Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Advanced Object Localization Pipeline for Robot Manipulation

Tim Schäfer

Course of Study:	Technische Kybernetik
Examiner:	Dr. Jim Mainprice
Supervisor:	Yoojin Oh, M.Sc.

Commenced:	July 1, 2020
Completed:	December 1, 2020

Abstract

The thesis objective is to develop a robust tracking pipeline for a Baxter robot system. The tracking pipeline includes the applications of robot tracking and object tracking, which localizes the surrounding objects for robot manipulation. By combining, extending, and comparing state-of-the-art approaches for object tracking and image segmentation, this thesis estimates the initial poses of the objects such that their pose can be used to auto-initialize object tracking algorithms. The pipeline is able to handle unfamiliar and dynamic environments. The system was implemented in simulation and can be applied to a real robotic system.
The resulting pipeline uses an RGB image, a depth image, and the 3D object models as input. The outputs are the tracked object poses in real time. A dataset was generated to train the instance segmentation network. Furthermore, the pipeline was evaluated with several conditions to test the robustness of the tracking. A comparison to other 6D pose estimation approaches is provided in the results.
The code of the pipeline is available on GitHub: https://github.com/timschaeferde/rai_baxter

Kurzfassung

Ziel dieser Arbeit ist die Entwicklung einer robusten Pipeline zur Ojektverfolgung für den Baxter Roboter. Die Pipline umfasst die Anwendungen der Roboterkalibrierung und der Objektverfolgung, die die Lage von Objekte im Raum bestimmt, um sie für weitere Aufgaben zu nutzen. Durch die Kombination, Erweiterung und den Vergleich von modernen Ansätzen zur Objektverfolgung und Bilderkennung, soll die initiale Lage der Objekte abgeschätzt werden. Die Pipeline ist in der Lage, Objekte in unbekannten und dynamischen Umgebungen zu lokalisieren. Das System wurde mit einer Simulation implementiert und kann jederzeit auf ein realen Roboter übertragen werden.
Die resultierende Pipeline verwendet das RGB Bild, das Tiefenbild sowie die 3D-Objektmodelle als Eingang und gibt die Lage des verfolgten Objekts in Echtzeit wieder aus. Für das Training der Bilderkennung wurde ein Datensatz generiert. Desweiteren wurde die Pipeline unter verschiedenen Bedingungen evaluiert, um die Robustheit der Objektverfolgung zu testen. Ein Vergleich mit ähnlichen Ansätzen ist auch in den Ergebnissen enthalten.
Der Quellcode der Pipeline is auf GitHub zu finden: https://github.com/timschaeferde/rai_baxter

Contents

1 Introduction **13**

2 Background **15**
2.1 Related Work 15
2.2 Terms 16
2.3 Object Recognition 16
2.3.1 Mask R-CNN 16
2.4 Object Tracking 19
2.4.1 6D Pose 19
2.4.2 DBOT - Depth Based Object Tracking Library 19
2.5 Point Cloud Registration 20

3 Tracking Setup **21**
3.1 Architecture Overview 21
3.2 Simulation 22
3.3 Instance Segmentation 24
3.4 Pose Estimation 26
3.4.1 mask_pose 27
3.4.2 mesh_pose 28
3.5 DBOT Initialization 30
3.6 Launching the Pipeline 30

4 Experiments and Results **31**
4.1 Own Datasets 31
4.1.1 Generating Datasets 31
4.2 Mask R-CNN experiments 33
4.2.1 Training with Synthetic Dataset 33
4.3 Pose Experiments 37
4.3.1 Metrics 37
4.3.2 General results 38
4.3.3 Low-Data and Inaccurate Masks 42
4.4 Noise in Depth Image 44
4.5 Results of other Papers 45

5 Conclusion and Outlook **47**

A Appendix **49**

Bibliography **51**

List of Figures

2.1 R-CNN Model 17
2.2 Faster R-CNN Model 18
2.3 Mask R-CNN Framework 18
2.4 Shortcut Connection 19

3.1 Tracking Pipeline 21
3.2 Simuation Example 23
3.3 *rai_baxter* package 23
3.4 RViz Visualization 24
3.5 Initialization Process 24
3.6 Localization by Color Segmentation 25
3.7 Example of Mask R-CNN Segmentation Result 26
3.8 Outlier Detection 27
3.9 2D Point Cloud Center Computation 27
3.10 Point Cloud Registration 28
3.11 RViz Visualization of Tracking Multiple Objects 29

4.1 Objects of the final dataset. 31
4.2 Training Loss Plots 35
4.3 Detection with Augmented Images 36
4.4 Accuracy-Threshold Curve 38
4.5 Training Loss Plots Low-Data 42
4.6 Detection with Low-Data Network 43
4.7 Tracking in Highly Distorted Point Clouds 44
4.8 YCB Dataset Objects 45

A.1 Training Loss Plots (unsmoothed) 49
A.2 Training Loss Plots Low-Data (unsmoothed) 50

List of Tables

4.1 Augmentation Definition 34
4.2 Average Precision of Augmentation Results 34
4.3 Results of the Standard Pipeline 39
4.4 Bias Experiment Results 40
4.5 Results of different Networks 41
4.6 Objects in Dataset of the Low-Data Experiment 42
4.7 Results of other Approaches 46

Listings

4.1 COCO annotations example. 32

1 Introduction

Manipulation of an object with a robot includes multiple sub-tasks for the robot, before completing the task. One of these key sub-tasks is to localize the object in the environment. The objective is to estimate the 6D pose, which contains position and the orientation of the object. This task is extremely challenging in unknown and dynamic environments, where the robot only receives limited and noisy sensor input. Nonetheless, inferring the pose of the object is crucial for robot grasping, as well as to predict user intention in a human-robot collaborated or shared environment.

In the last years, significant progress has been made in real-time object tracking [IWC+16], instance segmentation [HGDG18], and 6D-Pose estimation [WXZ+19; XSNF18]. Most of the real-time tracking approaches, like DBOT (depth bases object tracking) [IWC+16], require manual initialization of the object pose and the 3D model (mesh). By combining tracking, segmentation, and pose estimation, the robotic system can autonomously localize and manipulate objects with out manual initialization. This leads to a safer, more independent, and more reliable performance of the robot and can be widely applied.

The objective of this thesis is to develop a robust and advanced pipeline for 6D pose estimation and tracking by combining state-of-the-art approaches, comparing, extending them and providing a simulated development-environment for manipulation tasks, which can be transferred to a real robot environment. The robot should be able to handle the segmentation task, tracking task and further processing of the pose feedback autonomously. While the input for the pipeline is an RGB image, a depth image and a set 3D models of the objects, the output is the detected pose of the objects.

As the first step, a simulation environment was set up to provide a virtual environment including the Baxter robot. The scene contains a table or a shelf and objects to track. In the second step, a robot controller and the pose estimator were implemented using the simulation environment to initialize the tracker. The simulation environment uses ROS (Robot Operating System) topics to communicate with the tracker and other components like the visualization using Rviz (ROS visualization). The simulation can be extended to a real world scenario that includes the physical Baxter robot and a RGBD camera. To make the system more robust to noisy real-world data, different experiments with visual artifacts are evaluated during the evaluation of the models in simulation.w

The results of the experiments show that the pipeline is very robust and performs extremely well, also compared to other pose estimation and tracking frameworks. An important feature of the pipeline is that only very limited training (with transfer-learning) and data (with augmentation) is necessary to get reasonable results. After the successful initialization of the tracker, it provides very useful pose feedback for further manipulation tasks in simulation.

The thesis starts with Chapter 2 on related work and background knowledge. Chapter 3 describes the complete architecture including the simulation environment, the DBOT (depth bases object tracking) initialization and the ROS package with its components. Chapter 4, Experiments and

Results, starts with the explanation of the own dataset and the generation of the dataset. Experiments demonstrate the performance of the instance segmentation and the pose initialization of the pipeline under different conditions. The conclusion and the outlook are presented in Chapter 5.

2 Background

2.1 Related Work

6D Pose estimation and object tracking became popular over the last years with a rapidly rising number of papers and approaches for robot manipulation tasks [CMS11]. Especially, the advances in high performance computing and the use of convolutional neural networks lead to significantly better detection/prediction performance [SEZ+14; SSB15; XSNF18]. Other than the NOCS approach from Wang et al. [WSH+19], which performs instance-level 6D pose estimation and scaling, the thesis focuses on the pose estimation without scaling. The objective is to obtain 3D position and 3D rotation of the objects of known shape and size to automate the manual initialization process of current real-time trackers, like DBOT [IW+16]. The approach of the thesisis limited to familiar objects with fixed shape.

Traditional approaches use template-based methods, which obtain the pose by rendering the objects from different fixed camera poses and try to find the best matching template all across the image [RL18; SQLG15]. These methods with templates are simple and able to detect texture-less objects but often fail with bad lighting conditions or occlusions between the objects. By taking the depth information into account, the performance in bad lighting or partial occlusion increases [SSB15].

The thesis is related to most recent methods like the PoseCNN [XSNF18] and DenseFusion [WXZ+19] which use instance segmentation to get the mask and extract features to predict the 6D pose of the objects. PoseCNN refines the pose by using Iterative Closed Point (ICP)[BM92] method, while DenseFusion comes up with a new iterative refinement procedure within the neural network architecture, which replaces ICP. The method in this thesis uses another refinement method which outperforms ICP and enables successful initialization of DBOT [IWC+16; WPK+13]. For the DBOT initializer only a small dataset and minimal amount of training is necessary to perform a sufficient initialization.

2.2 Terms

term	explanation
pose	position and orientation
point cloud	A set of points in the 3D-space.
mesh	3D model stored as polygon object in 3D-space
label	The label of an object is an individual name, not only the class.
rai	Robotic AI research software package with a small simulation. [Tou18]
ROS	Robot Operating System with software tools for robot development.
Rviz	ROS visualization tool.
DBOT	Depth Bases Object Tracking implementation of [IWC+16]
ICP	Iterative Closest Point is an old algorithm for point cloud registration
CPD	Coherent Point Drift algorithm for point cloud registration

2.3 Object Recognition

Object recognition, in general, is part of computer vision and used in many applications such as autonomous driving, robotics or medical diagnosis. There are four tasks that are mainly related to object recognition: classification and localization, semantic segmentation, object detection, and instance segmentation.

Classification and localization detects the class and location for a known number of objects out of a fixed set of categories(classes). It does not differentiate between multiple instances of the same class.

Semantic segmentation annotates the classes pixel-wise, but does not create object masks. E.g. it shows us that all these pixels in the image are pixels which represent cars.

Object detection predicts objects and differs between objects of the same class. Therefore, it returns the number, the class and the location (often bounding boxes) of each object.

Instance segmentation, as used in this thesis, unites semantic segmentation with object detection. It provides the location and the category-label (class) for each individual instance detected in the image. The labels are chosen from a fixed set of categories. Additional to object detection, instance segmentation provides a pixel-wise annotated mask. But contrary to semantic segmentation, the masks are generated per instance.

2.3.1 Mask R-CNN

Mask R-CNN [HGDG18] adds instance segmentation to Faster R-CNN [RHGS16] which is an improved version of R-CNN [GDDM14].

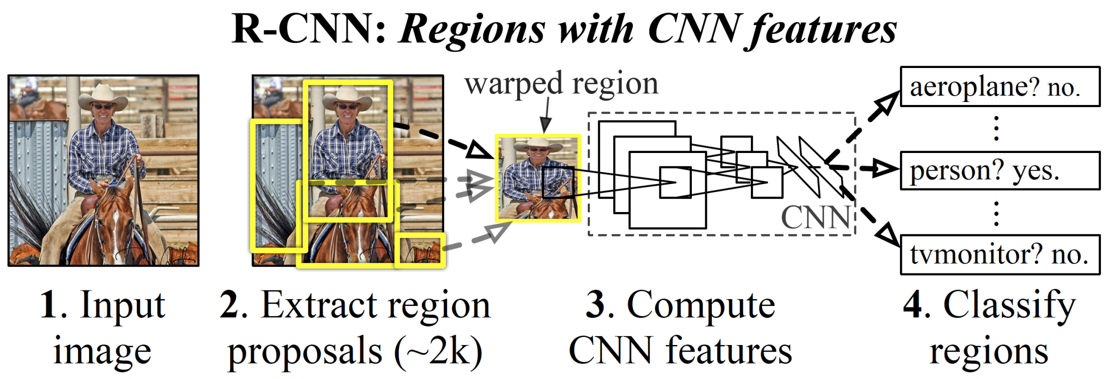


Figure 2.1: R-CNN object detection system overview. [GDDM14]

R-CNN

R-CNN is a Region-based Convolutional Neural Network [GDDM14] for classical object detection, which predicts labels and bounding boxes for objects in the image. First, it generates a manageable number (e.g. 2000) of independent region proposals all over the image and then evaluates each RoI (Region of Interest) with a CNN. (See Figure 2.1)

Faster R-CNN

Fast R-NN [Gir15] and Faster R-CNN [RHGS16] extend this approach by shared feature computation (with ResNet101)[HZRS15], RoI max pooling and a Region Proposal Network (RPN) attention mechanism. The RPN consists of additional convolutional layers which scan over so called anchors(regions/boxes) in the image, to evaluate how likely the anchor contains an object. RoI pooling makes sure to have a fixed input size for the following steps by resizing feature-maps from the anchors with max pooling. Thus, it only passes the proposed regions to evaluate the given features from RoI pooling and additionally uses the features from the RPN. This reults in a huge speedup in training and testing (see Figure 2.2). The classification and bounding-box regression are similar to R-CNN.

Mask R-CNN

Mask R-CNN [HGDG18] extended the object detection to instance segmentation and provides a label, a bounding box and an object mask for each object in the image, as shown in Figure 2.3.
The ResNet101 backbone starts by computing a feature map of the input image and in combination with a Feature Pyramid Network, which connects higher and lower level features, it returns an improved feature map. The features is passed to the RPN as described in Faster R-CNN (2.3.1) but instead of scanning the anchors over the image, Mask R-CNN scans the feature-map to avoid double computation. To refine the bounding box, the RPN predicts a delta for the change of the position and size for the box, to fit the object properly. Other than using max pooling, the authors of Mask R-CNN suggested to use bilinear interpolation for the "RoIAlign". The segmentation of the masks runs parallel to the class and bounding box prediction.

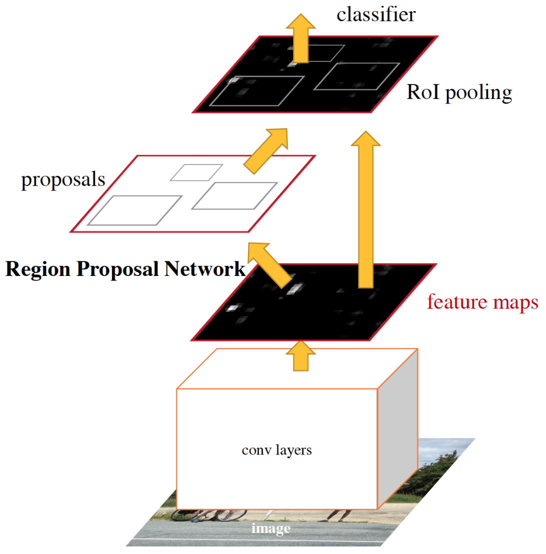


Figure 2.2: Part of the Faster R-CNN architecture. The RPN module serves as the 'attention' of this unified network. [RHGS16]

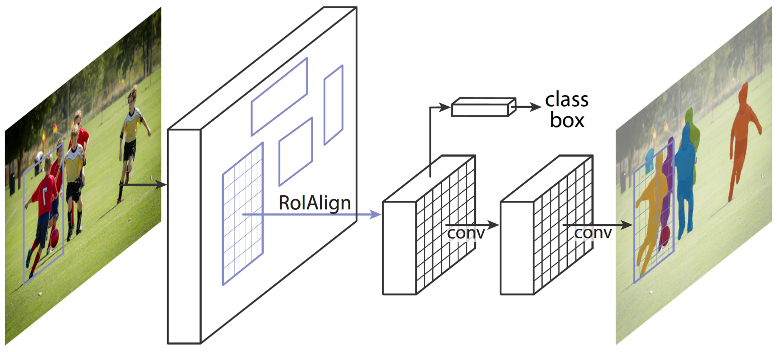


Figure 2.3: The Mask R-CNN framework [HGDG18].

Mask R-CNN creates 28x28 binary masks for each class in every RoI and evaluates the mask with a cross-entropy loss on the ground truth class-mask. During inference, it predicts 28x28 floating-masks which contain more information as a binary mask. The mask is scaled back to the size of the RoI and then binarized with a threshold of 0.5. The size of the masks is variable, but the implementation from Abdulla [Abd17] uses 28x28 as default.
The novelty of this approach is that the masks are generated per class and at the same time the classifier is not depending on the mask. On the contrary, the correct class-mask is choosen by the classifier.

Mask R-CNN benefits from the use of transfer learning, where existing knowledge is applied to a new, but related problem. By selecting a suitable pre-trained weight, e.g. a weight trained on COCO data, and using these as initial weights, Mask R-CNN produces high accuracy scores on small datasets, as well. The key to this behaviour is the pre-trained backbone, which already extracts very useful features out of the images, because it's already tuned on real images.

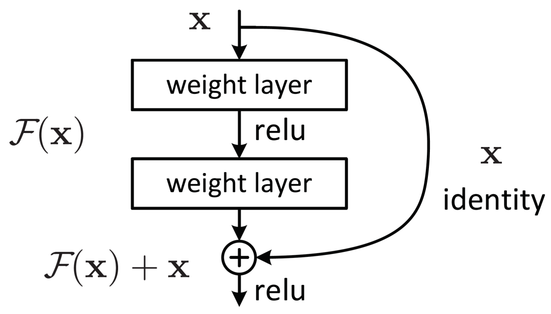


Figure 2.4: ResNet: Example ßhortcut connection"[HZRS15]

ResNet

ResNet is a deep residual neural network structure with “shortcut connections” (see Figure 2.4), which performs identity mapping and adds the mapping after the skipped layers. This structure tries to avoid the performance decrease in deeper networks and keeps the complexity on the same level. More details can be found here: [HZRS15; XGD+17].

2.4 Object Tracking

2.4.1 6D Pose

The term “pose” or “6D pose” is used to describe the position (3D) plus the orientation (3D) of an object or in an associated coordinate system.

2.4.2 DBOT - Depth Based Object Tracking Library

DBOT [IW+16] is the implementation of two papers: "Probabilistic Object Tracking Using a Range Camera"[WPK+13] and "Depth-based Object Tracking Using a Robust Gaussian Filter"[IWC+16], to perform 6D object tracking based on a depth image. Both trackers only require a full 3D model (mesh) of the object for tracking and there is no need to train the model for new objects.

The first paper [WPK+13] uses a dynamic Bayesian Network and a particle filter for the tracker. The Bayesian Network estimates the pose of the object by calculating a posterior distribution with respect to the previous poses and the observations. In every step, the likelihood of the observed depth input is calculated and passed down as an update to the network. They explicitly handle occlusion to gain robustness in their estimation. Therefore, their model has more dependencies in the Bayes network and is non-linear. Using a particle filter to compute the distributions during inference opens up the possibility of particle-wise parallelization.

The second method [IWC+16] uses a Gaussian filter plus some robustification methods, introduced by the same authors. One part handles occlusion, noise, and outliers (unmodeled effects), where they compute a probability of the measurement being produced by these effects. By applying this method, the unmodeled effects are handled better than with a normal Gaussian filter.

For high-dimensional measurements (e.g. depth-image) it is very complex to compute the standard Gaussian filter. Furthermore for the Gaussian tracker [IWC+16], the parallelization is not as trivial as for the particle tracker.

A comparison of robustness and accuracy for the two trackers is given in [IWC+16]. They mention that the particle tracker is more robust than the Gaussian filtering but the Gaussian tracker provides more accurate and smoother tracking. The Gaussian tracker sometimes loses track of the object. That's why Issac et al. [IWC+16] propose the particle tracker for fast and irregular scenes and the Gaussian tracker if more precise estimations are important.
The thesis also compares these two trackers and shows comparable results in Chapter 4.

2.5 Point Cloud Registration

The method of registration generally tries to register or align two point clouds with a nearly similar structure, but different positions and orientations. There are several types of registration like affine, rigid or non-rigid registration, but in the thesis only utilizes rigid registration.

With rigid registration the objective is to align a source point cloud or mesh, which initially does not have the same orientation and position, with a target object. Only rigid transformation is used to match the source with the target, therefore it only uses rotation, translation and scaling. It does not distort the shape of the object.
The registration is able to handle incomplete point clouds, like point clouds produced from a single point of view. Then the registration then tries to match the overlapping parts as good as possible.

3 Tracking Setup

The code of the tracking pipeline [Sch20] and its components are available on GitHub:
`https://github.com/timschaeferde/rai_baxter`

All related repositories such as the modified Mask R-CNN and the modified DBOT are linked in the README.md of the *rai_baxter* [Sch20] repository.

3.1 Architecture Overview

In Figure 3.1 the pipeline overview is shown. There are four major elements described in the following.
First starting with the simulation (Section 3.2), where the robot and the surrounding environment is simulated with a physics engine. Sensor and camera output of the robot are periodically published as ROS topics, which can be subscribed by RViz or DBOT. Secondly, instance segmentation (Section 3.3) and pose estimation (Section 3.4) are providing the DBOT initialization part of the pipeline, which is a major part in the thesis. Mask R-CNN [Abd17] is used for the instance segmentation. The masks are the input for the pose estimation. This pose estimation is a set of methods and algorithms to estimate the pose of the objects, from the mask and the depth information. Finally DBOT, which is described in Section 2.4.2, receives these initial poses. After receiving the

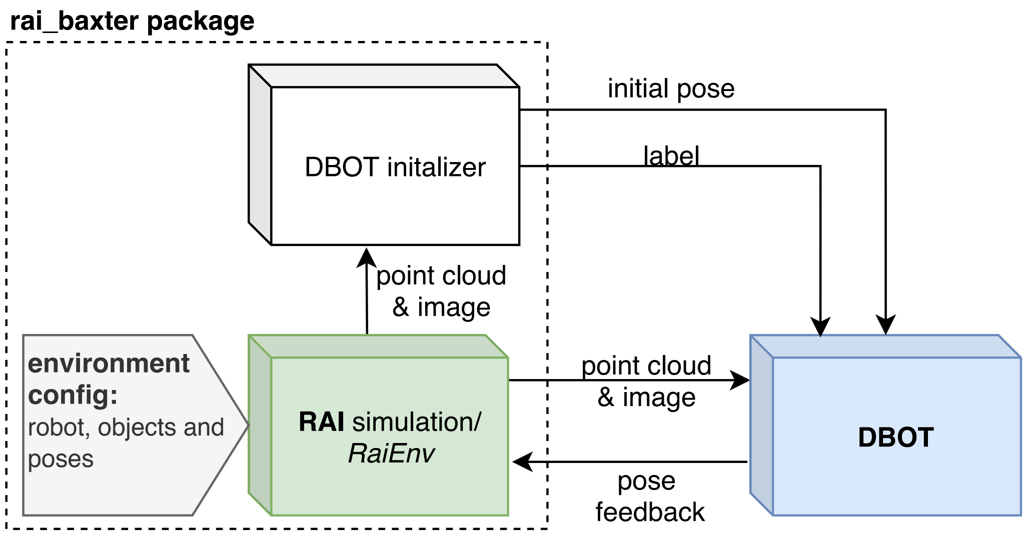


Figure 3.1: Overview of the tracking pipeline.

pose the tracker starts working by collecting the point cloud updates from the simulation via ROS topic subscription.

3.2 Simulation

In the thesis, only a simulation was used to build the environment for the robot.
The main arguments for using a simulation in the thesis are the flexibility, the reproducibility of scenes, and the speed (parallelization). Furthermore, it is simple to generate big datasets with any number of objects and variations.
Different approaches have been undertaken to avoid overfitting of the network, which is trained on the simulated inputs. These undertakings to make the predictions more robust to real inputs or to test the pipeline with "non-perfect" inputs, are described in the following sections and in Chapter 4, Experiments.

The pipeline is designed to operate with a real robot system instead of in simulation, by replacing the simulation with a ROS topic subscriber. In this case the subscriber passes the camera information, RGB image, and depth image to the DBOT initializer.

The simulation is based on the RAI library from Toussaint [Tou18], which provides rich functionalities in robotics, machine learning, and motion planning. Its main part is to handle configurations of the robot and objects in the environment. The simulation starts by adding and loading so called "graph"-files, where the initial configurations, like joints, orientations, colors, meshes, and sensor configurations are stored. It provides simple functions to edit all features in the configuration during run-time. After the initialisation of the simulation the environment is simulated step by step while reading the sensor outputs. The physics in the simulation are computed by *PhysX* and *bullet*.

In the thesis a model of the Baxter robot with a table in front (see Figure 3.2) was used as the environment. A simulated depth camera was added on the head of the simulated Baxter robot model to generate first-person view images. The resolution was chosen to 640x480 pixels, the focal length to 0.895 meter and the depth range is [0.4 meter, 7.0 meter].

To provide a useful interface, the *RaiEnv* (see Figure 3.3) class was developed during the thesis. *RaiEnv* comes up with a handful of functionalities to easily initialize and setup the configuration, read the camera output, and connect to a ROS interface. It handles to load the correct environment files, adds single or multiple objects in the scene, and enables the required components like the simulation or the visualizers.

To use the ROS structure to communicate with other nodes (components) and to add visualization via RViz, the hole simulation, environment files, and script's are wrapped into a ROS-catkin-package called *rai_baxter* (see Figure 3.3). Furthermore, publishers for the joint states, RGB images, depth images, and point clouds are implemented. These publishers can be used to publish the states and outputs of the simulation to ROS topics. With the *RaiEnv* class it is just one line of code to initialize the publisher-nodes and another one to publish the information.

When the information is published, RViz can be used to visualize those ROS topics. To visualize the joint states, the robot model needs to be loaded as *robot_description* in ROS and the *robot_state_publisher* node is required. Figure 3.4 shows how the visualization looks like.

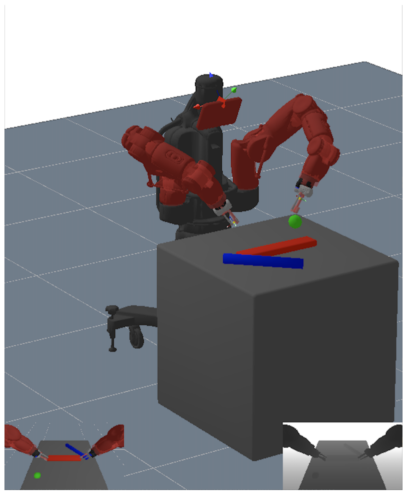


Figure 3.2: Simulation with Baxter robot and table with objects.

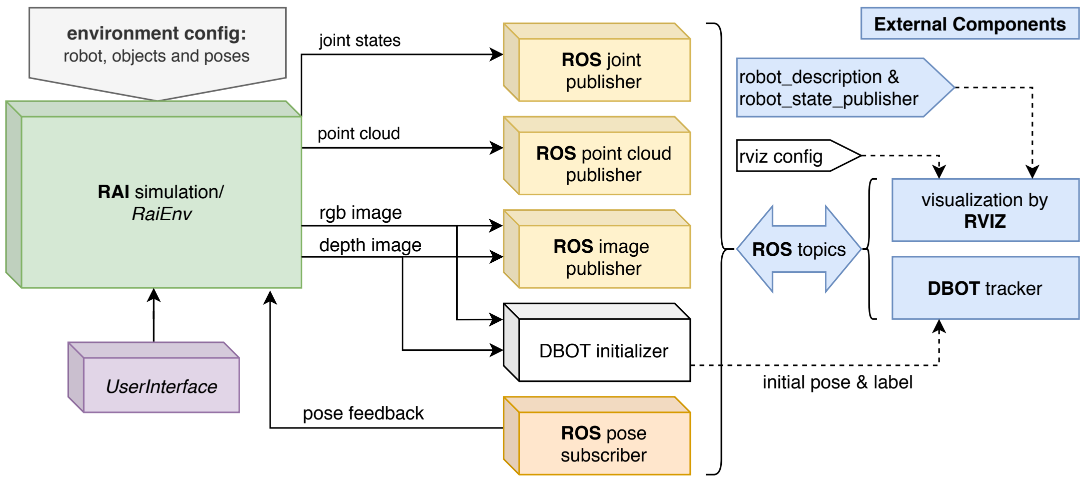
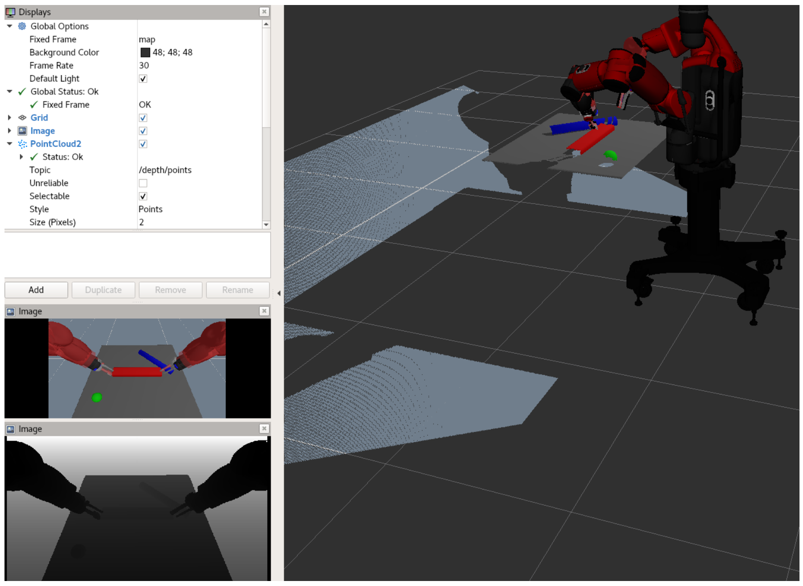


Figure 3.3: Overview of the rai_baxter package and its components.

The *UserInterface* (see Figure 3.3) class adds keyboard control to the simulation. There are keys to move the end effector position and a key to close the gripper.

With all these extension to the original RAI, this *rai_baxter* package comes up with a simple and extendable simulation. The simulation can be used for a wide range of tasks in robotic research or development. It is able to replace a real robot system for testing or evaluation of new methods. Even though the graphical rendering is not as realistic as in other engines, the simulation is lightweight and the results are adequate enough for most applications.


Displays

Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Fixed Frame	OK
Grid	✓
Image	✓
PointCloud2	✓
Status: Ok	
Topic	/depth/points
Unreliable	☐
Selectable	✓
Style	Points
Size (Pixels)	2

Figure 3.4: Visualization in RViz.

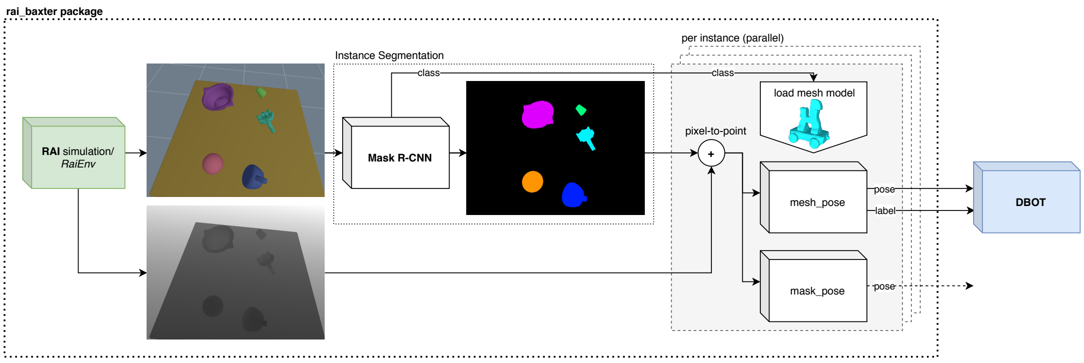


Figure 3.5: Initialization process of DBOT with the pose estimation.

3.3 Instance Segmentation

Instance segmentation is the first element of the DBOT initializer, which is shown in Figure 3.5.

In the beginning color segmentation was used with *openCV*[Ope15] to build up a proof of concept. Here the colors in the RGB-images were used to determine the mask of the object by simply searching the largest area of a single color. This mask of the object was applied to the depth image and the remaining pixels were converted to 3D points according to the camera specifications (focal-length, resolution). To receive the position of the objects, the center (coordinate-wise mean) of the resulting 3D points was calculated to provide an indication of the location.
For several instances, with a known number and the same color, the K-means algorithm was selected to get the K centroids of the objects. It is not just using the largest contour to get the mask, but it determines the K centers in images with the given color. The result of the proof of concept is shown in Figure 3.6.

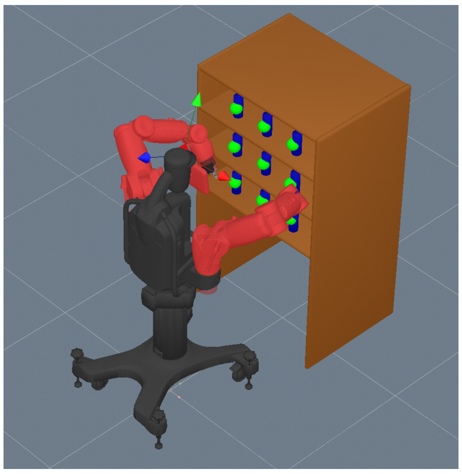


Figure 3.6: The green balls indicate the position calculated by using color segmentation and K-means.

The color segmentation can only produce acceptable results in a simulation with nearly no shadows and no reflections, because these effects aggravate the the determination of the color. Another challenge is, that the color either has to be fixed or must be passed manually to find the objects in the images. Therefore a more effective approach was used.

The implementation [Abd17] of the Mask R-CNN paper [HGDG18] was used for instance segmentation in the thesis. More Details on the architecture of Mask R-CNN are provided in Section 2.3.1.
Mask R-CNN is famous for instance segmentation, because it comes up with a flexible interface for own datasets and produces reasonable results even on small datasets (low-data problem) by transfer-learning. To support TensorFlow 2 the fork of Adam Kelly [fbAda18] was used.

Mask R-CNN has many advantages over color segmentation. The first advantage is that it can handle realistic images with noise or other distortions. Section 4.2.1 provides further details on different approaches how to make the Mask R-CNN more robust when training with little or synthetic data. The second advantage is that the result of Mask R-CNN is not only containing masks rather than class labels and bounding boxes. These class labels are especially used to fine tune the pose estimation and to initialize the DBOT tracker with the correct object. The benefit in knowing the object class during pose fine tuning is to use the according 3D model (mesh) to optimize the pose in a more precise way (see Section 3.4 for details). Additionally the Mask R-CNN is able to detect multiple objects in one detection step, even if they are close, occluded, overlapping or of the same color.

In a nutshell, the Mask R-CNN detects mask and class labels from the input RGB image. It can be trained efficiently by using pre-trained weights (transfer-learning) and is not limited to simulated images, single colored objects, or one object per image. In Figure 3.7, an exemplary output of Mask R-CNN together with an input image for the simulation is shown.

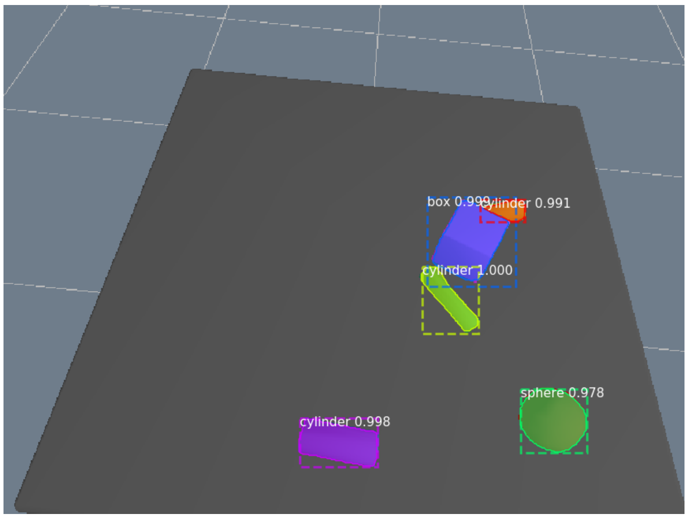


Figure 3.7: An example of a result from Mask R-CNN with different classes and overlapping objects.

3.4 Pose Estimation

The second part of the DBOT initializer (see Figure 3.5) is the final step to estimate the pose. The objective of the pose estimation part is to process the results of the instance segmentation, in this case Mask R-CNN, and to estimate position plus the orientation (pose) of the instances, in world coordinates.

The pose estimation, described in the following, runs parallel for each instance detected by Mask R-CNN. Equally to the Color Segmentation approach, described in the previous section, the depth image is masked by the instance mask. The masked depth pixels are transformed into 3D points with respect to the depth-value (d), the focal length (f_x, f_y), the center of the image (p_x, p_y), and the position of the pixel in the image (u_x, u_y). The equations for this pixel-to-point transformation are shown in Equations 3.1. In RAI the camera is positioned, so that the z-axis points towards the robot it self, therefore the minus sign in the brackets is required.

$$
\begin{aligned}
x &= d \cdot \frac{(u_x - p_x)}{f_x} \\
y &= (-)d \cdot \frac{(u_y - p_y)}{f_y} \\
z &= (-)d
\end{aligned}
\tag{3.1}
$$

After this transformation, the points are represented in the camera coordinates and not in world coordinates of the "real" robot environment. For the next steps, the poses remain in this coordinates and transformation into the world coordinates is performed in the end, because the initialization of DBOT requires the object pose in camera coordinates.

The pose estimation computes the *mask_pose* and the *mesh_pose* as visualized within the grey box in Figure 3.5, using the output of the pixel-to-point transformation. To achieve more precise results, the outliers in the observed point cloud are removed via the function *remove_statistical_outlier*

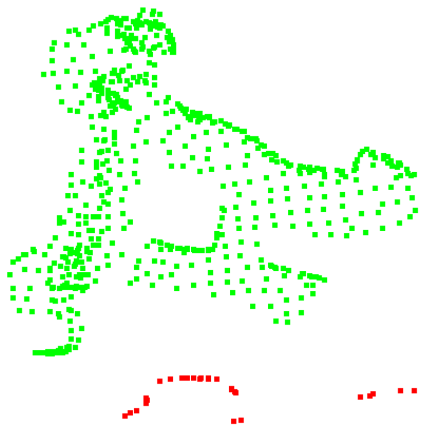


Figure 3.8: The red outliers getting removed from the observed point cloud.

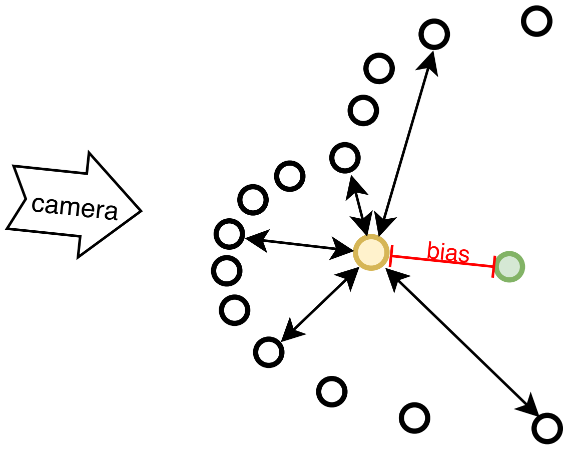


Figure 3.9: An example for computing the center of a 2D point cloud received by a camera. The yellow point is the computed center and the green point is the real center.

of *open3D* [ZPK18] (see Figure 3.8). The function uses the standard deviation of N neighbors to determine the outliers.

3.4.1 mask_pose

The *mask_pose* only estimates the position and not the orientation. It uses simple averaging per axis (coordinates) over the point clouds of every instance to identify the center. The center is a good approximation for the real position, center of mass, but is often biased in direction of the camera (see Figure 3.9). The reason for the bias is, that the camera provides an image from a single point of view. Therefore the depth image represents the object only from the side, which points to the camera. The same effect also occurs with concave objects, but with an opposite bias. To reduce this bias, a fixed offset in the Z-direction of the camera coordinates is manually added. Experiments with different offsets are shown in Chapter 4.

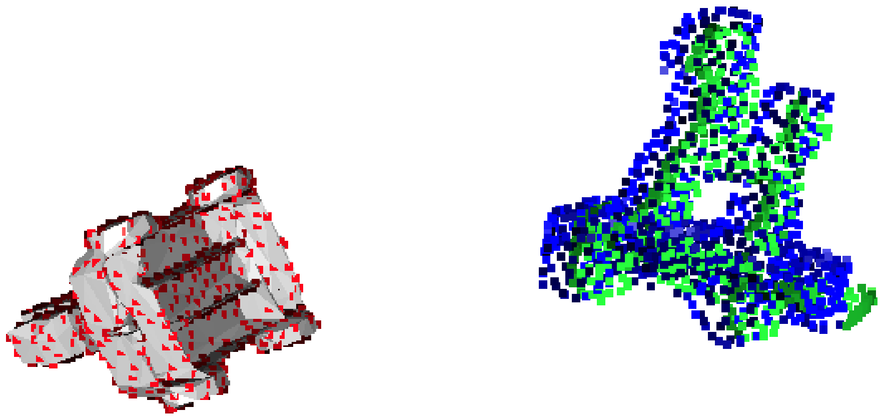


Figure 3.10: A point cloud registration used to align the red points (sampled on the mesh (gray) surface) with the green points from the depth camera. The result are the blue points.

3.4.2 mesh_pose

The *mesh_pose* estimates the complete pose of the objects. The label from the instance segmentation is used to load the 3D model of the object. A point cloud (set of points) with 1000 points is sampled via *open3D* [ZPK18] on the surface of the model for the following registration with the masked point cloud, referred to as the target point cloud. The are stored in a file, so they can be reused for the next estimation.
At this point, there is a sampled source point cloud and the masked point cloud. Next the normals of the point clouds are estimated, which supports the registration computation.
The main part of the "mesh_pose" estimation is the registration of these two point clouds.

The rigid registration, as defined in Section 2.5, is performed using the coherent-point-drift (CPD) algorithm, which is implemented in the *probreg* library [Ken] (see Figure 3.10). It is limited to rotation and translation, because scaling is not needed. As an initialization for the translation, the center (similar to *mask_pose*) of the target cloud is used. The output of the CPD is a 4x4 transformation matrix, which contains rotation and translation to align the object.
The whole process to estimate the *mesh_pose*s takes about 1-16s, depending on the number of objects and the number of iterations the mesh registration takes to find a sufficient pose. This process is no performance bottleneck, as this pose estimation is only performed once to initialize the actual tracker (DBOT).

Finally the positions from *mask_pose* and the *mesh_pose* are transformed with Equation (3.2) to world coordinates for external usage and evaluation. The orientations are transformed by the Hamilton Product [Wyn20] of the camera quaternion and the quaternion from the pose (see Equation (3.3).

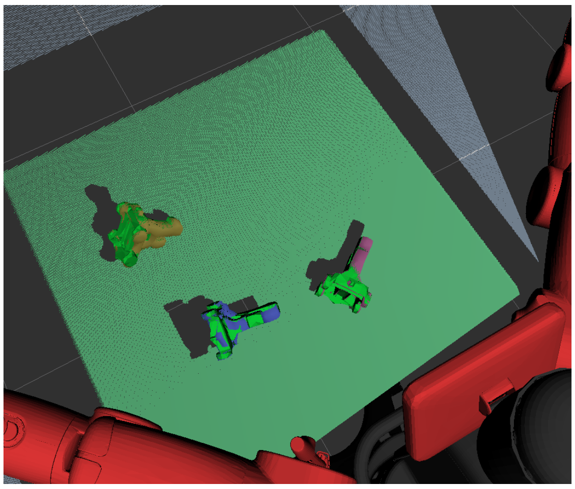


Figure 3.11: RViz visualization of tracking multiple objects parallel. The green objects are markers which indicate the current pose estimation by the tracker.

$$
\begin{aligned}
R &:= camera\ rotation \in \mathbb{R}^{3x3} \\
t &:= camera\ translation \in \mathbb{R}^{3} \\
T &:= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4x4}
\end{aligned}
$$

$$
\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} = T_{cam} \begin{bmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ 1 \end{bmatrix} \tag{3.2}
$$

$$
q_{world} = q_{cam} * q_{obj} \tag{3.3}
$$

3.5 DBOT Initialization

Originally DBOT requires manual initialization in RViz, where an interactive maker has to be aligned with the point cloud manually. To initialize DBOT [IW+16] without RViZ, DBOT provides a ROS service, which receives the initial pose for the particle tracker. Additionally, a new ROS service for the Gaussian tracker is implemented to initialize this tracker as well.

Both services are extended to receive initialization for multiple objects and to track them in parallel. A call to initialize the services contains a list of individual names, mesh-filenames (3D models) and poses. As mentioned in the previous chapter, DBOT receives the poses in camera coordinates.

After the initialization, the DBOT trackers compute a new position for every object, on each update of the point cloud, via the ROS publisher of the simulation. The computed pose is published as a ROS topic as well (see Figure 3.3). In RViz, the tracking can be visualized by adding markers as shown in Figure 3.11.

By using this pipeline, DBOT doesn't have to be initialized manually because initialization is automated, which is much more comfortable for the user of the robot. The advantage of successfully initialized DBOT trackers is that the simulation or a real robot system, can subscribe to the poses from DBOT (see Figure 3.1) and use this information to perform precise manipulations on the object.
Once initialized the DBOT tracker runs with 10Hz, for up to 7 objects, only on CPU. By using a GPU, the performance could be further improved.

3.6 Launching the Pipeline

The pipeline is launched via ROS-launch scripts in the *rai_baxter* package. These scripts start the ROS master, the tracker services, the RViz visualization (with robot model), and the simulation.

The launch scripts could also load the Depth-Based-Robot-Tracking (DRBT) [GIW+17]. The robot tracker subscribes to the same ROS topics as DBOT and computes calibrated joint states with the depth image and a model of the robot. These joint states could lead to a more precise manipulation for real robots, but this is not necessary in a simulation, so DBRT is disabled for the thesis.

4 Experiments and Results

4.1 Own Datasets

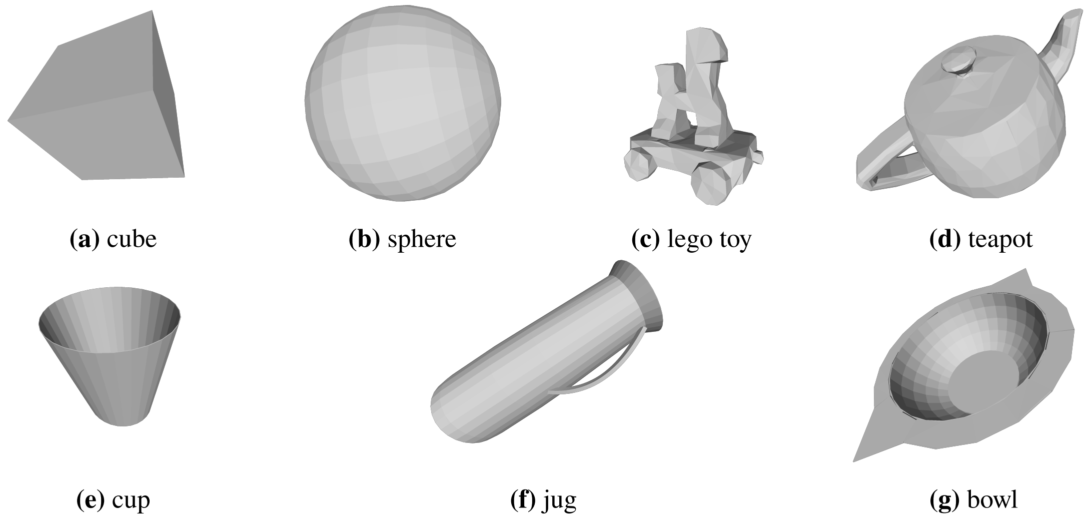
For the training of the Mask R-CNN Network, an own dataset is created. The dataset features are stored in a python file, which contains properties like the classes, object names, object mesh filenames, and path structures. The *Config* and *Dataset* classes from Mask R-CNN are extended to load the dataset correctly, in the same file. This configuration file is used for creation, training, and inference on the dataset.

The final dataset, used in the thesis, contains the seven objects presented in Figure 4.1. In the dataset the objects are randomly placed on a table. The final dataset contains about 49998 samples with 42498 images for training and 7500 images for validation. But this amount of data is not needed to train the Mask R-CNN sufficiently as shown in Section 4.2.1.

4.1.1 Generating Datasets

Each sample in the dataset includes an RGB image, a depth image, the masks for each object in the scene, and some informations for each object like position, quaternion, and color.

The complete dataset generation is implemented on top of the simulation. To generate a sample, the simulation is started and produces a random scene on the table. The function for generating this random scene is also used for creating evaluation scenes as well. The function either selects up to



(a) cube **(b)** sphere **(c)** lego toy **(d)** teapot

(e) cup **(f)** jug **(g)** bowl

Figure 4.1: Objects of the final dataset.

N objects or generates scenes with exact N objects. The position and orientations are randomly distributed on the table, while avoiding overlaps.
The color is selected exclusively to avoid that the objects have the same color as the background. The first step is to split the hue colorspace in N parts, starting at an arbitrary point. The second step is to check if one of the N colors is to close to a background color, because all colors have to be unique for the mask generation. If a color is to close, the same procedure starts with another arbitrary starting point in the colorspace. Saturation and Value for the HSV color are randomly selected in the interval: $saturation, value \in [120, 180]$
The masks are generated with the color segmentation algorithm, described in Section 3.3. By searching for pixels with the same hue value as the object, the mask is determined and stored as PNG. For this reason, the object colors have to be unique. This method for creating the masks could be replaced and improved by a method, which uses the object pose and the camera pose to calculate the mask. This improved method could work like an inverse of the pixel to point transformation (Equation (3.1)).All supplementary information is stored as a *dictionary* with the *pickle* library.

To store the masks and the annotations more efficiently, a script converts the datasets with the PNG masks (binary masks) into the format of the COCO [LMB+15] datasets. These datasets with the annotations of all samples are stored in a single *json* file. The masks are stored as polygons instead of binary masks. These polygons are described by a set of points in the image. The decreased accuracy of the masks does not significantly affect the training. Furthermore, a real dataset would be annotated with polygons as well. The bounding-boxes are described by the format [*top_left_x, top_left_y, width, height*]. An example format is shown in Listing 4.1.

```
{
  ...
  "annotations": [
    {
      "id": 0,
      "image_id": 0,
      "category_id": 3,
      "iscrowd": 0,
      "area": 5562,
      "bbox": [533.0, 300.0, 107.0, 106.0],
      "segmentation": [
        [ 639.0, 405.5, 624.0, 391.5, ... 639.5, 372.0, 639.0, 405.5]
      ],
      "width": 640,
      "height": 480
    },
    ...
  ]
  ...
}
```

Listing 4.1: COCO annotations example.

The RGB images are stored as PNG-files, because the encoding is efficient enough. The depth image can not be stored as PNG, due to the fact that PNG is not capable of storing the *float32* values of the depth image. Therefore, it is stored with *pickle*.

As many experiments showed that the robot arms were often detected as objects, the dataset is collected with arbitrary arm joint positions. Mask R-CNN is trained to treat the arms in the image as background.

4.2 Mask R-CNN experiments

4.2.1 Training with Synthetic Dataset

A dataset generated from a simulation is called a synthetic dataset. The samples in a synthetic dataset are typically perfect in terms of noise, blur, and lighting. To ensure the network is generalized towards real or non-perfect images, these distortions (noise,...) can be added to the images for training. Hence, the network learns to manage these kinds of visual distortions.

Tobin et al. [TFR+17] provides an approach where they successfully use randomization in datasets from the simulation to optimize detection in real images. They assume that enough randomization, results in the fact that the real world just represents another variant.

Augmentation

Mask R-CNN already provides the implementation of the library [JWC+20] to apply augmentation to the images. The library supports a tremendous number of image augmentations.
Augmentation is primarily used to artificially expand a small dataset by applying random augmentations on the images. The *imgaug* library [JWC+20] is used to add blur, add noise, resize the image, chop the images, change the contrast, and change the brightness.
Another experiment is using a small dataset and “expands” it with augmentation.

To evaluate the training with augmentation, a medium and a heavy augmentation is defined as shown in Table 4.1.

The results (see Table 4.2) of the trainings, with various augmentations, show that the networks trained with stronger augmentation perform better on the augmented test data while maintaining the precision on the non-augmented data on the non-augmented data. Presumably the precision of the less augmented trainings, on non-augmented data, is higher, because these trainings overfit to the simulated data. This effect is not desirable, if the objective is to obtain a more general and robust network.
The loss-plots in Figure 4.2 additionally show that the final loss from less augmented trainings is the smallest. This is not very surprising, because by augmentation there is always a loss of information in the augmented images.

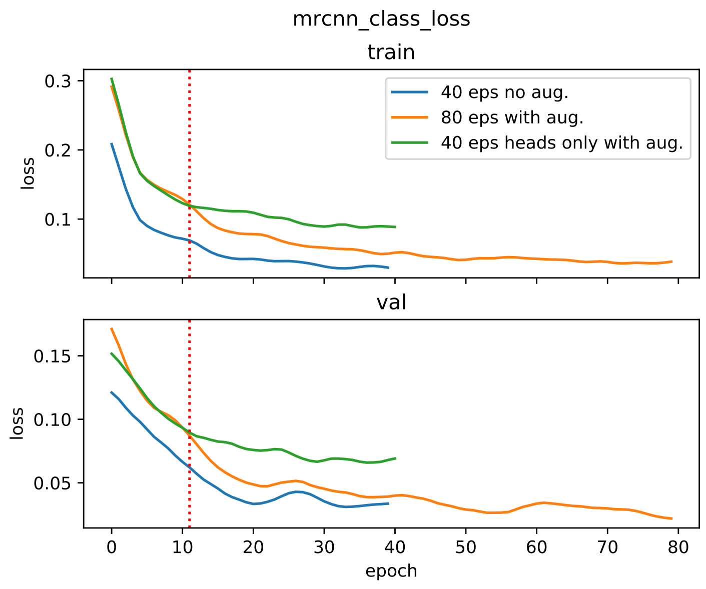
Figure 4.3 includes results of Mask R-CNN with the augmentation applied before inference.

	medium	heavy
	frequently used	
flip horizontal	X	X
flip vertical	x	X
chop & fill	X	X
affine trans.	X	X
light contrast	X	
multiply color	X	
	rarely used	
blur	X	X
sharpen	X	X
emboss	X	X
edge detection	x	X
noise	x	X
dropout	x	X
invert color	x	X
add color	X	X
multiply color		X
contrast		X
grayscale	X	X
elastic trans.		X
piece affine trans.		X

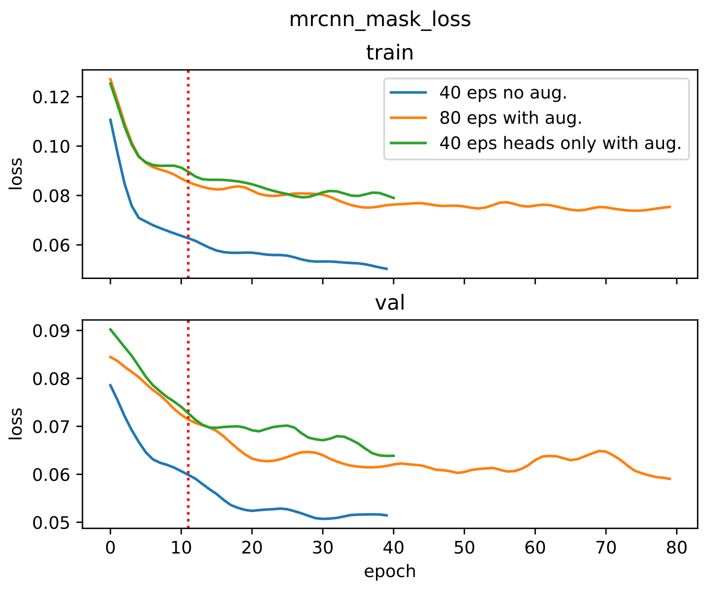
Table 4.1: Medium and heavy augmentations. A small x indicates lighter application.

	Augmentation in Training		Average Precision Test in %		
	train heads	fine-tuning	no aug.	med aug.	heavy aug.
training1	flip only	flip only	**99.072**	**94.975**	88.406
training2	med aug	med aug	98.206	94.953	87.720
training3	heavy aug	med aug	97.789	94.589	**93.844**
training4	med aug	heavy aug	diverged		

Table 4.2: Heads training 8 epochs and fine-tuning 8 epochs. Average Precision based on 200 samples.



(a) Smoothed plot of class loss



(b) Smoothed plot of mask loss

Figure 4.2: Loss plots. Smoothed by rolling average. The unsmoothed plots can be found in the Appendix, Figure A.1. The dotted red line indicates the boundary between heads only and fine-tuning training. Only medium augmentation was used in these trainings.

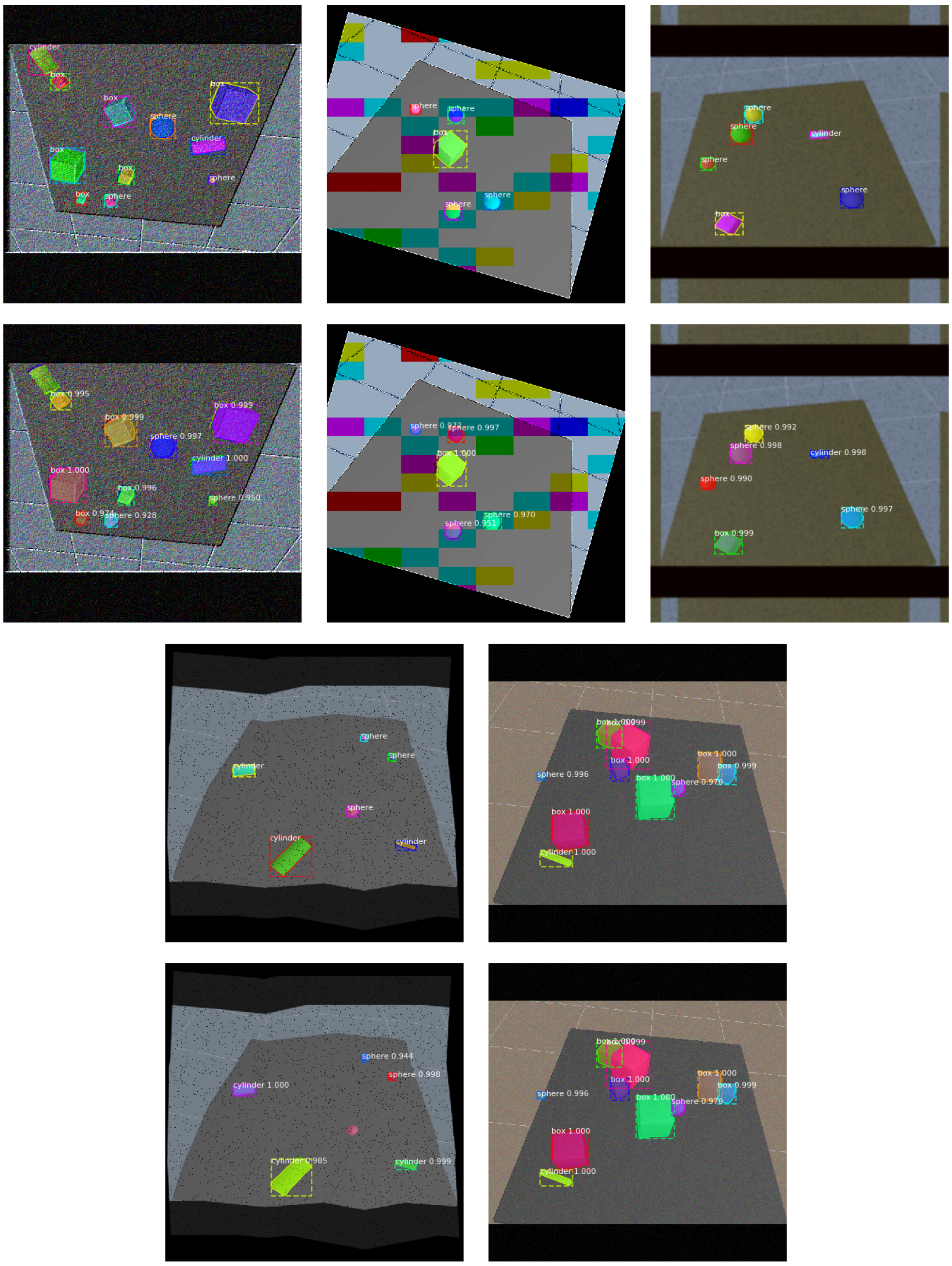


Figure 4.3: Detection with augmented images. The images are from a dataset with just cubes, cylinders, and spheres and not from the final dataset.

Optimal Number of Epochs?

Trainings of Mask R-CNN with more epochs have been performed to check how the losses behave over time.

Focusing on the most significant losses for the pose detection, only the *mrcnn_class_loss* and the *mrcnn_mask_loss* are presented and evaluated. The corresponding loss-plots are shown in Figure A.1a and Figure A.1b.
These plots (see Figure 4.2) show that the loss decreases only very little on the epochs above 40. Therefore, it could be enough to train just about 40 epochs.
The results of the network with 40 epochs and with 80 epochs are additionally compared by the predicted pose from the DBOT initializer subsystem. These results can be found in Table 4.5 in Section 4.3.2. They show that the accuracy of the the detected objects is exactly the same, only the detection rate increases from 92.726% to 94.33%. Considering the losses and the accuracies, it is obvious, that a extensive number of epochs is not mandatory for excellent performance.

Furthermore the training with fine-tuning on all layers results in a slightly lower *mrcnn_mask_loss* than just training the heads. Since the difference is bigger for the *mrcnn_class_loss*, it is recommended to train the network with fine-tuning.

4.3 Pose Experiments

4.3.1 Metrics

There are two methods of pose evaluation used in the thesis. The first method is to look at the position and the orientation separately. This is essentially for the comparison of the *mask_pose* and the *mesh_pose* position, because the *mask_pose* has no orientation. The second method is to evaluate the whole pose.

Especially for symmetric objects, it is challenging to find a metric which is able to handle the ambiguous poses of some perspectives. Therefore, the PoseCNN paper [XSNF18] provides an advanced average distance metric (ADD). The metric is called average closest point distance (ADD-S) (see Equation (4.1)) and handles the ambiguities of symmetric objects as well as normal objects. ADD-S transforms the 3D model (M with m points) of the object once by the predicted pose $(\hat{R}, \hat{t})$ and once by the ground truth pose (R, t). Then it computes the average distance of every point (x_2) in the predicted model to the closest point (x_1) in the target model.
In PoseCNN they do not calculate the accuracy by using a fixed threshold. Instead, they vary the threshold from zero to a maximum threshold and calculate the area under the curve (AUC) of the resulting accuracy-threshold curve (see Figure 4.4). The AUC divided by the maximum threshold results in the final accuracy score.

$$\text{ADD-S} = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} ||(Rx_1 + t) - (\hat{R}x_2 + \hat{t})|| \tag{4.1}$$

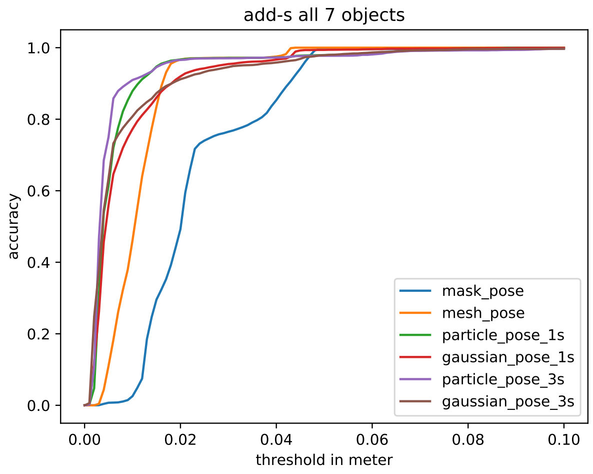


Figure 4.4: Accuracy-threshold curve of all objects with maximum threshold of 10cm for the different poses. The AUC score for the curves are e.g. *mask_pose* 76.69%, *mesh_pose* 88.49%, *particle_pose_3s* 93.60%

In the thesis, a maximum threshold of 10cm is used following DenseFusion [WXZ+19] and PoseCNN [XSNF18]. For the separate evaluation of the position or the orientation, the unconsidered part (orientation(R) or position(t)) is igno, when calculating the ADD-S. After the calculation for each sample the AUC, with 10cm threshold, is used as accuracy for the position or the orientation. When considering the whole pose for evaluation, the standard ADD-S and AUC with 10cm threshold is calculated.
DenseFusion has an additional metric, where they report the percentage of ADD-S under a threshold of 2cm. They argue that most of the robot grippers include a tolerance of 2cm and therefore all predictions with a distance under 2cm are tolerated.

4.3.2 General results

The evaluation of all pose experiments are performed on a set of 4,798 random samples. All samples are not part of the training for Mask R-CNN. These samples do not contain the arms of the baxter robot.
All the accuracy scores are calculated by the metrics described in Section 4.3.1. Only samples, where Mask R-CNN detects an object, are used for the accuracy-score. The detection rate and error rate, of Mask R-CNN, are provided with the results (below the tables). The detection rate is defined as number of classified objects with respect to the number of ground truth objects. The error rates are defined as the number of wrong class predictions with respect to the number of classified objects. The undetected objects are presumably simply not detected, out of sight or just partially visible.
The results from the pose estimation and DBOT initialization of the standard pipeline, which uses the unmodified images and a Mask R-CNN network with 40 epochs of training are shown in Table 4.3.
Note that the rotation precision of the sphere is always 97% but should be 100% for a perfect sphere. The reason is assumely the sampling of the point cloud models which are used for the accuracy calculation.

	mask	mesh		particle 1s		gaussian 1s		particle 3s		gaussian 3s	
	t	t	R	t	R	t	R	t	R	t	R
cube	89.1	90.3	96.0	97.4	97.0	93.9	97.0	**97.7**	**97.2**	90.7	96.9
sphere	76.1	79.2	**97.0**	97.9	**97.0**	97.5	**97.0**	97.9	**97.0**	**98.0**	**97.0**
lego_toy	66.7	80.2	90.2	**85.0**	92.2	84.8	92.3	84.9	**92.4**	83.0	92.3
teapot	77.8	80.0	90.7	91.7	93.3	86.5	92.8	**94.0**	**95.2**	91.2	94.8
cup	90.9	90.2	95.0	94.0	95.9	85.1	95.6	**94.3**	**96.2**	77.1	95.7
jug	69.8	77.3	90.2	93.6	92.4	74.8	92.8	**95.1**	95.3	89.9	**96.2**
bowl	69.3	70.4	84.4	81.0	87.7	**86.6**	84.9	81.1	**88.7**	80.8	85.6
mean	77.1	81.1	91.9	91.5	93.7	87.0	93.2	**92.1**	**94.6**	87.2	94.1

	mesh		particle 1s		gaussian 1s		particle 3s		gaussian 3s	
	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm
cube	93.8	100.0	96.8	100.0	95.6	99.7	**97.0**	100.0	94.3	98.8
sphere	88.6	100.0	**96.1**	100.0	96.0	100.0	**96.1**	100.0	**96.1**	100.0
lego_toy	90.8	99.3	93.3	98.1	**93.4**	96.7	**93.4**	97.3	92.3	88.4
teapot	88.2	100.0	93.7	100.0	91.3	94.0	**95.2**	100.0	93.8	95.2
cup	92.9	100.0	95.1	99.0	92.0	92.7	**95.3**	98.9	87.8	86.4
jug	84.6	98.8	91.8	99.6	86.6	82.7	**95.2**	99.6	93.3	94.5
bowl	80.4	78.7	83.6	80.2	**85.5**	78.1	82.9	80.2	83.9	75.0
mean	88.5	**96.7**	92.9	**96.7**	91.5	92.0	**93.6**	96.6	91.6	91.2

Table 4.3: Results of the standard pipeline. The upper table shows position and orientation individually. The lower table shows the AUC and <2cm metric.
detection rate(Mask R-CNN): 92.726%; error rate(Mask R-CNN):0.16%

mask_pose vs. *mesh_pose*

The *mask_pose* only contains position information and no information about the orientation. But the *mask_pose* is a good baseline for the position. The *mesh_pose* contains both position and orientation.
In the first table of Table 4.3, the comparison of the *mask_pose* position and the *mesh_pose* position shows that the *mesh_pose* additionally is slightly more accurate.
In the following evaluations only the *mesh_pose* will be considered.

Bias in Point Cloud Center

When calculating the position of a point cloud from a camera image, there is often a bias for the position in the direction of the camera. This effect especially occurs on convex surfaces. Concave surfaces cause an opposite effect. But all objects with positive volume show this bias effect, no matter how many convex or concave surfaces there are.

	mask	mesh	
offset	t	t	R
-0.01m	56.2		
-0.005m	62.1		
0.00m	66.0	77.0	91.9
0.015m		79.4	91.9
0.02m	77.1	**81.1**	91.9
0.03m	**77.6**	80.8	91.9

Table 4.4: Mean precision of all objects for bias compensation results with different offsets.

This bias is described and illustrated in Section 3.4.1 and Figure 3.9. The effect of the bias is also visible in the accuracy-threshold curve (see Figure 4.4) where the gradient of the *mesh_pose* (with offset) is very high but starts to rise barely after some millimeters. This effect is even worse when no offset is added.
Furthermore, the pipeline is tested with different values for the z-axis-offset (in camera coordinates), which should compensate the bias. The mean precision over all objects is presented in Table 4.4. In most cases the bias mainly depends on the size of the objects. Therefore, the bias should be adjusted for significantly bigger or smaller objects. The samples used for evaluation contain the same objects as the final dataset (see Figure 4.1). They all have a diameter between 4cm to 15cm.

The *mesh_pose* provides a more accurate pose even without offset. An offset of 2cm for the *mesh_pose* and 3cm for the *mask_pose* produces the best results. As expected the bias barely affects the orientation. Another result is, that the offset affects the *mask_pose* more than the *mesh_pose*, because the precision of the *mask_pose* increases with a higher rate, when adding more offset. Additionally, the bias for the *mesh_pose* is not as depended on the object size as the bias of the *mask_pose*.

An alternative approach to compensate the bias of the *mask_pose*, could be a higher weighting of the pixels, which are further away form the camera, during the center computation. This method is not evaluated in the thesis, because the *mask_pose* is just used as a baseline.

Result of long Mask R-CNN training

As already mentioned in Section 4.2.1, the pose estimation does not depend on perfect segmentation to produce excellent results. The corresponding results of different trainings are visualized in Table 4.5.

DBOT Initialization

To evaluate the behaviour of DBOT after an automated initialization with the *mesh_pose*, the pose of DBOT is captured after 1s and 3s for both trackers. As the objective is not to evalutate the DBOT tracking itself, the objects are not moved during the first three seconds.
The accuracy scores are presented in Table 4.3. These results show that both trackers achieve better

	40 epochs		80 epochs		40 epochs (ld)		20 epochs (ld)		40 epochs (noise)	
	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm
cube	93.8	100.0	93.8	100.0	93.5	100.0	93.6	100.0	93.7	100.0
sphere	88.6	100.0	88.9	100.0	89.1	100.0	89.3	100.0	88.3	100.0
lego_toy	90.8	99.3	90.6	98.7	90.6	98.9	90.7	98.8	90.6	98.8
teapot	88.2	100.0	88.0	99.9	87.9	99.8	87.9	100.0	88.1	100.0
cup	92.9	100.0	92.9	100.0	92.9	100.0	92.9	100.0	92.9	100.0
jug	84.6	98.8	84.5	98.3	84.4	98.7	84.7	98.7	84.2	96.9
bowl	80.4	78.7	80.3	77.9	80.6	78.8	80.8	78.9	80.3	78.4
mean	88.5	96.7	88.4	96.4	88.5	96.6	88.6	96.6	88.3	96.3

Table 4.5: Accuracy results of the *mesh_pose* from different experiments. All networks trained with medium augmentation.
Networks from left to right:
40 epochs on the standard dataset with about 50k samples;
80 epochs on the same dataset (50k samples);
40 epochs on a very small dataset (low-data) with 107 samples;
10 epochs the same very small dataset (107 samples);
40 epochs same network as the first column but with noise and blur in evaluation.
Mask R-CNN rates from left to right:
detection rate: 92.73% ; error rate:0.16%
detection rate: 94.33% ; error rate:0.13%
detection rate: 95.16% ; error rate:0.68%
detection rate: 95.14% ; error rate:0.74%
detection rate: 94.00% ; error rate:0.70%

accuracies, for translation (t), rotation (R), and the AUC metric, than the initial pose. This indicates a successful initialization of the trackers, because they are able to refine the pose.

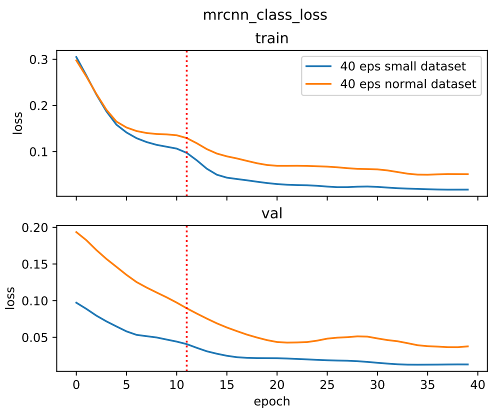
Particle Tracker vs. Gaussian Tracker

The particle tracker refines the pose faster and is accurate than the Gaussian tracker. But both trackers still outperform the initial pose (*mesh_pose*). A reason for the more inaccurate results of the Gaussian tracker may be that the tracker is less robust to inaccurate initialization. As observed in the simulation, the tracked objects sometimes drift away from the correct pose towards the table or other objects. Once the Gaussian tracker tracks the object correctly, it maybe even more precise than the particle tracker as the authors of DBOT mentioned in their paper [IWC+16].

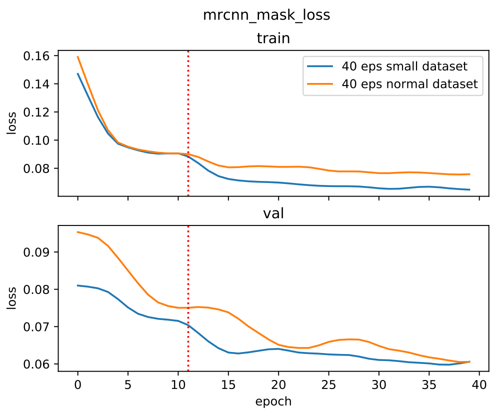
The "<2cm" metric does not behave like the other metrics, for the Gaussian tracker. The metric decreases over the time. This effect could also be related to the pose drift mentioned before. The "<2cm" metric penalizes such drift, away from the true pose, more than the other metrics. Because the drift does not happen very often, the other metrics increase for the Gaussian tracker as well.

	val	train
cube	4	46
sphere	7	54
lego_toy	7	56
teapot	5	54
cup	8	51
jug	6	51
bowl	7	55

Table 4.6: Occurrences per class in a small dataset for the low-data experiment.



(a) Smoothed plot of class loss



(b) Smoothed plot of mask loss

Figure 4.5: Loss plots of the low-data experiment. Smoothed by rolling average. The unsmoothed plots can be found in the Appendix, Figure A.2. The dotted red line indicates the boundary between heads only and fine-tuning training.

4.3.3 Low-Data and Inaccurate Masks

Another experiment indicates the performance of the DBOT initialization with a Mask R-CNN network, which is trained on a small dataset (low-data training). The dataset contains 91 training and 15 validation images, the occurrences of each class are presented in Table 4.6.

The network is trained for 40 epochs with this small dataset and medium augmentation. The evaluation shows that the average precision of Mask R-CNN on unseen data is 91.2098%, which is still good. It is important to include augmentation during training with small datasets, otherwise the trainings tends to overfit. The loss plots of these trainings compared to the network trained with a big dataset are presented in Figure 4.5. Generally the loss of the low-data training is smaller. The reason maybe some overfitting to the small dataset. Another interesting aspect is that the final mask loss on the validation dataset is almost the same as for the network trained with the big dataset.

To confirm the results, the accuracy of the pose from the DBOT initializer, which receives more inaccurate masks from the low-data network, is shown in Table 4.5.
These results show that this pose estimation produces the same accuracy as the pose estimation with

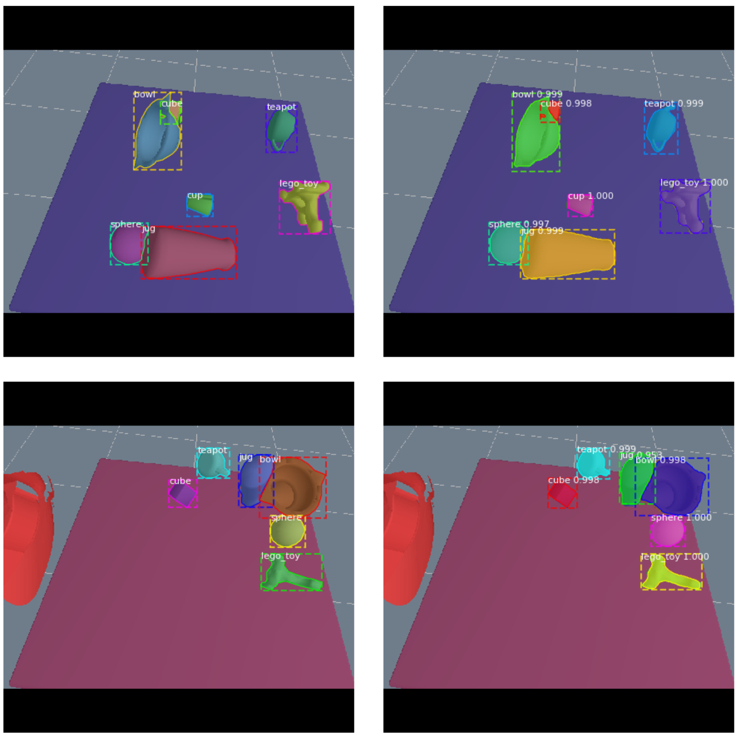
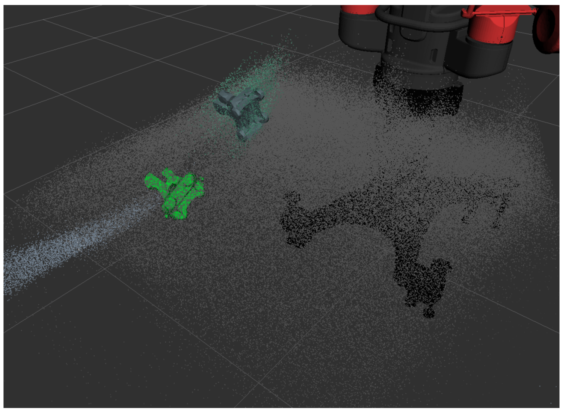


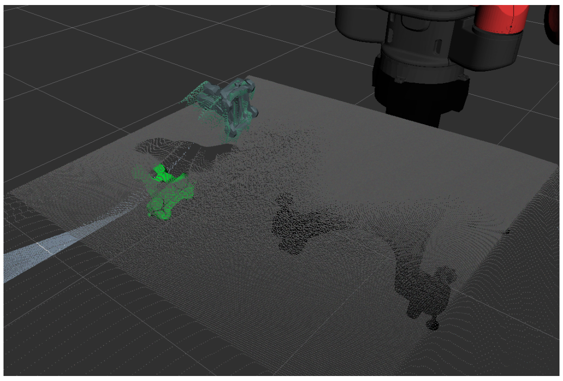
Figure 4.6: Mask R-CNN results from the low-data experiment network after 16 epochs. On the left are the ground truth images and on the right the segmented images.

the more accurate masks. Even the detection rate is higher than for all the other networks in the table. The error rate is higher as well, but in sum the low-data network obtains the highest number of correct labeled objects. The reason for the robust pose refinement, even with inaccurate mask inputs, is the outlier detection. It compensates the inaccuracies and leads to the robustness of the coherent-point-drift point cloud registration.
Furthermore, when selecting the weights from epoch 20 on the low-data network the pose results are practically identical. Only the error rate is a little bit higher, than for the 40 epochs weights.
The images presented in Figure 4.6 are the results from the network after 16 epochs training on the small dataset. These results are excellent, considering the small dataset and short training period. It is quite impressive, how fast the Mask R-CNN is able to learn from a remarkably small dataset.
The fact that the accuracy of the masks plays a minor role, is observed during the evaluation of the network with 80 epochs, as well. This network has even more accurate masks than the standard network, but the performance is still the same.

All these results prove, that the complete pipeline performs very well even with small datasets and is very robust to non-perfect segmentation masks. Combining the facts that the pose estimation is very robust and the Mask R-CNN network has a high learning speed, the pipeline is able to adapt to new objects or other datasets very fast.



(a) Heavy noise and blur in the point cloud.



(b) Some images later there is less noise, this shows that the grey Gaussian tracker still tracks the object very good. The green particle tracker is not able to follow the object.

Figure 4.7: Tracking in highly distorted point clouds.

4.4 Noise in Depth Image

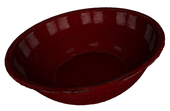
In this experiment, the visual distortions, like noise and blur, are added to the RGB and depth image during the initialization. These effects are frequently updated on every captured image (10Hz). The *imgaug* library [JWC+20] assists to generate these effects. Both effects are not applied uniformly. The strength of the effect changes randomly across the image and over time. The objective is to simulate a "real" camera form the synthetic images and compare the performance. The range of the noise for the depth image is between 0mm to 7mm. Therefore, the maximal difference from one pixel to its neighbor, on a planar surface, is about 14mm.

The results of the experiments are shown in Table 4.5. They show only a minimal decrease of accuracy compared to the standard evaluation. Furthermore, the point cloud registration takes about 50% more time for the distorted point clouds than for the standard registration. Because the results are very close to standard evaluation, another experiment is started with even more noise and blur. The maximum noise is set to 60mm, which is quite high. The results of the second experiment are almost as good as the results from the first one. The mean precision for AUC metrics is 87.2% (1.3% lower than the standard precision) and for the <2cm metric is 95.1% (1.6% lower than the standard precision). Additionally, the results of the second experiment are verified manually. This shows that the outlier detection and the registration performing excellent in refining towards the best-matching position, even in highly distorted point clouds.
Furthermore, the tracking of DBOT in the second experiment showed that the Gaussian tracker is able to track the object really well with the distortions in the input point cloud (see Figure 4.7). On the contrary, the particle tracker is not able to handle the tracking with the distortions. The reason here is not the initial pose, as the successful tracking of the Gaussian filter proves, but the particle tracker itself.
In conclusion, the pose estimation of the DBOT initialization is extremely robust against noisy or blurry images.

(a) potted meat can

(b) wood block

(c) pitcher base


(d) bowl

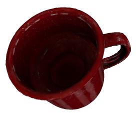

(e) mug

Figure 4.8: Some YCB Dataset Objects. [CWS+15]

4.5 Results of other Papers

This section presents a brief overview over the results form the DenseFusion paper [WXZ+19] and compares them to the results from the thesis. In their paper, they compare DenseFusion [WXZ+19] with PoseCNN [XSNF18]. The paper uses the same metrics as used in the thesis (see Section 4.3.1).

DenseFusion [WXZ+19] evaluated their work on the YCB [CWS+15] and the LineMOD[HHC+11] datasets. Therefore, a direct comparison of the results is not completely valid, due to the different datasets used for evaluation, but it provides an idea how this approach performs compared to others.

The following objects from the YCB dataset are comparable, in terms of size and form, to objects from the final dataset in the thesis:

YCB dataset		**own dataset**
potted meat can (4.8a)	<->	cube
wood block (4.8b)	<->	cube
pitcher base (4.8c)	<->	jug
bowl (4.1g)	<->	bowl
mug (4.8e)	<->	cup

The results from the DenseFusion paper compared to the own results are presented in Table 4.7. For all the compared objects, the own results are pretty close, some are even better compared to their results.
The mean accuracy of DenseFusion is 93.1% for the AUC metric. The mean AUC of the *mesh_pose* with 88.5% is already very close and compared to the particle tracker after 3s with 93.6%, the DBOT pipeline is even better than the accuracy of DenseFusion.
As already mentioned, the direct comparison is not totally fair here, but the pipeline presented in the thesis produces excellent results, which can definitely challenge current state of the art frameworks like DenseFusion.

	PoseCNN [XSNF18]		DenseFusion [WXZ+19]		Own results				
					own	mesh		particle 3s	
	AUC	<2cm	AUC	<2cm	class	AUC	<2cm	AUC	<2cm
master chef can	95.8	100.0	96.4	100.0					
cracker box	92.7	91.6	95.5	99.5					
sugar box	98.2	100.0	97.5	100.0					
tomato soup can	94.5	96.9	94.6	96.9					
mustard bottle	98.6	100.0	97.2	100.0					
tuna fish can	97.1	100.0	96.6	100.0					
pudding box	97.9	100.0	96.5	100.0					
gelatin box	98.8	100.0	98.1	100.0					
potted meat can	92.7	93.6	91.3	93.1	cube	93.8	100.0	97.0	100.0
banana	97.1	99.7	96.6	100.0					
pitcher base	97.8	100.0	97.1	100.0	jug	84.6	98.8	95.2	99.6
bleach cleanser	96.9	99.4	95.8	100.0					
bowl	81.0	54.9	88.2	98.8	bowl	80.4	78.7	82.9	80.2
mug	95.0	99.8	97.1	100.0	cup	92.9	100.0	95.3	98.9
power drill	98.2	99.6	96.0	98.7					
wood block	87.6	80.2	89.7	94.6	cube	93.8	100.0	97.0	100.0
scissors	91.7	95.6	95.2	100.0					
large marker	97.2	99.7	97.5	100.0					
large clamp	75.2	74.9	72.9	79.2					
extra large clamp	64.4	48.8	69.8	76.3					
foam brick	97.2	100.0	92.5	100.0					
MEAN	93.0	93.2	93.1	96.8					

Table 4.7: Evaluation of 6D pose (ADD-S) on YCB-Video dataset. The first four columns are from Table 1 in the DenseFusion paper [WXZ+19] on page 6.

5 Conclusion and Outlook

Conclusion

The thesis comes up with a whole pipeline to simulate a robot environment, to accurately compute the initial poses of the familiar objects, and to initialize the object tracker, DBOT [IW+16]. The objective is to provide a robust initialization for the tracker and the feedback from the tracker for precise robot manipulation. The tracking pipeline is build from state-of-the-art components. The individual parts of the pipeline are cooperating successfully and they are flexible to integration into other systems.

The experiments show that the DBOT initialization in general performs excellent. Furthermore the pipeline only requires a minimal amount of training to adapt to new object sets, because Mask R-CNN is the only part which requires training. Other frameworks like DenseFusion requires more training than just the instance segmentation for each new object.
Moreover, it is shown that Mask R-CNN can be trained quite fast and with very little data due to the use of pre-trained weights e.g. from the COCO dataset. The main objective of using Mask R-CNN is to classify objects and provide labels. The masks gives a rough idea of the object location in the image or a bounding box could be enough. The *mesh_pose* estimation handles the final refinement, even with inaccurate masks or other distortions (noise or blur). DBOT only needs the initial pose and the 3D model of the object for the tracking.
All poses in the pipeline are in camera coordinates, therefore the tracking even works without knowing the exact location of the camera. Only when the real position is used for some manipulation, the camera pose is required.

On one hand, the advantages of this tracking pipeline are the handling of textureless objects, symmetric objects, and the utilization of transfer-learning for Mask R-CNN. Furthermore it is adaptable to other systems because it uses popular and well documented components.
On the other hand, the use of the 3D models limits the pipeline to objects, where the 3D model is available, and has a fixed shape (objects with no degrees of freedom). The pipeline is only able to handle one 3D model per class, therefore every model requires to have its own class for the instance segmentation. For example, the pipeline cannot estimate the pose of two different cars like a sports car and a SUV, which are both labeled as *car*. To properly handle this case, Mask R-CNN needs to be trained for the classes *sports_car* and *suv*.

Outlook

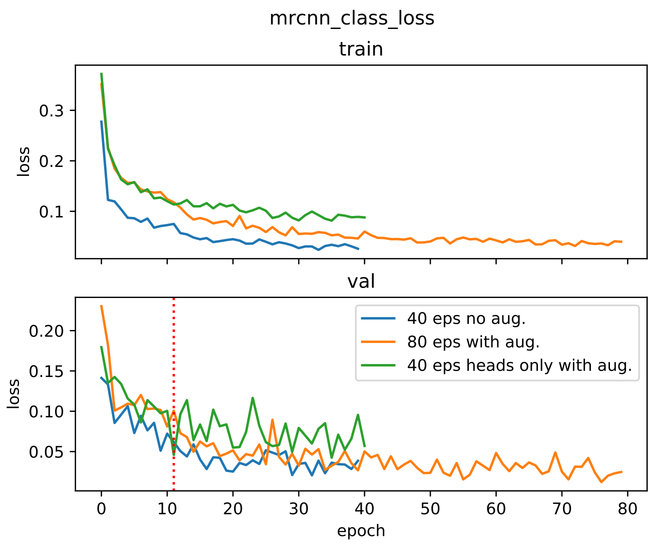
Overall, the results are promising. The pipeline and some individual parts may be used in future robotic research projects. The tracking could be an excellent base to develop advanced manipulation approaches or to accurately predict the intention of a person, who is manipulating the objects in front of the robot. This could lead to a better robot-environment or robot-human interaction.

Nevertheless, there are still many opportunities to improve the pipeline. One improvement could be the usage of a GPU for the mesh registration and for the DBOT tracking, to improve the speed of the initialization and the tracking. The *probreg* library [Ken], which performs the registration, is capable of using CUDA. But this approach was not used by the thesis. It may be worthwhile to look into the fast-coherent-point-drift approach from Feng et al. [FFZ20] for better performance.

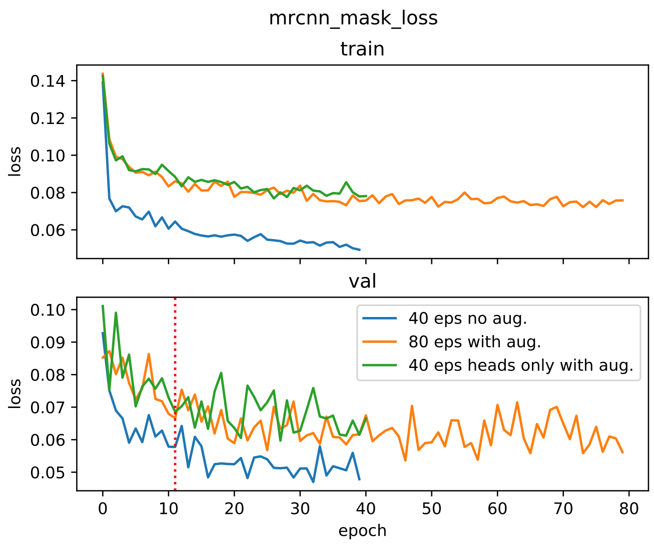
Another level of improvement could be to utilize a small convolutional network to refine the pose after the registration of the point clouds. By using the label and the pose of the object as input to the network, it tries to minimize the bias by adding a custom offset. This would add an additional training effort to the network, but since the network could be small, it should be no problem. Then the network may even detect additional biases in the registration, outside the z-direction.

Furthermore, there was no access to the lab during the time of the thesis due to CoVID-19 pandemic. The drawback is, that there are no results about, how the approach would perform in a real environment with noisy sensors, blur, real light conditions or cluttered background scenes. The complete tracking could be tested with a real dataset on a real robot or at least with more realistic data and simulation. Another possibility is to render objects into real background images, as done in other approaches.

A Appendix

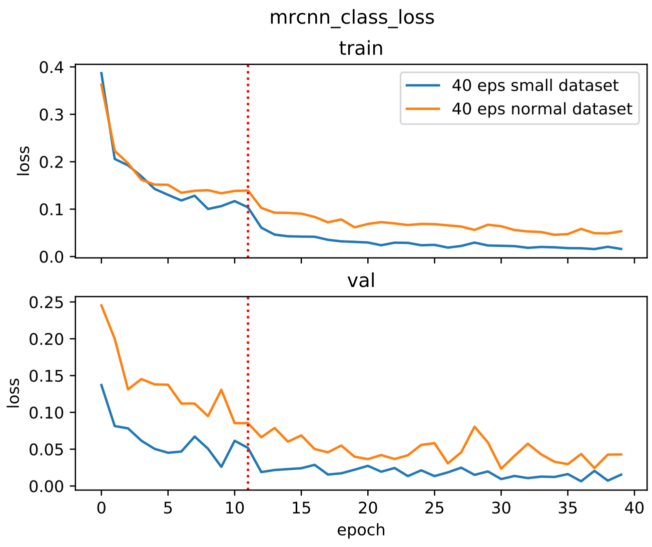


(a) Unsmoothed plot of class loss

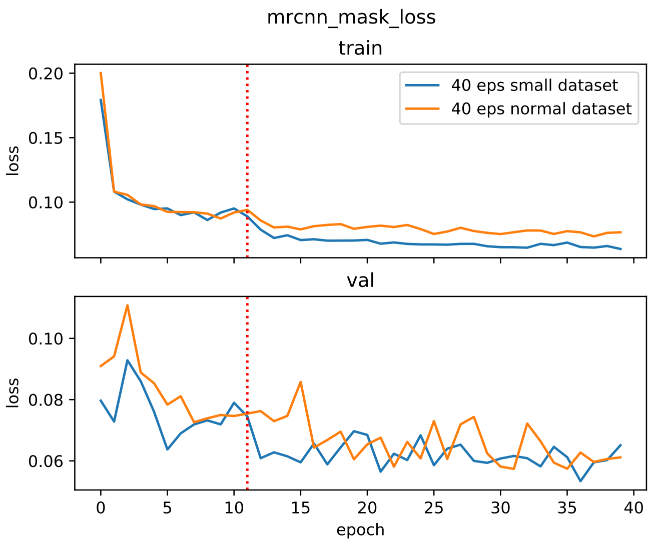


(b) Unsmoothed plot of mask loss

Figure A.1: Loss plots. The dotted red line indicates the boundary between heads only and fine-tuning training.



(a) Unsmoothed plot of class loss



(b) Unsmoothed plot of mask loss

Figure A.2: Loss plots of the low-data experiment. The dotted red line indicates the boundary between heads only and fine-tuning training.

Bibliography

[Abd17] W. Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. https://github.com/matterport/Mask_RCNN. 2017 (cit. on pp. 18, 21, 25).

[BM92] P. J. Besl, N. D. McKay. “Method for registration of 3-D shapes”. In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606 (cit. on p. 15).

[CMS11] A. Collet, M. Martinez, S. S. Srinivasa. “The MOPED framework: Object recognition and pose estimation for manipulation”. In: *The international journal of robotics research* 30.10 (2011), pp. 1284–1306. DOI: 10.1177/0278364911401765 (cit. on p. 15).

[CWS+15] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, A. M. Dollar. “Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols”. In: *arXiv preprint arXiv:1502.03143* (2015) (cit. on p. 45).

[fbAda18] forked by Adam Kelly. *Mask R-CNN fork to support Tensorflow 2*. https://github.com/akTwelve/Mask_RCNN. 2018 (cit. on p. 25).

[FFZ20] X.-W. Feng, D.-Z. Feng, Y. Zhu. *Fast Coherent Point Drift*. 2020. arXiv: 2006.06281 [cs.CV] (cit. on p. 48).

[GDDM14] R. Girshick, J. Donahue, T. Darrell, J. Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV] (cit. on pp. 16, 17).

[Gir15] R. Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV] (cit. on p. 17).

[GIW+17] C. Garcia Cifuentes, J. Issac, M. Wüthrich, S. Schaal, J. Bohg. “Probabilistic Articulated Real-Time Tracking for Robot Manipulation”. In: *IEEE Robotics and Automation Letters (RA-L)* 2.2 (Apr. 2017), pp. 577–584. DOI: 10.1109/LRA.2016.2645124. URL: https://doi.org/10.1109/LRA.2016.2645124 (cit. on p. 30).

[HGDG18] K. He, G. Gkioxari, P. Dollár, R. Girshick. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV] (cit. on pp. 13, 16–18, 25).

[HHC+11] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, V. Lepetit. “Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes”. In: *2011 international conference on computer vision*. IEEE. 2011, pp. 858–865 (cit. on p. 45).

[HZRS15] K. He, X. Zhang, S. Ren, J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on pp. 17, 19).

[IW+16] J. Issac, M. Wuthrich, et al. *Depth Based Object Tracking Library (dbot)*. https://github.com/bayesian-object-tracking/dbot. 2016 (cit. on pp. 15, 19, 30, 47).

[IWC+16] J. Issac, M. Wuthrich, C. G. Cifuentes, J. Bohg, S. Trimpe, S. Schaal. “Depth-based object tracking using a Robust Gaussian Filter”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)* (May 2016). DOI: 10.1109/icra.2016.7487184. URL: http://dx.doi.org/10.1109/ICRA.2016.7487184 (cit. on pp. 13, 15, 16, 19, 20, 41).

[JWC+20] A. B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, Z. Rui, J. Borovec, C. Vallentin, S. Zhydenko, K. Pfeiffer, B. Cook, I. Fernández, F.-M. De Rainville, C.-H. Weng, A. Ayala-Acevedo, R. Meudec, M. Laporte, et al. *imgaug*. https://github.com/aleju/imgaug. Online; accessed 01-Feb-2020. 2020 (cit. on pp. 33, 44).

[Ken] Kenta-Tanaka et al. *probreg*. Version 0.1.6. URL: https://probreg.readthedocs.io/en/latest/ (cit. on pp. 28, 48).

[LMB+15] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, P. Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV] (cit. on p. 32).

[Ope15] OpenCV. *Open Source Computer Vision Library*. 2015 (cit. on p. 24).

[RHGS16] S. Ren, K. He, R. Girshick, J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (cit. on pp. 16–18).

[RL18] M. Rad, 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] (cit. on p. 15).

[Sch20] T. Schäfer. *Tracking Pipeline for Baxter Robot with DBOT and RAI*. https://github.com/timschaeferde/rai_baxter. 2020 (cit. on p. 21).

[SEZ+14] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2014. arXiv: 1312.6229 [cs.CV] (cit. on p. 15).

[SQLG15] H. Su, C. R. Qi, Y. Li, L. Guibas. *Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views*. 2015. arXiv: 1505.05641 [cs.CV] (cit. on p. 15).

[SSB15] M. Schwarz, H. Schulz, S. Behnke. “RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 1329–1335. DOI: 10.1109/ICRA.2015.7139363 (cit. on p. 15).

[TFR+17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. arXiv: 1703.06907 [cs.RO] (cit. on p. 33).

[Tou18] M. Toussaint. *RAI bare code*. https://github.com/MarcToussaint/rai. 2018 (cit. on pp. 16, 22).

[WPK+13] M. Wuthrich, P. Pastor, M. Kalakrishnan, J. Bohg, S. Schaal. “Probabilistic object tracking using a range camera”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Nov. 2013). DOI: 10.1109/iros.2013.6696810. URL: http://dx.doi.org/10.1109/IROS.2013.6696810 (cit. on pp. 15, 19).

[WSH+19] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, L. J. Guibas. *Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation*. 2019. arXiv: 1901.02970 [cs.CV] (cit. on p. 15).

[WXZ+19] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, S. Savarese. *DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion*. 2019. arXiv: 1901.04780 [cs.CV] (cit. on pp. 13, 15, 38, 45, 46).

[Wyn20] K. Wynn. *pyquaternion*. Version 0.9.9. Oct. 15, 2020. URL: http://kieranwynn.github.io/pyquaternion/ (cit. on p. 28).

[XGD+17] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He. *Aggregated Residual Transformations for Deep Neural Networks*. 2017. arXiv: 1611.05431 [cs.CV] (cit. on p. 19).

[XSNF18] Y. Xiang, T. Schmidt, V. Narayanan, D. Fox. *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*. 2018. arXiv: 1711.00199 [cs.CV] (cit. on pp. 13, 15, 37, 38, 45, 46).

[ZPK18] Q.-Y. Zhou, J. Park, V. Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: *arXiv:1801.09847* (2018) (cit. on pp. 27, 28).

All links were last followed on November 15, 2020.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature