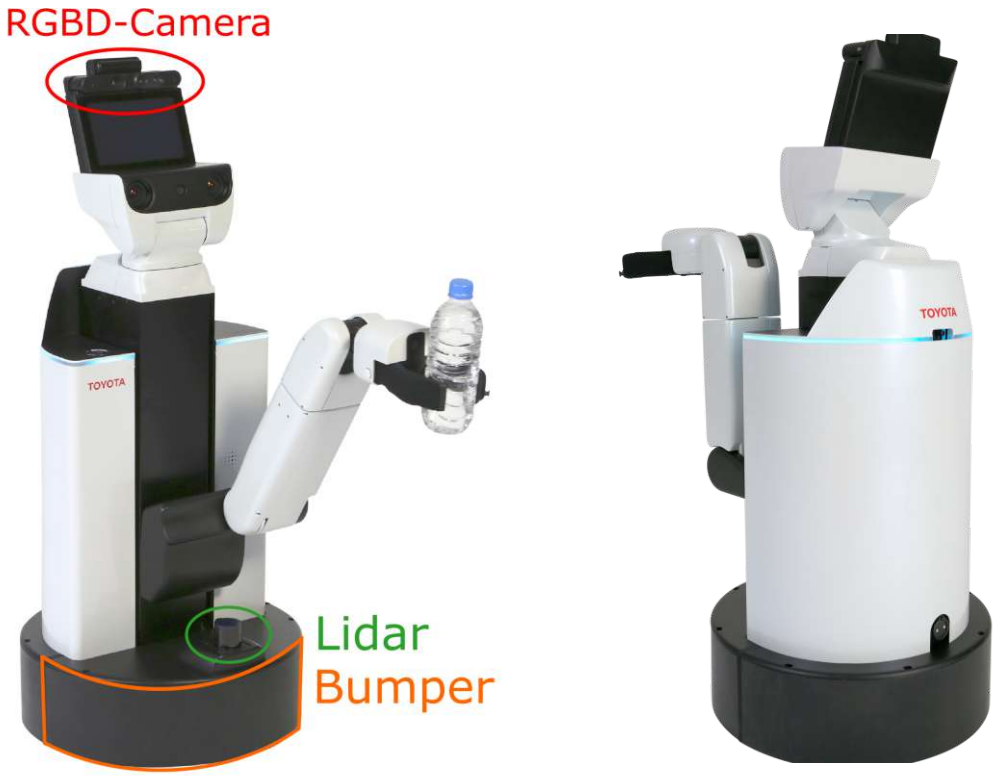


Figure 3.4: Toyota HSR [56]

4 Object Reconstruction, Deformation and Pose Estimation

This chapter describes the steps from object modeling to pose estimation. It starts with photographing the object. A 3D model is then created (4.1). The mesh deformation pipeline then takes the model and creates deformed submodels (4.2). Training data is generated from these submodels (4.3). After training the object detector and the pose estimator (4.4), the method can be tested and evaluated.

4.1 Object Reconstruction Process

The first step of the process is to reconstruct the object, with a focus on accuracy. This is illustrated in Figure 4.1, and the objects are defined in section 3.1. For each object, two sequences of 30 images were taken from four angles with the aim of reconstructing the entire object. To achieve better results, the overlapping areas should be as large as possible, resulting in 240 images of each object in the database. To reduce the workload, the images are captured by a robot in the lab. This uses a KUKA LBR arm equipped with a Realsense camera, which provides images with a resolution of 1280x720 pixels and also a matching depth image. The images are then loaded into Agisoft Metashape Professional software (Section 3.2.1) and the important part of the image is marked with the selection tool, creating black and white masks. Between three and six images are selected for masking. The camera positions for each image are estimated by searching for features between the images. The result is a point cloud containing the object, the table plane and some background artifacts with a good resolution. A mesh is then created from the point cloud and the texture is calculated, resulting in an authentic reconstruction of the object. To obtain a more accurate point cloud, a PLY file is created from the detailed mesh.

The Metashape process in detail:

1. Load images (Workflow → Add Photos ...)
2. Mask images (Choose photo, use selection tool)
3. Align images (Workflow → Align Photos ...)
   - Accuracy: Highest
4. Build mesh (Workflow → Build Mesh ...)

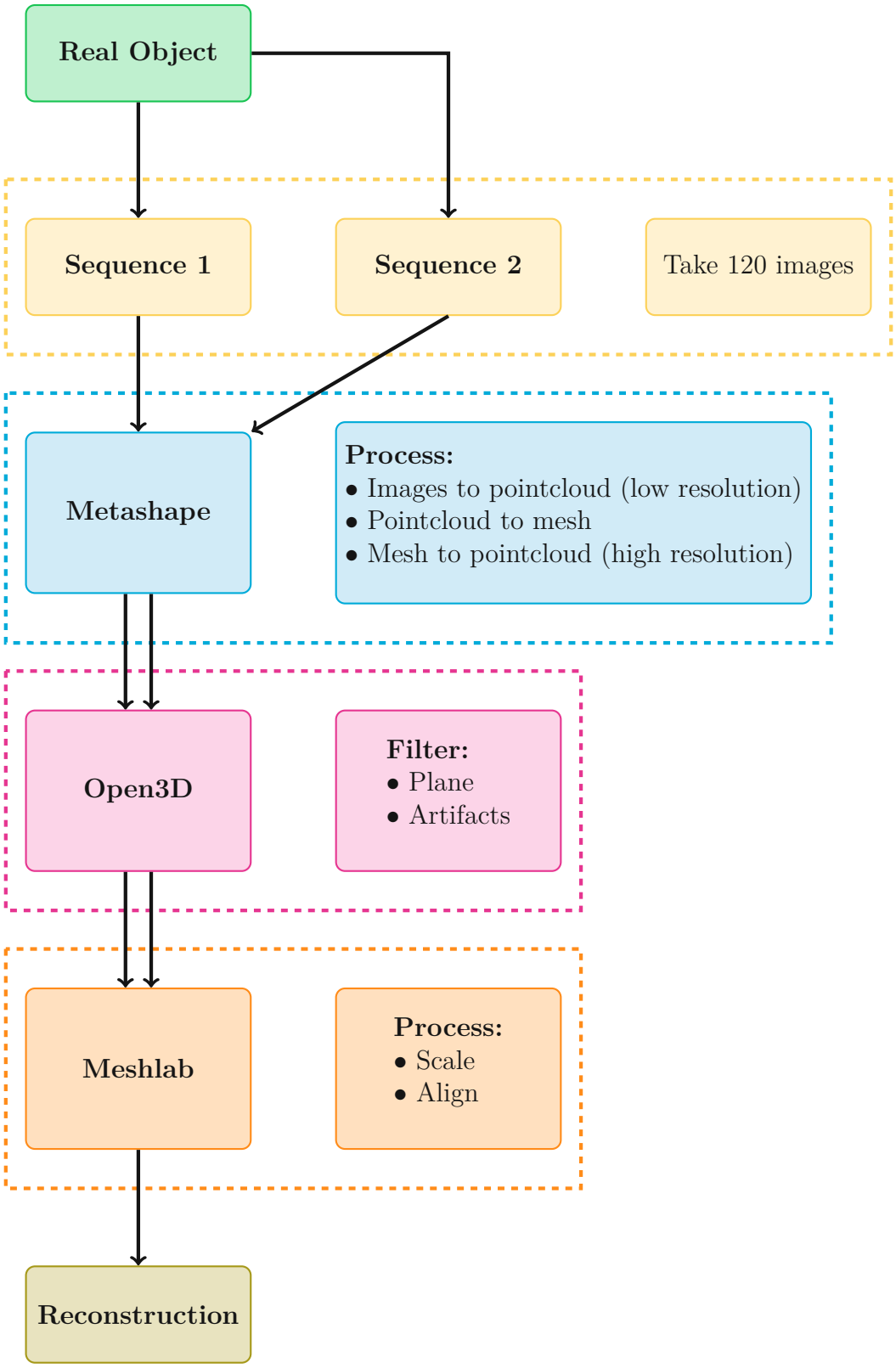


Figure 4.1: Object reconstruction process

- Quality: Ultra High

5. Build texture (Workflow → Build Texture ...)
6. Build pointcloud (Workflow → Build Point Cloud ...)
   - Quality: Ultra High

The Open3D library is used to remove the artifacts of the table plane and the background. A bounding box filter is used to remove the background artifacts, as the world coordinate system is located at the center of the object, resulting in a large distance between the background and the object. A RANSAC plane detection is used for the plane, whereby the outlier points are retained and the inlier points are removed. The object is then separated and can be processed further.

The segmentation process in detail:

1. Segment plane (open3d.geometry.PointCloud.segment_plane)
   - distance_threshold = 0.001
   - ransac_n = 3
   - num_iterations = 1000
2. Crop cloud (open3d.geometry.PointCloud.crop)
   - bounds = [[x_min, x_max], [y_min, y_max], [z_min, z_max]]

In a final step, the two sequences of the same object must be aligned with each other. Meshlab software is used for this. Both point clouds are opened in Meshlab and all artifacts that were not filtered by the previous steps can be removed manually. The option "Align" is used for the alignment step. One of the meshes is set to a fixed position in the specified coordinate system using the "Glue Here Mesh" option. For the second mesh, the option "Point-based gluing" is used to mark the same points in different meshes. This information is used to calculate a transformation between the meshes. It is recommended to select more than four points if possible. The transformation can be improved with ICP refinement using the "Process" option. Finally, the meshes are combined into a final mesh. The measurement method can be used to scale the final mesh and compare the result of the 3D model with the real object.

1. Load two meshes
2. Align (Edit → Align)
3. Fix first mesh (Align Window → Glue Mesh Here)
4. Find transformation (Align Window → Point Based Gluing)
   - use False Color → uncheck
   - use Point Rendering → check

- Allow Scaling → check

5. ICP Refinement (Align Window → Process)
6. Create final mesh (Filters → Mesh Layer → Flatten Visible Layers)
   - Merge Only Visible Layers → check
   - Delete Layers → check
   - Merge duplicated vertices → check
   - Keep unreferenced vertices → check
7. Measure feature (Edit → Measuring Tool)
   - Choose first point
   - Choose second point
8. Scale mesh (Edit → Quality Measure and Computations → Transform: Scale, Normalize)
   - X Axis → Scaling Factor

4.2 Object Deformation Algorithm

The deformation pipeline is outlined in the following section. The Blender software is used for this. With the additional pipeline for Blender, known as Blenderproc, which is discussed in Section 3.2.3, it is possible to use Python code in conjunction with almost all of Blender's functions. This means that everything can be calculated automatically. First, the mesh, including the vertices and faces, is loaded for each object. The terms face, edge and vertex are easy to understand when looking at a cube. The vertices are the corner points of the cube, the edges connect two vertices, and a face is the surface enclosed by the edges. This is shown in Figure 4.2.

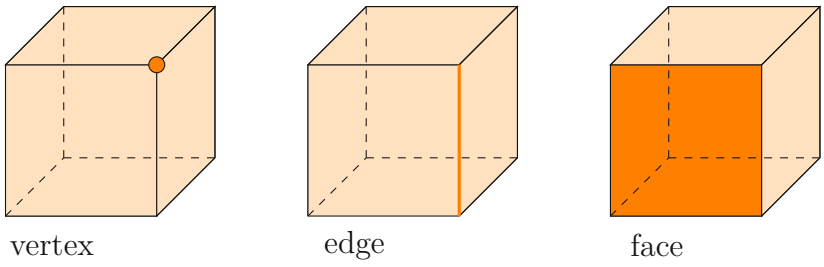


Figure 4.2: Vertex, edge and face explained with a cube

The object is then moved to the origin so that the lower middle point of the calculated bounding box is at (0/0/0). Equation 4.1 is then used to calculate the lengths of all

edges between the vertices, and any length value that is greater than a certain threshold (corresponding to the maximum length between two vertices) is marked as selected. A new vertex is then inserted between the edges using the subdivision method, and this process is repeated until no more edges fulfill the given condition. This results in a homogeneous edge length along the object and all edges at the same position are removed. A painted version of the mesh is then created, which helps to find a clear match between the original and the deformed mesh. Each vertex is selected and the RGB value is calculated according to its position in XYZ, shown in Equation 4.2. A "subsurf" filter is then used to increase precision, close holes and ensure that each vertex is connected to an edge. The painted undeformed object is then exported, which serves as a reference for all other objects.

$$\begin{aligned} \text{vertices} &= [v_1, ..., v_l] \\ \left\{ v_i \mid |v_i - v_{i+1}| > \frac{\max\limits_i v_i}{\text{th}}, i = 1, ..., l-1 \right\} \end{aligned} \tag{4.1}$$

$$\begin{aligned} R &= \frac{x_i - \min(x)}{\max(x) - \min(x)} \\ G &= \frac{y_i - min(y)}{max(y) - min(y)} \\ B &= \frac{z_i - min(z)}{max(z) - min(z)} \end{aligned} \tag{4.2}$$

The following deformation technique is modeled on the human body and involves the use of armatures. These armatures consist of bones that are connected by joints. The bones form the skeleton of the object and all have the same length. When a bone is moved, this has an effect on the connected bones, similar to a chain with rigid elements. The bones move in the X and Y directions, with the Z axis aligned with the longest part of the object. The skeleton S is made up of $n_S = [5, 14]$ bones:

$$S = \begin{pmatrix} b_{1,x} & b_{2,x} & ... & b_{n_S,x} \\ b_{1,y} & b_{2,y} & ... & b_{n_S,y} \end{pmatrix} \tag{4.3}$$

Next, the deflection of each bone in the X and Y directions is calculated. For this a random goal $g = [-270°, 270°]$ is calculated and the sum of all bones have to fit this value by a tolerance of $\pm 5\,°$:

$$g = \begin{pmatrix} g_x \\ g_y \end{pmatrix} \tag{4.4}$$

$$g_x = \sum_{i=1}^{n_S} b_{i,x} \qquad g_y = \sum_{i=1}^{n_S} b_{i,y} \tag{4.5}$$

The deflection of a bone is determined randomly between $b_{limit} = [30°, 50°]$ due to restrictions in the properties of the material. However, if the goal is to reach 270° and

the maximal limit of 50° is too low, the limit is adjusted to accommodate the situation.

$$b_{limit,x} = \frac{|g_x|}{n_S} + 10° \qquad b_{limit,y} = \frac{|g_y|}{n_S} + 10° \tag{4.6}$$

The bones have to meet the condition:

$$|b_{i,x}| \leq b_{limit,x} \qquad |b_{i,y}| \leq b_{limit,y} \tag{4.7}$$

In approximately 20% of all calculations, the bone values are set to zero, that allows the object to be deformed in a single direction, rather than along both the X- and Y-axes.

Table 4.1 presents a skeleton made up of six bones, the result of which is depicted in Figure 4.3.

g	b_{limit}		b_1	b_2	b_3	b_4	b_5	b_6	$\sum$	± 5
0	0	b_x	0	0	0	0	0	0	0	0
63	34	b_y	25	10	-19	30	24	-5	65	-2
-171	49	b_x	3	-39	-31	-45	-24	-38	-174	3
0	0	b_y	0	0	0	0	0	0	0	0
-83	41	b_x	-24	-6	3	-30	-10	-12	-79	-4
36	37	b_y	28	-3	-30	12	34	-5	36	0
207	47	b_x	40	37	46	7	32	41	203	4
120	35	b_y	31	10	12	32	35	3	123	-3
5	31	b_x	-4	10	-14	-16	0	25	1	4
-260	50	b_y	-49	-42	-50	-35	-48	-34	-258	-2

Table 4.1: Example bones deflection

Mapping Vertices between Models

To calculate the correspondence between the RGB image and the XYZ color model, a color mapping procedure is required. This algorithm uses the location information of a vertex to store both the color and the original texture value in an array of the form (3, 256, 256, 256). This array contains up to 16 million colors, converting the color information of the colorized model from the range [0.0, 1.0] to the range [0, 255], which allows the use of an unsigned integer instead of a floating point value and reduces the

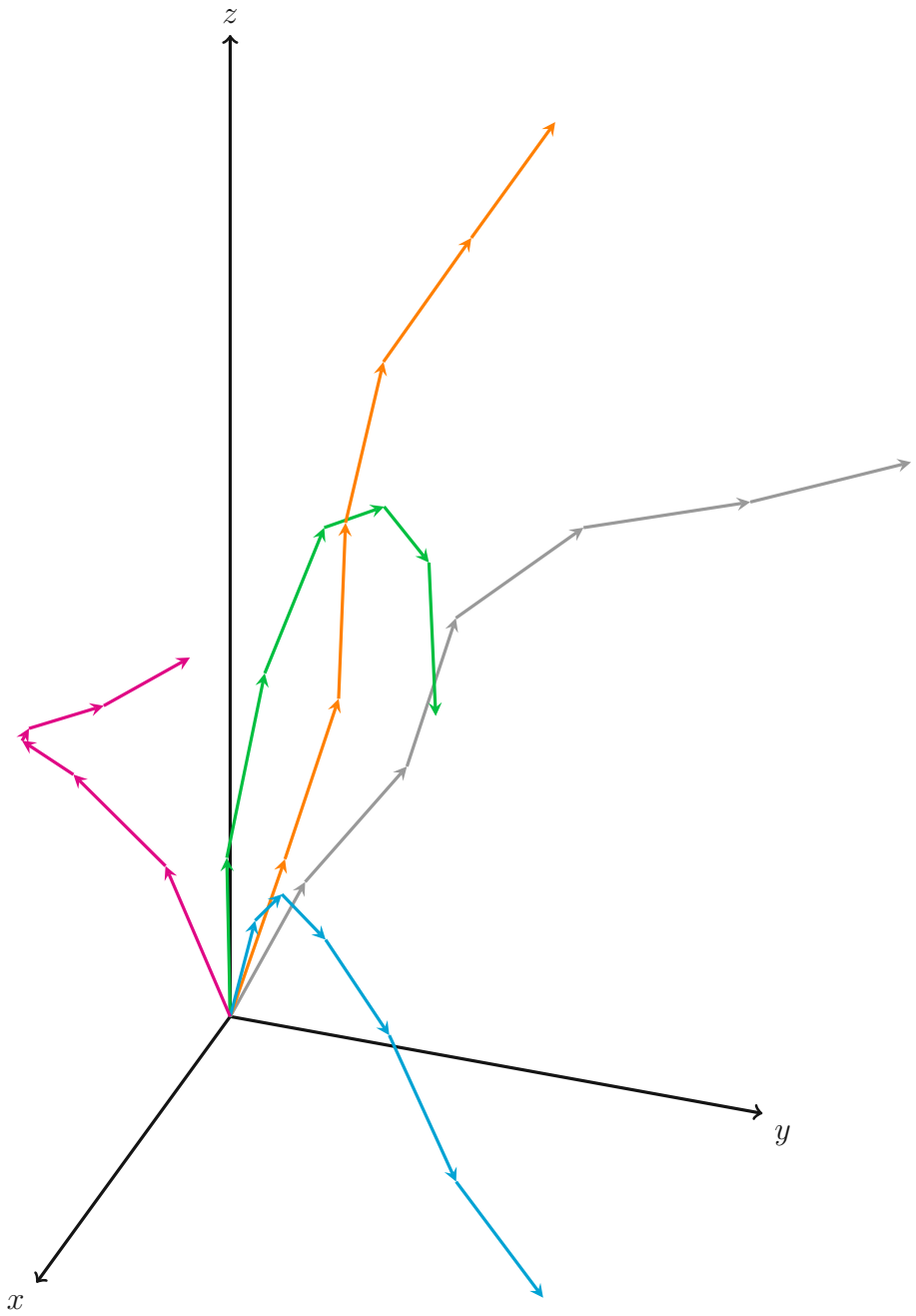


Figure 4.3: Bones movement

memory requirements of the mapping file by a factor of 10. The colored value can then be used to access the texture color value as explained in Section 4.2.

$$R_{texture} = mapping[0][R_{color,uint8}][G_{color,uint8}][B_{color,uint8}]$$

$$G_{texture} = mapping[0][R_{color,uint8}][G_{color,uint8}][B_{color,uint8}]$$

$$B_{texture} = mapping[0][R_{color,uint8}][G_{color,uint8}][B_{color,uint8}]$$

with

$$X_{color,uint8} = X_{color} \cdot (256 - 1) \mid datatype = uint8$$

Since the original colored model and all its deformed versions have the same number of vertices and the same colored data, the mapping also works for them. The mapping file can be used in any part of the project, but the calculation requires a lot of effort as each vertex must be examined. For the object chips_01 this means that about $600{,}000^2$ operations have to be performed. To make this process more efficient, the algorithm was optimized to use GPU support.

Outlier Removal

Once the color mapping file is created, the models can be further enhanced. All clusters with connected vertices are identified and only the largest is retained. As all objects have the same number and order of vertices, these can be calculated in one model and then quickly removed from the other files. Therefore, the outlier indices for each model are stored in a combined array. The repeated indices are then removed from all objects in the same group.

4.2.1 Model Manipulation Tools

The object is deformed in Blender using a series of filters and manipulation operations. The most important of these are listed below in the correct order.

Move_origin_to_bottom_mean_point: This operation is applied directly to the object, moving it so that its bounding box is at the center of the specified coordinate system. This means that the bounding box is in the middle of the X and Y axes and the lower center of the bounding box is at Z = 0.

Subdivide: This operation is applied to the given mesh, dividing the selected edges by the number of intersections and creating new vertices evenly distributed between them. For this purpose, a vertex group must be assigned. In this project, all edges with a length greater than 1/7 of the maximum edge length are selected and a vertex is placed in the center of the edge to obtain a more detailed mesh.

Remove_doubles: This operation is applied to the mesh, whereby all vertices that lie within a certain distance are combined. The pipeline merges vertices that are less than

1/x the maximum edge length. This is a common step in the pipeline, with the number of merged vertices ranging from 200 to 2000, depending on the object. The value x was determined empirically. In this step, unnecessary vertices are eliminated.

Subsurf Modifier: Once the edges and the number of vertices have been optimized, the surfaces must be refined. Blender offers the "Subdivision Surface Modifier" which divides the mesh surfaces into smaller surfaces. This also helps to create a smooth look with a mesh that has a small amount of vertices. In addition, a large amount of data can be stored. To reduce the calculation time, a subsurf level of one is used. This modifier is crucial for the success of the following deformation step.

Displace Modifier: The "displace modifier" is used to manipulate the surface of the object. A "DISTORTED NOISE" texture is applied to the object and the parameters noise_scale and distortion are set to 0.7. The result is a slightly distorted surface that looks more realistic. This modifier is only applied to the previously selected vertices.

Lattice Modifier: Next, a "lattice modifier" is inserted. The lattice object serves as a support for the base object. The deformation of the lattice object has a direct effect on the base object. This approach is particularly advantageous for objects with a large number of vertices. The lattice modifier does not change the texture of the mesh surface.

Armatures: Blender's armature system is modeled on the human body. It consists of several bones that can be manipulated to move. All parts connected to these bones move in the same way, allowing the mesh to be deformed based on the armature. Between 8 and 14 armatures are connected and used in the deformation pipeline. The upper end of each armature is connected to the lower end of the next. A bone is created and connected for each armature. If a bone moves, all other parts of the chain are affected. In combination with the lattice object, the entire object can be deformed.

Bones: The skeletal structure of the body influences the shape of the mesh by changing its position. The bones are an integral part of the skeleton and change the mesh depending on their position.

4.3 Generate Training Data

The following section of the thesis deals with the generation of training data. First, a scene with objects is created in Blenderproc in which only one of the four degrees of deformation is present. This scene consists of one plane, seven objects from the Deformable Object Dataset and six deflection objects. Blender's physical engine is used to place the objects randomly, with a drop point defined in a specific area and the final position depending on the existing scene objects and the object shape itself. The area has the shape of a box with a base surrounded by walls, the size of which is determined by the maximum camera radius parameter. Two random objects from each of the TLESS,

TYOL and YCBV datasets are used for the distractor objects, and the texture of the base and walls is also chosen randomly. This is done to avoid dependencies, e.g. on the background or the positioning of the scene objects.
Each block of training images contains 20,000 RGB images for each degree of deformation. These images are divided into 800 different scenes, with 25 images per scene from different camera angles. Figure 4.4 shows a complete scene. This means that each deformation submodel occurs on average eight times. In addition to the camera angle, the light source of the scene also varies from scene to scene, which leads to more realistic rendering results. RGB images, depth images and segmentation maps are saved for each scene. In addition, the mask and UV mask are also saved for each object in the deformation dataset in the scene. In addition, all information about the camera and the object position is stored in another file so that the scene can be reproduced later.
The structure of the training data is identical to that of the BOP challenge so that the pose estimation algorithm can be exchanged.

To sum up the resulting dataset consists of:

- 80,000 RGB images
- 80,000 depth images
- 80,000 segmentation map images
- 560,000 mask images
- 560,000 UV mask images
- 80 scene files

The number of mask images and UV mask images can be reduced if an object is not in the camera's field of view. An example of a scene is shown in Figure 4.5.

These files form the basis for the next two steps. First, an object detector such as Mask-RCNN must be trained on the objects. Then the pose estimator must be trained.

4.4 Training Procedure

This section explains the training procedure for the object detector (4.4.1) and the pose estimator (4.4.2). In the first step, the training data must be prepared for the training process. After adjusting the parameters and defining the training epochs, the process can be carried out.

4.4.1 Train Object Detector

To train Mask-RCNN, a modern object detector created with Python3, Keras and TensorFlow, the training data from the previous section must be modified. For each scene,

Figure 4.4: Scene with training images from different camera views

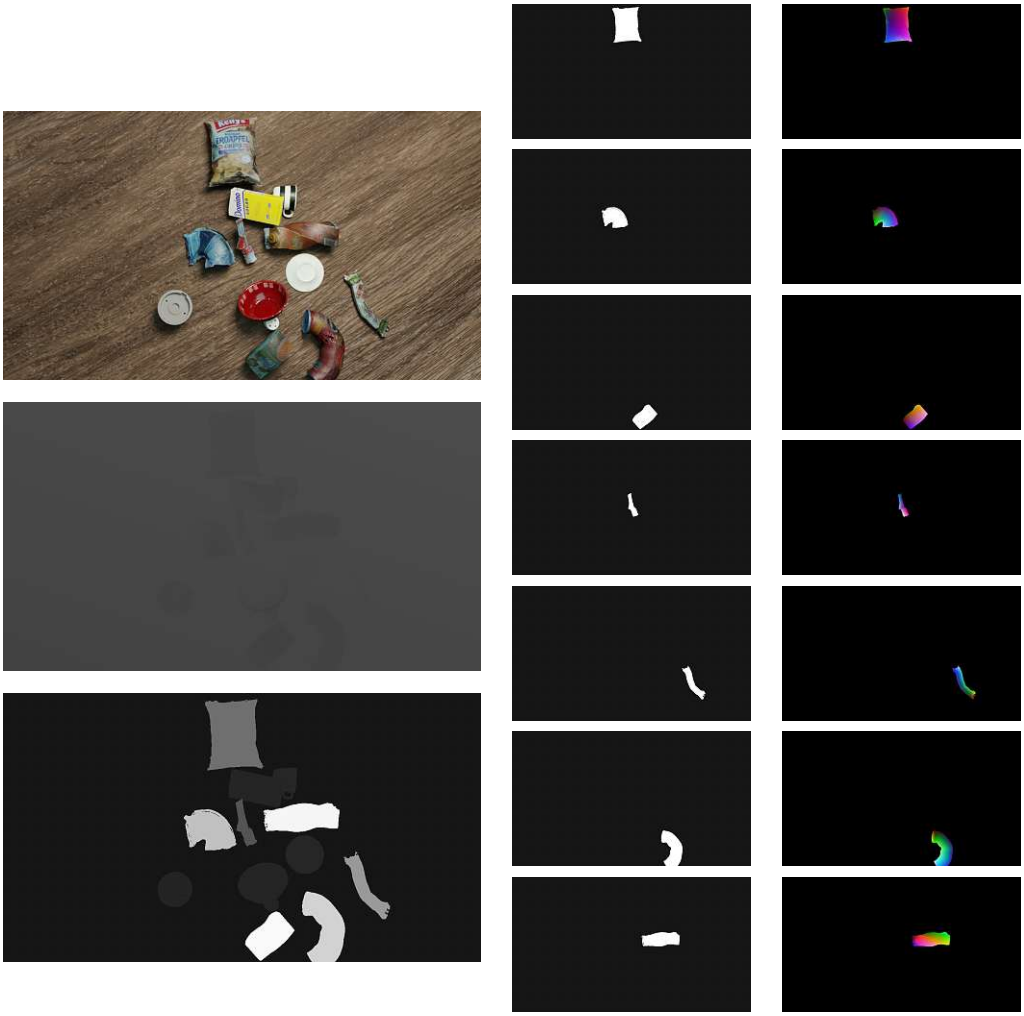


Figure 4.5: Training image with RGB, depth, segmentation map and masks

the objects are extracted from the RGB image of the scene based on the segmentation mask. To increase the amount of training data, a simple data augmentation step is used. The extracted image parts are rotated by a random value between -180 and 180 degrees and an occlusion is added. In addition, the size of the object in the image is manipulated. This step makes it possible to increase the amount of data with minimal effort. After this step is completed, the data is ready for the training process.
Mask-RCNN offers a wide range of settings to improve the training process. However, the memory requirements of the GPU are quite high: 12 GB are needed to train an image with 1280 x 1280 pixels. As this is not always available, the process must be optimized. The parameters in Table 4.2 are used for this purpose.

Parameter	**Value**	**Explanation**
BACKBONE	resnet50	Backbone network architecture
GPU_COUNT	1	NUMBER OF GPUs to use
IMAGES_PER_GPU	1	Number of images to train with on each GPU. Adjust based on your GPU memory and image sizes. Use the highest number that your GPU can handle for best performance
RPN_ANCHOR_SCALES	(16, 32, 64, 128, 256)	Length of square anchor side in pixels
TRAIN_ROIS_PER_IMAGE	32	Number of ROIs (Region of Interest) per image to feed to classifier/mask heads.
STEPS_PER_EPOCH	50000 / IMAGES_PER_GPU	Number of training steps per epoch
VALIDATION_STEPS	5	Number of validation steps to run at the end of every training epoch. A bigger number improves accuracy of validation stats, but slows down the training.
DETECTION_MIN_CONFIDENCE	0.5	Minimum probability value to accept a detected instance. ROIs below this threshold are skipped
IMAGE_MAX_DIM	800 px	Maximum image dimension
IMAGE_MIN_DIM	800 px	Minimum image dimension

Table 4.2: Mask-RCNN config parameters

This configuration of the Mask RCNN object detector is trained over five epochs with five or more layers. The result is a .h5 file containing the calculated weights. This file can then be used by the object detector implementation, e.g. on a robot.

4.4.2 Train Pose Estimator

To train the pose estimator, similar data as before must be generated. However, instead of storing all objects in one image, each object is stored separately, as shown in Figure 4.5 on the right column. The RGB image is then cropped to the dimensions of the mask and the resulting snippet is saved together with the UV mask in an .npy file. To expand the data, the snippet is rotated clockwise every 30 degrees, increasing the training dataset by a factor of twelve.
The training process can be performed with these files, creating a file with the corresponding weights. The next step is to convert this file into an inference file, which is the input for the pose estimator. For each of the seven elements there is an inference file.
Pix2Pose was used as the pose estimator, and the algorithm from the original work was modified for better results. The cut and paste on a random background was eliminated and a cutout with the scene background and an enlarged bounding box was used instead. This was possible because the backgrounds in the scenes were already taken from a random background dataset, making them as unbiased as possible. In addition, the occlusion value was removed as the scenes always contained a number of objects, resulting in a natural occlusion. Since the training was performed on a single GPU, the process of loading PLY files was also restructured. Instead of loading them all at the beginning, which requires a large amount of memory, the PLY files were only loaded when needed. A probability value has been added to the original code to improve the augmentation tools for processing the input images. This value is used in the pose estimator, which includes the following:

- CoarseDropout
- GaussianBlur
- Add
- Invert
- Multiply
- LinearContrast

It is important that the training images reflect the various factors that can influence an image, such as lighting conditions, camera resolution, movement, focus, etc. To weight the tools, the augmentation probability value was used, which ranges from zero to one. A value of zero means that the original image was used, while a value of one means that the entire range of the filter is applied to the image. For example, if an “Add filter“ with a range of -25 to 25 is used on all three channels (RGB) with an expansion probability

of 1, a value between -25 and 25 can be added to each color. Not all filters are used in every iteration, which allows for different results.
Table 4.3 show the training parameters of Pix2Pose.

Parameter	Value	Explanation
backbone	resnet50	Backbone network architecture
outlier_th	[0.15, 0.25, 0.35]	Threshold value for outliers (evaluation)
inlier_th	0.15	Threshold value for inliers (evaluation)
score_type	2	score methode (evaluation)
task_type	2	task type, e.g. BOP, ViVo (evaluation)
cand_factor	2	candidate factor (evaluation)
model_scale	1	model scale factor to get model object dimensions in m
aug_prob	[0.2, 0.4, 0.6, 0.8, 1]	augmentation probability factor (training)

Table 4.3: Pix2Pose config parameters

5 Experiments

This chapter primarily deals with the evaluation of the object detector (5.1) and the pose estimator (5.2). First, different training results are compared with adjustable training parameters. The object pose is then evaluated using the Deformable Object Dataset (3.1). Finally, the determined pose is tested and evaluated using the rendered dataset.

5.1 Evaluate Object Detector

The two-stage deformation pipeline consists of an object detector and an object pose estimator. Unfortunately, the results of the trained object detector were not satisfactory, as it often segmented the wrong objects or assigned them incorrectly. Since the pose estimator, Pix2Pose, only requires the region of interest as a bounding box, the object detector part was removed and the bounding box information was obtained from the simulation data.
To ensure that MaskRCNN was not the only one having difficulties with the deformed dataset, YoloX was also trained. The results were slightly better than those of MaskRCNN, but still not satisfactory. In contrast to MaskRCNN, the implementation of YoloX is much newer and has a higher performance. The training data for YoloX and MaskRCNN are almost identical. Both require RGB images with texture and the mask as input. The training showed that the data got better and better with each epoch. After more than 50 epochs, the results were still not good enough to be used as input for Pix2Pose. It is possible that with 100 to 200 epochs the results will become stable enough for the pose estimator, but there was not enough time in the experiments to test this. Since the focus of this master thesis is on pose estimation, the object detector part was not used.

5.2 Evaluate Pose Estimator

Common evaluation metrics such as mean loss (5.2.1) and translation and rotation error (5.2.2) are used to evaluate the pose estimator. To explain the deviation of the pose, the UV prediction error (5.2.3) is also important. And finally, there is a more detailed consideration of stable areas (5.2.4).

5.2.1 Mean Loss

At the beginning of the training process for the pose estimator, the Deformable Object Dataset was divided into three parts: 80 % for training, 10 % for evaluation and 10 %

for testing. To increase the diversity of the dataset and reduce bias, an augmentation probability factor was introduced during training. This augmentation probability factor was added to the original training procedure. The training method required several iterations, referred to as epochs, to improve the results. A loss function was used to compare the results. The purpose of a loss function is to map a high-dimensional problem to a single comparable value, usually represented as a real number. Figure 5.1 and Table 5.1 show that the mean loss decreases from epoch to epoch during training, which is the desired result. This shows that the results improve with each iteration. After the tenth epoch, the improvement becomes linear with only a slight increase. This indicates that with a large number of objects, training can be completed after ten epochs, halving the training time with minimal impact on performance. It can also be observed that a smaller augmentation probability has a positive effect on the resulting mean loss of the dataset. The loss function used is the binary cross entropy, which calculates the cross entropy between the real classification and the predicted classification. As the name suggests, the binary cross entropy has only two conditions, in contrast to the categorical cross entropy. The binary cross entropy falls under the category of probabilistic losses. The optimization discriminator used is an implementation of the Adam algorithm [57], which estimates first and second order moments based on the stochastic gradient descent method. The advantage of the Adam algorithm lies in its efficient implementation, which is invariant to gradient rescaling and optimized for applications with large amounts of data. In addition, the memory requirement is low.

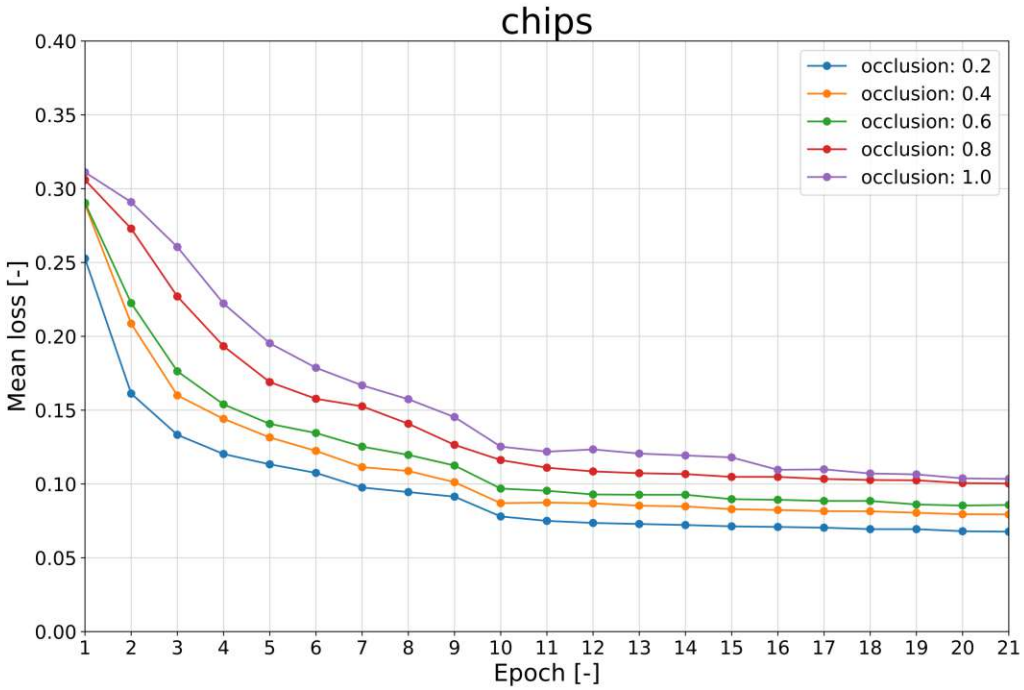


Figure 5.1: Chips training result at different augmentation probabilities

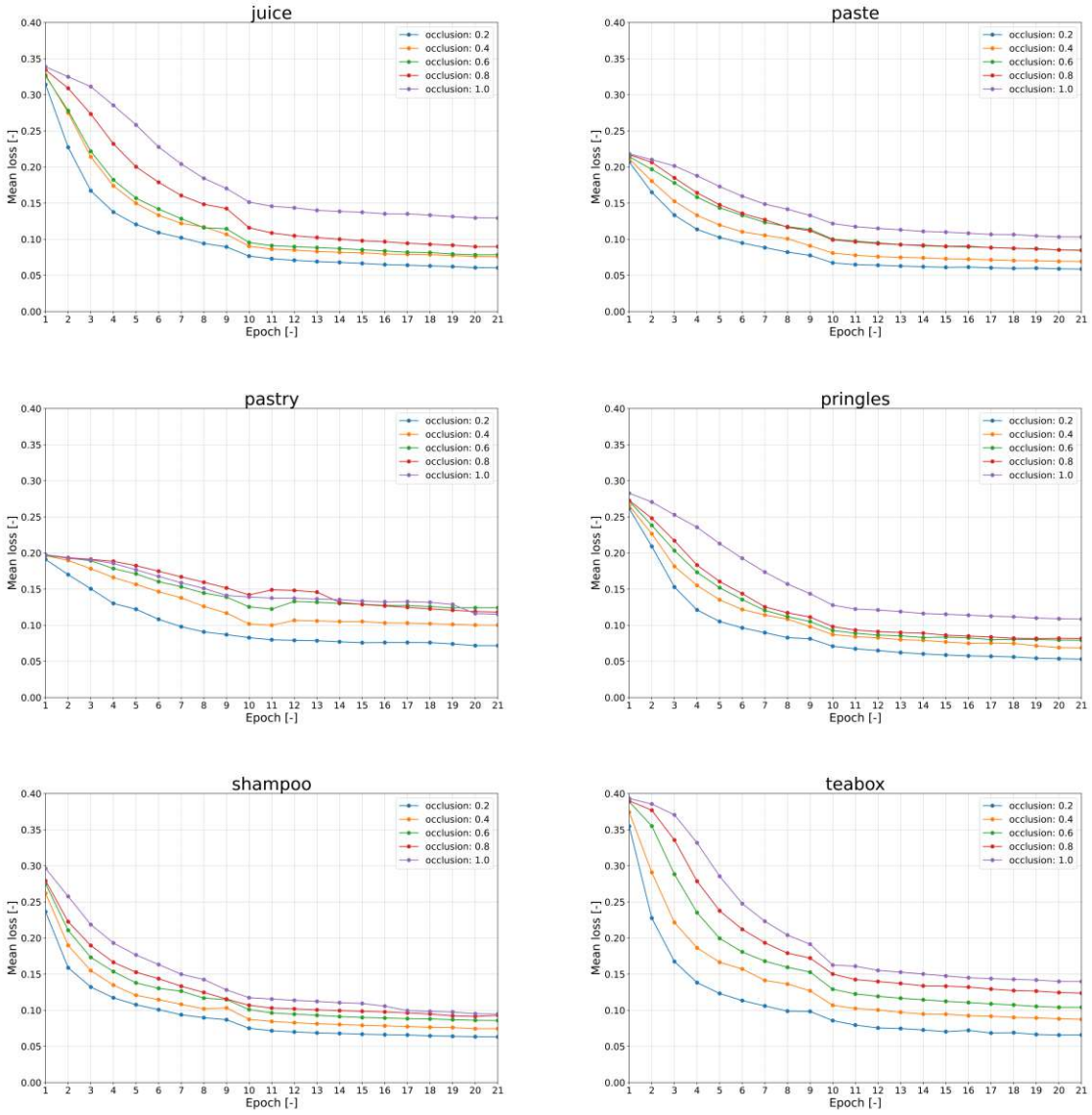


Table 5.1: Dataset object training result at different augmentation probabilities

5.2.2 Translation and Rotation Error

This section deals with the evaluation of the quality of the estimated property location. For this purpose, a suitable evaluation method is required. The simplest method is to measure the translation and rotation error between the estimated and the rendered pose. This method has two requirements: First, the rendered pose must be known, and second, the coordinate frame must be aligned between the original and the deformed object. Firstly, attention is paid to the issue of the coordinate frame. During the design phase of the deformation pipeline, the world coordinate frame was placed at the center of the bottom plane of the object. This positioning ensures that the deformation process, as described in Section 4.2, does not change this frame. Consequently, a comparison can be made between the estimated pose and the rendered pose. However, since the rendered pose is only available from synthetic data, no real data can be used at this stage. In practical applications, the objects would have to be annotated using a tool, which is more challenging due to the unavailability of the deformed object model. For the training of the pose estimator, a split into 80 % training data, 10 % evaluation data and 10 % test data was used. To calculate the translation and rotation error, the rotation and translation matrices of the estimated and rendered system are required. In both cases, these matrices represent the transformation from the model to the camera. Equation 5.1 illustrates the resulting matrix.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{5.1}$$

The following step involves computing the translation error in Equation 5.2 and the rotation error in Equation 5.3.

$$e_t = \sum_{i=1}^{n=3} \frac{(t_{1,i} - t_{2,i})}{n} \tag{5.2}$$

$$e_R = \left\| R_1 \cdot R_2^T \right\|_2^2 \tag{5.3}$$

Figure 5.2 shows the translation errors for each object in all scenes. The boxplot provides several important pieces of information. First, the box shows 50 % of all deviation errors. The bottom line of the box denotes the first quartile, which is 25 % or the median of the bottom half of the data. The top line of the box indicates the third quartile or 75 %. The lower and upper lines represent the minimum and maximum error without possible outliers. In addition, the blue line shows the median of the total errors. The green dashed line shows the arithmetic mean. It can be seen that the median error for all objects is around 10 cm. As a rough estimate, the 10 cm offset is useful, but in most applications the value is too high. For example, gripping will result in a failed grip. The distance between the mean and median value is unusually high. For example, for the object “chips“ the mean value is out of range. This happens when a number of values are at the

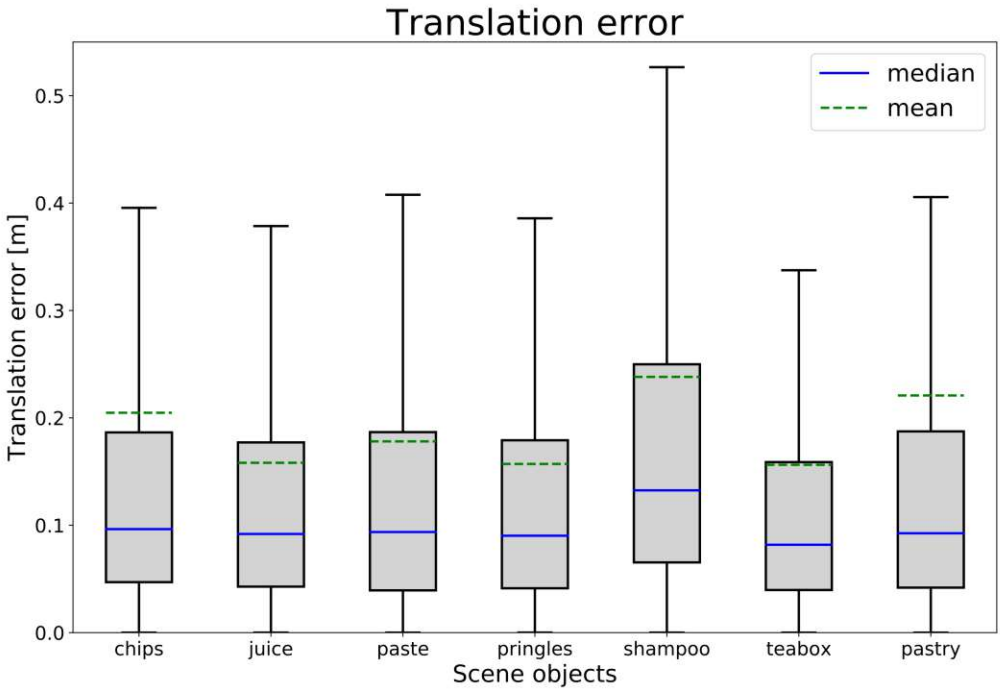


Figure 5.2: Translation error statistical evaluation without score filter

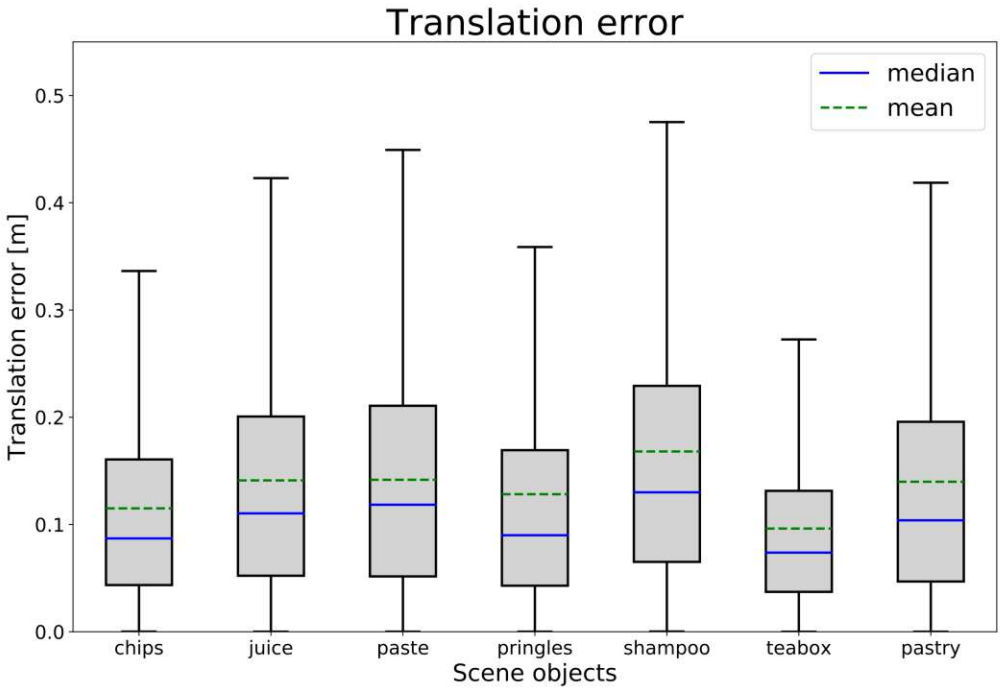


Figure 5.3: Translation error statistical evaluation with score filter

upper end of the scale. This means that there are some results where the translation error is 30 cm or more. The predicted score is also calculated using the translation and rotation matrix. This score is intended to indicate whether a result is good or bad. In theory, the score is higher for a good prediction and lower for a bad one. Therefore, predictions that lie on the upper side of the boxplot should be filtered out using this score. In Figure 5.2, all score values are allowed, which means that no filter was used. If we use the median score from Figure 5.6 as a filter and only allow higher scores, the results will be slightly better. This is shown in Figure 5.3. The median and the box do not move very well, as most values are not filtered. It is noticeable that the mean value has decreased significantly. The dimension of the box is also smaller. Another expected behavior is that the maximum translation error decreases for all objects.

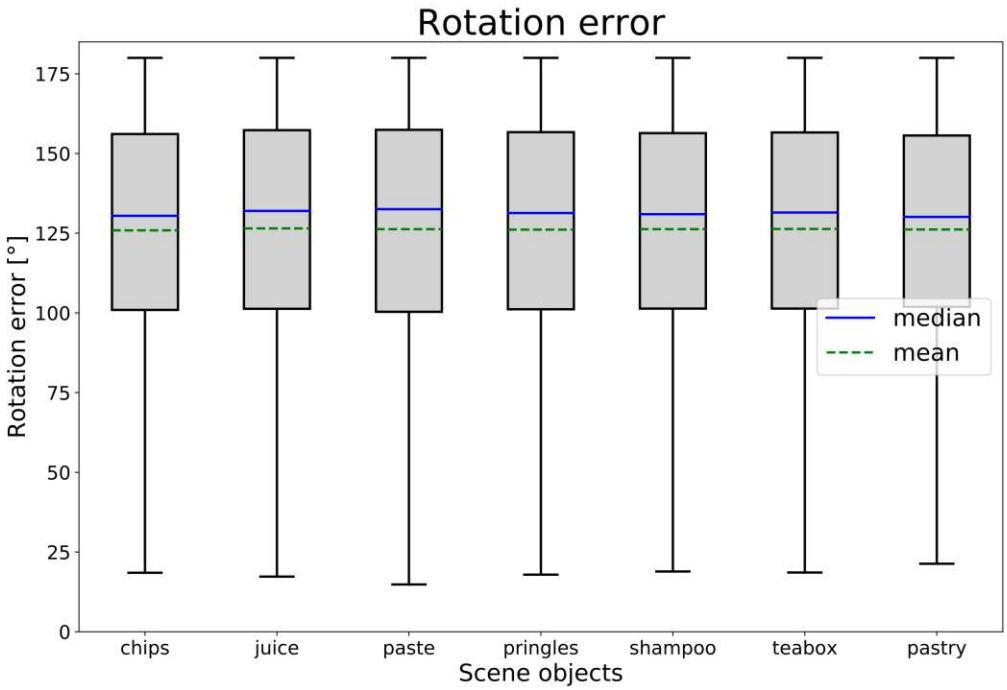


Figure 5.4: Rotation error statistical evaluation without score filter

The calculation of the rotation error for the object was also carried out in Figure 5.4. The boxplot shows that the accuracy of the object rotation is not very high. The most optimal object has a constant rotation error of about 125 °. Interestingly, the median value as well as the first and third quartiles for each object are in a similar range, indicating that there is a constant error during the training or data modeling process. Furthermore, the maximum value consistently reaches 180 °. It is noteworthy that the mean value is lower than the median value, suggesting that smaller rotation errors are rare. Using the same scoring filter that was used for the translation error did not lead to different results (see Figure 5.5). This indicates that a constant rotation error was introduced during the training phase. A possible solution for real data would be to consistently add a rotation offset to eliminate the error.

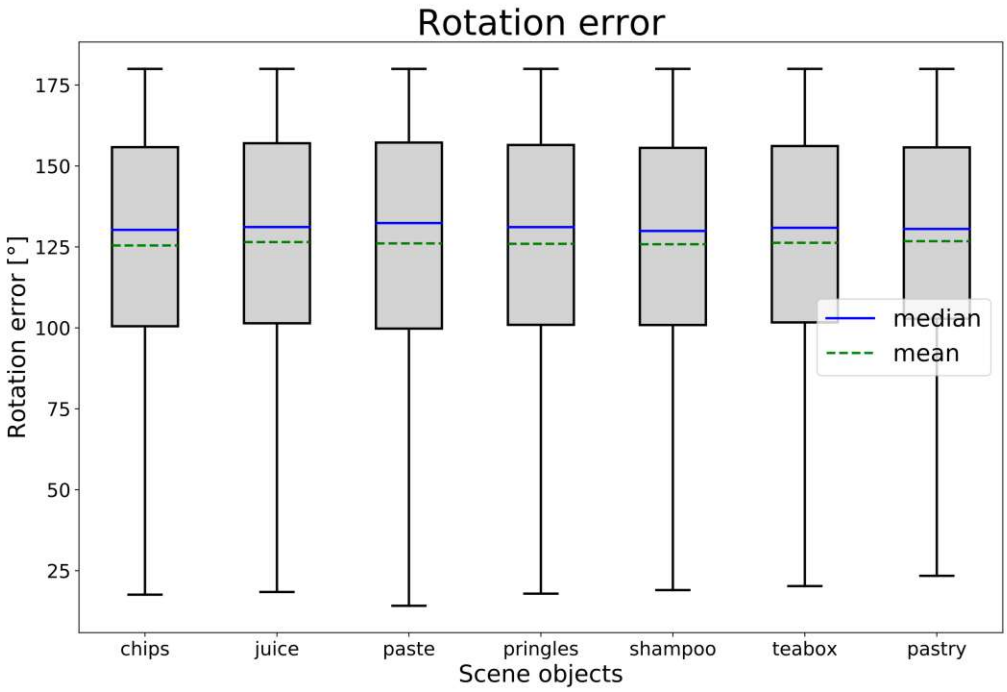


Figure 5.5: Rotation error statistical evaluation with score filter

The Pix2Pose score is currently being investigated. It is not easy to interpret. In general, the object detector and the object pose estimator affect the score. The formula for the score is shown in Equation 5.4. The final Pix2Pose score is calculated by multiplying the object detector score (s_d), the ratio of outlier to mask of the object pose estimator (s_i), the intersection between the object detector and the predicted mask of the object pose estimator (s_m) and a factor of 1,000. Equation 5.4 describes the final prediction score, s_d the object detector score, s_i the pose estimator score, s_m the mask score, m_i the number of outliers in the predicted mask from the pose estimator, m_p the total mask from the pose estimator and m_d the object detector mask.

$$\begin{aligned} s &= s_d \cdot s_i \cdot s_m \cdot 1000 \\ s_i &= \frac{m_i}{m_p} \\ s_m &= \frac{m_d \cap m_p}{m_d \cup m_p} \end{aligned} \tag{5.4}$$

Figure 5.6 shows the score for each object. The “pringles“ has the highest score, which is probably due to its almost rectangular shape in the image. The score is strongly influenced by the given mask. Since the results of a trained object detector were not useful, bounding boxes were used instead. These bounding boxes contain not only the mask of the object, but also many background pixels. Equation 5.4 shows that the intersection between the mask of the object detector and the mask of the pose estimator is used. The more background pixels are contained in the mask, the lower the score,

which is particularly noticeable for small objects such as the paste and the pastry. To improve the scoring, a segmented mask can be used instead of a bounding box mask. The bounding box mask is sufficient for the project, as the pose estimator Pix2Pose requires it as input. As a higher score indicates a better estimate, a filter can be developed on the basis of this result. For example, all estimates that are below the median value could be discarded. The diagram also shows that not all objects have the same score. The score of the "chips" is between 0 and 5,000 with a median value of 1,000. In contrast, the pastry have a score range of 0 to 600 with a median value of 200. In an application, the score value must therefore be determined empirically. Another approach is to use the training data and evaluate the results as here and then use the median or the third quartile limit as a filter.

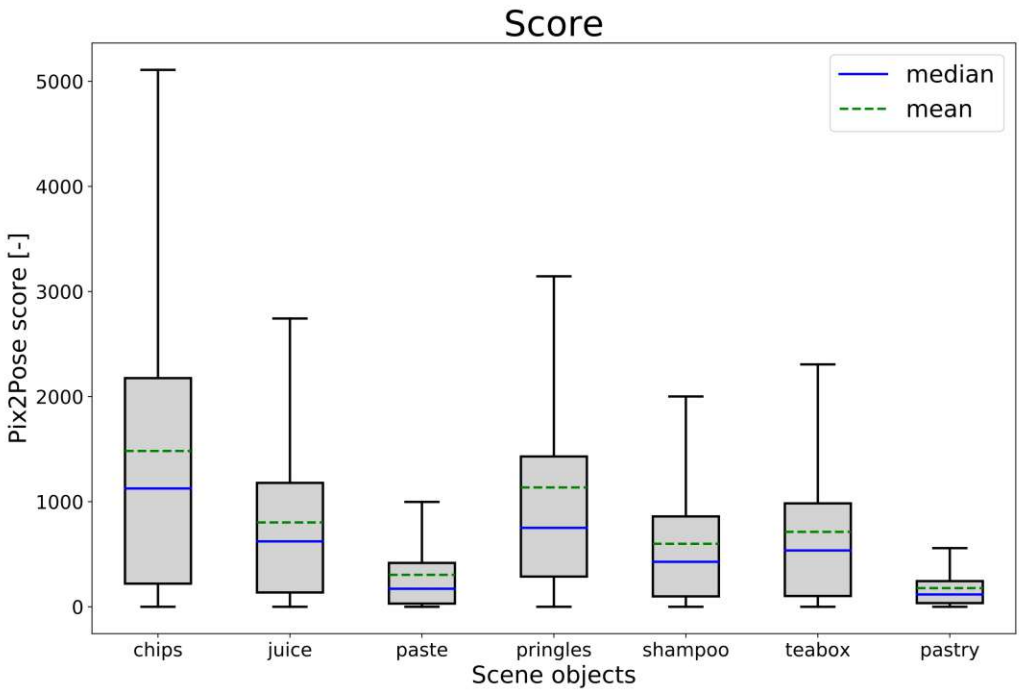


Figure 5.6: Pix2Pose score statistical evaluation

5.2.3 UV-Prediction Error

This section discusses the UV prediction error with Pix2Pose. The UV prediction is represented by an RGB color code, where a value of (255, 0, 0) represents the point (x_{max}, 0, 0), (0, y_{max}, 0) by (0, 255, 0), and (0, 0, z_{max}) by (0, 0, 255). This representation enables the estimation of objects without texture. As the RGB value is correlated with the object dimensions, the prediction should be as accurate as possible. The Section 4.2 contains a more detailed description of the process. When a pose is estimated with

Pix2Pose, the predicted mask and the predicted image section are returned. Together with the Region of Interest (ROI), this enables the extraction of the UV prediction for the object. In the next step, the rendered data is used to compare the results. The metric used is Equation 5.5, where e_X is the error for the specific color or coordinate, X_p is the predicted color value, X_r is the rendered color value and n is the number of predictions in the given scene. The average error over all color values is represented by e_{RGB}.

$$\begin{aligned} e_R &= \frac{1}{n} \cdot \sqrt{\sum_{i=1}^{n} R_{p,i} - R_{r,i}} \\ e_G &= \frac{1}{n} \cdot \sqrt{\sum_{i=1}^{n} G_{p,i} - G_{r,i}} \\ e_B &= \frac{1}{n} \cdot \sqrt{\sum_{i=1}^{n} B_{p,i} - B_{r,i}} \\ e_{RGB} &= \frac{1}{n} \cdot (e_R + e_G + e_B) \end{aligned} \tag{5.5}$$

Figure 5.7 shows the error in the UV prediction for the object “chips“. Since the discussion applies to all objects shown in Figure 5.7 - 5.13, it is conducted specifically for the “chips“ object. The boxplot illustrates that the approximation to the Z value provides the best results. The median of the blue value is around 85 and the spread of the values is comparable across all color channels. In this diagram, a value of zero represents a perfect match, while 255 represents the worst value. The proximity of the median and mean value indicates that the values are evenly distributed around a constant value of around 80-90. As mentioned earlier, a rotational and translational error was identified in the previous section. This error is clearly visible in the RGB values. However, this is to be expected as it forms the basis for the estimation of the posture itself. To find an explanation for this deviation, knowledge of the actual object dimensions would be helpful. The dimensions of the “chips“ object are approximately [25 cm, 10 cm, 38 cm]. Another interesting observation is that all objects have a similar boxplot. This uniformity in the plots is beneficial for future research as it suggests that any errors can be minimized through improvements in the estimation pipeline. However, it should be noted that the current values from the estimation process are not as accurate as desired.

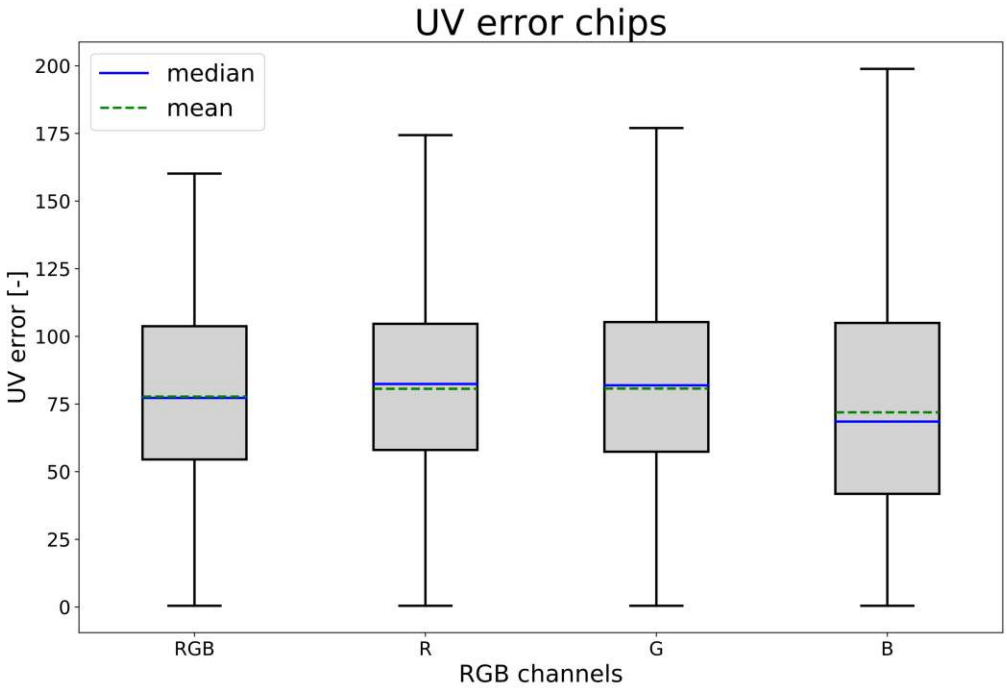


Figure 5.7: UV-Prediction error for the object chips

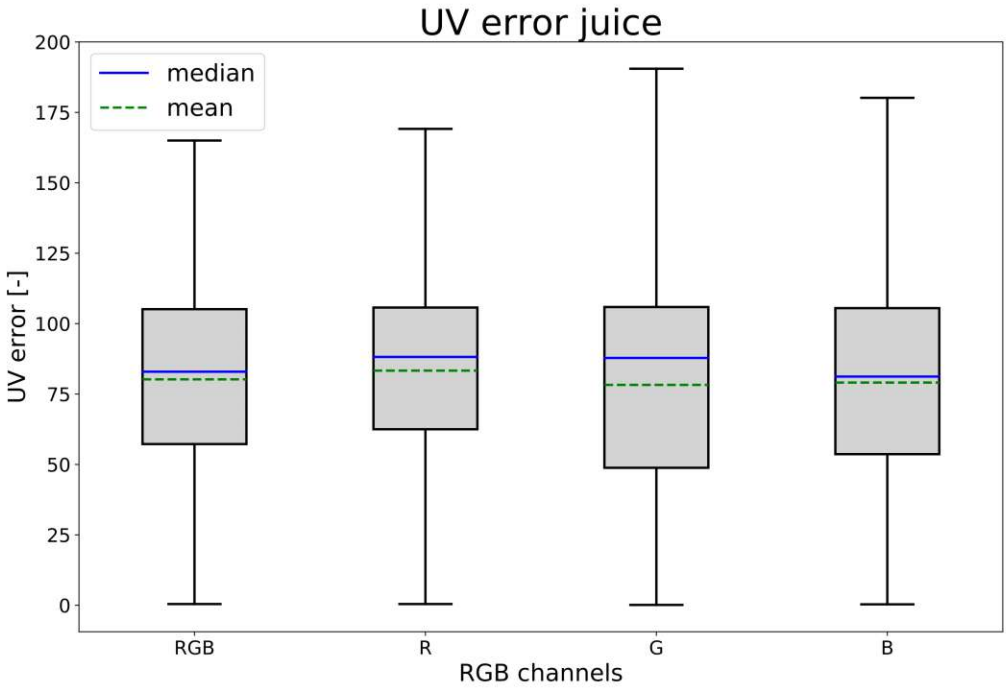


Figure 5.8: UV-Prediction error for the object juice

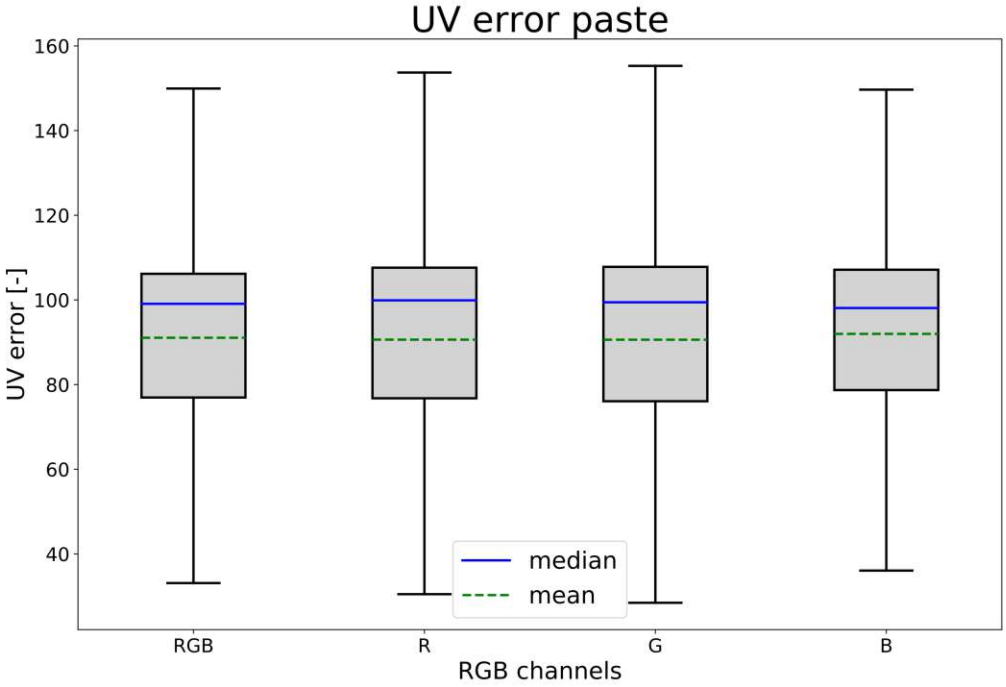


Figure 5.9: UV-Prediction error for the object paste

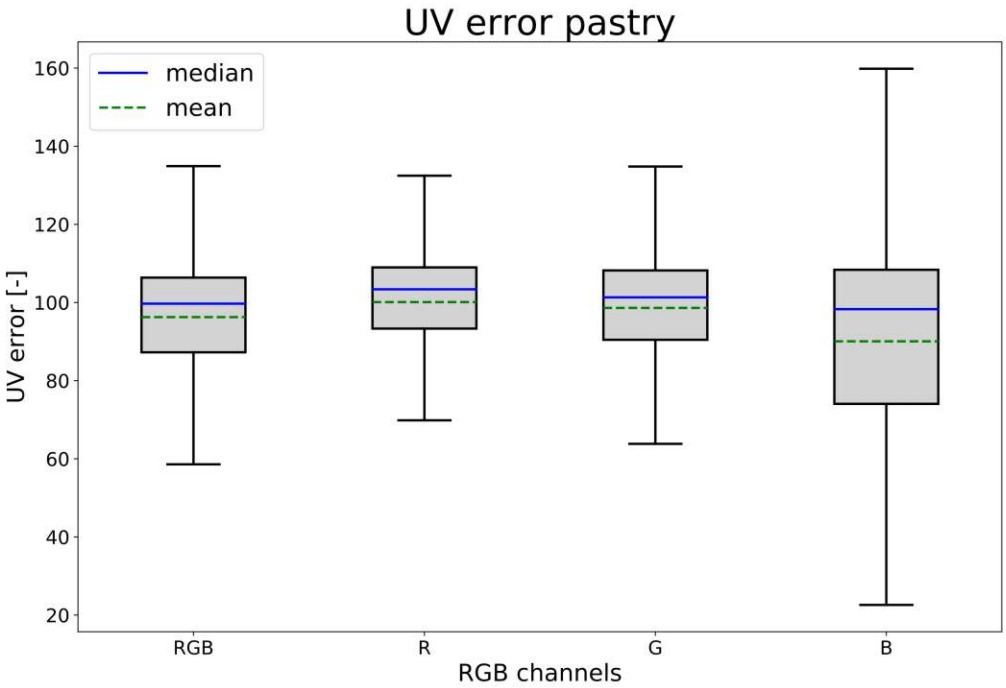


Figure 5.10: UV-Prediction error for the object pastry

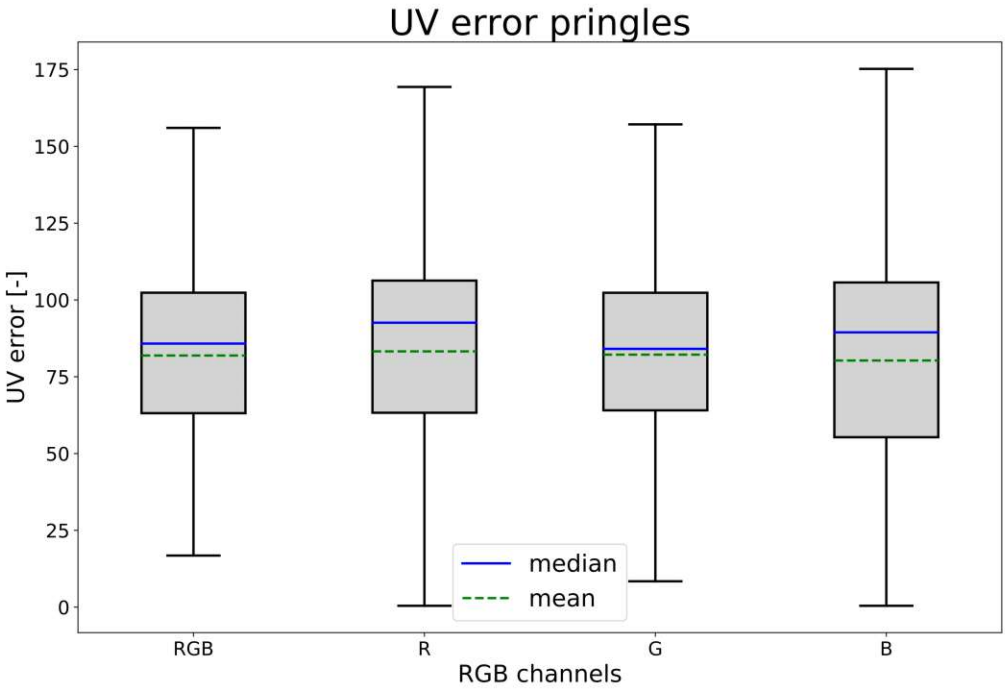


Figure 5.11: UV-Prediction error for the object pringles

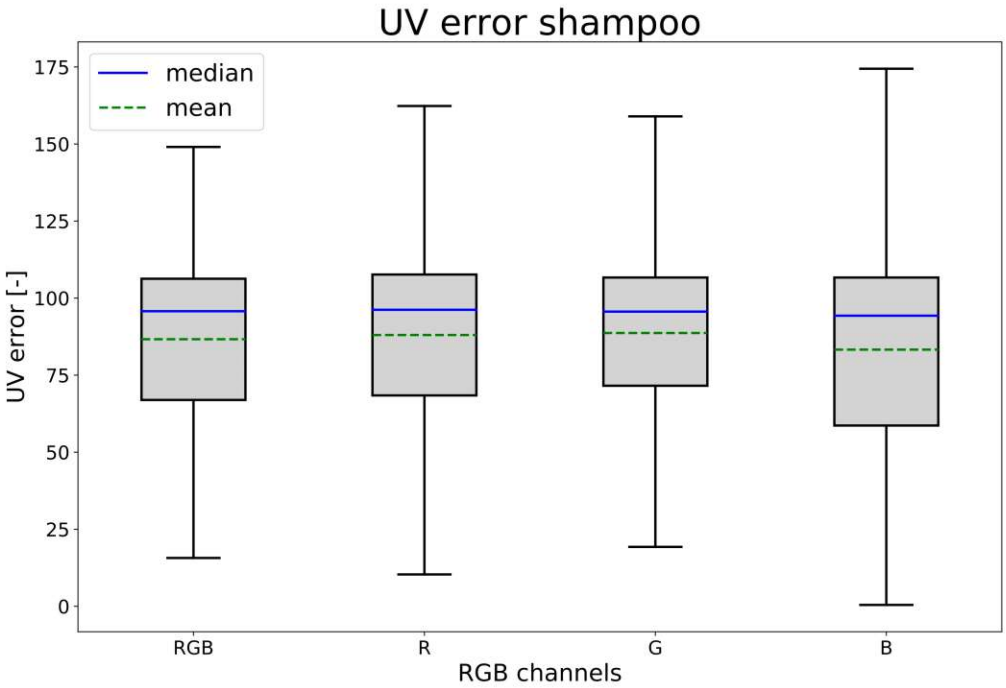


Figure 5.12: UV-Prediction error for the object shampoo

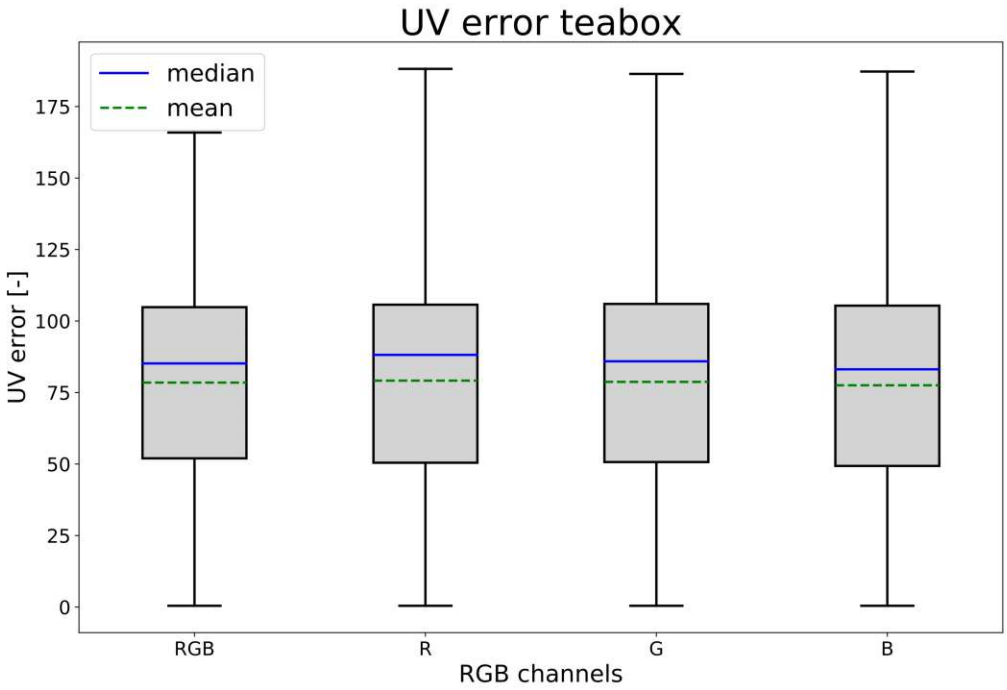


Figure 5.13: UV-Prediction error for the object teabox

Previously, a translation error of around 10 cm was determined for the object chips. To compare the results, we can now use the information from Figures 5.7 and 5.2 with the correct dimensions of the object. The expected translation error for the "chips" object is also around 10 cm. Using Equation 5.6, it can be determined that the total translation error is 12.51 cm. This value agrees quite well with the observed mean translation error. It is important to note that the calculation was not performed with exact values, but with estimates derived from the given diagrams.

$$\begin{aligned} e_R &= \frac{75}{255} \cdot 25\,\text{cm} = 7{,}35\,\text{cm} \\ e_G &= \frac{75}{255} \cdot 10\,\text{cm} = 2{,}94\,\text{cm} \\ e_B &= \frac{65}{255} \cdot 38\,\text{cm} = 9{,}69\,\text{cm} \\ e_{RGB} &= \sqrt{(e_R^2 + e_G^2 + e_B^2)} = 12{,}51\,\text{cm} \end{aligned} \tag{5.6}$$

5.2.4 Stable Areas

After taking into account the translation, rotation and UV prediction error, the next step is to identify stable regions in the prediction. The results provided are a summary of all predicted values. It may be possible to identify sub-regions where the prediction is more accurate than the rest. For this purpose, a random prediction was selected and a heat map was created. In the heatmap, green indicates a good match, while red

indicates a poor match. Figure 5.14 illustrates the evaluation of the “pringles“ object, specifically scene 19 image 0. The figure shows a part of the original image with the mask, the predicted and the rendered UV mask, the difference between them and the heatmap. The heatmap shows areas of both good and poor prediction accuracy. Most of the object appears light green, indicating that the prediction, while not perfect, is reasonably accurate. However, it is worth noting that the prediction is not very accurate overall, as almost half of the object is not green. This evaluation is made possible by the availability of the rendered UV data. Without this data, only the UV-colored original model could be used for comparison, making it difficult to identify stable regions in the object. One approach to overcome this challenge could be to divide the prediction into smaller clusters and search for corresponding regions in the original object.

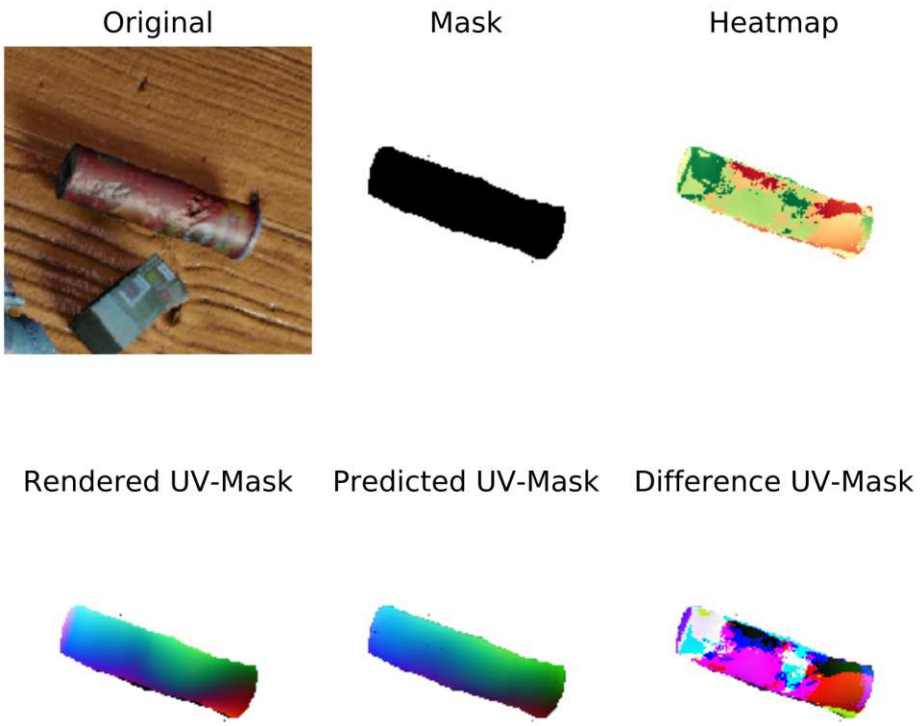


Figure 5.14: Heatmap of pringles sample

6 Conclusion

This thesis focuses on training and testing a state-of-the-art pose estimation algorithm on a newly created dataset of deformed objects, the Deformable Object Dataset. The following chapter evaluates the strengths and weaknesses of the proposed approach and also discusses possible future work. The discussion is divided into two main topics: the dataset with deformed objects (6.1) and the pose estimation results (6.2).

6.1 Deformed Object Dataset (DOD)

The dataset forms the basis for each trained method. However, as the area of deformed objects is relatively new, there was no suitable dataset. For this reason, a new dataset was created. The Simple Deformed Object Dataset consists of seven main objects (chips, juice, paste, pastry, pringles, shampoo, teabox) in 3,200 scenes. Each scene contains seven objects from the dataset, six random distractor objects and a randomly selected unbiased background. The objects were specifically selected based on their shape, deformability and availability. For each scene, there are 25 different views. In total, the dataset includes 80,000 RGB images, 80,000 depth images, 80,000 segmentation masks, 560,000 mask images and 560,000 UV mask images.
The study included four different models of each object, including the original object and three real deformations. In addition, 100 additional deformations were created through software manipulation, resulting in a total of 404 unique deformations per object. The 3D model files contain RGB and UV texture information. The deformation algorithm used in the study is implemented as open-source Python code, which can be adapted for future applications in conjunction with the open-source software Blender. In this work, a simple approach based on a skeleton consisting of bones was used. To increase realism, material properties, surface stiffness and external forces can be included.
To facilitate future use, the structure of the dataset corresponds to the specifications of the BOP challenge. This allows for easy integration of the dataset into both new and existing projects, with minimal customization required. Due to the large volume of data, it is advisable to read the PLY files individually and not all at once.

6.2 Pose Estimation

An object pose estimator called Pix2Pose was used for the evaluation. This pose estimator was chosen for several reasons. First, it is capable of working with objects that do not have texture. Also, UV prediction masks are integrated into the methodology. Finally,

the supporting team has a wealth of knowledge and experience in using this particular pose estimator. The training process was based on feature matching between the base object and different deformations. The UV prediction masks played a crucial role in establishing a unique connection due to their different color codes. As a result, the pose estimator was able to learn the deformations effectively.
The results show that the UV prediction leads to an error across all color channels. This error subsequently leads to translation and rotation errors. The translation error for each object is about 10 cm, while the mean weighted rotation error is about 125 degrees. It is possible to reduce the error by using the predicted score. However, since the score is different for each object, there is no universal score filter. Furthermore, the evaluation shows a correlation between the UV prediction and the translation and rotation errors. This indicates that the pose estimation process is theoretically feasible if the UV prediction is sufficiently accurate. Another notable observation is that a smaller augmentation probability factor improves the mean loss or quality of the training process. One possible explanation for the error in UV prediction could be due to the use of the entire dataset for training. In particular, there were three actual deformed recorded models, and the UV mask did not match the real objects in these models. The intention behind this approach was to evaluate different levels of deformation, ranging from simple to severe. However, due to the unexpected challenges encountered during the training process, this assessment was ultimately discarded.
Another aspect of the work was the requirement that the pose estimator should be compatible with common hardware. In particular, it should be ensured that the pose estimator can run on a laptop connected to the laboratory robot. This laptop was used throughout the training and evaluation process, demonstrating its potential for future use. The object mask or the Region of Interest (ROI) is a typical input for any pose estimation process. In this work, two different object detectors, namely MaskRCNN and YoloX, were trained on the dataset. However, both proved to be unusable due to incorrect segmentation and labeling. The results indicate that either not enough training epochs were performed or the object detector was not able to process deformed objects. As the main focus of this work is on the object pose estimator, the cause of this problem was not investigated further.

6.3 Future Work

This section discusses possible future improvements for the pose estimator. To improve the accuracy of the score predicted by the pose estimator, it is recommended to incorporate a reliable object detector. Instead of using the ROI to calculate the score, it would be more effective to use the mask generated by the object detector. Another approach could be to use a scene segmenter, where the prediction results would reflect the most appropriate suggestion. However, a disadvantage of this method is that the estimation process would have to be performed for each learned object.
The main problem with the trained pose estimator lies in the inaccuracies of UV prediction.

To get a deeper insight into the UV prediction, the training process can be divided into four different parts corresponding to the different levels of real deformations: light, medium, large and no deformation. This evaluation is expected to provide valuable information.
In addition, the Deformable Object Dataset can be used to train another algorithm for pose estimation. The advantage of the dataset lies in its existing BOP format, which allows the use of all listed methods for pose estimation. This leads to the next improvement. The evaluation of the results is done with knowledge of the generated data. For future work, it would be beneficial to have a metric that evaluates the quality of the pose. Also, an additional method could be developed to draw a 3D bounding box around the deformed object. Currently, this is not possible due to the limited availability of information about the 3D model of the original object.
As already mentioned, the current deformation algorithm offers considerable potential for improvement. Factors such as material properties, surface texture and shape can be used as input for a new algorithm. In addition, the use of physical machines commonly used in video games to capture the deformed object scene by scene could be explored, especially with the increasing computing power available. Once the pose estimation process is sufficiently improved, it would be highly interesting to evaluate the algorithm on real data. The ultimate goal would be to implement the algorithm on a robot for object manipulation tasks, such as grasping. To achieve this, it could be beneficial to identify stable areas on the object, and a possible approach could involve the use of heat maps similar to those discussed in this thesis. In summary, there is considerable potential for future work in this area. The basic steps have been completed in this work, and the next challenge is to improve the results for practical applications on robots.

Bibliography

[1] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, „Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," *CoRR*, vol. abs/1809.10790, 2018. arXiv: 1809.10790. [Online]. Available: http://arxiv.org/abs/1809.10790.

[2] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*, 2018. arXiv: 1711.00199 [cs.CV].

[3] F. Manhardt, W. Kehl, N. Navab, and F. Tombari, *Deep Model-Based 6D Pose Refinement in RGB*, 2018. arXiv: 1810.03065 [cs.CV].

[4] J. Sock, K. I. Kim, C. Sahin, and T.-K. Kim, *Multi-Task Deep Networks for Depth-Based 6D Object Pose and Joint Registration in Crowd Scenarios*, 2018. arXiv: 1806.03891 [cs.CV].

[5] W. Abdulla, *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*, https://github.com/matterport/Mask_RCNN, 2017.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016. arXiv: 1506.02640 [cs.CV].

[7] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. arXiv: 2207.02696 [cs.CV].

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, „SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, pp. 21–37. [Online]. Available: https://doi.org/10.1007%2F978-3-319-46448-0_2.

[9] S. Ren, K. He, R. Girshick, and J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2016. arXiv: 1506.01497 [cs.CV].

[10] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. DollÃ¡r, *Focal Loss for Dense Object Detection*, 2018. arXiv: 1708.02002 [cs.CV].

[11] Z. Tian, C. Shen, H. Chen, and T. He, *FCOS: Fully Convolutional One-Stage Object Detection*, 2019. arXiv: 1904.01355 [cs.CV].

[12] S. Thalhammer, T. Patten, and M. Vincze, *COPE: End-to-end trainable Constant Runtime Object Pose Estimation*, 2022. arXiv: 2208.08807 [cs.CV].

[13] S. Zhang, W. Zhao, Z. Guan, X. Peng, and J. Peng, „Keypoint-Graph-Driven Learning Framework for Object Pose Estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 1065–1073.

[14] M. Rad and V. Lepetit, *BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth*, 2018. arXiv: 1703.10896 [cs.CV].

[15] S. Peng, Y. Liu, Q. Huang, H. Bao, and X. Zhou, *PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation*, 2018. arXiv: 1812.11788 [cs.CV].

[16] Y. Hu, J. Hugonot, P. Fua, and M. Salzmann, *Segmentation-driven 6D Object Pose Estimation*, 2019. arXiv: 1812.02541 [cs.CV].

[17] B. Tekin, S. N. Sinha, and P. Fua, *Real-Time Seamless Single Shot 6D Object Pose Prediction*, 2018. arXiv: 1711.08848 [cs.CV].

[18] T. Hodan, D. Barath, and J. Matas, *EPOS: Estimating 6D Pose of Objects with Symmetries*, 2020. arXiv: 2004.00605 [cs.CV].

[19] Z. Li, G. Wang, and X. Ji, „CDPN: Coordinates-Based Disentangled Pose Network for Real-Time RGB-Based 6-DoF Object Pose Estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 7677–7686.

[20] K. Park, T. Patten, and M. Vincze, „Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2019. [Online]. Available: https://doi.org/10.1109%2Ficcv.2019.00776.

[21] G. Wang, F. Manhardt, F. Tombari, and X. Ji, *GDR-Net: Geometry-Guided Direct Regression Network for Monocular 6D Object Pose Estimation*, 2021. arXiv: 2102.12145 [cs.CV].

[22] Y. Di, F. Manhardt, G. Wang, X. Ji, N. Navab, and F. Tombari, *SO-Pose: Exploiting Self-Occlusion for Direct 6D Pose Estimation*, 2021. arXiv: 2108.08367 [cs.CV].

[23] Y. Hu, P. Fua, and M. Salzmann, *Perspective Flow Aggregation for Data-Limited 6D Object Pose Estimation*, 2022. arXiv: 2203.09836 [cs.CV].

[24] R. L. Haugaard and A. G. Buch, *SurfEmb: Dense and Continuous Correspondence Distributions for Object Pose Estimation with Learnt Surface Embeddings*, 2021. [Online]. Available: https://arxiv.org/abs/2111.13489.

[25] I. Shugurov, F. Li, B. Busam, and S. Ilic, *OSOP: A Multi-Stage One Shot Object Pose Estimation Framework*, 2022. arXiv: 2203.15533 [cs.CV].

[26] L. Huang, T. Hodan, L. Ma, L. Zhang, L. Tran, C. Twigg, P.-C. Wu, J. Yuan, C. Keskin, and R. Wang, *Neural Correspondence Field for Object Pose Estimation*, 2022. arXiv: 2208.00113 [cs.CV].

[27] M. A. Fischler and R. C. Bolles, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, 1987.

[28] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbé, E. Brachmann, F. Michel, C. Rother, and J. Matas, „BOP Challenge 2020 on 6D Object Localization," *European Conference on Computer Vision Workshops (ECCVW)*, 2020.

[29] M. Sundermeyer, T. Hodan, Y. Labbe, G. Wang, E. Brachmann, B. Drost, C. Rother, and J. Matas, *BOP Challenge 2022 on Detection, Segmentation and Pose Estimation of Specific Rigid Objects*, 2023. [Online]. Available: `https://arxiv.org/abs/2302.13075`.

[30] Y. Wu, A. Javaheri, M. Zand, and M. Greenspan, *Keypoint Cascade Voting for Point Cloud Based 6DoF Pose Estimation*, 2022. [Online]. Available: `https://arxiv.org/abs/2210.08123`.

[31] Y. Su, M. Saleh, T. Fetzer, J. Rambach, N. Navab, B. Busam, D. Stricker, and F. Tombari, „ZebraPose: Coarse to Fine Surface Encoding for 6DoF Object Pose Estimation," *arXiv preprint arXiv:2203.09418*, 2022.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, „ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[33] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, „Microsoft COCO: Common Objects in Context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: `1405.0312`. [Online]. Available: `http://arxiv.org/abs/1405.0312`.

[34] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, „Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[35] A. Gupta, P. Dollár, and R. B. Girshick, „LVIS: A Dataset for Large Vocabulary Instance Segmentation," *CoRR*, vol. abs/1908.03195, 2019. arXiv: `1908.03195`. [Online]. Available: `http://arxiv.org/abs/1908.03195`.

[36] T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. Glent Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, and C. Rother, „BOP: Benchmark for 6D Object Pose Estimation," *European Conference on Computer Vision (ECCV)*, 2018.

[37] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, „Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," in *Computer Vision – ACCV 2012*, K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 548–562, ISBN: 978-3-642-37331-2.

[38] E. Brachmann, *6D Object Pose Estimation using 3D Object Coordinates [Data]*, version V1, 2020. [Online]. Available: `https://doi.org/10.11588/data/V4MUMX`.

[39] T. Hodaň, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, „T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[40] B. Drost, M. Ulrich, P. Bergmann, P. HÃ¤rtinger, and C. Steger, „Introducing MVTec ITODD â€" A Dataset for 3D Object Recognition in Industry," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 2200–2208.

[41] R. Kaskman, S. Zakharov, I. Shugurov, and S. Ilic, „HomebrewedDB: RGB-D Dataset for 6D Pose Estimation of 3D Objects," *International Conference on Computer Vision (ICCV) Workshops*, 2019.

[42] S. Tyree, J. Tremblay, T. To, J. Cheng, T. Mosier, J. Smith, and S. Birchfield, „6-DoF Pose Estimation of Household Objects for Robotic Manipulation: An Accessible Dataset and Benchmark," in *International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[43] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, „PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," *CoRR*, vol. abs/1711.00199, 2017. arXiv: 1711.00199. [Online]. Available: http://arxiv.org/abs/1711.00199.

[44] C. Rennie, R. Shome, K. E. Bekris, and A. F. D. Souza, „A Dataset for Improved RGBD-based Object Detection and Pose Estimation for Warehouse Pick-and-Place," *CoRR*, vol. abs/1509.01277, 2015. arXiv: 1509.01277. [Online]. Available: http://arxiv.org/abs/1509.01277.

[45] R. Kouskouridas, A. Tejani, A. Doumanoglou, D. Tang, and T.-K. Kim, „Latent-Class Hough Forests for 6 DoF Object Pose Estimation," *CoRR*, vol. abs/1602.01464, 2016. arXiv: 1602.01464. [Online]. Available: http://arxiv.org/abs/1602.01464.

[46] A. Doumanoglou, R. Kouskouridas, S. Malassiotis, and T.-K. Kim, „6D Object Detection and Next-Best-View Prediction in the Crowd," *CoRR*, vol. abs/1512.07506, 2015. arXiv: 1512.07506. [Online]. Available: http://arxiv.org/abs/1512.07506.

[47] V. E. Arriola-Rios, P. Guler, F. Ficuciello, D. Kragic, B. Siciliano, and J. L. Wyatt, „Modeling of Deformable Objects for Robotic Manipulation: A Tutorial and Review," *Frontiers in Robotics and AI*, vol. 7, 2020, ISSN: 2296-9144. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2020.00082.

[48] H. Delingette, „General Object Reconstruction Based on Simplex Meshes," *International Journal of Computer Vision*, vol. 32, 1999. [Online]. Available: https://doi.org/10.1023/A:1008157432188.

[49] S. I. Nikolenko, „Synthetic Data for Deep Learning," *CoRR*, vol. abs/1909. 11512, 2019. arXiv: 1909.11512. [Online]. Available: http://arxiv.org/abs/1909.11512.

[50] M. Pasieka, *A comparison of synthetic data generation methods and synthetic data types*, 2022. [Online]. Available: `https://mostly.ai/blog/comparison-of-synthetic-data-types`.

[51] OpenAI, *GPT-4 Technical Report*, 2023. arXiv: `2303.08774 [cs.CL]`.

[52] R. Thoppilan, D. D. Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, Y. Zhou, C.-C. Chang, I. Krivokon, W. Rusch, M. Pickett, K. S. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. H. Chi, and Q. Le, „LaMDA: Language Models for Dialog Applications," *CoRR*, vol. abs/2201.08239, 2022. arXiv: `2201.08239`. [Online]. Available: `https://arxiv.org/abs/2201.08239`.

[53] A. Glaese, N. McAleese, M. Trebacz, J. Aslanides, V. Firoiu, T. Ewalds, M. Rauh, L. Weidinger, M. Chadwick, P. Thacker, L. Campbell-Gillingham, J. Uesato, P.-S. Huang, R. Comanescu, F. Yang, A. See, S. Dathathri, R. Greig, C. Chen, D. Fritz, J. S. Elias, R. Green, S. MokrÃ¡, N. Fernando, B. Wu, R. Foley, S. Young, I. Gabriel, W. Isaac, J. Mellor, D. Hassabis, K. Kavukcuoglu, L. A. Hendricks, and G. Irving, *Improving alignment of dialogue agents via targeted human judgements*, 2022. arXiv: `2209.14375 [cs.LG]`.

[54] Q.-Y. Zhou, J. Park, and V. Koltun, „Open3D: A Modern Library for 3D Data Processing," *arXiv:1801.09847*, 2018.

[55] M. Denninger, D. Winkelbauer, M. Sundermeyer, W. Boerdijk, M. Knauer, K. H. Strobl, M. Humt, and R. Triebel, „BlenderProc2: A Procedural Pipeline for Photorealistic Rendering," *Journal of Open Source Software*, vol. 8, no. 82, p. 4901, 2023. [Online]. Available: `https://doi.org/10.21105/joss.04901`.

[56] Toyota Motor Corporation, *Toyota HSR*, 1995-2021. [Online]. Available: `https://global.toyota/en/detail/8709541`.

[57] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: `1412.6980 [cs.LG]`.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Christian Eder, BSc

St. Leonhard am Forst, 25.01.2024