MASTER THESIS

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Mechatronics/Robotics

Semi-Automatic Generation of Training Data for Neural Networks for 6D Pose Estimation and Robotic Grasping

By: Johannes Nikolaus Rauer, BSc.

Student Number: 1710331019

Supervisors: Wilfried Wöber, MSc.
Prof. (FH) Dr. mont. Corinna Engelhardt-Nowitzki

Wien, September 2, 2019

FH University of Applied Sciences
TECHNIKUM WIEN

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Wien, September 2, 2019 Signature

Kurzfassung

Produktionsstätten setzen seit den 1960er Jahren erfolgreich klassische fest programmierte Roboter in der Massenproduktion ein, wo diese trotz fehlender Sensoren zur Umgebungswahrnehmung sicher agieren können. Die produzierende Industrie zeigt einen Trend weg von der Massenproduktion hin zu hoch individualisierten Produkten, die mehr Flexibilität und Automatisierung erfordern. Der Einsatz mobiler Manipulatoren, eine Kombination aus mobilen Robotern und Knickarmrobotern, für innerlogistische Transportaufgaben verspricht diese gewünschte Modularität. Da die Genauigkeit der Navigation von mobilen Robotern nicht ausreicht, um Objekte zuverlässig greifen zu können, benötigen sie Sensoren um Objekte und deren 6D Pose autonom zu erkennen. Die vielversprechendsten Ansätze für die Bestimmung von Objektposen sind Machine-Learning-Methoden, die auf Kameradaten angewandt werden. Diese werden mithilfe annotierter Ground-Truth-Daten trainiert – Bilder des Objekts und Informationen zu dessen Pose. Aktuelle Methoden zur Erstellung solcher Daten verwenden an den Objekten angebrachte Marker, die in umständlicher Nachbearbeitung entfernt werden müssen oder benötigen menschliche Annotatoren zur Ausrichtung von 3D-Modellen in Videos.

In dieser Arbeit wird ein Ansatz zur semi-automatischen Erzeugung von Trainingsdaten mit annotierten 6D Posen vorgestellt. Dabei wird ein Knickarmroboter verwendet, um mehrere Bilder eines Objektes mit nur minimaler manueller Interaktion zu erstellen. Zunächst wird die Position des Objekts mithilfe eines Markers, der auf dem Objekt platziert wird, in Bezug zur Basis des Roboters bestimmt. Nach dem Entfernen des Markers können Trainingsbilder aus verschiedenen Perspektiven aufgenommen und annotiert werden, ohne dass 3D-Modelle oder Nachbearbeitungsschritte erforderlich sind.

Datensätzen unterschiedlicher Art und Größe (reale, photometrisch erweiterte, synthetische Daten) werden erstellt und verwendet, um ein neuronales Netz zur Bestimmung von Posen zu trainieren und hinsichtlich Trainingszeit und Genauigkeit zu analysieren. Kleine Datensätze, die in der Zieldomäne aufgezeichnet und photometrisch erweitert werden (Data Augmentation), führen zu höheren Genauigkeiten als umfangreichere synthetische Datensätze. Eine höhere Anzahl von Epochen und größere Datensätze steigern die Trainingszeit, führen jedoch aufgrund von Overfitting und falscher Generalisierungen nicht unbedingt zu höheren Genauigkeiten. Die Ergebnisse bestätigen, dass vortrainierte Modelle besser abschneiden als Netze, die mit zufälligen Gewichten initialisiert werden. Eine qualitative Analyse zeigt, dass ein mobiler Manipulator, der das vorgestellte System zur Posen-Bestimmung verwendet, in realen Logistikanwendungen eingesetzt werden kann, um den Automatisierungsgrad zu erhöhen.

Schlagworte: 6D Lagebestimmung, Autonomes Greifen, Deep Learning, Datensatzerstellung

Abstract

Production facilities have successfully deployed classic fixed-programmed robots since the 1960s. Due to their inability to perceive the environment, such robots have mostly been used in mass production, where a static physical setup can be assumed. The production industries' move away from mass production towards highly customized goods, requires increased flexibility and automation. Deploying mobile manipulators, a combination of mobile and articulated robots, for intra-logistical transport tasks, promises this desired modularity. Since the accuracy achieved by mobile robot navigation is not sufficient to reliably grasp objects, robots need sensors to perceive their surroundings and autonomously detect objects and their 6 degree-of-freedom pose. The most promising approaches for object pose estimation and grasp-detection are machine-learning-based methods applied to camera data. Deep neural networks are trained using annotated ground-truth data – images showing the object and information of its pose. State-of-the-art methods for creating such data use marker-boards rigidly attached to the objects, which have to be removed in cumbersome post-processing, or need human annotators that align accurate 3D models to video-streams.

In this thesis, an approach to semi-automatically generate 6D pose-annotated training data is presented, using an articulated robot to create multiple images of an object with only minimal manual interaction. The pose of the object with respect to the robot's base is first determined using a marker placed on the object of interest, viewed by the robot's camera. After removing the marker, training images from different perspectives can be recorded and accurately labeled using the transformations between the robot's base and the camera, without a need for 3D models or post-processing steps.

A publicly available neural network for pose estimation is selected and trained using datasets varying in size and type (real, augmented, synthetic data). The trained network models are analyzed regarding different dataset parameters and training times in terms of pose estimation accuracy. Small datasets recorded in the target domain and supplemented with augmented images achieve more robust results than larger synthetic datasets. Bigger datasets and a higher number of epochs increase training time but not necessarily accuracy, due to wrong generalizations and overfitting. Results verify that pre-trained models achieve better results than networks initialized with random weights. A qualitative evaluation confirms that a mobile manipulator using the proposed pose-estimation system could be deployed in real-life logistics applications to increase the level of automation.

Keywords: 6D pose estimation, Robotic grasping, Deep learning, Dataset generation

Acknowledgements

I want to thank all who have supported me throughout my years of study and in writing this thesis. Thank you to my master thesis supervisor Wilfried Wöber MSc, who always helped me with constructive feedback and helpful solutions. Willie's door was always open whenever I ran into a trouble spot or had questions about my research or writing. I would also like to acknowledge Prof. (FH) Dr. mont Corinna Engelhardt-Nowitzki as the second supervisor for her very valuable comments on this thesis. I would further like to express my gratitude to all colleagues who made my years of study an extraordinary time and whom I became friends with. My special thanks go to my family and my girlfriend Silvana for providing me with support and encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without you.

Thank you!

Contents

1 Introduction

Since the 1960s, classic fixed-base industrial robots have been deployed in factories for performing repetitive tasks [1]. These fixed-programmed robots are unable to perceive their environment and have mainly been used in mass production where a static physical setup can be assumed. Adapting control-programs to changing conditions or new products and workflows requires expert knowledge and is therefore expensive [2, 3]. The production industry shows a trend away from mass production towards highly customized goods. To enable flexible productions, versatile robots that can react to changing environmental conditions are required [4, 5, 6]. For instance, the production of cars has been highly individualized in recent years, which involves delivering many different parts to the production line just in time [5]. This task is mostly performed by warehouseman since searching, grasping, and transporting specific objects requires high flexibility and cognitive skills [7]. Thus, the level of automation of part handling during the assembly processes in the automotive industry is only 30% [5].

Mobile manipulators increase automation when being deployed for such intra-logistical transport tasks. To enable human cooperation, mobile manipulators consisting of autonomous mobile transport vehicles and sensitive articulated robots are commonly used. Due to the combination of mobile and articulated robots, grasping individual objects all over a shop floor and transporting them to a goal position is possible [8, 4, 7].

Localization of mobile robots accurate to a few centimeters and degrees is possible using well-established methods for path planning and navigation [9, 10]. These accuracies are not sufficient for articulated robots to grasp objects reliably. Thus, robots need sensors to perceive their environment and autonomously detect objects and their 6 degree-of-freedom pose [11, 12]. Raw data from cameras is simply a grid of numbers to the robot, which has to be interpreted to extract high-level semantic information, such as locations and orientations of objects as well as possible grasp positions [13]. Due to that, object pose estimation and grasp-detection are necessary skills for flexible robots. Furthermore, pose estimation plays a crucial role in many other areas, such as augmented reality or autonomous driving [13].

Numerous publications are addressing the problem of pose estimation and robotic grasping in different ways. Most promising results are delivered by learning-based methods applied to camera data [13]. Deep neural networks are trained using large amounts of annotated ground-truth data – images showing the object and information of its pose [14]. There exist some datasets containing training data of standard objects [15, 16, 17, 18, 19] and different methods for generating labeled images. Some tools try to fit a 3D model in RGB-D videos to obtain labels for each frame. These techniques require an accurate 3D model and human annotators and lead to redundant samples due to videos being used [20, 21, 22]. To increase automation of the

labeling process, markers or calibration boards rigidly attached to objects are commonly used. These markers can involuntarily "help" the pose estimation algorithms and therefore have to be removed in cumbersome post-processing steps [23, 19, 24, 25, 26]. For complete automation, some approaches use multiple robots performing trial-and-error experiments [27, 28], but such generated data can only be used for training grasp-systems, not pose-estimation systems.

In this thesis, an approach for semi-automatic generation of 6D-pose-annotated training data is presented, using an articulated robot equipped with a camera at the end of its kinematic chain. The pose of the object with regard to the robot's base is first determined, using a marker placed on the object of interest. After removing the marker, annotated images from different perspectives can be recorded. As long as the object is not moved during this process, its pose can be calculated for every image, using the transformations between the robot's base and the camera. This process does not need any 3D models or cumbersome post-processing for removing markers from images and involves only minimal manual interaction for placing and removing a marker at the beginning of data recording. Furthermore, datasets recorded using this technique can also be used for 2D object detection or semantic segmentation since necessary ground-truth data is provided.

Different approaches for 6D object pose estimation are analyzed and compared. A publicly available network is selected and trained using diverse datasets. It is analyzed how the performance of the pose-estimation system is influenced by factors such as pre-training and dataset-size. Moreover, the effects of training using synthetic (computer rendered) images and data augmentation are evaluated and verified using different objects. By analyzing these dataset factors and training time in terms of pose estimation accuracy, guidelines for composition, size and type of training datasets for neural networks are devised.

This work is structured as followed: In Chapter 2, the state of the art of robotic grasping and pose estimation are presented, focusing on deep-learning-based methods. The performance of different approaches is reviewed considering various error metrics and a system for further implementation and evaluation is selected. Chapter 3 gives an overview of different types of training data for neural networks and discusses problems and solutions for domain gaps. Dataset formats and approaches for generating ground-truth annotated data are explained. In Chapter 4, the procedure for creating images with 6D ground-truth labels is presented and the software architecture of the recording-pipeline is explained. The mobile manipulator used for capturing training data and evaluation are introduced. In Chapter 5, an overview of the methods for synthetic data generation and data augmentation is given. The different generated datasets and their biases regarding scene diversity and pose distribution are analyzed and the training parameters are discussed. Moreover, this chapter presents the software architecture for the quantitative and qualitative evaluations. Chapter 6 provides an analysis of the accuracy of the calculated ground-truth pose from markers. Furthermore, the results of the evaluations are discussed and guidelines regarding optimal dataset-parameters and training time are given. Finally, Chapter 7 summarizes this work and gives directions for future research.

2 Robotic Grasping and Pose Estimation

In order for robots to interact effectively with their environment, they must be able to recognize their surroundings and grasp individual objects. To enable autonomous operation, awareness of the 3D position and orientation – the 6 degree-of-freedom pose – is necessary [29].

In this section, the state of the art of robotic grasping and pose estimation are reviewed. Different fundamental strategies for pose estimation are presented, focusing on deep-learning-based approaches in section 2.2.3. Furthermore, state of the art methods for 6D pose estimation are compared. One approach for further implementation in this work is selected and presented in more detail. Finally, different evaluation metrics for robotic grasping and pose estimation are discussed and compared.

2.1 Robotic Grasping

Grasping individual objects is a challenging task to perform with robots since it involves perception and understanding of a scene, as well as planning and controlling a grasping movement [30, 31]. Hence, robotic grasp-detection systems lag far behind human performance levels [32]. Grasp-detection is a visual recognition problem in which a robot's sensors are used to detect graspable objects in an environment and determine corresponding grasping poses to maximize a success metric [30, 33]. The general procedure for robotic grasping can be divided into object localization, pose estimation, grasp point determination and motion planning, as visualized in Figure 1.

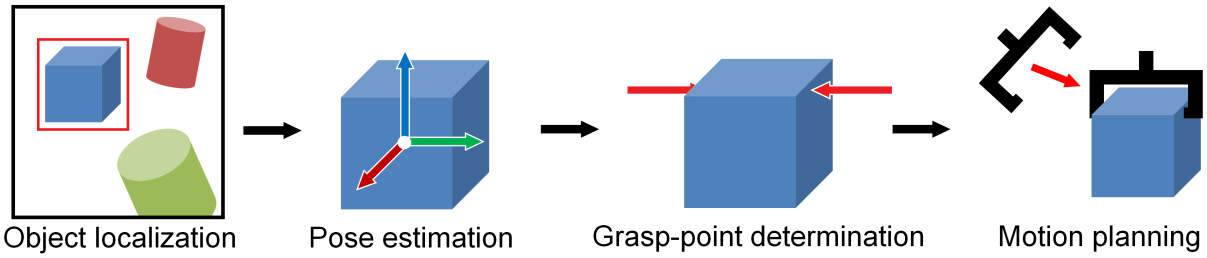


Figure 1: Grasping procedure: Consisting of object localization, pose estimation, grasp-point determination and motion planning [13].

A grasping pose describes a configuration of the end-effector in a given image plane to pick up an object [12], by partial or complete form- or force-closure [34]. Since a comprehensive survey of robotic grasp-detection systems is out of the scope of this work, it may be referred to

the standard survey on this subject by Bohg et al. [35]. This section gives a short overview of basic categorizations and approaches in the field of robotic grasping, as visualized in Figure 2.

Traditionally, robotic grasping has been limited to predefined objects at specific locations. Algorithms for grasping such objects have been implemented manually with prior knowledge [12]. Especially for mobile robots that are deployed in a flexible production or in highly unstructured environments, neither predefined objects nor specified locations can be assumed, which makes grasp-detection systems necessary.

Grasp-detection methods can be grouped according to the knowledge of the query object into approaches for grasping *known*, *familiar* and *unknown* objects [35]. Systems that are designed for grasping *known* items commonly provide a database containing object models with associated good grasps. The workflow in such systems usually consists of scene segmentation and object recognition to estimate the pose of the object and retrieving a proper grasp from the database. *Familiar* objects can share visual features such as shape, color and texture, or can be defined based on their object category. Approaches enabling grasping of familiar objects need to find an object-category representation and a similarity metric to allow the transfer of grasp knowledge. Approaches for grasping *unknown* objects work without models or any sort of grasp experience. These methods try to identify the structure of perceived features and analyze heuristics that link this structure to grasp candidates [35].

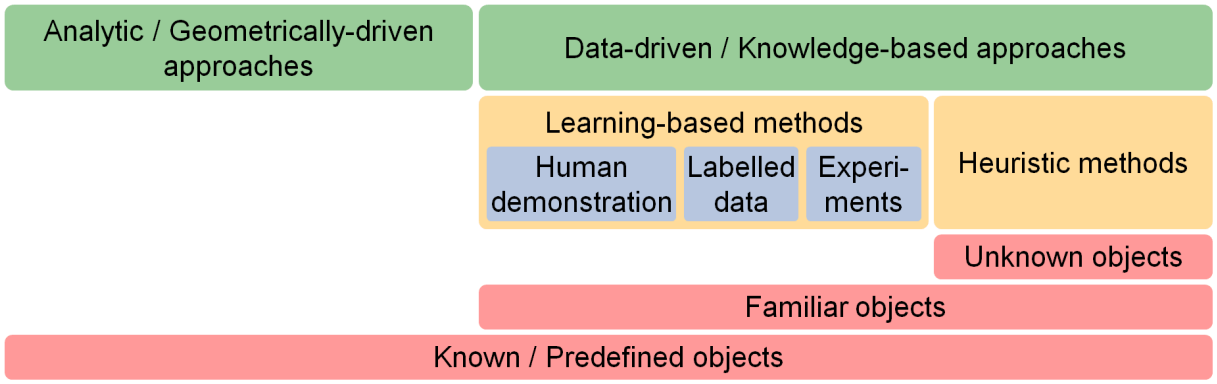


Figure 2: Categorization of grasping approaches into analytical and data-driven methods for known, familiar and unknown objects.

Grasp-detection approaches can furthermore be categorized into *analytic or gemoetrically-driven*, as well as *data-driven or knowledge-based* methods. *Analytic* approaches use knowledge of the 3D geometry and a contact model to find grasps that maximize a particular metric considering criteria such as dexterity, stability and dynamic behavior [35, 31]. Since availability of complete object knowledge is necessary [28] and typically restrictive assumptions have to be made, the popularity of data-driven methods has risen since the early 2000s [36, 35]. *Data-driven methods* do not rely on understanding the physics of an object [36] since they sample and rank grasp candidates by comparison to existing grasp knowledge [35]. *Learning-based* approaches are the most common data-driven methods and are usually trained with labeled color or depth images, human demonstrations or by performing trial-and-error experi-

ments [35]. Although good grasp performance for known and familiar objects is possible using such attempts, these approaches require hand-engineering to design input features that are able to generalize to unknown objects [32]. Thus, grasp-detection systems for unknown objects typically use heuristics, which link structures of the sensor data directly to possible grasps [35].

Amongst all methods there exists a variety of critical aspects which have to be considered when developing or using a robotic grasping system: During *data acquisition*, sensor and segmentation errors should be reduced since they affect the performance directly [13]. By obtaining high-level information using *semantic perception*, better segmentation and estimation of surface parameters and weight information is possible [13]. To further increase success probability, *closed-loop architectures* such as [37] that use feedback instead of one-shot estimations are necessary, which furthermore enable robots to react to pose-modifications while already reaching for an object. Another central problem of learning-based methods is the high number of *training data*, which can partially be solved using simulated data [13]. Furthermore, due to the complexity of most grasp-detection methods, processing in real-time is only possible using special approaches [38] and high computational power [12, 32].

A goal of this work is developing a grasp-detection system for predefined objects, why a new learning-based approach is developed. One main component of such systems is pose estimation to calculate gripper configurations. Therefore, in the following section, different methods for 6D pose estimation are presented.

2.2 6D Pose Estimation

Numerous approaches for 6 degree-of-freedom (DoF) object pose estimation have been developed for problems such as robotic manipulation and augmented reality (AR). While in the field of AR precision is not as crucial as avoiding jittering, high accuracy is essential when manipulating objects [26, 13].

Estimating the pose of an object is the task of *object detection*, followed by *pose estimation*. Object detection involves localization and classification. It aims to find an estimation vector (c, x, y, h, w) where c corresponds to an object's class provided by classification and (x, y, h, w) specify the location and size of an object respectively in image coordinates provided by localization [39]. The pose P of a rigid 3D object is typically presented by a 4×4 matrix $P = [R, t; 0, 1]$ with R being a 3×3 rotation matrix and t a 3×1 translation vector [40]. The goal of pose estimation is the calculation of the most likely pose $\hat{P}$ of an object, which is the closest pose to the ground-truth pose $\tilde{P}$ [11, 41].

2.2.1 Regression and Classification

Pose estimation can be formulated as a classification and regression problem. Classification means that all possible configurations are quantized into different classes, while regression tries

to regress a pose that fits the object's pose best [39]:

$$P = [R, t; 0, 1] \qquad \begin{array}{ll} \textbf{Regression}: & \textbf{Classification}: \\ R \in \mathbb{R}^{3\times3} & R \in \{R_0, R_1, ..., R_n\}, R_i \in \mathbb{R}^{3\times3} \\ t \in \mathbb{R}^{3\times1} & t \in \{t_0, t_1, ..., t_n\}, t_i \in \mathbb{R}^{3\times1} \end{array} \tag{1}$$

It seems natural to use regression since rotations live in a continuous space. Though, especially when estimating the pose of symmetrical objects, ambiguities are possible due to identical-appearing views being assigned to different poses in ground-truth data [42]. Special attention also has to be given to equivalences. E.g., a rotation of θ about ω is equivalent to the rotation $2\pi - \theta$ about $-\omega$, but these forms resolve to different (negative) unit quaternions [43]. If a rotation matrix is estimated, it has to be paid special attention since not all 3×3 matrices are valid rotation matrices. It is possible to use Lie-Algebra as proposed by Do et al. [44] to represent a 3D rotation matrix with only three scalar values without any constraints. Due to the listed limitations, regression approaches have not been very successful in the field of 6D pose estimation and experiments show that classification is more reliable [42, 45].

On the other hand, quantizing poses leads to over 50,000 classes if rather coarse intervals of 5° are used. Learning-based approaches often have problems converging since each class is represented sparsely in the training dataset [42]. Due to that, Kehl et al. [45] propose decomposition of a 6D pose into viewpoint and in-plane rotation with 642 different viewpoints and 19 in-plane rotations at an interval of 5°. However, this representation can lead to ambiguous class combinations since changes of the viewpoint are challenging to distinguish from small in-plane rotations [42].

Oñoro-Rubio et al. [46] observed that classification models are more likely to estimate poses at the opposite side, while regression-based approaches tend to accumulate errors at nearby poses which is not as important as errors associated with opposite poses. Though, if a small and in terms of pose annotations poorly balanced training dataset is available, classification methods perform better.

Due to coarse intervals between different poses at classification approaches, as well as to correct imprecisions occurring at regression methods, pose refinement using iterative closest point algorithms (ICP) is commonly performed [45]. ICP has been initially developed by Besl & McKay [47] and is one of the most widely used algorithms for three-dimensional model registration and point matching when an initial guess of the target transformation is available. One point cloud is kept fixed while the other one is iteratively transformed to minimize an error metric, usually the sum of squared distances between the matched points [48].

2.2.2 Methods for Pose Estimation

Approaches for pose estimation can be categorized into *template-based*, *feature-based*, *voting-based* and *learning-based* methods [49, 50] (see Figure 6). This section is intended to give an overview of fundamental strategies for solving pose estimation problems. Learning-based methods are presented in more detail in section 2.2.3.

Since feature-, template- and voting-based approaches usually use the same evaluation metrics, these methods can be compared as followed: *Feature-based* methods can only be applied for pose estimation of textured objects and are fairly robust to occlusions due to the local search strategy. Although they need high computational times (up to 50 seconds), they often do provide low accuracy. *Template-based* methods, which are also able to detect the pose of texture-less objects, do only need about one-third computational time of feature-based approaches and gain about ten times more accuracy, but are vulnerable to occlusions. *Voting-based* methods show slightly better results than template-based approaches in about 1/10 to 1/20 of the time needed for feature-based methods (2 to 5 seconds) and are also suitable for occluded texture-less objects [13].

Template-based Approaches

The most traditional approaches are *template-based*. These methods use a rigid template with assigned 6D poses which is compared to an object region in different scales and orientations, trying to maximize a similarity score to find the best match, commonly followed by pose refinement [49, 51, 50] (see Figure 3).

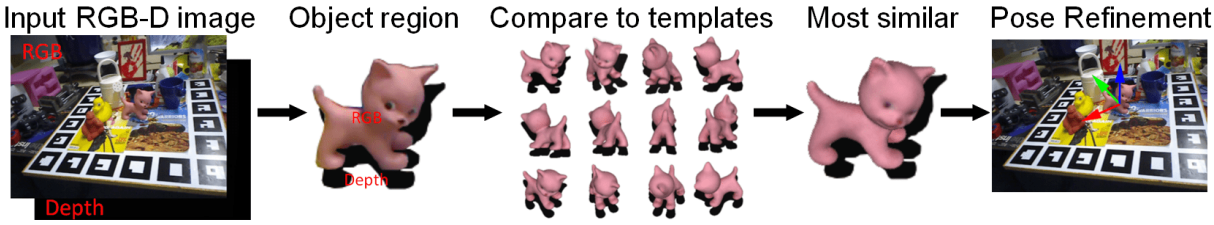


Figure 3: Template-based methods: Object poses are estimated by comparing image regions to template views [13]. Image source: Du et al. [13].

Template-based approaches are suitable for texture-less and texture-poor rigid objects. Since a complete template is matched, most algorithms cannot handle deformable objects, changing lighting conditions or occlusions. They show weak performance for cluttered scenes and high number of different classes since a large number of templates is needed, which leads to higher computational times [49, 52, 16]. Hence, approaches try to increase robustness for illumination and object shape by using robust feature descriptors [53, 54], speed up matching by deploying hierarchical tree structures of templates [55, 56] and improve accuracy in cluttered environments using RGB-D sensors [57, 51].

Feature-based Approaches

Feature-based or also called correspondence-based methods extract points of interest in an input image, construct local descriptors and match this set of features with a database to find correspondences [49, 26], as illustrated in Figure 4.

Since texture-less objects usually do not provide descriptive feature points in sufficient number, these approaches only work for textured objects, but are robust to occlusions and show reasonable performance [49, 26, 52, 50]. The various methods differ depending on the avail-

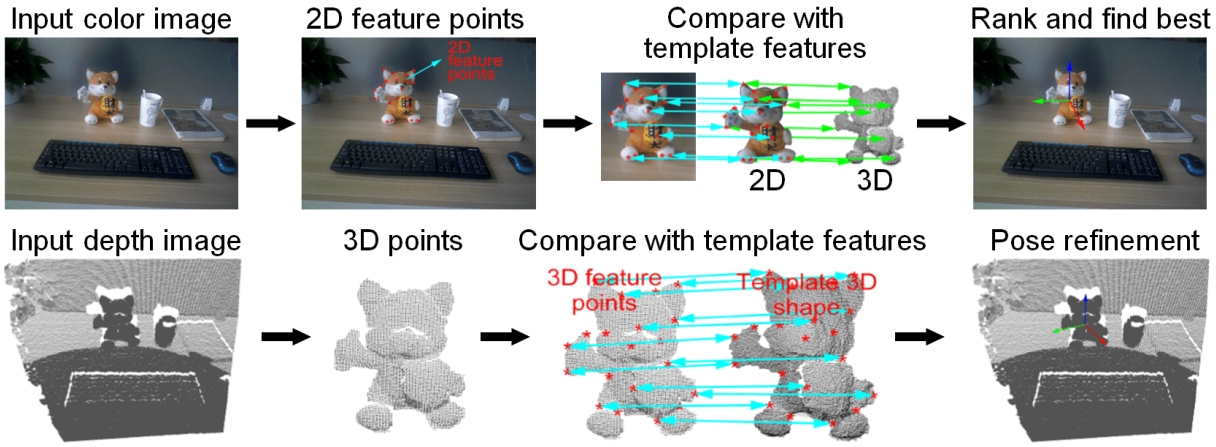


Figure 4: Feature-based methods: Extracted keypoints are matched to template features to find the best pose in RGB-images (top) and depth-images (bottom) [13]. Image source: Du et al. [13].

able input data. For RGB images, rendered images of 3D models from various angles are used for matching 2D feature points and a Perspective-n-Point (PnP) algorithm [58] is applied subsequently to retrieve the 6D pose from the projected 2D points. If 3D point clouds are available, the full object is matched with the partial 3D data to obtain a rough pose which is then refined using an iterative closest points (ICP) algorithm [13].

Voting-based Approaches

Voting-based methods are mainly used for pose estimation in scenes with occlusions since every single pixel or image patch casts a prediction about the output [13]. Basic strategies of voting-based approaches for RGB-D and depth-images are presented in Figure 5.

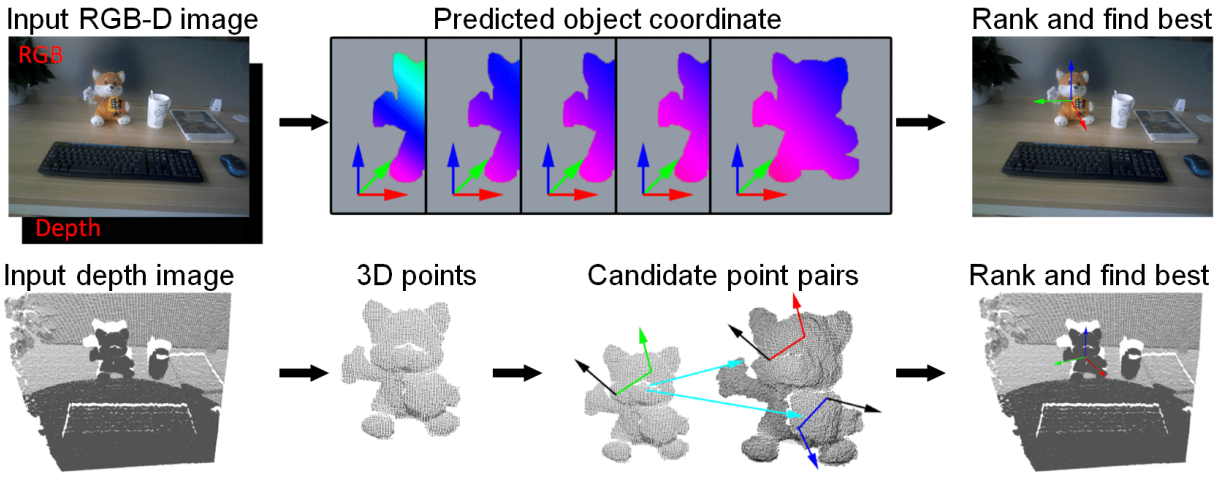


Figure 5: Voting-based methods: Each pixel or image patch votes for the prediction outcome using RGB-D (top) or depth images (bottom) [13]. Image source: Du et al. [13].

For RGB-D data, Brachmann et al. [49] propose a combined dense 3D object coordinate and class prediction to generate pose hypotheses. Great success has been shown for depth-only

data, using point pair features that contain the distance and normals of two arbitrarily chosen 3D points [59, 60]. Wang et al. [61] use a novel neural network for extracting pixel-wise dense features from RGB and depth data separately and use the information combined for estimating a 6D pose.

Learning-based Approaches

The idea behind *learning-based* approaches is to use images of an object from various viewpoints to learn an abstract representation of this object. Given a new input image, the object's pose can be determined using the learned model. Learning-based methods show reasonable performance for textured and texture-less objects, even under partial occlusions, but need a high number of training images [26, 13]. Most approaches are computationally expensive, but calculation in real-time is feasible on high-end GPUs [39]. More detailed information on learning-based pose estimation approaches is given in section 2.2.3.

Pose Estimation using Fiducial Markers

Since most presented methods need high computational power and effort to detect the pose of new objects, some pose estimation approaches make use of *fiducial markers* for 6d pose estimation. This is only possible if a marker can be attached to an object permanently [62, 63]. If the pose of a new object is to be estimated, it is only necessary to attach a marker to it and determine the rigid transformation between marker and object.

A widely used system is the OpenSource library ArUco [64], which is able to detect the ID and estimate the pose of a square marker from a given 2d image using the camera's calibration matrix. This algorithm extracts the most prominent contours from the image and tries finding 4-sided polygons. By computing the homography matrix, perspective projection is removed from all polygons and the marker's ID can be calculated by evaluating the mean pixel values of all grid-elements of the equalized image segment. If this ID can be found in the used dictionary, the corners of the marker are refined. To estimate its pose with respect to the camera, the reprojection error of the corners is iteratively minimized [65]. To deal with occlusions, marker-boards with a grid of multiple markers are used to increase detection likelihood [65]. Classical computer vision methods, as described above, often fail at poor lighting conditions or for blurred images. Therefore, Hu et al. [66] propose a learning-based approach that extracts marker locations and ids using deep neural networks and retrieves the pose using a PnP algorithm.

2.2.3 Neural Networks for Pose Estimation

Deep learning-based methods for 6D pose estimation have become popular since the introduction of deep convolutional neural networks (CNNs) [35]. Compared to hand-crafted methods such as feature-, template- or voting-based approaches, better and more robust results can be obtained using deep learning approaches. However, these methods need a large number of training data to generalize. Like hand-crafted methods, they show poor performance on objects

which are not included in the training dataset [13]. As visualized in Figure 6, learning-based approaches can be categorized into direct and indirect methods [35]. For the sake of completeness, Figure 6 shows all discussed types of approaches for pose estimation.

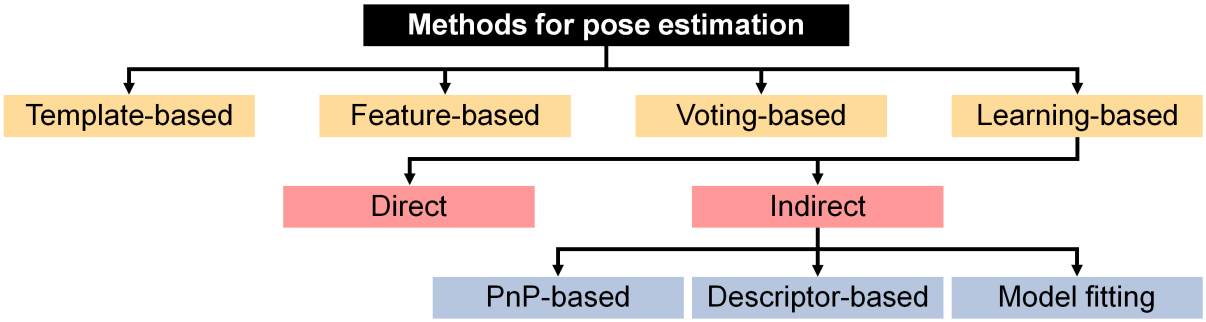


Figure 6: Methods for pose estimation: template-, feature-, voting- and learning-based approaches. Learning-based approaches can be categorized into direct and indirect methods – including PnP-based, descriptor-based and model-fitting methods [13].

Direct approaches use a deep neural network to estimate object poses directly from an input image without intermediate representations or steps, why they are also called one-stage methods. *Indirect* or multi-stage methods on the other hand use neural networks to predict intermediate cues, using semantic segmentation, keypoint extraction or descriptor estimation, which are used to compute 6D poses [13, 67]. Based on these intermediate information, it can be differentiated between *PnP-based*, *descriptor-based* and *model-fitting methods*.

PnP-based approaches use a neural network for estimating 2D-projections of 3D keypoints, which are passed to a PnP (Perspective-n-Point) algorithm to estimate a 6D object pose. The Perspective-n-Point (PnP) problem has first been introduced by Fischler & Bolles in 1981 [68]. The goal of PnP algorithms is determining the pose of a camera given its intrinsic parameters and a set of n 3D points and their projected 2D correspondences in an image [58]. Therefore, given estimated 2D keypoints in an image and their 3D object location, the object's pose can be predicted solving the PnP problem.

Descriptor-based methods predict a low dimensional representation of a given input image and match it with pre-computed descriptors in a database to retrieve the pose [35]. *Model-fitting-based* approaches segment images using a neural network and match 3D models to this segmentation, typically using ICP-algorithms [47] (For an explanation of ICP see section 2.2.1). PnP-, as well as model-fitting-based approaches, do need extra time for calculating 6D poses. Especially ICP algorithms are computationally expensive [69, 70].

Direct Methods

As visualized in Figure 7, direct approaches for pose estimation use no intermediate steps when estimating object poses. They vary widely in terms of basic strategy. One of the most famous is PoseCNN [71], which decouples pose estimation into different components to explicitly model dependencies and independencies between them: For each pixel, an object label and an unit

vector pointing to the object's center is predicted. All pixels corresponding to the same object vote for the center location and translation from the camera. Assuming given camera intrinsics, the 3D object translation is calculable. 3D rotation represented by quaternions is regressed from the convolutional features extracted from the bounding box [71].

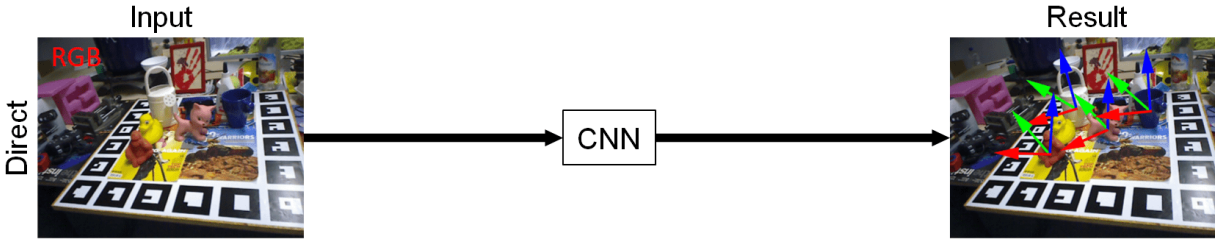


Figure 7: Direct methods for pose estimation: Object poses are estimated from input images without any intermediate steps [13]. Image source: Du et al. [13]

Another possibility is using a network for semantic segmentation to predict object masks in combination with a pose interpreter network, which estimates the objects' 6D pose from their silhouette. Since authentic silhouettes can easily be rendered from 3D models, training the pose interpreter network only on easily generatable synthetic data is possible [43]. Billings et al. [70] overcome problems provoked by occlusions by deploying a similar approach, which estimates an unoccluded silhouette in a parallel network branch for pose regression. The intersection of the two silhouettes can furthermore be used to calculated unoccluded grasp points. Do et al. [44] extend Mask R-CNN [72] with an additional branch for 6D pose estimation. This network estimates the translation by predicting the object's center position with respect to the camera and outputs three values representing the object's rotation matrix using Lie algebra. Another considerable method extends YOLOnet [73] for direct pose regression [69]. The approach by Kehl et al. [45] predicts in a single step very rough orientations and 2D bounding boxes whose sizes are used for depth estimation.

DenseFusion combines color and depth features into a new color-depth space, which is then used to estimate the object's pose. Also, an iterative self-refinement algorithm is proposed, which uses the last estimated pose to refine the current pose [61]. Most presented methods need labeled real-world training data for pose estimation or semantic segmentation. The approach by Rad et al. [74] requires only annotated depth images that can easily be generated synthetically. A depth feature extractor and a pose-estimator are trained on labeled depth images only. Furthermore, another deep network is trained on RGB-D images to learn mapping color-features to depth-features. At runtime, the features of a color input image are mapped to depth-space and the pose-estimator trained on synthetic depth data estimates the object's pose.

PnP-based Methods

Most indirect approaches are PnP-based. They detect objects of interest in an input image, calculate a set of sparse keypoints in 2D image coordinates – often represented as belief maps –

and perform a Perspective-n-Point (PnP) algorithm to compute their 3D coordinates and thereby the object's pose [29, 75, 76] (see Figure 8). Commonly, the 3D bounding box's corners are used as keypoints since they frame the object and are well spread in space [76].

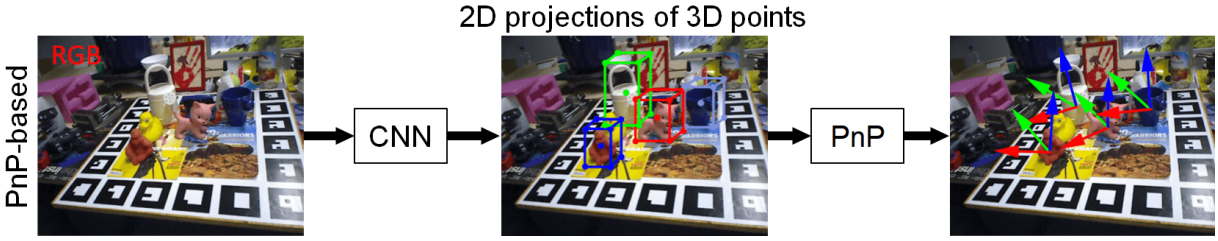


Figure 8: PnP-based methods for pose estimation: Neural networks are used for estimating 2D-projections of 3D keypoints. 6D poses are calculated using PnP algorithms [13]. Image source: Du et al. [13]

Since multi-stage methods separately estimate object type and keypoints, well-established and pre-trained object detection networks such as YOLOnet [73] can be used for former [77]. The networks for keypoint-extraction are often sensitive to occlusions, if not all keypoints are visible. To overcome this problem, not only the keypoints themselves can be used to estimate their position. The locations of the keypoints can be estimated by every visible object patch in an image together with a confidence value. These predictions are combined according to their score and the resulting keypoint locations can be used for a PnP algorithm [78]. Peng et al. [41] introduced PVNet, which uses a CNN to predict an object label and unit vectors that represent the direction from each pixel to all keypoints, together with a confidence score. An uncertainty-driven PnP algorithm (which solves the PnP problem considering the keypoints' confidence score) is then used for computing the pose. This method is robust against occlusions and truncations of keypoints since the points themselves do not have to be visible [41].

Another promising strategy is using correspondence-maps [79]. These maps are two-channel images with values between 0 and 255, which map an RGB-image to a 3D model. A deep network is trained to predict these correspondence-maps which are then used to estimate the object's pose using a PnP algorithm. This pose is then improved in a deep-learning model-based refinement-network, which compares the cropped image with a rendered image in the estimated pose [79]. This recently presented approach DPOD outperforms PoseCNN [71] by a large margin and performs similarly to PVNet [41] achieving an ADD (see Section 2.3) of 95.15 at average 33 fps. The RGB-based refinement method of DPOD also outperforms the quite similar method DeepIM [80].

Descriptor-based Methods

As Figure 9 shows, descriptor-based methods match a predicted representation of an input image with pre-computed descriptors to retrieve object poses.

Wohlhart et al. [22] use a neural network to calculate descriptors for RGB images and store them in a database together with an object label and the corresponding pose. These descriptors

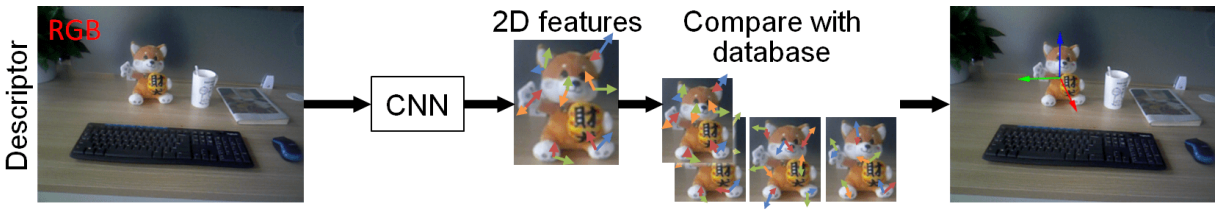


Figure 9: Descriptor-based methods for pose estimation: A low-dimensional representation of an input image is predicted and matched with pre-computed descriptors to retrieve object poses [13]. Image source: Du et al. [13]

efficiently capture the object identity and pose in a low dimensional space. The similarity-score to test images can be calculated using the Euclidean distance of the descriptors. Kehl et al. [81] propose a similar approach using RGB-D data but do not just calculate one descriptor per view and do not directly output the pose with the smallest Euclidean distance. They divide an input image into a regular grid, calculate descriptors and lookup the corresponding object pose for each cell. Then the pose estimations of all image patches are merged into a final object pose according to their confidence value. For generating the codebook containing object representations and related poses, Sundermeyer et al. [42] suggest using an augmented autoencoder. The great advantage of approaches that represent poses implicitly in a database is that identical appearing views – e.g., occurring at symmetric objects – are mapped to the same 6D pose in latent space [42].

Model-fitting Methods

Model-fitting based approaches for processing RGB-D data first perform a semantic segmentation on RGB images and crop the corresponding point cloud in the depth images. Then they align a 3D model using an ICP algorithm [25, 11] or match 2D keypoints with 3D object class models if no depth data is available [82] (see Figure 10).

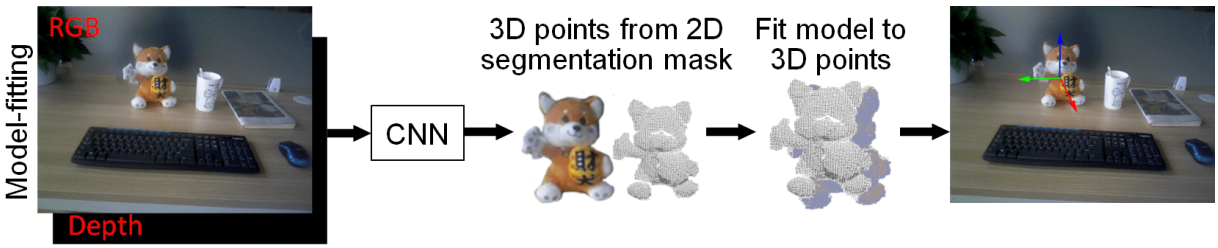


Figure 10: Model-fitting methods for pose estimation: Neural networks are used for semantic segmentation on RGB images. Detected objects are cropped in the point clouds and matched to 3D models to retrieve object poses [13]. Image source: Du et al. [13]

Pose estimation performance depends on high accuracy segmentation, but especially the output of neural networks for semantic segmentation trained on synthetic data only is error-prone. Therefore, Mitash et al. [83] introduced "soft" segmentation, which takes into account probability of the segmentation mask. By using only point-sets with high confidence scores,

more robust object fitting and pose estimation is possible. Increasing robustness can also be done by using multiple views of a scene and merging the pointclouds before model-fitting [84]. Sock et al. [85] propose a similar multi-view approach, but use a deep network for directly generating pose hypotheses for each view and predicting next-best-view positions to increase performance.

Comparison of Approaches

Different deep learning-based methods for 6D pose estimation and their evaluation metrics are listed in Table 1 and categorized according to the structure introduced above. The first three columns show author and approach-names (if given), as well as the year of publication and reference to the related paper. The download-links for the sourcecode are provided in Table 6 in Appendix A. Metrics are given for computation time (frames per second [fps]) and accuracy (ADD-metric - see section 2.3) as provided by authors. For the ADD-values, the highest reported values on the LINEMOD-dataset [16] are listed. Two star-symbols [**] show results for YCB-dataset [71]. One star-symbol [*] indicates that this value is reached after a refinement algorithm such as ICP. In these cases, also computation time is including refinement. An "x" in the corresponding column shows that the approach is direct or indirect with the sub-categories PnP-based [P], descriptor-based [D] or model-fitting-based [M]. The training data used by the authors is divided into real images [R], synthetic images [S] – including photorealistic, domain randomized or simply rendered data – and mixed-reality images [M] – which means that rendered objects are placed in front of real backgrounds (for details, see Section 3.1.2). The last column indicates if the method needs depth data [x], if processing depth data is possible [(x)] or if the methods are designed for RGB images [-].

Selection of a Pose Estimation Approach for Implementation

Intensive research shows that 12 of the approaches presented in Table 1 provide their sourcecode online. This is necessary for smooth implementation since this work focuses on training data generation and robotic grasping, not on neural network development. PoseCNN [71], DOPE [29] and the approaches by Wu et al. [43] and Zeng et al. [84] also provide a ROS-wrapper (Robot Operating System) for easy deployment on a robotic platform. All approaches which publish hardware-information test computational speed on high-end graphics processors such as NVIDIA GTX-1080Ti or TitanX. Mobile robots usually do not feature such high-performance graphics cards, which means that speeds listed in Table 1 unlikely can be reached. Since real-time performance is not required but preferable, the approaches by Zeng et al. [84], Brachmann et al. [86] and PoseCNN by Xiang et al. [71] are excluded despite latter stating high accuracies. The approaches by Tekin et al. [75] and Sundermeyer et al. [42] fail in terms of reliability since their detection accuracies are under 70%. SilhoNet [70] does not provide speed or accuracy rating and therefore, has not been further considered due to missing comparability. The remaining five networks would be suitable for robotic grasping on a mobile platform in terms of accuracy and computational-speed. Another goal of this work is to

discover how synthetic and real training data, as well as sizes of datasets affect the performance of pose estimation networks. DenseFusion by Wang et al. [61] and the approaches by [43] and [77] consist of multiple networks for semantic segmentation or keypoint extraction and pose estimation, which have to be trained separately. Besides higher training time, sources of performance-affecting influences would be impossible to identify why these methods are excluded. The remaining methods DOPE [29] and PVNet [41] meet all requirements and latter shows slightly better performance. However, the sourcecode of this method has been published after the network-selection process of this work. Without this, PVNet may have been selected for implementation, although it has never been tested with rendered training-data. Since DOPE by Tremblay et al. [29] meets all requirements and shows sufficient computational-speed and accuracy, this system has been chosen for further implementation and evaluation.

Selected Pose Estimation Network

The selected 6D object pose estimation approach *DOPE* (Deep Object Pose Estimation) has been published by Tremblay et al. in 2018 [29]. One main goal of their work is developing an integrated 6D pose estimation method for robotic grasping which can be trained on synthetic data consisting of photorealistic and domain randomized images (see Section 3.1.2). The proposed system consists of a neural network estimating keypoints, combined with a PnP-solver for object pose calculation. The network uses the first ten layers from VGG-19 [87] pre-trained on the ImageNet-dataset [88] for computing image features. In multiple convolutional stages, a sequence of belief maps – indicating 2D positions of keypoints – and vector fields – describing a vector pointing from each keypoint to the object's center – are estimated. The stages use increasingly larger receptive fields to leverage more context from stage to stage, considering the extracted image features as well as the outputs of previous stages (see Figure 11).

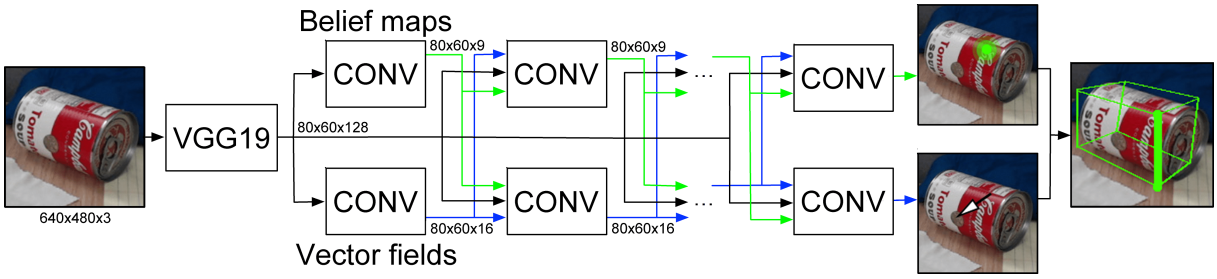


Figure 11: Structure of DOPE-Network: Belief maps (2D positions of corners or 3D bounding box) and vector fields (pointing from the corners to the centroid) are predicted in multiple stages to estimate a 6D pose using a PnP algorithm (Courtesy of Tremblay et al. [29]).

This network outputs nine belief maps – one for each corner of the object's 3D bounding box and its centroid – and eight vector fields – one for each corner pointing to the centroid. Local peaks in the belief maps above a threshold are searched to locate the corners. A greedy assignment algorithm associates these points to the closest centroid to assign them to an object. A PnP algorithm is used to finally calculate the object's pose from the projected vertices, given camera intrinsic parameters and object dimensions, if at least four vertices are detected [29].

Table 1: Overview of deep learning-based object pose estimation approaches: Reference to paper, author-name (method-name), year of publication. Computation time (frames per second [fps]), ADD-Metric for LINEMOD-dataset [16] (** for YCB-dataset [71], * pose is refined after network-processing). Background-colors indicate performance for each column (green: best, red: worst). Direct or indirect approach – PnP-based [P], Descriptor-based [D], Model-fitting-based [M]. Network trained on Real-data [R], Synthetic-data [S], Mixed-reality-data [M]. Depth-data is necessary [x], possible [(x)], RGB-only [-]. The top table shows approaches with publicly available source-code. For links to the code see Appendix A.

Approach			Metrics		Direct	Indirect			Training			Depth
			fps	ADD		P	D	M	R	S	M	
[61]	Wang et al. (DenseFusion)	2019	16	94.3	x				x			x
[86]	Brachmann et al.	2016	2	99*	x				x			(x)
[70]	Billings et al. (SilhoNet)	2018	-	-	x				x	x		x
[71]	Xiang et al. (PoseCNN)	2018	0.09	88.6*	x				x	x		(x)
[43]	Wu et al.	2018	20	-	x					x		-
[29]	Tremblay et al. (DOPE)	2018	20	80		x				x	x	-
[75]	Tekin et al.	2018	50	55.9		x					x	-
[77]	Zhao et al.	2018	25	72.6		x					x	-
[41]	Peng et al. (PVNet)	2018	25	86.3		x			x		x	-
[42]	Sundermeyer et al.	2018	5	64.7*			x		x	x		(x)
[84]	Zeng et al.	2017	0.05	-				x	x			x
[69]	Liu et al.	2019	55	-	x				x			-
[44]	Do et al.	2018	10	65.2	x				x			-
[74]	Rad et al.	2018	300	51.6	x				x	x		x
[85]	Sock et al.	2018	-	-	x					x		x
[45]	Kehl et al. (SSD-6D)	2017	10	79	x					x		(x)
[76]	Rad et al. (BB8)	2017	2	62.7*		x					x	-
[78]	Hu et al.	2018	20	32.9**		x					x	-
[79]	Zakharov et al. (DPOD)	2019	33	95.2*		x				x		-
[22]	Wohlhart et al.	2015	-	-			x		x	x		(x)
[81]	Kehl et al.	2017	1.5	-			x		x	x		x
[82]	Pavlakos et al.	2017	3.5	-				x	x			-
[25]	Wong et al. (SegICP)	2017	14	-				x	x			x
[11]	Xu et al.	2019	0.4	-				x	x			x
[83]	Mitash et al.	2018	1.7	-				x		x		x

2.3 Evaluation Metrics

The evaluation and therefore comparison of 6D object pose estimation- and robotic grasping systems is not straightforward. Due to object symmetries and (self)occlusions, different object poses are indistinguishable in a given image and thus should be treated equivalent [40]. Since grasping objects involves object detection and pose prediction, following evaluation metrics for these sub-tasks as well as for whole grasping systems are shortly reviewed.

Precision, recall and F-score, as well as average precision (AP) and intersection over union (IoU) are used to evaluate approaches for *object detection*. Common metrics in the field of *pose estimation* are translational and rotational error, Average Distance of Model Points (ADD), 2D Reprojection Error and Visual Surface Discrepancy (VSD). For evaluating *grasping systems*, point- and rectangle metric, as well as success rate and planning time are typically used. To enable easy comparability, most approaches do report the percentage of correct estimations for certain error thresholds [75, 89].

Precision, Recall and F-Score

These three values are most commonly used in object detection. Precision p measures the accuracy of predictions, i.e. the fraction of all predictions that are correct, and is therefore also called positive predictive value. Recall r on the other hand is the fraction of true events that were detected and thus can be understood as sensitivity [14]. These values are often merged into an F-score to characterize systems with only one value, as illustrated in Equation 2 (TP = True Positives, FP = False Positives, FN = False Negatives).

$$p = \frac{TP}{TP + FP} \qquad r = \frac{TP}{TP + FN} \qquad F = 2 \cdot \frac{p \cdot r}{p + r} \tag{2}$$

AP and mAP – (Mean) Average Precision

Average precision is also a typical metric for object detection. It summarizes the shape of the precision-recall-curve by calculating the mean precision at a set of eleven equally spaced recall levels [90]. AP is computed for each object category individually. The mean value of all average precisions across all classes (mAP) is used to measure the general performance of a system [91, 39, 13].

IoU – Intersection over Union

The standard way of evaluating the accuracy of object detection and semantic segmentation methods in 2D domains is using IoU. The regions to be intersected are rectangles for former or masks for latter systems [40, 39]. In Equation 3, the formula for calculating the IoU for a given predicted bounding box $\hat{b}$ and a ground-truth $\tilde{b}$ is presented [40, 13].

$$IoU(\hat{b}, \tilde{b}) = \frac{area\ of\ overlap}{area\ of\ union} = \frac{area(\hat{b} \cap \tilde{b})}{area(\hat{b} \cup \tilde{b})} \tag{3}$$

The output of a detector can be notated as $(\hat{b}, \hat{c}, \hat{p})$ where $\hat{b}$ denotes the predicted location, $\hat{c}$ the object category and $\hat{p}$ a confidence score. Usually, an estimation is counted as true positive if the class prediction is correct and the confidence score is above some limit, as well as the IoU is larger than 0.5 which means an overlap of 50% [86]. This procedure is also commonly used in the evaluation of pose estimation using the ground-truth projection of a 3D model [75, 45], although this metric provides only weak information about object surface alignment since operating in projective space [40].

2D Reprojection Error

The 2D reprojection error measures the mean distance between the true image projection of an object calculated with the ground-truth pose and the estimated projection calculated by using the predicted pose. This metric is commonly used for augmented reality applications since pose accuracy in the z-direction (direction pointing away from the camera) is less critical for visual impression [75, 86]. A typically used threshold is 5 pixels [39].

Translational and Rotational Error

Translational and rotational errors are model-independent pose error functions which indicate deviations of an estimated pose $\hat{P} = (\hat{R}, \hat{t})$ with respect to the ground-truth pose $\tilde{P} = (\tilde{R}, \tilde{t})$ in mm and degrees accordingly. R is thereby a 3×3 rotation matrix and t a 3×1 translation vector. Errors can be calculated according to Equation 4 [92, 40].

$$e_T(\hat{t}, \tilde{t}) = \|\tilde{t} - \hat{t}\|_2 \qquad e_R(\hat{R}, \tilde{R}) = arrcos\ \frac{Tr(\hat{R}\tilde{R}^{-1}) - 1}{2} \tag{4}$$

Shotton et al. [89] introduced a 5cm5° threshold to consider poses as correct, which is only sufficient for augmented reality applications, not robotic grasping. It should be noted that neither translational nor rotational errors are invariant to ambiguities and direct comparison of pose deviations do not provide an intuitive visual understanding [40, 13].

ADD – Average Distance of Model Points

The most widely used evaluation metric for pose-estimation systems is the ADD-metric proposed by Hinterstoisser et al. [16]. The ADD-error is defined as the average distance of all model points x from *their transformed versions* if the model has no indistinguishable views (ADD), and from the *closest model point* if it has indistinguishable views (ADD-S). It can be calculated as stated by Equation 5, where M denotes the set of 3D model points and m is the number of points [16, 71, 13].

$$e_{ADD} = \frac{1}{m} \sum_{x \in M} \|(\tilde{R}x + \tilde{t}) - (\hat{R}x + \hat{t})\| \qquad e_{ADD-S} = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} \|(\tilde{R}x_1 + \tilde{t}) - (\hat{R}x_2 + \hat{t})\| \tag{5}$$

A drawback of the ADD-S metric is that it yields relatively small errors even for distinguishable views, why objects evaluated with this metric are advantaged [40]. e_{ADD} specifies the mean Euclidean distance between ground-truth and estimated model points in millimeters. Since

object sizes vary, such values should not be compared directly. ADD itself is a percentage describing the number of ADD-errors e_{ADD} being smaller than a specified threshold [40]. The literature proposes a threshold of 10% of the model's diameter for pose estimation tasks in general [16, 71, 75, 45, 42] or 20 mm for robotic grasping application since this value is a typical tolerance value for grippers [61].

VSD – Visual Surface Discrepancy

To achieve true ambiguity-invariance, Hodaň et al. [40] propose the visual surface discrepancy, which uses only the visible part of the object's surface for calculating an error. VSD is determined by the distance between the ground-truth and estimated visible object depth surfaces [40, 42]. Differentiating from the two-dimensional IoU, VSD uses 3D model points. In contrast to ADD, which considers all model points, VSD uses only the visible part of the surfaces and indistinguishable poses are therefore treated equivalently [13, 92].

AUC – Area under Curve

A common possibility to combine some of the metrics mentioned above is using an area under curve information – e.g., a e_{vsd} vs. recall curve can be created. The greater the area under the curve, the better the performance [42]. Authors often also publish accuracy-threshold curves printing a cumulated ADD-metric on the y-axis against an increasing threshold on the x-axis to inform about the overall system performance [71, 29].

Point- and Rectangle Metric

Many systems for grasp-detection are evaluated using the popular ADD metric, as stated above. Some also make use of a point metric, which indicates the distance between a predicted and ground-truth grasp center without considering a grasp angle. The rectangle metric is mainly used by approaches calculating a grasp-rectangle as described in section 3.3.4. An estimated grasp-rectangle specified by the 2D center, width, height and angle is typically considered correct if the angle is within 30° of ground-truth and the IoU of the rectangles is more than 25% [30, 38, 12, 13].

Success Rate and Planning Time

Mahler et al. [33] furthermore propose a success-rate and planning-time metric. The success-rate is defined as the percentage of grasps that can lift, transport and hold a desired object after shaking. Planning time characterizes the time between receiving an image and returning a planned grasp.

As stated above, no standard method for object pose estimation has been developed yet, but numerous promising approaches exist. Also, no norm for evaluation of such systems is available and most authors use a combination of the presented metrics.

3 Training Data for Neural Networks

For extracting information such as object types, positions and 6D poses from datasets, deep neural networks are commonly used [93, 94]. They consist of layers with artificial neurons whose weights are determined during training for matching estimated outputs to expected outputs [14]. Convolutional neural networks (CNNs) are special types of deep neural networks that are designed for working with data that has a grid-like topology – such as images [14]. They consist of a series of convolutional and pooling layers for feature extraction, supplemented with fully connected layers [94]. A convolutional layer generates feature maps by doing a convolution between a sliding filter and the input. The dimension – and thereby the number of parameters – of the feature maps is reduced using a pooling layer and the thereby extracted high-level features are used for processing the desired output using fully connected layers [14, 94]. During the training process of a neural network, the model's weights are manipulated such that the predicted output matches the expected output. Hence, a large number of training data with annotated ground-truth is necessary [14].

In this section, different types of training data for neural networks concerning their data format and data origin are introduced. Furthermore, problems caused by domain gaps are presented and possibilities to reduce these gaps are discussed. Finally, different approaches for generating ground-truth annotated data for different deep learning tasks and dataset formats are presented.

3.1 Types of Training Data

The type of data needed to train a network strongly depends on the network's architecture and on the type of data the system has to deal with at runtime. Training data should be as similar as possible to the data provided for detection in the real application [95, 96]. Training data for convolutional neural networks for pose estimation and robotic grasping can be classified in different ways. It can be differentiated between *data format* – monochrome-, color- or depth-images – and *data origin* – real or synthetic images.

3.1.1 Monochromatic, Color and Depth Data

Input data can be captured and stored at different levels, which are illustrated in Figure 12. The amount of information available for estimation is strongly influencing the performance of neural networks and mostly depends on the data format, why it is important to choose the correct format for an application [97].

Color- or RGB-images map environmental information to three channels for red, green and blue pixels respectively. Monochromatic- or grayscale-images consist of only one channel containing the color of each pixel as shades of grey [39, 98]. Depth-images do not include color information, but the distance between the image sensor and a scene at each pixel [39].

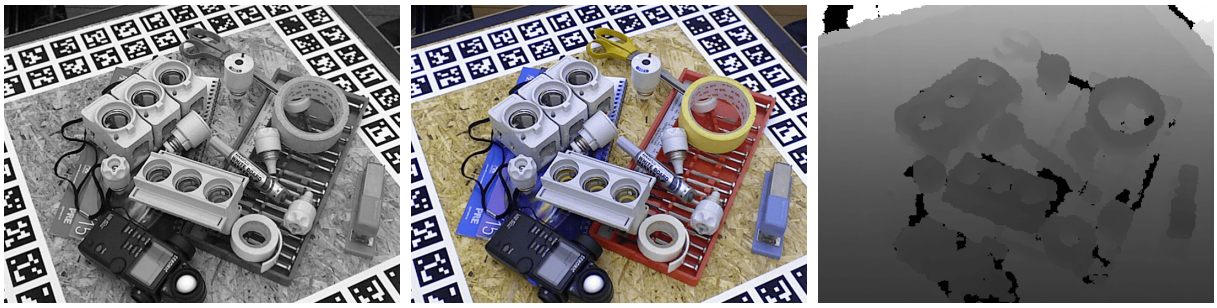


Figure 12: Monochromatic (left), color (center), depth image (right). Image source: T-LESS dataset [15].

Santana et al. [97] investigated how the training data format influences the performance of convolutional neural networks by training the same network with RGB-D- and RGB-data. They show that better results can be obtained when the depth-channel is treated on equal bases as the color channels [97]. RGB-D-cameras include sensors for color-images as well as for depth maps and hence can provide colorized pointclouds. The prices for depth-sensors have dropped recently and therefore such cameras have gained popularity [99]. Since providing the highest amount of information, using RGB-D data is preferable for 3D vision tasks such as 6D pose estimation [97].

3.1.2 Real and Synthetic Data

Training data can furthermore be classified into *real* data – real-world scenes captured with real image sensors – and *synthetic* data – objects or scenes rendered with virtual cameras. Synthetic data can furthermore be distinguished into photorealistic renders with or without a physics simulation, as well as domain randomized and mixed-reality data.

While real-world data can be captured with a standard camera, generating synthetic training data requires a rendering engine, such as Unreal Engine[1] [100, 17] or OpenGL[2] [100, 101, 102]. Synthetic data benefits from easily controllable environmental conditions and complete definition of scenes, which enables precise calculation of application-specific annotations with low effort [100]. The main difficulties are physically realistic scene layouts, random but lifelike camera trajectories and photorealistic rendering [103]. Therefore, some approaches do not focus on modeling own environments but using environments from big consumer games such as Grand Theft Auto (GTA) [104] since the game industry has spent much effort into creating realistic and interactive 3D worlds [105]. Games can simulate different weather scenarios and thus can

[1] Unreal Engine: https://www.unrealengine.com
[2] OpenGL: https://www.opengl.org

be used for extracting labels for semantic per-pixel segmentation from outdoor scenes for autonomous driving [104, 106]. Challenging for such approaches is catching appropriate data with special plugins or reconstructing scenes from the communication between the game and the graphics hardware since usually no programming interface is provided [104, 107]. Furthermore, high computational power and sophisticated rendering techniques are necessary to generate images looking like real-world data for bridging the reality gap [108, 109].

3.2 Domain Gap and Reality Gap

Annotated synthetic data is relatively easy to collect, but neural networks trained with synthetic data only often show poorer results than when trained on a much smaller number of real images – due to the so-called *reality gap* [108]. A reality gap is a special case of domain gap which occurs when the test data (target domain – real-world images) is different from the training data (source domain – synthetic images) [110]. Domain adaption aims to reduce the gap between the feature distributions of the source and target domains [110]. Bridging a domain gap is possible using photorealistic rendering, domain randomization (DR) and data augmentation (DA) [42], as visualized in Figure 13.

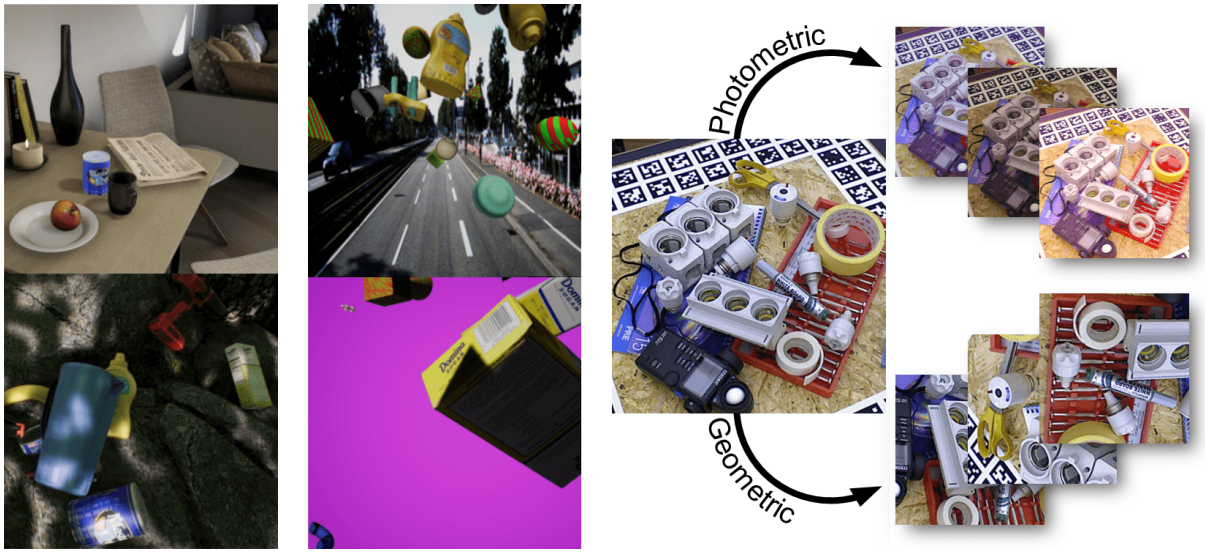


Figure 13: Methods for reducing the domain gap and expanding datasets: Photorealistic rendering (left), Domain randomization (center), Photometric data augmentation (top right), Geometric data augmentation (bottom right). Image sources: Tremblay et al. [29], Hodaň et al. [15].

3.2.1 Photorealistic Rendering

Photorealistic rendering tries to imitate reality to the greatest possible extent. Necessary physics simulations need high computational power and are time-consuming. Though, it has been shown that improvements of up to 24% are possible if objects are rendered realistically in

contrast to placing them on random photographs [109]. Zhang et al. [111] verify this result by showing that the level of photorealism of rendered training data directly influences the robustness and performance of deep neural networks when tested on real data. It is common practice to train a network on synthetic data and re-train it on the real domain [109], which provides a significant increase in accuracy at the best cost-to-benefit ratio [112, 107].

3.2.2 Domain Randomization

Domain randomization (DR) aims to provide high variability in semi-realistic synthetic data at training time such that the model generalizes to real-world data and real images appear as just another variation [95, 42]. Approaches for domain randomization do not need any knowledge about the target domain [110], can use a low-quality non-photorealistic rendering engine and do not need textures, scene configurations or lighting conditions matched to real-world scenarios [95]. Randomized scene factors include poses and textures of objects, occlusions by a random number of differently shaped distractor objects, pose and field of view of the camera, noise and background as well as the number, pose and type of light [95, 113].

3.2.3 Data Augmentation

Data augmentation (DA) is commonly used to increase the amount and diversity of data. Raising the number of training data helps networks to reduce overfitting and is possible using data augmentation without extra labeling costs. Higher diversity reduces the domain gap, especially if information from the target domain is used to augment training data [114, 115]. Data augmentation can be used to reduce domain gaps caused by geometric influences like positions of objects in the images and by photometric impacts such as scene lighting and sensor noise or synthetic training data (reality-gap) [108, 116, 117].

Geometric Augmentation

Geometric augmentations aim to achieve invariance to spatial factors not relevant to a learning task by rotating and translating images [116]. Since especially pictures captured by a camera applied to a robot show objects often at unusual angles and scales while training images tend to show them centered, creating photos that display zoomed-in parts of objects can lead to performance boosts of up to 7% in object recognition [118]. Even higher performance increases can be obtained by using a type of mixed reality, where images of objects are separated from the background, randomly transformed and pasted on backgrounds using blending. Thereby, also occlusions and distractors can be augmented [119]. More realistic pictures are possible by rendering object models with plausible indoor-lighting conditions, placed on detected planes in RGB images [120]. Data augmentation can also be used to improve domain randomized data. Since DR samples rendering parameters uniformly, the generated dataset tends to include a high number of easy and redundant samples. By defining distribution scenarios [121] or

using reinforcement learning if unlabeled data from the target domain is available [110], an augmentation policy can be identified to enable more realistic domain randomization.

Photometric Augmentation

Photometric augmentations are used to increase robustness to different lighting conditions and camera noise by changing individual pixels [116]. Some approaches randomly add chromatic aberration, blur, shadows, noise, spotlights and shadows or change brightness and color temperature of synthetic [116] and real images [66, 122] to extend the training dataset.

Training a data augmentation network to learn the distributions of such impacts from real pictures [113, 114] or transferring the style of an unlabeled real to a labeled synthetic image [108] increases the effectiveness of these methods. Rambach et al. [123] use one channel edge enhanced images looking like pencil drawings to gain invariance to lighting conditions. They show great success using a network for pose estimation on real data, trained entirely on synthetic images since synthetic and real pictures processed in a pencil filter look similar. It is also possible to not directly augment training data but use a high number of different augmentations of unlabeled images to pre-train a network to gain invariance of the feature extractors [115].

Data augmentation can also be applied to depth data. Simple approaches add noise to imitate a real depth sensor [124]. More sophisticated methods calculate augmentations from surface geometry and material reflection [125] or train an augmentation network with real depth data to gain realistic depth maps [126]. As visible in Figure 12 (right), real depth images show no-depth-return pixels in black due to reflections (e.g. scissors) or occlusions, which do not occur in rendered depth maps. Therefore, Goodfellow et al. [127] first introduced generative adversarial nets (GAN), which consist of a generator and a discriminator network with competing losses. The generator network tries to produce realistic images from synthetic data, whereas the discriminator tries to distinguish them from real images. GANs trained on unlabelled real depth maps can be used to augment synthetic depth images to look like real images [36, 128, 129].

Great success has also been shown by not augmenting most realistic images, but most reasonable pictures for learning. An augmenter-network is trained to merge multiple samples from a class into a new image, which is used to train a target network, which informs the augmenter about the loss [130]. Although most approaches focus on augmenting synthetic data to train a network on, it is also possible to modify real test data to look synthetic by denoising and retaining features, which is easier than the opposite way [131].

3.3 Generation of Ground-Truth Annotations

To train neural networks, data with annotated ground-truth is necessary. The type of annotation depends on the network's structure and the desired output of the neural network [98]. In the following sections, types of ground-truth information for object detection, semantic segmentation, object pose estimation and grasp prediction are introduced.

3.3.1 Types of Ground-Truth Information

If a network is to be trained for object detection, training data containing the 2D bounding box coordinates around the objects (Fig. 14a) are necessary [132, 73]. For semantic segmentation and scene understanding, per-pixel-labels (Fig. 14b) which indicate, to which object-class or class-instance a pixel belongs, are used [124, 133]. Neural networks for 6D pose estimation (Fig. 14c) commonly use the position and rotation of an object-fixed coordinate frame with respect to a camera (see section 2.2.3). Some approaches make use of similar features such as the coordinates of a box enclosing the objects of interest, which are calculable from the 6D pose [29]. Some neural networks for pose estimation require furthermore segmentation masks or bounding boxes to perform object detection first [61, 84, 70] or 3D-models for pose-correction as post-processing step [71, 45, 42]. Object masks or 2D bounding boxes can effortlessly be generated from given pose-data by projecting the 3D object poses into the 2D image space [20]. Many methods where neural networks are trained to calculate graspable regions for object manipulation are trained for pose estimation and look up corresponding grasp points or gripper poses in databases [71, 70, 134]. For grasping objects with a parallel-jaw gripper, representations defining a graspable region with additional gripper opening width (Fig. 14d) have been developed [135, 34].

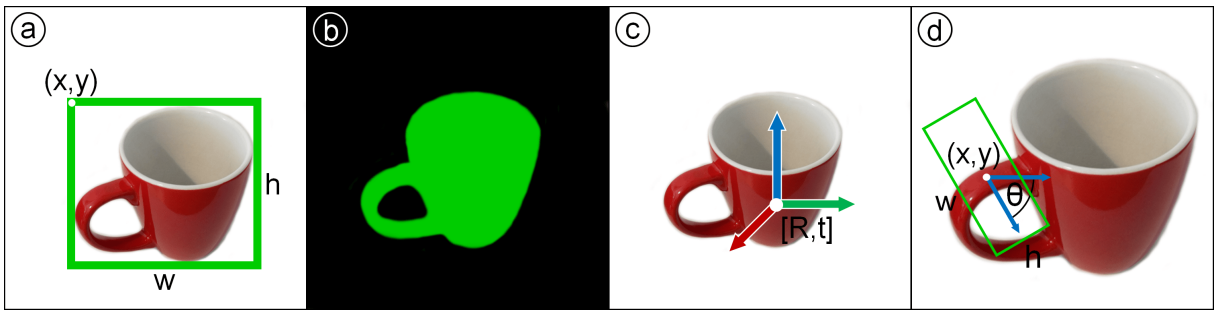


Figure 14: Different types of annotations: Bounding box (a) [73], Segmentation mask (b) [124], 6-DoF pose (c) [13], Grasp rectangle (d) [135].

Generating accurate ground-truth information for the introduced tasks is cumbersome. Especially generating grasp-annotations is currently a big challenge [134], because complexity rises with the number of dimensions necessary to completely define data [136]. Bounding boxes, per-pixel labels and grasping-poses can be derived from 6D object poses. If the pose of an object in an image is known, all other label types can be retrieved, why a dataset for object pose estimation can also be used for 2D object detection or grasping tasks.

3.3.2 Bounding Boxes and Per-Pixel Labels

Bounding boxes and semantic per-pixel labels for object detection and semantic segmentation can be described in 2D-space and are therefore the simplest annotations. Some neural networks for pose estimation require segmentation masks or bounding boxes to locate objects in

images first [61, 84, 70]. There exist numerous approaches for semi-automatic extraction of ground-truth 2D-bounding boxes and semantic per-pixel labels from RGB and RGB-D data.

LabelMe [137] is an online annotation tool where users draw outlines of objects onto pictures and label them. Once an appropriate number of labeled images from a class exists, an algorithm searches online for similar pictures and labels them – fully controlled by humans [137]. A similar approach with less human effort is pursued with "humans in the loop" [138]. Humans create per-pixel labels for images which a neural network is trained on. The system then annotates new pictures and only images with low confidence scores are labeled by humans, which are again used to train the network iteratively. Human labor can further be reduced by exploiting depth changes at object edges in RGB-D images, which limits manual effort to rearranging scenes and labeling object classes for segmented data [139]. Dai et al. [140] introduced a pipeline for automatic scene reconstruction from RGB-D data, supplemented with manual object labels. Video frames are merged and object surfaces are automatically reconstructed and manually labeled to extract ground-truth images from real 3D scenes. It is also possible to let users align CAD (Computer-Aided Design) models to the reconstructed scenes for simpler annotation [141].

There also exist multiple approaches for creating photorealistic scenes from synthetic data since automatic ground-truth annotation is possible with knowledge of the camera-to-object transformations as well as the camera's intrinsic parameters. It can be differentiated into approaches focusing on labeling generated indoor scenes – mostly used for scene understanding [124, 103, 111] – and outdoor scenes – primarily for autonomous driving [104, 107, 106]. Another method proposed by Dwibedi et al. [119] using mixed reality is masking an object and pasting this image in random background scenes to increase the number of training images autonomously and add diversity to a dataset.

3.3.3 6-DoF Object Poses

To describe the pose of an object with respect to a camera, six degrees of freedom are necessary – three rotations and three location coordinates. The tool LabelFusion [20] can be used to generate a large number of annotated training images from RGB-D videos. A CAD model has to be aligned to a reconstructed scene manually and is fitted exactly using iterative closest point algorithms (ICP) and projected to all frames [20]. This straightforward method has the advantage of fast labeling many images but can only be applied for rigid objects with accurate CAD models available and generates many redundant images due to a big overlap of video frames being necessary for 3D-reconstruction. All methods depending on human annotators make use of similar CAD-model fitting with iterative local search in projected 2D images or ICP for local refinement [142]. To assign the pose automatically to multiple images, reconstructing the scene and thereby the camera viewpoints of each frame to transform the pose for each image is possible [21]. It is conversely also possible to place objects of interest on a turntable and to calculate the transformations using the table's rotation [15].

To increase automation in the labeling process, some approaches use calibration-boards rigidly attached to objects and calculate the object's pose from the transformation between the camera and the detected marker [24, 26]. For higher accuracy, this pose can be refined using ICP-algorithms [19]. If images with markers are used for training neural networks, they involuntarily "help" the algorithm and therefore have to be removed in a post-processing step. Furthermore, the camera movement, as well as the number of objects presentable per scene, are restricted. Hence, Garon et al. [23] propose not using square markers but small spherical marker-points 3 mm in size applied to multiple positions directly onto the object and onto the cameras. By using a visual capture system with eight cameras calculating the marker poses, an object's ground-truth pose can be obtained. Since the markers are visible in the depth data, the objects are rendered in their ground-truth pose and the pixels containing the marker are replaced with the rendered depth image [23]. It is also possible to use active markers to locate the object relative to its environment and track the camera movement [25].

To estimate ground-truth without any markers, a single-view object detector can be applied to each frame of an RGB-D video. These object hypotheses can be projected into the reconstructed scene and the representation describing the scene best can be chosen [143]. Mitash et al. [134] also take a multi-view approach for generating ground-truth pose data. A neural network is first trained with simple synthetic images to detect objects. Multiple cameras are attached to defined positions on a robotic arm and used to capture multiple objects per scene. If an object is successfully recognized in at least two views, the pointclouds containing it are merged and the object's pose is calculated using model registration. Poses with high confidence are projected to all views to calculate ground-truth data for all images.

Using rendering engines to generate ground-truth data synthetically simplifies data generation and increases accuracy [100]. Thus, Tremblay et al. [17] developed a plugin for Unreal Engine to generate photorealistic scenes with ground-truth pose annotations. Even higher realism can be provided through more complex illumination effects using physically based rendering, at the cost of high rendering times [109]. Martinez-Gonzales et al. [101] built a framework upon Unreal Engine, which enables generating training data while a robot – controllable in virtual reality – interacts with objects. The created data can also be used for hand pose estimation, visual grasping or in-hand manipulation of objects.

3.3.4 Annotations for Grasping

The type of data needed to grasp an object successfully depends on the used grasping device. Estimation of a 6-DoF grasp point enables grasping with a suction cup. For grasping objects with a parallel-jaw gripper Jiang et al. [135] proposed a 7D annotation, which includes a 3D grasp-point, a 3D grasp-orientation and the gripper opening width. It has later been simplified to a 5D representation – 2D grasp point, rotation, length and width of a grasp-rectangle – assuming that a 2D pose can be transformed into a 3D pose by a robot with an RGB-D camera [34]. The dimensions of annotations can further be reduced to 4 – 3D point and gripper orientation

– when restricted to planar grasps, where the gripper is parallel to the plane objects are placed on [31]. Assuming that a 2D pose can be transformed into 3D space using RGB-D data [34], another reduction to 3 dimensions – 2D point and gripper orientation – is possible [28, 33].

Labeling data for grasping tasks manually is challenging and time-consuming since objects can be grasped in multiple ways and labels being of high dimensions and gripper-dependent. Also, exhaustive manual labeling is impossible and hence, it can not be assured that proposed grasp-configurations from a network that do not comply with the ground-truth actually lead to unsuccessful grasps. Furthermore, human annotations are biased by semantics since we tend to label handles as grasp location even though objects can be grasped at several other locations [28]. Therefore, most data for training neural networks for grasp-detection is either generated by robots or created synthetically: Pinto et al. [28] use a robot with an RGB-D camera, randomly define a 3D planar grasp and try to lift the object. The number of successful grasps could be increased by training a network iteratively to sample more reasonable grasps. Over 50k annotated images could be created in about 700 robot hours doing trial-and-error experiments. Levine et al. [27] used 14 robots doing trial-and-error grasps for two months to generate more than 800k images annotated with possible grasp-poses and their success-probability. Dex-Net 2.0 [33] is a synthetic dataset, containing 6.7 million point clouds with grasp metrics generated from 3D models. Grasp success probability is calculated using a function that considers object pose, gripper pose and friction coefficient uncertainty.

Since economic exhaustive manual labeling of grasp annotations is impossible and robots need much time to gather useful data in trial-and-error experiments, many approaches for robotic grasping focus on estimating object poses and look up related grasp configurations in databases [71, 70, 134].

3.4 Dataset Formats

Analyzing different frameworks for synthetic data-generation (e.g. NDDS [144], UnrealROX [101]) or real-world data-annotation (e.g. LabelFusion [20], approach by Aldoma et al. [143]), as well as existing datasets for pose estimation (e.g. FAT [17], T-Less [15], RobotriX [100], SceneNet RGB-D [103]) shows that no standardized format for pose-annotated datasets exists.

A meta-data format which enables easy writing and reading of data and also provides the maximum number of information for different tasks without further calculations has been proposed for the falling things dataset (FAT) [17]. The dataset provides two setting files and for each image an annotation file:

- **Object settings** – For each object:
  - Object name
  - Class- and Instance-ID
  - Dimensions of the Object's 3D bounding box
  - Pose w.r.t. the origin

- **Camera settings** – For each camera:
  - Camera name
  - Intrinsic parameters
  - Dimensions of image
- **Annotations** – For each image:
  - Pose of capturing camera w.r.t. the origin
  - For each object in the image:
    - Object name
    - Class- and Instance-ID
    - Visibility
    - Location and quaternion-rotation
    - Homogenized transformation-matrix
    - Corners of the object's 3D bounding box (3D coordinates and 2D projections)
    - 2D bounding box coordinates

The dataset includes for each object an RGB and depth image, as well as a category- and instance-level segmentation mask[3]. All this information is stored as JSON-files. JSON (JavaScript Object Notation) is a lightweight, language-independent data-interchange format. It consists of structured data and therefore, is easily read- and writable for humans and parse- and generate-able for machines [145].

The stored 2D bounding box coordinates can be used for object detection. Using the segmentation masks, networks for semantic segmentation on a category- or instance-level can be trained. The two different pose representations can be used for different pose-estimation approaches, while the cuboids and centroid are useful for the training of PnP-based pose-estimation methods. Using object- and camera-settings, as well as the camera's pose at each photograph, the scenes can be reconstructed synthetically and further information obtained.

Redundancies arising from providing the same information in multiple annotations lead to slightly higher memory usage, but data can be used directly for a large number of tasks and approaches. Hence, this dataset format has been chosen for implementation in this work.

Although synthetic training data for neural networks can easily be generated, the performance of networks trained on real data can currently not be achieved. Photorealistic rendering, domain randomization or data augmentation are effective ways for reducing this reality gap. Annotated 6D object pose data contains the highest amount of information and enables retrieving bounding boxes, per-pixel labels and grasping-poses, why it is reasonable to store 6D poses, if possible.

[3] FAT Dataset format: https://research.nvidia.com/sites/default/files/pubs/2018-06_Falling-Things/readme_0.txt

4 Semi-Automatic Data-Generation

For training neural networks for pose estimation, large scale annotated training datasets are necessary. Tremblay et al. [29] observed best results for 600,000 photorealistic training images including five objects. The commonly used YCB-dataset [71] features more than 130,000 video frames of 21 objects, and the dataset of [146] contains 277,000 images of 89 objects. Gathering these datasets manually or using special tools is cumbersome. Therefore, a method for semi-automatic creation of 6-DoF pose-annotated training data for neural networks, using a mobile manipulator, is presented.

In this section, the mobile manipulator and its kinematic structure are introduced. The procedure for capturing training data is presented and the software architecture for controlling all components is explained in detail.

4.1 Mobile Manipulator

To record training data in a real-world scene and test if the accuracy of the pose estimation is sufficient for robotic grasping, a mobile manipulator is used. In the following section, the components and structure of this robot are described and the calibration of the robot's camera is explained.

4.1.1 Components and Structure of the Mobile Manipulator

The mobile manipulator used for capturing data and qualitatively evaluating the neural network is presented in Figure 15 (left). A mobile platform MiR100[4] (1) serves as base for the robot. This mobile robot is able to calculate and travel paths autonomously and is aware of safety aspects. A collaborative, articulated UR5[5] (2), with a payload of up to 5 kg at a maximum grasping-distance of 850 mm is mounted on top of the mobile robot. At the robot's flange, a 3D-printed fixture is located for mounting the parallel-jaw gripper Robotiq[6] 2F-85 (4) with an opening width of 85 mm and the RGB-D-camera Realsense D435[7] (5), which can capture images between 0.2 and 4.5 meters in depth. All components are controlled by a ROS-Kinetic-Core [147] on Ubuntu 16.04, running on the industrial computer ECS-9100-GTX1050T[8] (3), which is equipped with

[4] MiR100 mobile robot: www.mobile-industrial-robots.com/de/products/mir100
[5] UR5 robot: www.universal-robots.com/de/produkte/ur5-roboter
[6] Robotiq Gripper 2f-85: www.robotiq.com/products/2f85-140-adaptive-robot-gripper
[7] Intel Realsense RGB-D camera D435: click.intel.com/intelr-realsensetm-depth-camera-d435.html
[8] Industrial Computer ECS-9100-GTX1050T: www.vecow.com/dispUploadBox/PJ-VECOW/Files/3238.pdf

an NVIDIA GeForce GTX 1050 Ti graphics processor for image processing tasks.

The kinematic structure of this robot with all coordinate frames and their connections is presented in Figure 15 (right). Orientations of the coordinate systems are exemplary. Frames which are not necessary for the understanding are not shown in terms of reduced complexity (e.g. in all joints of the robotic arm). The orange lines visualize all mutual dependencies of the frames.

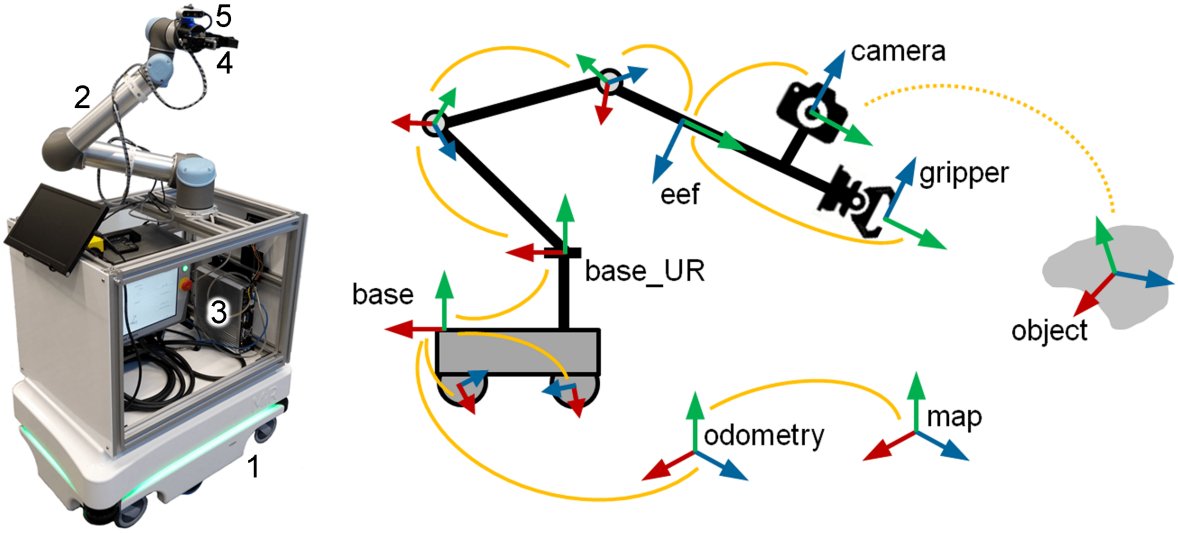


Figure 15: Picture of the mobile manipulator (left) and its frames and kinematic structure (right) [148]. Components of the mobile manipulator: MiR100 (1), UR5 (2), industrial computer (3), gripper (4) and camera (5).

The *map*- and *odometry*-frames are world-fixed and the *base*-frame of the robot is moved in these frames according to signals from a localization module, the inertial measurement unit and the wheel odometry [149]. The frame *base_UR* is connected to the *base*-frame statically and used for calculating transformations of the robotic arm joints. At the end of the kinematic chain, the *eef*-frame (End Effector Frame) serves as reference for the *gripper* and *camera*-frame. Objects are detected in the camera frame using the given transformations for calculating their poses with respect to the *base_UR*-frame. For determining the transformations between the *object* and *camera*-frame, as well as between the *camera* and *eef*-frame, calibrating the camera is necessary.

4.1.2 Calibration of the Camera

To calculate the pose of recorded objects with respect to the robot, two transformations are necessary: The transformations between the *eef*-frame and the *camera*-frame (extrinsic), as well as between the *camera*-frame and the *object*-frame (intrinsic).

There are 11 free parameters necessary for a complete definition of a camera: 3 rotations, 3 translations and 5 intrinsic parameters [98]. Intrinsic parameters are necessary to calculate 3D metric information from 2D images. Cameras are usually calibrated by showing a planar

pattern to a camera in at least two different, unknown orientations [150]. Thereby, a camera calibration matrix containing the focal lengths and principal points for both image directions is obtained [151]. This matrix indicates the relationship between 3D world coordinates and 2D image coordinates. For capturing annotated data, the camera has been calibrated intrinsically according to the manufacturer calibration guide [152] using a special calibration target.

Extrinsic parameters define the pose of a camera in world coordinates [98]. In the field of robotics, determining extrinsic parameters is also known as hand-eye calibration, which aims to find the transformation between the robot's gripper (hand) and the camera (eye) [153]. Traditional approaches for extrinsic calibration formulate the hand-eye calibration problem as solving the homogeneous transform equation $AX = XB$. A is the known change of the wrist position, B is the accompanying displacement of the sensor and X is the sensor's position relative to the wrist. By moving the robotic arm into two or more known positions, a system of equations that is solvable for X results [153, 154]. Accuracy can be increased by complying with the following [153]: rotation between poses should be maximized, while translations should be minimized. The distance between target and camera should be minimal, and there should be redundant poses. There exist also newer and more accurate yet more complex solutions for extrinsic calibration without a calibration target [155, 156]. For capturing annotated data, the camera has been calibrated extrinsically using the ROS package easy_handeye[9] and an ArUco marker-board [64].

4.2 Procedure to Capture Training Data

To train a neural network for estimating object poses from an image, training data has to consist of images showing the object and information of the pose of the object in the image. Therefore, a semi-automatic technique for generating training data with a camera placed at the end of the kinematic chain of a robot has been developed. The procedure is explained in more detail in the following sections.

The pose of the object with regard to the robot's base is first determined with a marker placed on the object of interest, viewed by the robot's camera. After removing the marker, training images from different perspectives can be recorded. As long as the object is not moved during this process, its pose can be calculated for every image, using the transformations between the robot's base and the camera. The basic procedure for generating training data in a real scene is illustrated in Figure 16.

In the first step, the object is placed in front of the robot. In the case of a robotic arm mounted on a mobile robot, the robot can be moved to the object. The distance between the robot and the object of interest should be approximately the same as the distance at pose estimation. Following, a fiducial marker is put onto a defined position and orientation onto the object of interest. The pose of the marker with respect to the camera can be calculated from the captured

[9] easy_handeye extrinsic camera calibration: https://github.com/IFL-CAMP/easy_handeye

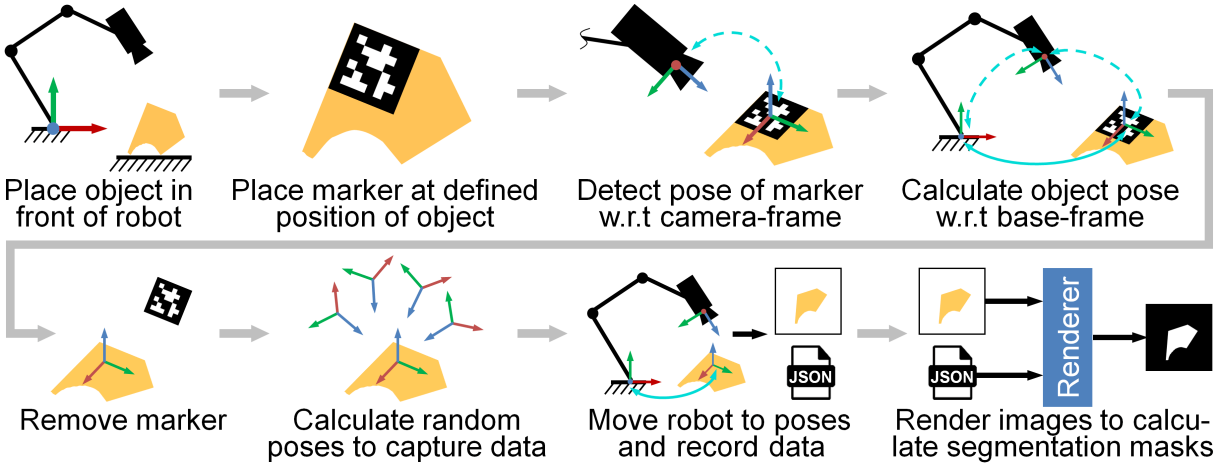


Figure 16: Procedure for generating annotated training data using a marker and a camera mounted to a robot: The object with marker is placed in front of the robot. The pose of the marker is computed and used to calculate the object's pose with respect to (w.r.t.) the robot's base. After removing the marker, images can be captured at random poses to create a large scale 6D pose-annotated dataset.

image, as explained in Section 2.2.2. This transformation is used to calculate the pose of the object with respect to the robot's base. The transformation between camera and base, as well as between marker and object-origin are used for this computation. Afterwards, the marker is no longer needed and can be removed, whereby special attention has to be placed on not changing the object's pose. The robot arm can drive to different positions around the object and capture images and associated object-pose data automatically. To supplement the training data, the object is finally rendered in a virtual environment to calculate a segmentation mask and the bounding box of the real data.

The proportion of human labor in this process is very small. It is only necessary to put the marker onto the object, move the robot and capture images of the marker, and remove it again, to enable capturing several thousand training images fully automatically. In the following sections, the implementation of the software and all steps necessary for generating training data in a real scene are described in more detail.

4.3 Software Architecture

The robot's software for capturing annotated data is entirely based on ROS (Robot Operating System) [147]. It is structured in different nodes: The basic nodes establish communication to all hardware, provide a virtual robot model for calculation of transformations and a module for marker pose estimation. Successively nodes for determination of the ground-truth object-pose (*get_gt_pose*), calculation of data-recording poses (*capture_pose_calc*) and capturing data (*data_capture*) are launched. Figure 17 presents the software architecture and is intended to supplement the following descriptions.

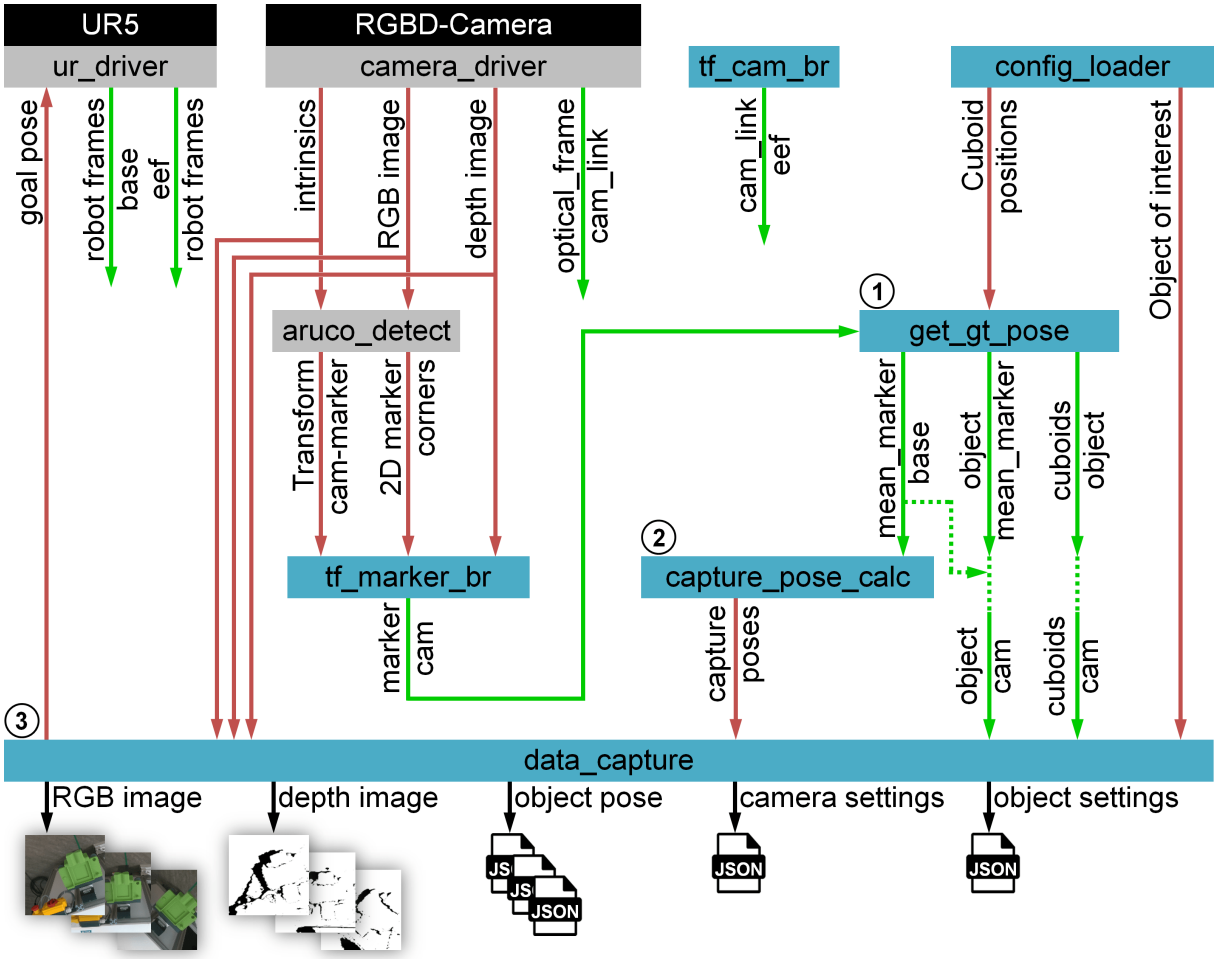


Figure 17: Software Architecture of data generation pipeline: Hardware (black boxes), Provided ROS-Packages (gray boxes) and implemented-Packages (blue boxes). ROS-topics (red arrows) and tf-frames (green arrows – frame on top side of arrow with respect to (w.r.t.) frame on bottom side of arrow). Tf-frames used by multiple nodes are not connected in terms of readability. The numbered packages are not part of the basic nodes and must be launched in correct order. For easier readability, only necessary topics and frames are visualized.

4.3.1 Basic Nodes

First, *config_loader* provides a configuration-file with information such as image size, object name, cuboid positions, paths to 3D files and parameters for the generation of random poses to the ROS-Core and thereby to all nodes. *ur_driver*[10] [157] establishes a connection to the UR5. This node controls the robot's joints and offers transformations between the coordinate-frames to tf-tree[11] [158], which manages all frames in the system. *camera_driver*[12] connects to the camera and publishes RGB- and depth-images, as well as the intrinsic camera parameters. It furthermore provides transformations between the different image sensors in the camera to

[10] UR5 ROS-Driver: https://github.com/ros-industrial/ur_modern_driver
[11] TF ROS-Package: http://wiki.ros.org/tf
[12] Realsense Camera ROS-Driver: https://github.com/IntelRealSense/realsense-ros

the tf-tree, to enable alignment of depth- to color-images. With *tf_cam_br*, the transformation between the robot's last frame and the camera, determined by hand-eye-calibration, is broadcasted to tf-tree.

4.3.2 Marker-Pose Determination and Refinement

For marker detection and pose estimation, ArUco markers [64] in combination with the ROS-Package aruco_detect[13] are used. Given an image and the camera's intrinsic parameters, this package calculates the 6D-marker pose with respect to the optical frame, as well as the 2D-projection of the marker's corners onto the image. This information is published and processed in *tf_marker_br*, which performs a pose-refinement with the help of the camera's depth data. It determines the marker's center $\vec{c}$ by calculating the average of all x- and y-coordinates, as stated in Equation 6 [159].

$$\vec{c} = \left(\frac{1}{4} \sum_{k=1}^{4} x_i, \ \frac{1}{4} \sum_{k=1}^{4} y_i \right)^T \tag{6}$$

The depth-value of this point is looked up in the depth-image to refine the z-value of the marker's pose, which is then published to tf-tree. The effects on the accuracy of the marker detection are investigated in section 6.1.

4.3.3 Determination of Ground-Truth Object Pose

Storing the calculated pose of the marker at an arbitrary point of time and using this as ground-truth for the captured data would lead to inaccurate results due to errors of the camera-, as well as the hand-eye-calibration and inaccuracies of the robot's joints. Therefore, the node *get_gt_pose* calculates a mean object pose from different views of the marker.

Using the transformation between the robot's base and the camera – which is provided by *ur_driver* – and the refined marker-pose with respect to the camera-frame – which is calculated by *tf_marker_br* – the marker's pose relative to the robot's base can be computed. This transformation stays the same when the camera's viewpoint is changed. To eliminate influences of the noise of the marker detection, several poses are stored in a ring-buffer and a mean pose is calculated for each perspective. By averaging the captured poses from all viewpoints, impacts of the described inaccuracies are minimized. The final average pose $\overrightarrow{P_\mu}$ is computed for the position coordinates and rotation around the axes separately, as described exemplarily for the position-coordinate μ_x in Equation 7.

$$\overrightarrow{P_\mu} = (\mu_x, \mu_y, \mu_z, \mu_{rx}, \mu_{ry}, \mu_{rz}) \qquad \mu_x = \frac{1}{n_{views}} \sum^{n_{views}} \Bigg(\underbrace{\frac{1}{n_{buffer}} \sum_{k=1}^{n_{buffer}} x_i}_{Mean\ pose\ at\ a\ viewpoint} \Bigg), \mu_y = \dots \tag{7}$$

[13] Aruco-Detect ROS-Package: http://wiki.ros.org/aruco_detect

Figure 18 illustrates the deviations and averaging process: The object with attached marker is captured in 4 different poses for calculating the mean marker pose (Figure 18 left). The central image shows the 3D coordinate systems for each captured marker-pose and an object model at the calculated mean pose overlaid on the 3D pointcloud. The plot on the right-hand side shows the deviations of all translations and rotations. Each recorded marker-pose (red, blue, green, black) shows a deviation due to noise of the marker detection (small thin dashes). A mean position or rotation at each viewpoint is calculated (thick dashes) and these values are again averaged using data of all four recordings for calculating the final mean pose, which has been shifted to zero to show deviations.

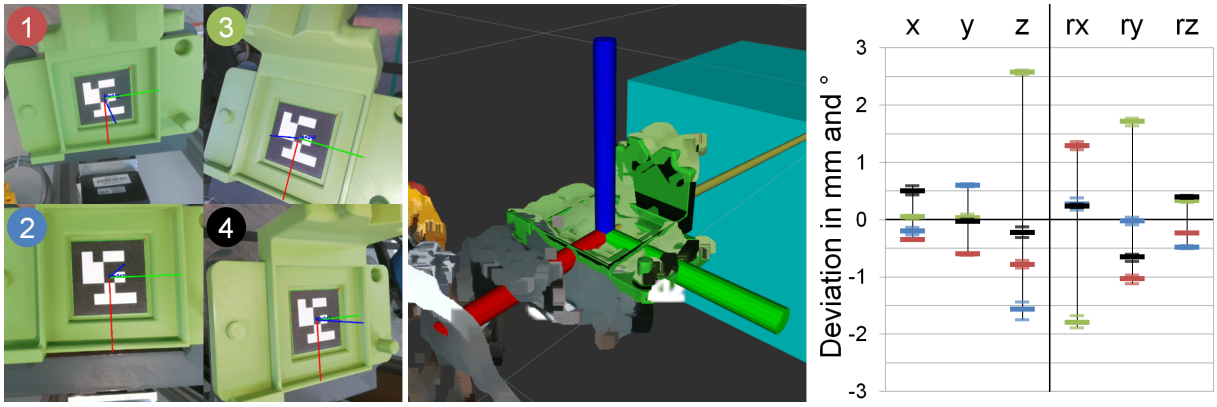


Figure 18: Ground-truth pose estimation using an ArUco marker: Captured images (left), calculated 6-DoF object-poses (center) and deviations for each captured image (right) in all directions (x, y, z) and around all axes (rx, ry, rz). Colored lines indicate pose deviations for different captured images: Pose-deviation (small thin dashes) and mean pose (thick dash).

The node *get_gt_pose* broadcasts the transformation between the calculated mean marker pose and the object's origin, as well as the cuboid-positions with respect to the object to tf-tree. A 3D model of the object is added to the robot-scene and can be shown in the 3D visualization for verification of the ground-truth pose. The marker can then be removed from the object, whereby the object's pose must not be changed.

4.3.4 Calculation of Data-Recording Poses

Training data is captured at different positions around the object. To ensure a structured way of proceeding with uniform pose-distribution and as short ways between positions as possible, the poses where the data is to be recorded are calculated in advance by *capture_pose_calc*. To achieve consistent training data from all perspectives, the camera should record images hemispherically around the object. Thus, the robot is moved to the beginning of the capture-area by the user. Originating from this pose, all other poses are computed by calculating the vector between the camera and the object in spherical coordinates and increasing the angles. The definition of the parameters of polar coordinates with the origin placed in the object is visualized in Figure 19.

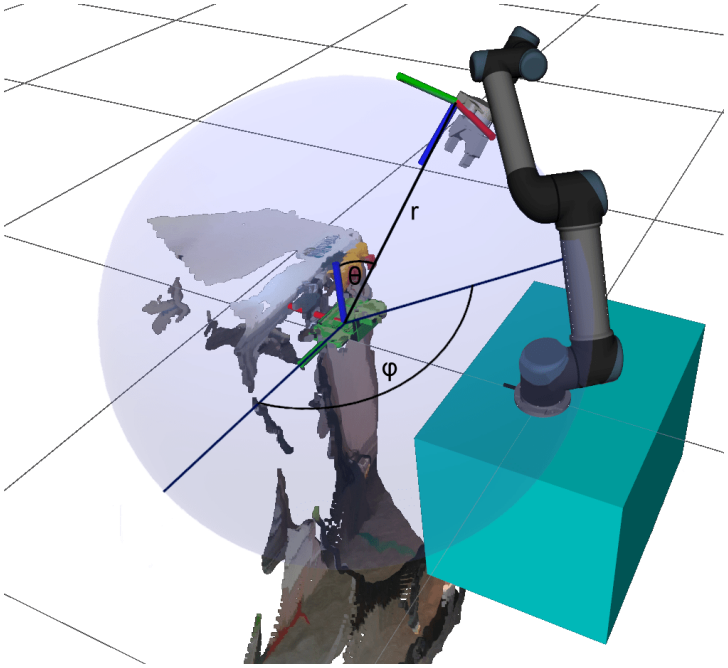


Figure 19: Spherical coordinates: A 3D point is defined by a distance r and two angles θ and φ.

The calculation of capture poses is performed, as illustrated in Code 1. First, θ is set to a fixed value and φ is enhanced until a specified maximum φ is reached. Afterwards, θ is increased to the next step and φ successively reduced until the starting angle is reached. This procedure is repeated until a specified maximum θ is reached. The orientation at each point is calculated to make the z-axis point exactly to the object since the object should be in the camera's center. Only if the robot can reach the calculated pose, it is added to the list. Whether a pose is reachable is determined by an inverse kinematics solver in *ur_driver*.

```
1    if user moved robot to start pose:
2        r, phi, theta = get_polar_coordinates(transformation_cam_to_object)
3        while theta < theta_max:
4            phi = phi + phi_inc
5            if phi >= phi_max:
6                theta = theta + theta_inc
7                phi_inc = -phi_inc
8            for i in num_random_poses:
9                pose = calculate_pose_pointing_to_object(r, phi, theta)
10               if is_reachable(pose):
11                   goals.add(pose)
12               r, phi, theta = manipulate_randomly()
13       publish_and_broadcast(goals)
```

Code 1: Pseudocode for calculation of capture poses: φ is increased from a user-given start pose until the maximum value is reached. Afterwards, θ is raised and φ successively reduced until the starting angle is reached. This is repeated until the maximum value of θ is reached. For each position, a pose pointing to the object's center is calculated. The poses are randomly modified to increase diversity and if reachable by the robot, added as capture poses.

Without randomization, the algorithm would produces an even distribution of reachable poses around the object of interest, as could be seen in Figure 20 (left). To increase variability – to make sure the object is not captured in the same distance at constant angle increments – θ, φ and the radius are sampled randomly from a uniform distribution in certain boundaries to calculate additional poses. This leads to the result visible in Figure 20 (right).

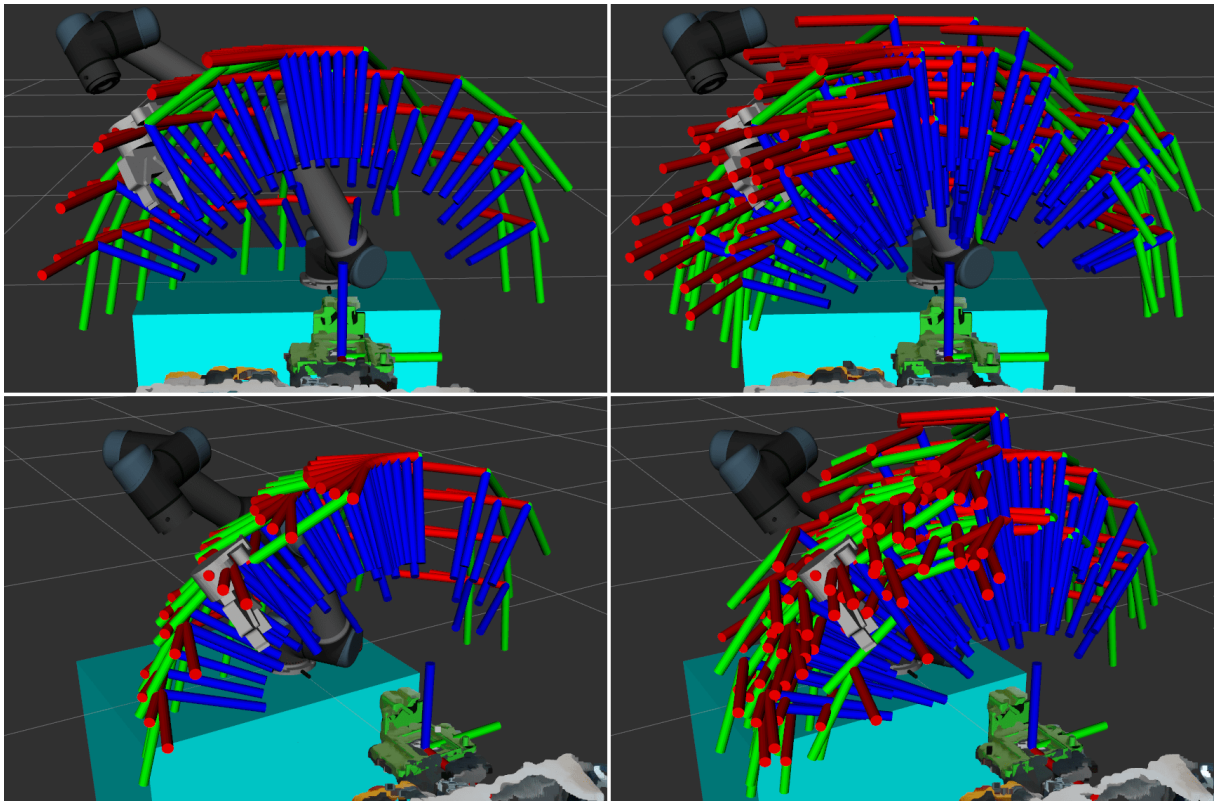


Figure 20: Generated poses for capturing data: None-random poses (left) and poses with uniformly sampled parameters r, θ and φ (right).

4.3.5 Capturing of Data

The ROS-node *data_capture* is responsible for recording data. It subscribes to the calculated capture-poses, moves the robot to these goals and stores the following data at each position (for an explanation of the format of the dataset see Section 3.4):

- poseNumber.png - RGB image published by the camera
- poseNumber.csv - depth image published by the camera
- poseNumber.json - annotation of the captured data

Furthermore, for each scene the following files are created:

- camera_settings.json - intrinsic parameters of the camera
- object_settings.json - the object's pose with respect to the robot and its dimensions

When storing training data, a focus is on saving as much useful information as possible. Therefore, the color images are stored as PNG-files since this format allows lossless, compressed saving [160]. As pointed out by Dodge et al. [117], image quality e.g. reduced by JPEG-compression can affect the output of a neural network. The depth value at each pixel is stored in a CSV-file to enable calculation of pointclouds or depth-images from these data. The JSON-file for each pose includes, besides the transformations base-to-camera and camera-to-object, the positions of the cuboid-points in the scene, as well as their 2D-projection onto the image plane. Since many neural networks – such as the selected – process only square images, a dataset with squared pictures is created simultaneously with the full-scale dataset, as illustrated in Figure 21.

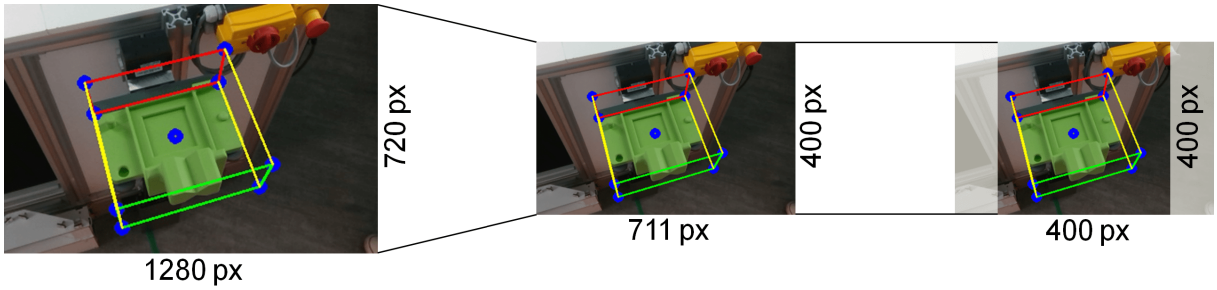


Figure 21: Resizing images to a square: Rescaling until the smaller edge fits the final size and cropping off pixels symmetrically at the longer side.

To resize the rectangular picture of the camera to a square image with a lower number of pixels, the shorter side is resized to match the desired number of pixels in the final image, and the edges of the longer side are cropped symmetrically. The camera's intrinsic parameters are divided by the scale-factor and the principal point is moved by the number of cropped pixels to stay in place with respect to the original image. The equation for transforming the camera's intrinsic parameters from a landscape picture to a smaller quadratic picture is presented in Equation 8 [151, 161].

$$K_{transform} = scale \cdot \begin{bmatrix} f_x & * & c_x \\ * & f_y & c_y \\ * & * & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & x_{offset} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8}$$

Because of the poses calculated by *capture_pose_calc* pointing exactly to the object, it is located in the center of every recorded image. This is not desirable since the object of interest will not always be centered in the real-world application. Thus, the robot's last three joints are moved a random positive and negative angle to increase on the one hand variation of data, and on the other hand the total number of data in the dataset. Using this method, 15 records are created per pose – five images with the camera pointing to the object and slightly up/down/left-/right, captured in the original pose and with the last axis turned positively and negatively. The axis angles are manipulated randomly, but in a way that no cuboid-point-projection is located more than 50 px outside the picture. The 3D cuboid-points are broadcasted to the tf-tree by the

node *get_gt_pose*. Their 2D-projections can be calculated using the camera's intrinsic parameters and the 3D-coordinates through Equation 9 [161]. The image coordinates u and v in pixels can be computed with the principal point represented by c_x and c_y, the focal lengths f_x and f_y, as well as the 3D-cuboid-coordinates with respect to the camera x_c, y_c and z_c in meters.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \frac{1}{z_c} \cdot \begin{pmatrix} f_x \cdot x_c \\ f_y \cdot y_c \end{pmatrix} \tag{9}$$

Figure 22 illustrates the robot's tf-tree starting with *base_UR*. This tree structure of the frames is necessary for calculating the correct object pose, which complies with the transformation between the camera's optical frame *RGB_optical_frame* and *object*. *ur_driver* broadcasts all frames of the robot and a link for the end-effector. The module *tf_cam_br* adds the frame *cam_link* to the end of the kinematic chain which serves as reference for the camera's optical frames. The module *get_gt_pose* calculates the mean marker pose and broadcasts the corresponding *object*-frame and the positions of all cuboids. Due to that, transformations between the camera and the cuboids can be read out for data annotation. The poses, where the robot drives to capture data, are calculated with respect to the *object*-frame and thus, they are broadcasted in relation to this frame.

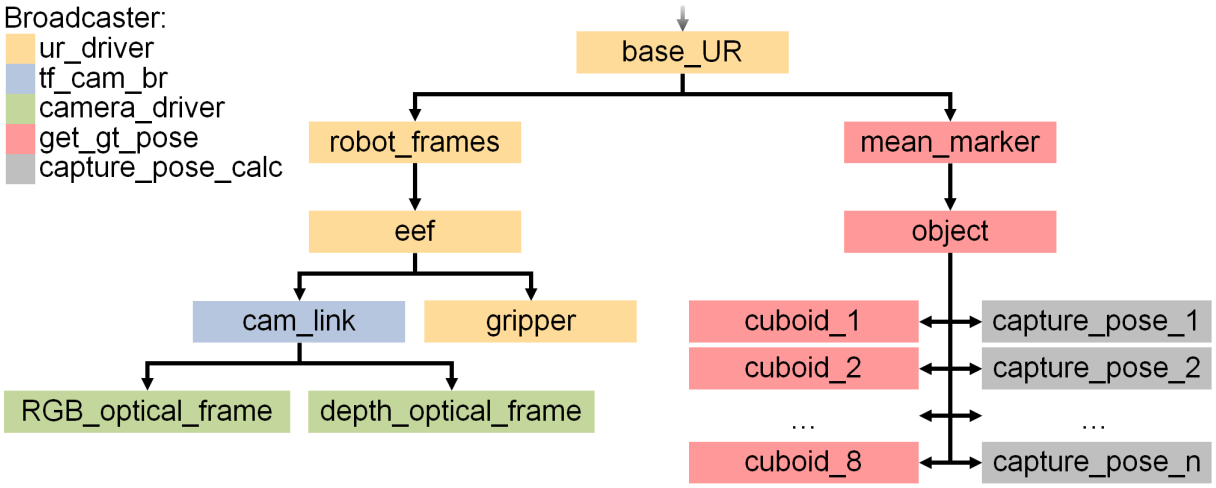


Figure 22: Frames and their relations during data-generation: The left branch represents the robot's kinematic chain. The object's pose is computed with respect to *RGB_optical_frame* and broadcasted relatively to *base_UR* to achieve independence from robot movements. Colors indicate the broadcasting nodes.

4.3.6 Post-Processing

After training data of a real-world scene has been captured, several optional post-processing steps are possible to supplement the recorded data. The segmentation mask and the object's bounding box can not be calculated directly from the pictures, but are determinable with knowledge of the object's pose and the camera's parameters. As illustrated in Figure 23, the 3D

model is first rendered using Blender[14]. Blender is an open-source 3D creation suite providing a graphical user interface and programming interface for python code [162]. These renders are then binarized to create segmentation masks and calculate the object's 2D bounding box.

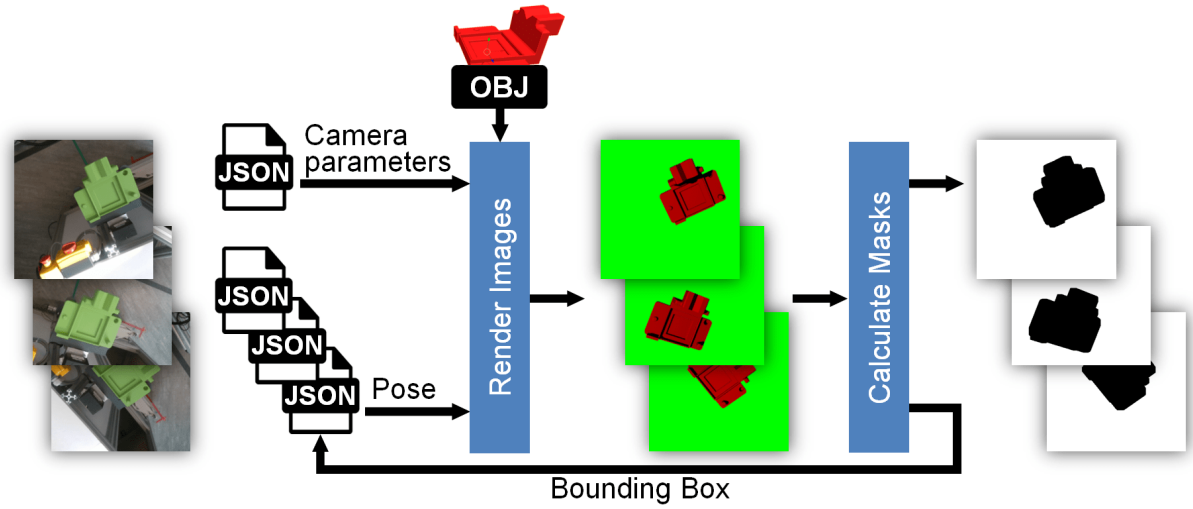


Figure 23: Post-processing steps: A 3D model is rendered in specified poses using the camera parameters of the recorded data. The rendered images are binarized to calculate segmentation masks and bounding boxes, which are written back into the meta-data files.

To render the object synthetically in the real-world pose, a 3D-model of the object is loaded into Blender. It is oriented as specified in the recorded JSON-file and a virtual camera is set to the parameters of the real camera. A red material is assigned to the object and the background-color is set to be green, to ensure easy distinguishability between object and background. Then the scene is rendered and the resulting image is stored.

Afterwards, image-processing tasks are performed, as visualized in Figure 24, to create a binary segmentation mask: First, the picture's channels are split to separate the green background from the red foreground. Following, the green channel is binarized to obtain black object contours on white background. Finally, the object's shape is eroded to enlarge its area since inaccuracies during the capturing process could lead to the mask being slightly displaced, which would result in segmentation masks excluding some pixels belonging to the object. Empirical experiments show a necessary enlargement of the shape by 2 to 5 pixels to include the complete object in all pictures. Figure 24 (d) shows the rendered and eroded contour overlaid on the original image to visualize deviations. Using the eroded segmentation mask, the coordinates of the object's bounding box can be calculated by finding the first black pixel from every side of the picture. The segmentation mask is stored as image-file and the bounding box's coordinates are written to the JSON-file.

It would also be possible to calculate the segmentation mask directly from the recorded data without a render of the scene, if it would be possible to reliably separate object and background. This would require strongly different color shades and could lead to problems in different lighting

[14]Blender – open-source 3D creation suite: https://www.blender.org/

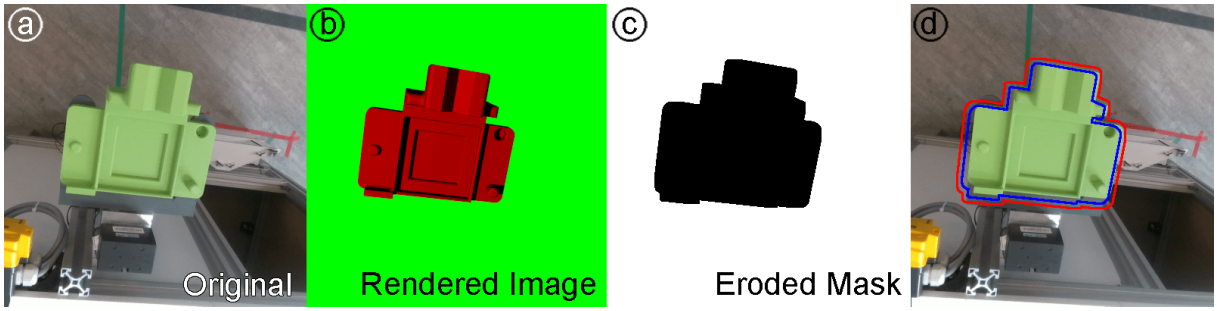


Figure 24: Post-processing steps and contour visualization: Original image (a), Rendered image (b), Binarized and eroded segmentation mask (c), Contours of original (blue) and eroded (red) segmentation mask overlaid on original image (d)

conditions since thresholds could vary. Another possibility to create segmentation masks from real-world data would be capturing the scene with a depth camera from the same perspective with and without the object. This would enable calculating an accurate segmentation mask by simply subtracting the images, but would require capturing the scene from all poses twice.

Using the introduced method for 6D pose-annotation of real-world scenes, large scale annotated datasets can be generated rapidly. The created datasets can be used for 6D pose estimation, object detection and image segmentation tasks. Due to the modular structure and clear interfaces, deploying the dataset-generation software for different hardware would be possible effortlessly.

CAD models of the object are only necessary, if segmentation masks have to be calculated, not if only a 6D pose annotated dataset is to be recorded. Annotation tools such as LabelFusion [20] need very exact CAD models to work correctly. Furthermore, depth data is used for refinement of the marker pose and for capturing additional training data only. The approach would also work with an RGB camera. Using a marker for object pose determination at the beginning of the capturing process brings the advantages, that no cumbersome post-processing for removing markers from images or computationally expensive fitting of CAD models are necessary. On the other hand, this limits the approach to objects big enough for placing a marker on them. Due to random poses for capturing single images being calculated and not extracting data from video streams, a diverse dataset with few redundancies is created.

5 Pose-Estimation System Training and Evaluation

To analyze the performance of pose estimation networks when trained using different datasets, annotated datasets of real scenes are created using the presented data generation pipeline. These datasets are enhanced with synthetic datasets of rendered scenes and extended using data augmentation.

This section gives an overview of the methods for synthetic data generation and augmentation, and the different datasets and their biases regarding scene diversity and pose distribution are analyzed. Furthermore, the training procedure and parameters, as well as the trained network models using the given datasets are presented. Finally, the procedures for quantitative and qualitative evaluation are described.

5.1 Generation of Synthetic Training Data

Synthetic ground-truth training data can be generated in different ways (see Section 3.1.2). While photorealistic rendering tries to imitate reality to the greatest possible extent [109], domain randomization aims to provide high variability in semi-realistic data such that the model generalizes to real-world data and real images appear as just another variation [95, 42].

Since generating photorealistic data is computationally expensive and rendered images look equally to real-world views, only a dataset with domain randomized images is created. Therefore, the Unreal Engine plugin *NVIDIA Dataset Synthesizer (NDDS)*[15] [144] is used, which has also been deployed for capturing the FAT-dataset [17]. This plugin is able to capture data at a rate of 50-100 Hz and stores an RGB and depth image, masks for instance- and category-level segmentation and meta-data including the objects' poses [144] (see Section 3.4).

NDDS provides different modules for capturing and spawning data. A virtual camera is used as scene capturer defining which data should be captured in which format. It is set to the parameters of the real camera used in this work. The virtual camera points at a volume for spawning training objects, enclosed by a bigger volume where distractor objects can spawn. When recording starts, a random number of training objects spawn in the inner volume while simple geometric shapes appear in the distractor volume. All objects are moved around and rotated randomly while the background of the scene switches between a set of random textures to generate unique images. A subset of the generated data is visible in Figure 25.

[15] NVIDIA Dataset Synthesizer: https://github.com/NVIDIA/Dataset_Synthesizer

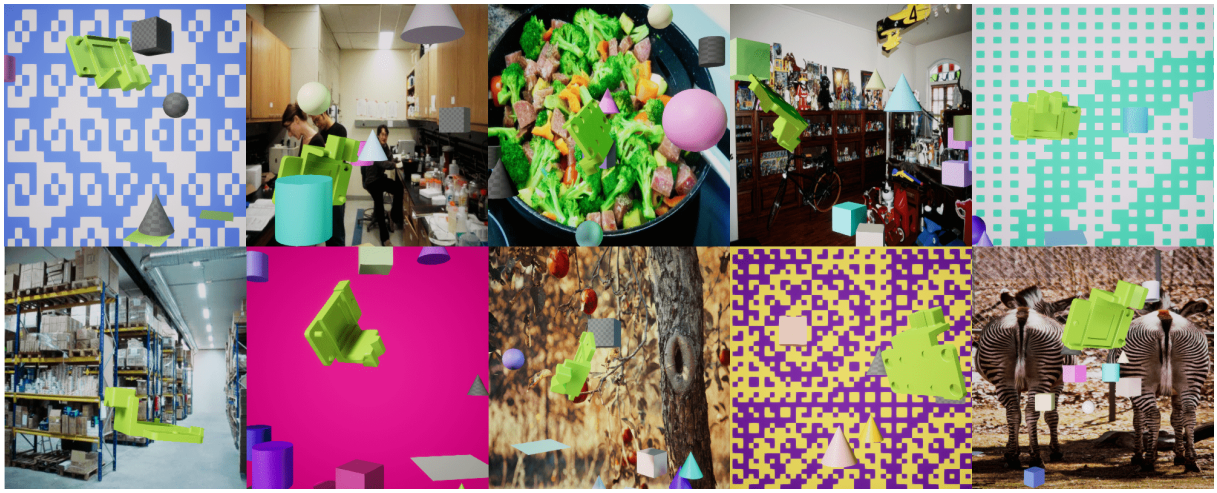


Figure 25: Examples of generated DR training data: The green object of interest (carrier), surrounded and partially occluded by distractor objects, placed in front of random pictures and patterns.

The following aspects of scenes have been randomized, similar to Tremblay et al. [29]:

- Distractors: Number, types and textures
- Backgrounds:
  - Solid colored
  - Procedurally generated with random multicolored grid patterns
  - Photograph from a subset of 1000 images from the COCO dataset [163] and 1200 from the dataset by Quattoni et al. [164]
- Poses: Sampled uniformly for training objects and distractors
- Directional lights: Orientation, color, intensity

5.2 Data Augmentation

Data augmentation (see Section 3.2.3) is commonly used to increase the size and diversity of datasets without extra labeling costs by applying geometric and photometric transformations. Thereby, networks learn to ignore influences like object positions in images or scene lighting and become more robust [114, 116, 115, 108].

Most deep learning frameworks implement basic image transformation such as flipping, rotating, scaling and cropping [122]. Also, the implemented network DOPE applies augmentations to every image before each epoch [29]. Augmentations are not used to produce more training data, so only augmented versions of the images are used for training. The framework adds random contrast and brightness to the training data, as well as a constant Gaussian noise. Furthermore, geometric transformations of up to 15° and 50 px in all directions are applied.

To evaluate the performance of the pose estimation network using augmented data and investigate differences between using a bigger dataset with unique images and a smaller dataset that is expanded by augmenting data, a small dataset is augmented multiple times. For aug-

mentation, the python framework albumentations[16] by Buslaev et al. [122] is used. This library provides a wide variety of different photometric and geometric augmentations and focuses on high computational speed.

As described above, at each capture-position, multiple images are recorded with the robot's last axes slightly turned, which leads to a broad distribution of object poses. Furthermore, the DOPE framework applies random rotations and translations to each image. Hence, only photometric augmentations are performed. If also geometric transformations would be computed, not only training images but also related meta-data would have to be transformed to correctly de-

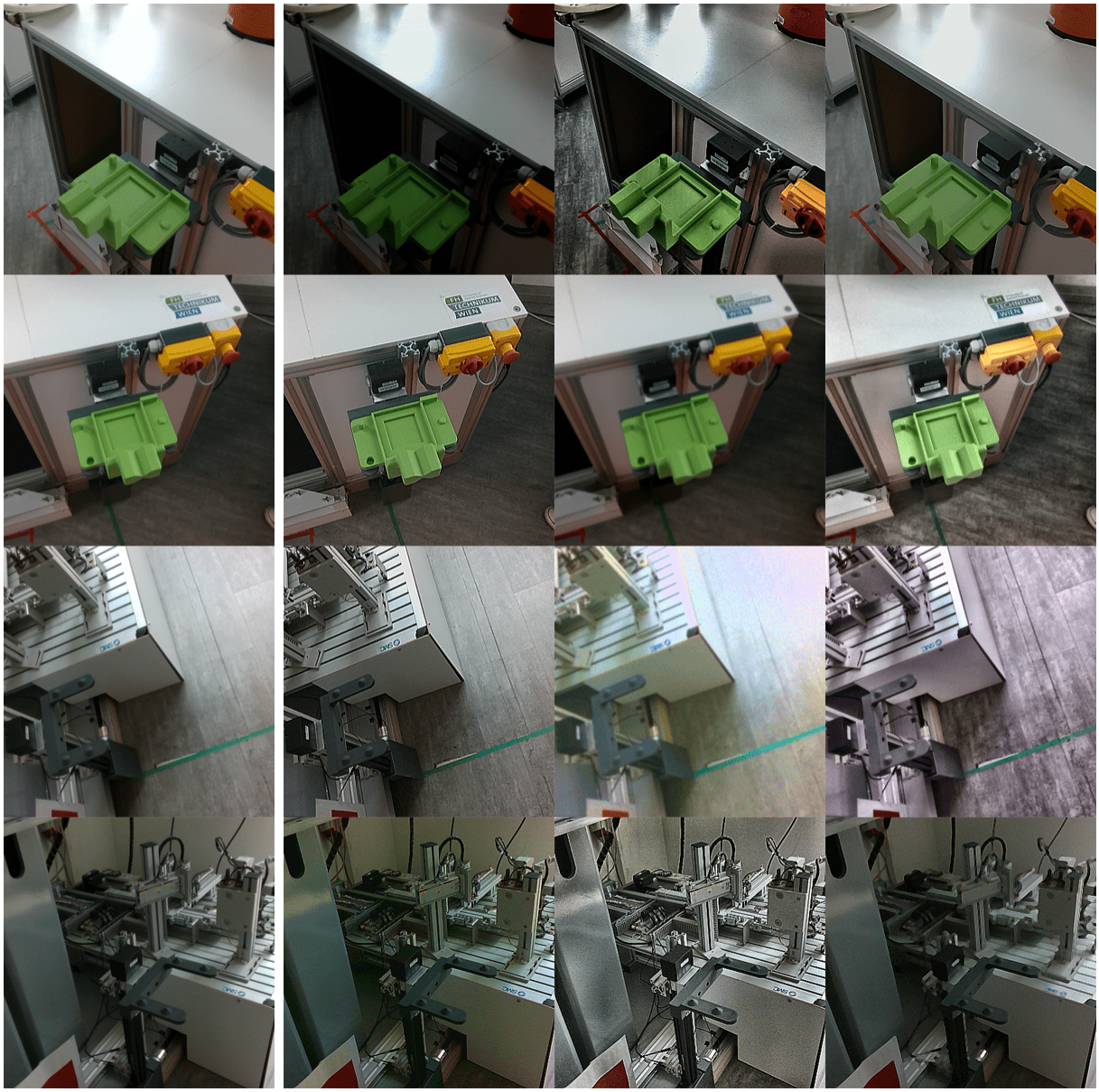


Figure 26: Examples of applied augmentations: Multiple augmentations applied combined to each picture. The left column shows original photos.

[16] Albumentations – fast image augmentation library: https://github.com/albu/albumentations

scribe object poses and corner positions for the transformed images. To ensure high variability of augmented data, each image is transformed a randomly chosen number of times, and also type and degree of augmentation are chosen randomly. Augmentations are composed from Table 2, according to the given probabilities. Values in square brackets are limits for the applied filters. Figure 27 shows the different augmentations for an average image of the dataset.

Table 2: Applied types of augmentations: Multiple main categories (grey background) are chosen according to the given probabilities. One augmentation of each selected category is chosen and applied using random values in the given ranges in square brackets.

Augmentation	Probability	Degree of Augmentation
Color	**30%**	
HSV-Shift (Hue Saturation Value)		[0, 10][0, 30][0, 20]
RGB-Shift (Red Green Blue)		[0, 15][0, 15][0, 15]
Gamma-shift		[50, 130]
Brightness and Contrast	**70%**	
Contrast Limited Adaptive Histogram Equalization (CLAHE)		[0, 3]
Brightness		[0, 0.3]
Contrast		[0, 0.3]
Brightness and Contrast		[0, 0.35][0, 0.35]
Blur	**20%**	
Gaussian		[0, 5]
Median		[0, 5]
Noise	**20%**	
Gauss		[5, 10]
Edges	**30%**	
JPEG-Compression		[40, 90]
Sharpen		[0.2, 0.5]

The final image transformation applied to a photo is composed of multiple augmentations by first choosing the main categories (color, brightness and contrast, blur, noise, edges) according to their probability. Since brightness and contrast are especially difficult to control in the given robot's environment, it is most likely that an augmentation from this category is included. Then, from each selected main category, one augmentation is randomly chosen. All augmentations are equally probable. The degree of augmentation is selected randomly from the given ranges in square brackets. Some examples of augmented images of the dataset with multiple augmentations applied are presented in Figure 26.

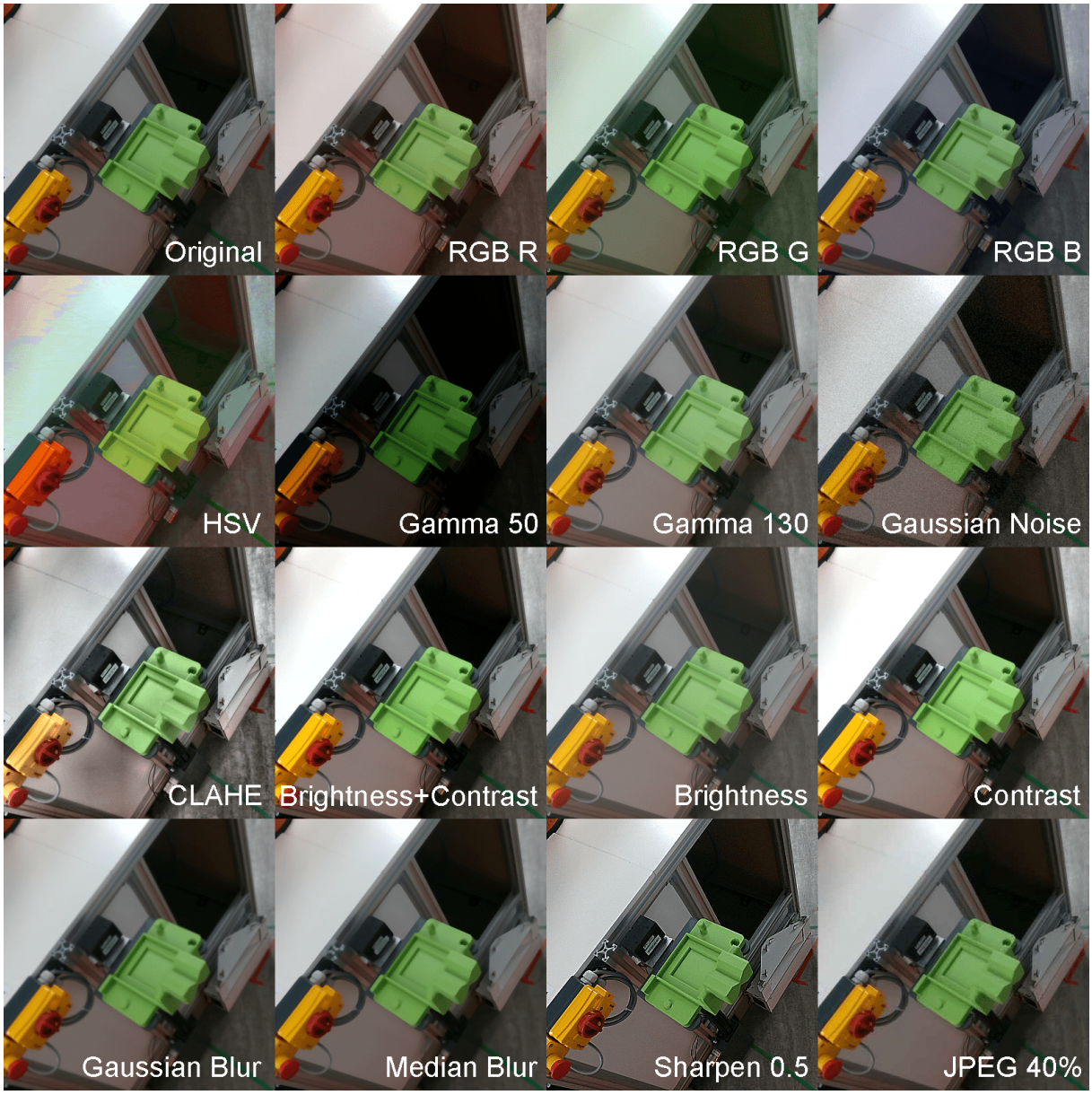


Figure 27: Examples of applied augmentations: Only one augmentation applied to a typical image in the dataset. If no value is given, the maximum level of transformation has been applied. The image in the top left corner shows the original picture.

5.3 Datasets

Datasets can be categorized into training, validation and test datasets. While a training dataset is a set of examples used to fit the parameters of a neural network, a validation dataset is used for tuning parameters of the network such as the number of hidden units. A test dataset contains a set of examples for evaluating the performance of a fully trained network [165]. The test dataset should be completely locked away while training and validation to ensure independent evaluation [166].

Since an already implemented network is used in this work and therefore no hyperparameters are to be chosen, no validation datasets are necessary. For analyzing the performance of the neural network depending on type and size of the training dataset, different subsets have been created which are listed in Table 3. There are datasets for analyzing influences of dataset-size (datasets 1-5), real and synthetic data (datasets 8-9) and unique and augmented data (dataset 6). The network performance is evaluated using a test dataset (dataset 7) and the results of the carrier are reviewed using datasets of the holder (datasets 10-12).

Table 3: Created datasets of the objects carrier and holder: For each dataset, the total number of samples, as well as its size relative to the biggest dataset are given. The test-datasets used for evaluation are marked grey.

ID	Object	Number of Samples		Data Type	Purpose
1	Carrier	2958	20% of train	Real	Train
2	Carrier	5916	40% of train	Real	Train
3	Carrier	8874	60% of train	Real	Train
4	Carrier	11832	80% of train	Real	Train
5	Carrier	14789	100% of train	Real	Train
6	Carrier	8874	60% of train	Real 2958 + Augmented (2x)	Train
7	Carrier	3698	20% of all	Real	Test
8	Carrier	15000		Syntehtic (DR)	Train
9	Carrier	14789 + 15000		Real + Synthetic (DR) mixed	Train
10	Holder	5284	100% of train	Real	Train
11	Holder	13170	250% of train	Real 5284 + Augmented (1-2x)	Train
12	Holder	1321	20% of all	Real	Test

About 18,500 labeled real images of the carrier have been captured. All pictures have been split into a training and test dataset by choosing images evenly distributed in an 80:20 ratio (datasets 5, 7). The training dataset (dataset 5) containing about 15,000 images has furthermore been evenly divided into five subsets containing about 3,000 images each. These subsets have been merged again to form datasets containing approximately 3/6/9/12/15,000 images (datasets 1-5) for analyzing the performance of the neural network for different dataset sizes.

All images of the smallest dataset (dataset 1) have been copied and augmented twice according to Section 5.2 to form another dataset containing about 9,000 images from only 3,000 unique images (dataset 6). This dataset is used for analyzing the effects of data augmentation on the neural network performance. For analyzing how synthetic domain randomized data influences the network output, a dataset containing 15,000 DR images (dataset 8) and another one

using the 15,000 DR images (dataset 8) and about 15,000 real images (dataset 5) has been created (dataset 9).

Best results in terms of effort and grasp-success with respect to capturing and training time for the carrier are observable for medium-sized datasets with real images that are augmented multiple times (see Section 6). Therefore, for training the neural network on the holder, a dataset containing about 13,100 images (dataset 11) has been generated from about 5,300 individual images (dataset 10), which have been augmented once or twice, leaving about 20% of all initially captured images aside for testing (dataset 12).

5.3.1 Scene Diversity

Scene diversity has a large impact on the performance of neural networks. Thus, training should be carried out in all possible environments in which detection should take place. Training images have to show the object in all poses, which should be detectable [39]. Therefore, all training images have been captured in the real environment at three different stations where the objects possibly could be placed. Since the environment features big windows, training data has been captured at different times of the day and on different days to include images with varying lighting on sunny, cloudy and rainy days, as well as direct sunlight. In Figure 28, images captured at different environments (three different stations) and lighting conditions (rainy, cloudy, sunny, direct sunlight) are visualized. This diversity ensures high detection rates at different environmental conditions.

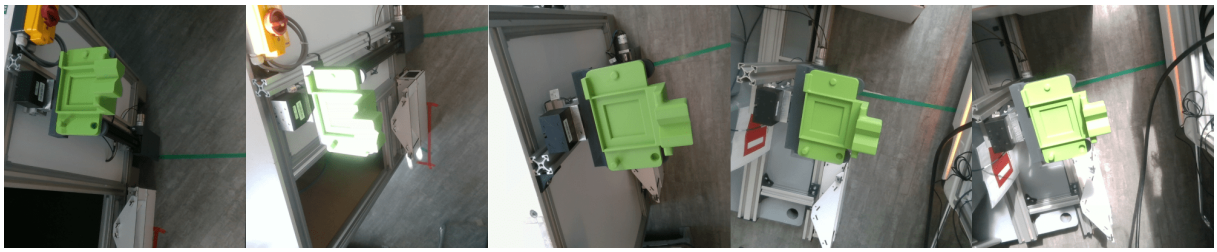


Figure 28: Images captured at different stations (SX) and lighting conditions: S1 rainy, S1 direct sun, S2 sunny, S3 cloudy, S3 direct sun

5.3.2 Distribution of Poses

For reliable detection and pose estimation, images showing the objects in every possible pose are necessary. Since the object is always located towards a station on the same side, only images showing it from the opposite side are captured. To check if all datasets are evenly distributed, all subsets are analyzed according to included poses, distances between the object and the camera, as well as location of the centroid in the images. The distributions of the poses in the biggest real and synthetic training datasets (datasets 5, 8) of the carrier are visualized in Figure 29. The distributions of the test dataset, as well as of smaller datasets are similar due to uniformly sampling images from all recorded data when creating sub-datasets.

The recorded real data (blue) is not as equally distributed as the synthetic data (red). When the domain randomized dataset is generated, the object is randomly placed in a defined region that is cubically shaped. Thus, the object is captured in every possible pose and roll, pitch and yaw angles are equally distributed. The distribution of the object's centroid in the 2D images, as well as the distance from the camera, also show uniform distributions. Thus, also images with e.g. the object's bottom side visible are included in the dataset. Since the object is not to be detected in such poses in real-life applications, these unnecessary images increase dataset size and thereby training time.

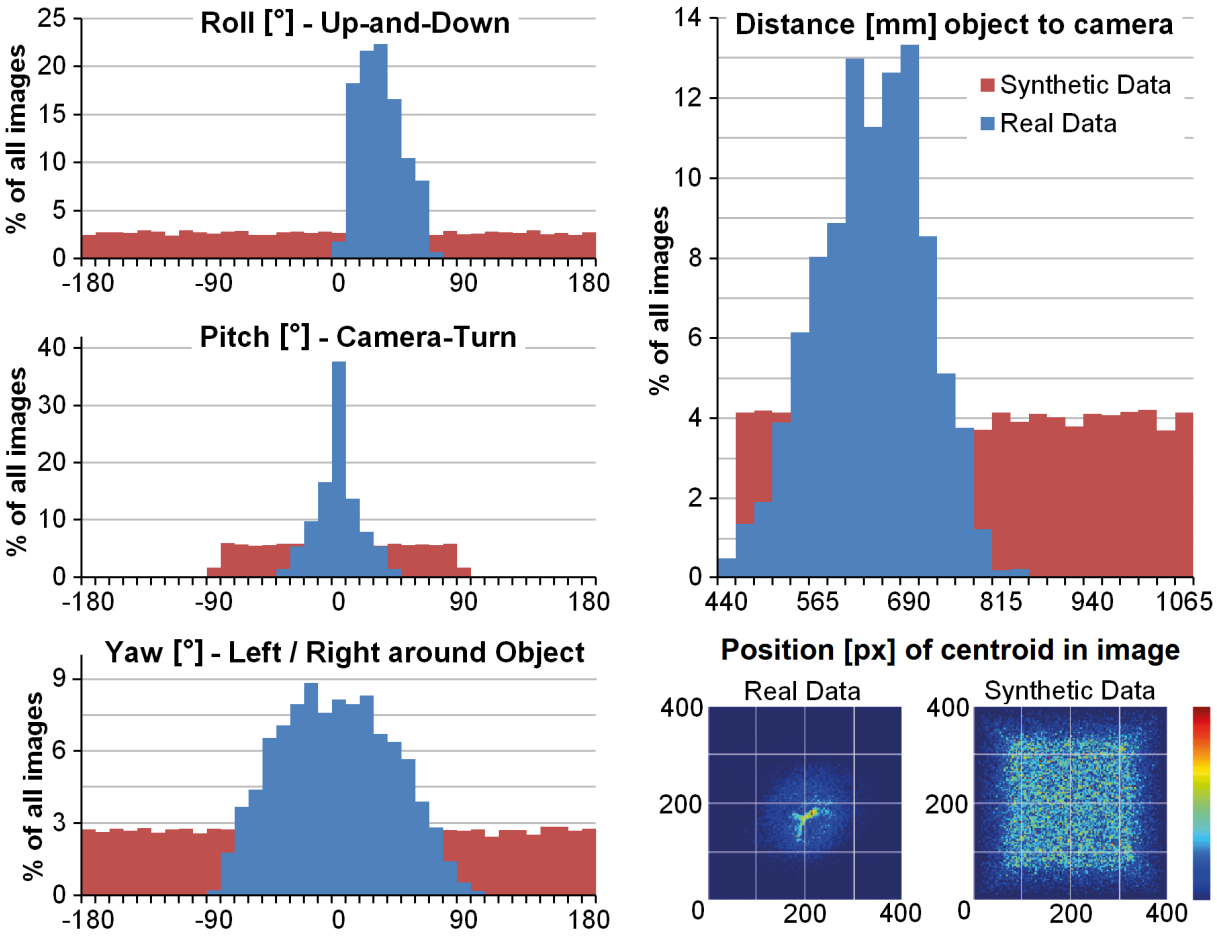


Figure 29: Distribution of poses in dataset 5 (real – blue) and dataset 8 (synthetic – red): The charts on the left-hand side show the distributions of the spatial angles (roll, pitch, yaw). The top right chart visualizes the distribution of the distances between object and camera and the bottom right pictures the positions of the centroids in real and synthetic images.

The recorded dataset is strongly biased due to the robot capturing images hemispherically with the camera turning around its axes randomly. Since the object is only to be detected from the side it is graspable from, the angles are distributed on the halved (yaw between -90° and +90°) upper hemisphere (roll between 0° and 90°) with the camera randomly turning (pitch between -45° and +45°). The distance between object and camera is approximately normally

distributed since the distances of the capture poses are varied randomly. These distributions are tailored for the given application of driving a mobile robot to a volatile position and grasping the object from its graspable side. The plot shows that most pictures capture the object with its centroid located at the image's center or in a 200 px field around it. Since it is desirable to detect the pose independent of the object's image location, a geometric augmentation is performed during training to eliminate this bias. The synthetic data shows even distributions for all parameters. The distribution of the centroids in the image is rectangularly shaped since the volume, in which the object is randomly relocated, is a cuboid.

5.4 Training of the Neural Network

Gathering a large-scale annotated, domain-specific training dataset is cumbersome [134, 136]. Hence, pre-training is commonly used to obtain better performance of networks trained on small datasets. Pre-training is a type of transfer learning, used to increase the generalization capacity of a deep network by taking knowledge learned by one network and transferring it to another [167]. In the case of CNNs, weights from other networks trained for specific tasks are used as starting point for the training of another task [130]. In this way, training starts with superior feature extractors enabling faster converge [168]. Tremblay et al. [169] show that even when trained with 1 million synthetic images, pre-training significantly improves the results.

As visualized in Figure 11 on page 15, the implemented neural network uses the first ten layers from VGG-19 [87] for computing image features. These layers can be initialized pre-trained on the ImageNet-dataset [88] as proposed by Tremblay et al. [29]. To analyze the impacts of pre-training for the created datasets, networks have been trained with and without being initialized with pre-trained weights.

During the training process of a neural network, the model's weights are manipulated such that the predicted output matches the expected output. Finding these weights is a gradient-based optimization problem which is solved by minimizing a loss function. As described in Section 2.2.3, the used neural network for pose estimation DOPE [29] uses an indirect approach. The network predicts the 2D locations of 3D keypoints and the direction from each keypoint to the object's centroid, which are used in a PnP algorithm for retrieving a pose. During training, these belief maps and vector fields are compared to ground-truth belief maps and vector fields, which are generated by placing 2D Gaussians at the cuboid locations and setting pixels to the normalized x- and y-components of the vectors respectively. The L_2 loss, as presented in Equation 10 [14], is calculated for both the belief maps and vector fields.

$$L_2Loss = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y_i} - \hat{y_i})^2 \tag{10}$$

This loss is defined as the average of all squared distances between ground-truth $\tilde{y}$ and estimation $\hat{y}$ [14] – it is the Euclidean distance between the ground-truth and estimated keypoint locations and vector coordinates.

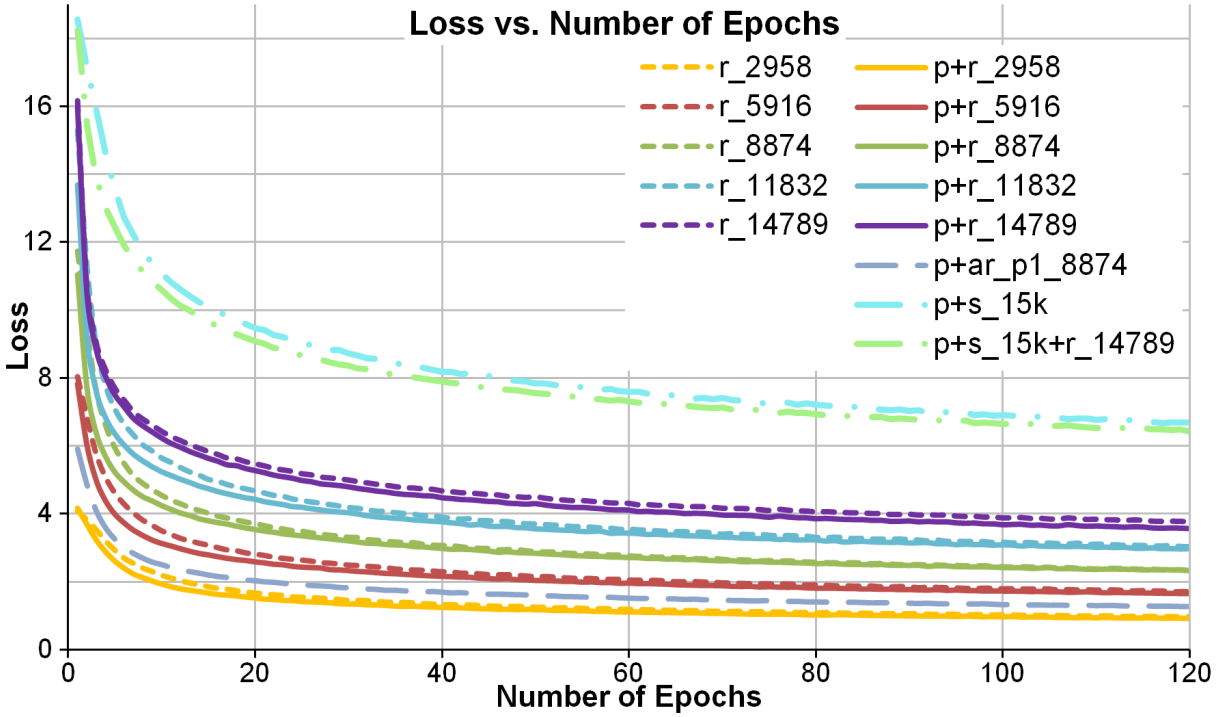
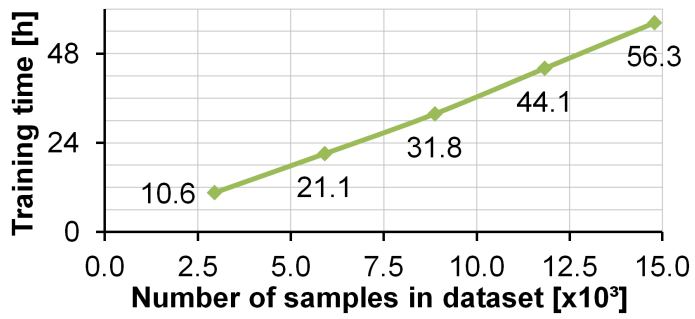


Figure 30: Loss vs. epochs: Networks initialized using random weigths (r_ – dashed lines), with pre-training (p+r_ – solid lines), trained on augmented data (p+ar_ – long-dashed) and synthetic data (p+s_ – dot-dashed). Different colors indicate varying dataset sizes.

In Figure 30, the development of the loss over training epochs is visualized for different networks, trained on the carrier. None-pre-trained networks (dashed lines) show especially at the begin of training higher loss values than pre-trained models (solid lines) because the model weights have to be adapted to a greater extent. Furthermore, losses are higher for models trained on bigger datasets since the accumulated loss per epoch is computed including every sample. Interestingly, the losses of the network trained on the augmented dataset (blue long-dashed line) containing about 9,000 images are more similar to the losses of the smallest dataset with 3,000 images, which is the base dataset augmented to create the bigger dataset.



Number of Samples	Training Time per Epoch [min]
2958	5.3
5916	10.6
8874	15.9
11832	22.0
14789	28.2

Figure 31: Training time vs. number of samples in a dataset: Necessary time increases linearly with the number of samples. Grey bars in the table illustrate training time (Full bar = 60 minutes).

Losses calculated for the synthetic and the mixed synthetic-real dataset (dot-dashed lines) are similar although the second dataset is about double the size. The losses of all networks converge to a final value during the last epochs of training. Since loss stays approximately constant, training for more epochs would not lead to significantly better results.

The networks have been trained and tested on an NVIDIA GTX 1080[17] graphics card. Training has been performed for 120 epochs with a learning rate of 0.0001 as proposed by Tremblay et al. [29], using a batch-size of 16 images. All trained networks, as well as the used datasets, are listed in Table 4. In Figure 31, the training times using the given parameters are visualized. The time necessary to train an epoch increases approximately linearly with the number of samples in a dataset. This means that a double-sized dataset needs about the doubled training time. On average, about 110ms are needed to train a single sample.

Table 4: Trained neural networks with different datasets: The first column gives the network's ID. An *X* in the second column means that pre-trained weights have been used for network initialization. Dataset-IDs correspond with IDs of Table 3. The columns *Real* and *DR* indicate the number of samples in the datasets. The last column gives the network names used in the results (r: real data, s: synthetic data, p: pre-trained, ar: augmented real data). Networks I to XIV are trained on data of the carrier, XV and XVI of the holder. Augmented datasets are marked grey.

ID	Pretrained	Dataset ID	Real	DR	Epochs	Network Name
I		1	2958		120	r_2958_120
II		2	5916		120	r_5916_120
III		3	8874		120	r_8874_120
IV		4	11832		120	r_11832_120
V		5	14789		120	r_14789_120
VI	X	1	2958		120	p+r_2958_120
VII	X	2	5916		120	p+r_5916_120
VIII	X	3	8874		120	p+r_8874_120
IX	X	4	11832		120	p+r_11832_120
X	X	5	14789		120	p+r_14789_120
XI	X	6	8874		120	p+ar_p1_8874_120
XII	X	8		15000	120	p+s_15k_120
XIII	X	5+8	14789	15000	120+120	p+s_15k_120+r_14789_120
XIV	X	9	14789	15000	120	p+s_15k+r_14789_120
XV	X	10	5284		120	p+ar_13170_120
XVI	X	11	13170		120	p+r_5284_120

[17] NVIDIA GTX 1080: https://www.nvidia.com/de-at/geforce/products/10series/geforce-gtx-1080/

5.5 Evaluation of the Neural Network

Two different evaluations are carried out to analyze the system-performance. In a quantitative analysis, the pose-estimation system is used for processing each sample of a test dataset. The estimated poses are compared to the ground-truth poses, considering different error metrics. The goal of the qualitative analysis is evaluating if the accuracy of the system is high enough for a real-life robotic manipulation use case. In this section, the implementations of both evaluation methods are described. The results of these analyses are presented in Section 6.

5.5.1 Quantitative Analysis

In the quantitative analysis, the trained network is tested with each sample of the test dataset. As described in Section 2.3, different evaluation metrics for analyzing the performance of neural networks for pose estimation are possible. Due to their wide distribution and high informative value concerning pose estimation for grasping, translational and rotational errors, as well as ADD metrics are calculated for each image. These metrics are computed using a library[18] by Hodaň et al. [40]. Furthermore, a modified ADD metric is calculated, considering all detected cuboids and the centroid. To evaluate besides the accuracy of the complete system the performance of the neural network only, functions for visualizing the belief maps and vector fields

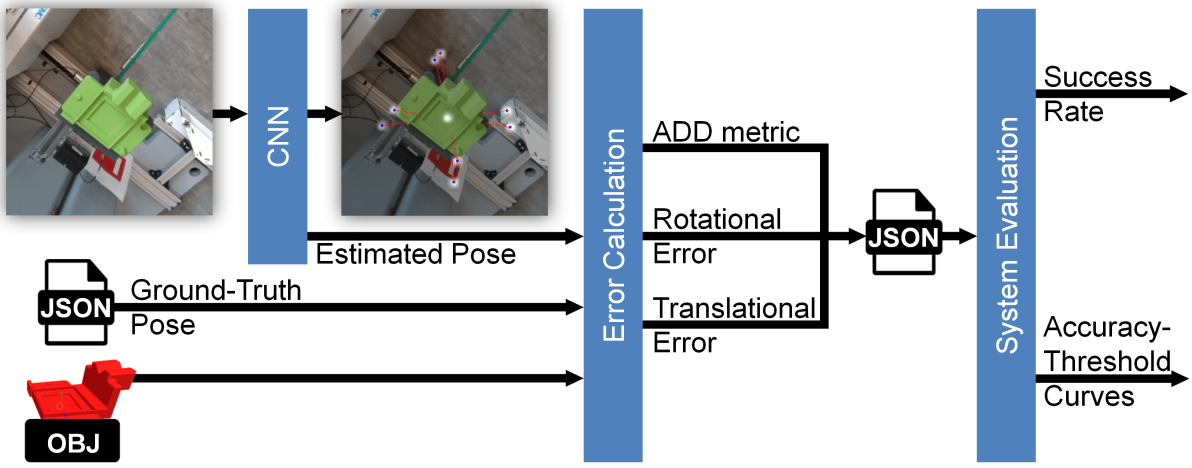


Figure 32: Structure of quantitative analysis: In the first step, the object's pose is estimated for every single sample and different error metrics are calculated by comparing it to the ground-truth pose. Furthermore, the estimations are visualized: Estimated cuboids (blue) and centroid (white) with confidence (white glow). Predicted vectors from cuboids to centroid (red). Secondly, the system performance is computed by merging the results for each image.

[18] Tools for evaluating 6D pose estimations: https://github.com/thodan/obj_pose_eval

predicted by the network are implemented. These overlay the belief maps – showing predicted cuboid positions and the centroid position – onto the image and draw a line for visualizing the directions of the vector fields – indicating the direction from each cuboid to the centroid.

As Figure 32 shows, the quantitative analysis is divided into two steps: first, the object's pose is estimated and metrics are calculated for each sample. This step uses the image, the ground-truth pose and the 3D model for the ADD metric. The results for each image are stored and can be analyzed in a second step to evaluate the performance of the complete pose-estimation system. This has the advantage that the computational and time expensive pose estimation and error calculation step must be performed only once to enable evaluation of the network in different ways using the stored data.

5.5.2 Qualitative Analysis

The goal of the qualitative analysis is proving that the accuracy of pose estimations is high enough for a robot to perform grasping tasks. For this purpose, a use case is developed: An articulated robot is mounted onto a mobile robot. This mobile platform is able to drive to specified goals but shows a deviation of several centimeters and degrees. Due to these inaccuracies, it is not possible to grasp "blind" for the object.

The robot presented in Section 4.1 is used to drive to a goal and search for the carrier. If its pose is detected, it is grasped and put down on the mobile platform. The mobile manipulator then navigates to a goal where the carrier is to be put down again. There the holder is searched and upon successful pose estimation the carrier put down on it. These steps are repeated multiple times to evaluate the reliability of the pose-estimation system and reasons for failures. The system architecture for the described process is visualized in Figure 33 and described in more detail below.

The software for the qualitative analysis is based on ROS [147]. In the first step, basic nodes are launched which provide configuration information, the structure and frames to the tf-tree and communication to the UR5 and the camera, as described in Section 4.3.1. Since movements of the mobile robot are necessary for the evaluation of the complete system, communication to the MiR100 is established using the ROS-package *mir_driver*[19]. The MiR runs its own ROS-core with packages for movement control and path planning. *mir_driver* works as interface between the MiR's core and the local ROS-core on the industrial computer. This ROS-bridge reads all topics of the MiR's core and provides the same topics in the local ROS-environment. When a message is sent on the MiR, the same message is sent on the local ROS-core and vice versa. If e.g. a goal-pose is sent in the local environment, it is forwarded to the MiR's core to make the robot move. The pose of the robot on the other hand is read from the corresponding MiR-topic and published locally to provide this information to all nodes on the local computer.

The DOPE ROS-package [20] [29] subscribes to an image-topic and publishes the poses of

[19] MiR-Driver: https://github.com/dfki-ric/mir_robot
[20] DOPE ROS-Package: https://github.com/NVlabs/Deep_Object_Pose

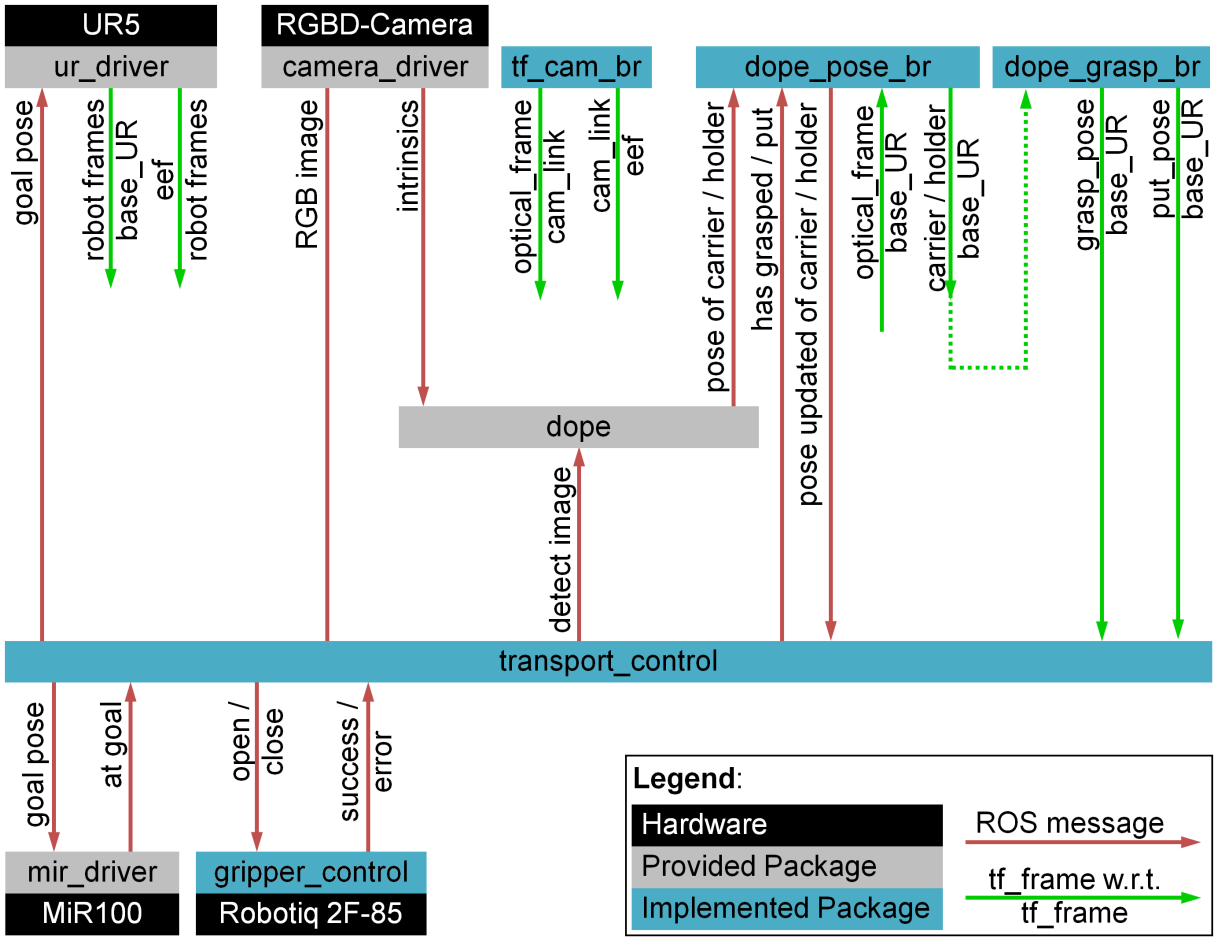


Figure 33: Software architecture for qualitative analysis: Hardware (black boxes), Provided ROS-Packages (gray boxes) and implemented-Packages (blue boxes). ROS-topics (red arrows) and tf-frames (green arrows – frame on top side of arrow with respect to (w.r.t.) frame on bottom side of arrow). Tf-frames used by multiple nodes are not connected. For easier readability, only necessary topics and frames are visualized.

detected objects with respect to the camera frame. To enable grasping, this pose must be represented in the robot's tf-tree. If broadcasted directly with respect to the camera, it would change with the manipulator moving, which is not desired. Therefore, the ROS-node *dope_pose_br* subscribes to the object's pose with respect to the camera and broadcasts it to the tf-tree once, only to read it out with respect to the robot's base. The pose of the object in relation to the base is then periodically broadcasted to the tf-tree to stay stable, also if the robotic arm moves. Since the object can not be grasped at the broadcasted pose, which is located at the centroid of the object, the module *dope_grasp_br* broadcasts static transformations from the detected pose to e.g. the grasp point or the put-down point the robot must be moved to.

Using these functionalities and communications, the node *transport_control* implements the logic of the use case, which is visualized in Figure 34 and described in more detail below. Given a pick-up and put-down goal with associated robot search pose, this node first sends the MiR to the pick-up position. The UR5 is moved to the search pose and an image is published for

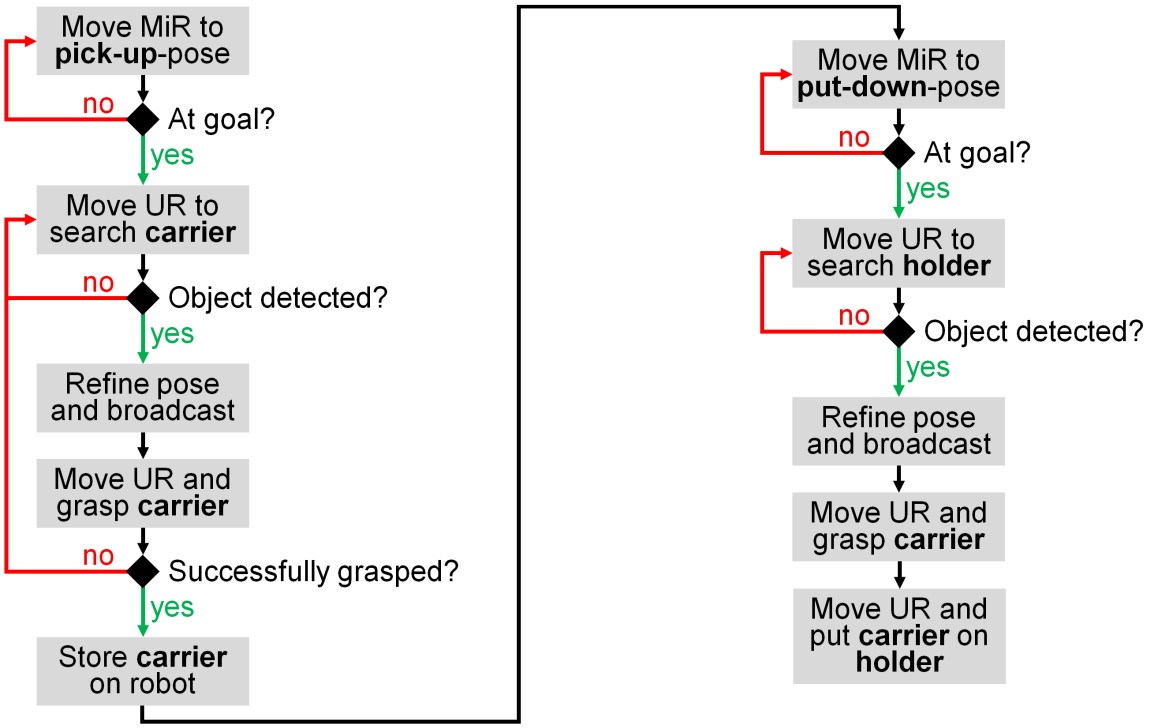


Figure 34: Program flow of *transport_control*: The MiR is sent to a goal, where the UR searches and grasps the carrier. At the put-down-goal the UR searches for the holder and puts the carrier down on it.

pose estimation using the module *dope*. If the carrier is not detectable, the environment is systematically searched. If the carrier's pose is detected, it is refined by moving the camera around and calculating the mean of all poses. This mean pose is broadcasted with respect to the robot's base, and a 3D model of the carrier is shown in the virtual environment. Then the robotic arm is moved to the broadcasted grasp position and the gripper is closed.

To enable controlling the gripper via the robot's teach pendant, it is connected directly to the UR5's control box. For opening and closing the gripper using the ROS node *gripper_control*, a special configuration of the robot's in- and outputs is necessary, as visualized in Figure 35. The robot's digital outputs can be set and read using the LAN-connection to the robot. A different

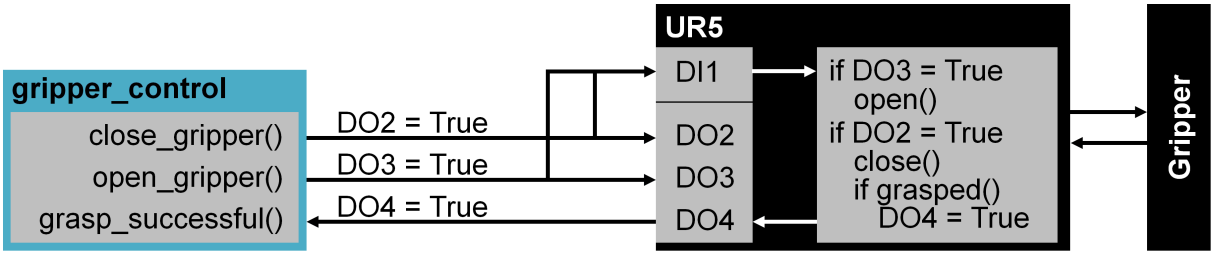


Figure 35: Opening and closing the UR5's gripper: *gripper_control* sets the UR5's digital pins (DO2 or DO3 – connected to DI1) to trigger a program-execution. The gripper is opened or closed and upon successfully grasping an object, DO4 is set to send a signal to *gripper_control*.

output is set, whether the gripper is to be opened (DO3) or closed (DO2). Both are connected to a digital input of the robot (DI1) which triggers the execution of a program on the robot's control unit. This program checks, which output is set, opens or closes the gripper respectively, and upon successfully grasping the object, it sets another output of the robot (DO4). This is read by *gripper_control* on the industrial computer, so it can be checked if a grasping attempt has been successful.

Upon successfully grasping the carrier, the robot moves it to a storing position on the mobile platform where it is temporarily put down. This is necessary because while holding an object, the robot's camera is covered and therefore, the pose of the holder – where the object should be placed – can not be estimated. The UR5 moves into a safe transport position and the MiR is sent to the put-down goal. Again, the search and refine-process starts, this time searching for the holder. If its pose is detected, the carrier is grasped from the temporary storage position and put down onto the holder. This process can be repeated multiple times to gather meaningful qualitative results.

Using the presented software for quantitative and qualitative analysis, evaluation of the different datasets and networks can be performed. Due to the modular structure and clear interfaces of the evaluation algorithms, deploying them for different hardware or neural networks would be possible without much effort.

6 Results and Discussion

This section provides an analysis of the accuracy of the calculated ground-truth pose during semi-automatic data generation. The accuracy mainly depends on the precision of the marker detection, why a quantitative analysis of impacts of the distance between camera and marker on the detected marker pose is given. Moreover, the results of the quantitative and qualitative evaluations are discussed. The performance of networks trained on different datasets using varying parameters are analyzed and guidelines regarding optimal dataset-parameters and training time are given. Furthermore, the results of qualitative experiments of the system using a real robot performing pick-and-place tasks are provided.

6.1 Accuracy of Ground-Truth Pose from Markers

The accuracy of the ground-truth object poses in the dataset influences directly the accuracy of the pose-estimation system. Calculating the overall accuracy of the system for capturing training data would be able using a motion capture system tracking the pose of the object and the camera mounted to the robot. In this case, only errors from this capturing system would be present. It is also possible to consider the errors of all components to estimate the error of the ground-truth pose annotations. The overall accuracy of the system for capturing training data is affected by errors from the robot and the marker detection. The UR5 provides repeatability of +/- 0.1 mm [170], which means if the robot is moved to the same pose multiple times, its TCP (tool center point) is always located at a maximum of 0.1 mm from the same position.

Neither the authors of the work presenting ArUco markers [64] nor papers that explicitly investigate the accuracy of marker detection [171, 172] provide absolute errors. That is due to errors depending mostly on the accuracy of the corner detection, which is influenced by numerous factors such as lighting and distance to the marker [64, 172].

The overall system accuracy influenced by these errors can be evaluated qualitatively by visualizing the ground-truth 3D-bounding box around the object in the dataset images, as illustrated in Figure 36. The top row shows positive examples with no deviations visible. In the bottom row, selected images with faulty poses are visualized. Figure 36 (a) shows imprecisions in x- and y- direction. In image (b), a deviation in the z-direction is visible and Figure 36 (c) shows slightly tilted ground-truth pose. These errors are caused by inaccuracies of the marker detection, marker placement and the robot.

It would be possible to increase the precision of marker detection by using a marker-board as proposed by Yoon et al. [173]. Using marker-boards has the advantage of more corner

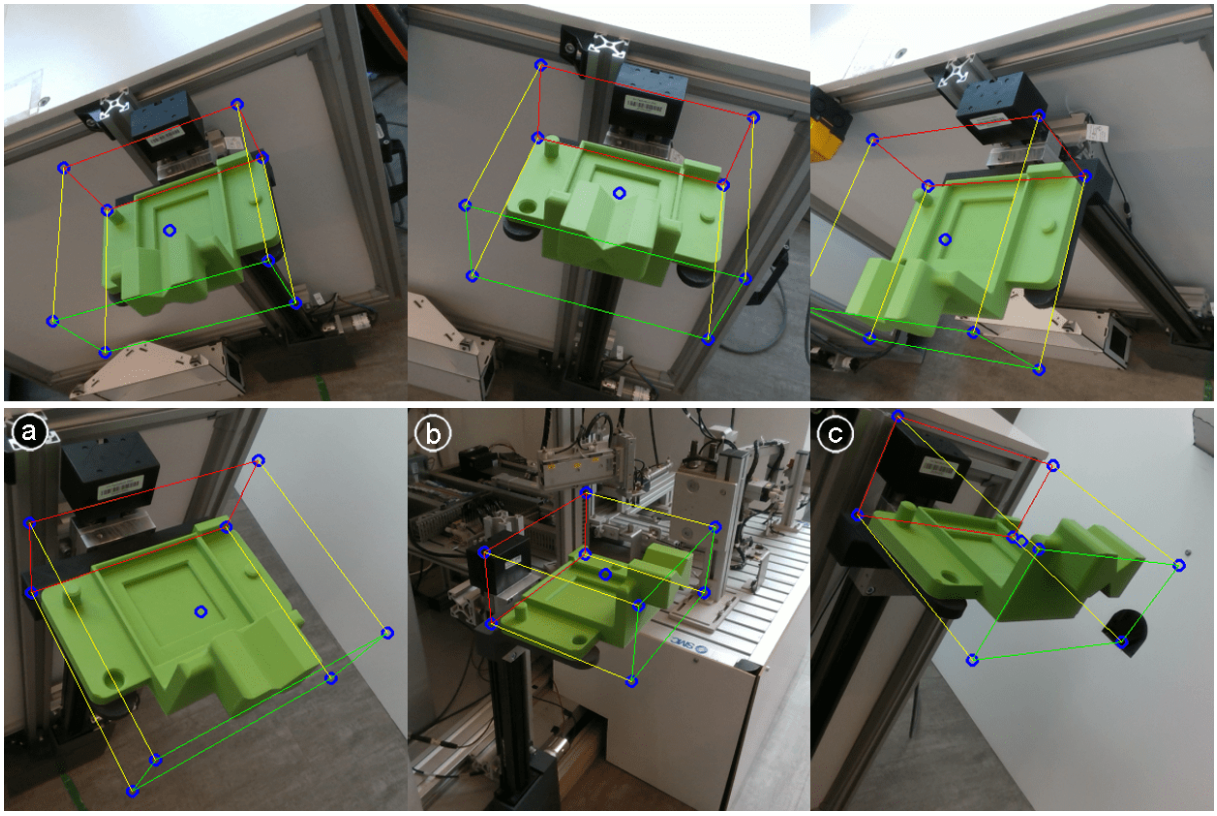


Figure 36: Qualitative analysis of accuracy of ground-truth bounding box: The top row shows selected examples with no recognisable deviations. In the bottom row, deviations in x-, y- (a) and z-direction (b), as well as rotational errors (c) are visible.

points being available, thus, enabling calculating a higher accuracy pose [65]. Using such a marker-board for ground-truth pose determination, which must be placed at a fixed position of the object, would be challenging due to its large size. It would also be possible to use other (active) markers, which could increase accuracy [23, 25]. Another option would be performing an ICP algorithm after pose determination using the marker. As described in Section 2.2.1, ICP algorithms fit a 3D model into a point cloud by minimizing the sum of squared distances between the matched points [48]. Thus the object's pose could be refined.

As described by Agnus et al. [172], the errors of the marker detection are smaller when the distance between marker and camera is smaller. The change of the detected marker pose with varying distance is analyzed in Figure 37. An experiment is performed where the marker is placed on the robot's base and the camera is located directly above it. The pose of the marker is stored in steps of 10 mm in a range of 150 mm between marker and camera (minimal distance for detection) and 360 mm (maximum distance which could be used for object pose determination). Lighting is kept constant throughout the experiment. The top right images of Figure 37 show the detected marker at a distance of 150 mm and 360 mm respectively between marker and camera. Since the marker is placed in the robot's base plane, a ground-truth z-value is present. As stated above, no ground-truth values for the other directions are determinable.

The deviation in the direction of the camera (z) is considered to be affected most by lighting [172]. Due to the use of a depth-camera in the setup, this position is refined using the camera's depth values, as described in Section 4.3.2.

Figure 37 shows that the x and y positions change linearly in a range of about 3 mm when the distance is increased by 200 mm. The starting point of the plot can not be considered ground-truth – the chart shows only how x and y change over increasing distance. This trend is also visible for the z direction. Since this chart shows absolute values related to ground-truth, an absolute deviation of the z value between -1.6 mm and -2.8 mm is visible. The dotted line shows the same graph shifted up by 1.5 mm which could be determined as systematic error through numerous marker-detections showing different perspectives. This systematic error is used for compensation during the pose-refinement process by subtracting 1.5 mm from the calculated distance if no depth-values are available. Measurements for the refined marker-pose are only present for distances greater than 260 mm since prior to that the camera does not provide any depth values, although according to the datasheet [174] the minimal depth-distance is 110 mm. An active stereoscopy camera is used, which means that a projector generates an infrared pattern on the surface for stereo-matching on texture-less surfaces. Though, better results can be obtained using features of strongly textured surfaces, which the marker does not provide [99]. Thus, usable depth values are only provided for a minimal depth-distance greater than

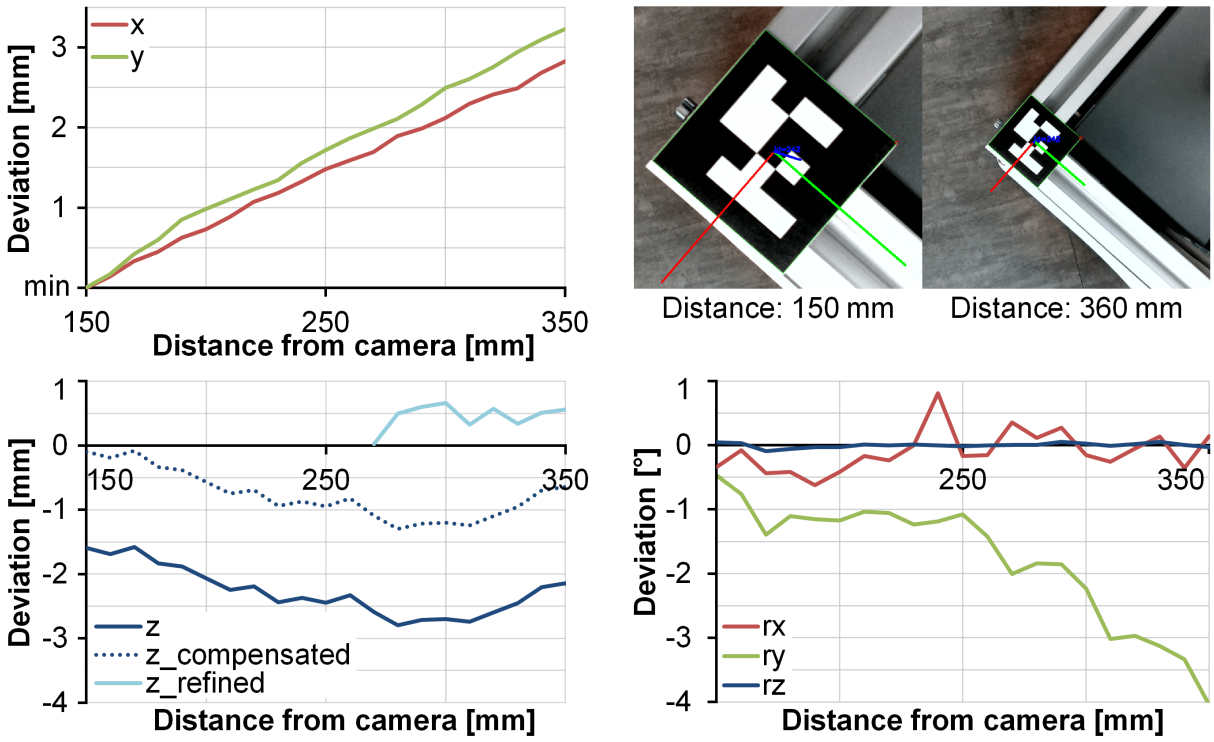


Figure 37: Errors of marker detection with increasing distance between object and camera: Deviation in x- and y-direction (top left) and z-direction with compensation of systematic error and depth-image-refined position (bottom left). Rotational errors around the axes (bottom right) and images of the marker detection at different distances (top right).

260 mm. If images are captured in smaller distances, the un-refined depth-values are used. The rotation around the z-axis is fairly stable while the rotation around the x-axis varies in a range of about one degree. The rotation around the y-axis shows errors of up to -4° increasing with the distance from the marker. This might be caused by incorrect detection of an edge of the marker. Since the marker is placed in the robot's base-plane, the rotations around the x- and y-axes can be considered ground-truth while the rotation around the z-axis shows only deviations over distance.

When determining the ground-truth object pose, distances between the marker and camera between 200 and 300 mm are used. Therefore, maximum errors of approximately 1 mm for the three directions and up to 2° for the rotations are possible. These results do include errors from marker detection along with imprecisions of the robot. To gather higher accuracy, multiple images from different perspectives are captured for computing the mean object pose, as described in Section 4.3.2. Thereby inaccuracies of ground-truth annotations can be minimized, as visualized in Figure 18 on page 36.

6.2 Quantitative Analysis of Pose-Estimation

In this section, the quantitative results of the pose-estimation system are analyzed and discussed. After a short comparison of different error metrics, impacts of pre-training, dataset-size, number of epochs, synthetic and real data and data augmentation are evaluated. Finally, a comparison between these dataset parameters, training time and accuracy is given for analyzing the best performance-to-effort-ratio. All experiments are carried out for the carrier and verified using the holder.

6.2.1 Comparison of Error-Metrics

As described in Section 5.5.1, during the quantitative analysis, different error metrics are calculated for evaluation. Due to their wide distribution and high informative value concerning pose estimation for grasping, translational and rotational errors, as well as ADD metrics for each sample are computed.

The ADD value (Average Distance of Model Points) is defined as the average distance of all model points x from *their transformed versions* if the model has no indistinguishable views (ADD), and from the *closest model point* if it has indistinguishable views (ADD-S) [16, 71, 13]. Since both analyzed objects do not have any indistinguishable views in the pose-range of the recorded data, the ADD-error is used. Translational and rotational errors are model-independent pose error functions which indicate deviations of an estimated pose $\hat{P} = (\hat{R}, \hat{t})$ with respect to the ground-truth pose $\tilde{P} = (\tilde{R}, \tilde{t})$ in mm and degrees accordingly. R is thereby a 3×3 rotation matrix and t a 3×1 translation vector [92, 40]. A more detailed description of the errors is provided in Section 2.3.

For comparison to ground-truth, the ADD metric uses all surface points of the model. This means that the ADD metric provides information about how equally the model's surface is aligned to a ground-truth model. Hence, this metric is especially meaningful for virtual and augmented reality tasks where the alignment of complete models to real-world models is demanded. Since the mean distance of *all* model surface-points is computed, small rotational and translational deviations lead to high distances for points far from the object's center. Only if the object is to be grasped at an exposed point, this error metric is a significant measurement for grasping performance.

The translational and rotational errors provide information about how the object's centroid is shifted and distorted. If the object is to be grasped near its center, this metric offers more meaningful information. The described behavior is visualized in Figure 38, which opposes the ADD and translational error and also shows the rotational error for the evaluation of a randomly chosen network. These charts show the accuracy of the networks plotted against the threshold – which is the number of samples in percent, which show a deviation smaller than the thresholds. E.g. About 90% of all analyzed samples show an error smaller than 8 mm.

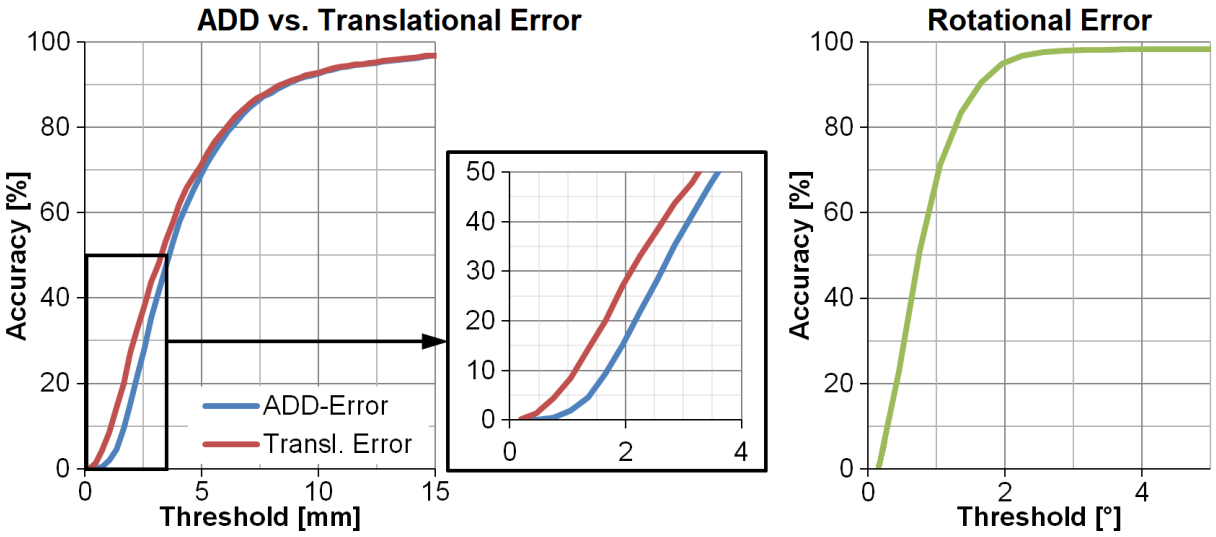


Figure 38: Comparison of error metrics: The ADD-metric (blue) shows higher deviations than the translational-error metric (red) in the lower range. At greater thresholds, the curves converge because the rotational errors (green) are relatively small against the translational errors.

It can be seen that the ADD metric shows higher deviations in the lower range up to about 10 mm threshold. Subsequent the curves proceed approximately equivalent. The zoomed-in chart in Figure 38 (center) shows that e.g. when considering the ADD-error about 18% deviate 2 mm or less, while the translational error states 28% in this range. At higher thresholds, the curves converge because the rotational errors, visualized in Figure 38 (right), are relatively small against the translational errors. About 90% of all samples show a rotational error smaller than 1.6°. As described above, the ADD metric shows higher deviations for exposed points due to rotational errors having bigger effects. Though, because of the rotational error being small for most samples, the ADD-errors deviate from the translational errors only for low thresholds.

Since the grasp-point for the carrier and the point where an object is placed on the holder are located near to the centroid, the translational error provides more valuable information for this analysis. Due to that, this error metric is used in the following evaluations. For grasping objects successfully, a deviation between 15 mm and 20 mm is allowed due to the gripper's opening width. For these thresholds, the errors of both metrics hardly differ.

6.2.2 Pre-Training and Dataset-Size

To analyze the impacts of pre-training and dataset-size, networks are trained with different parameters and datasets and the performances of these networks are compared. As stated in Section 5.3, all 14,789 training images are evenly divided into 5 different portions containing 2,958 samples each (20%). These subsets are merged to obtain datasets with 20%, 40%, 60%, 80% and 100% of the images for training. The pose-estimation system is trained using random weights and using pre-trained weights for the feature-extraction layers at initialization, as described in Section 5.4. In Figure 39, the translational errors of the resulting ten neural networks trained for 120 epochs are visualized. Networks without pre-training are plotted using dotted lines. The colors indicate different dataset-sizes.

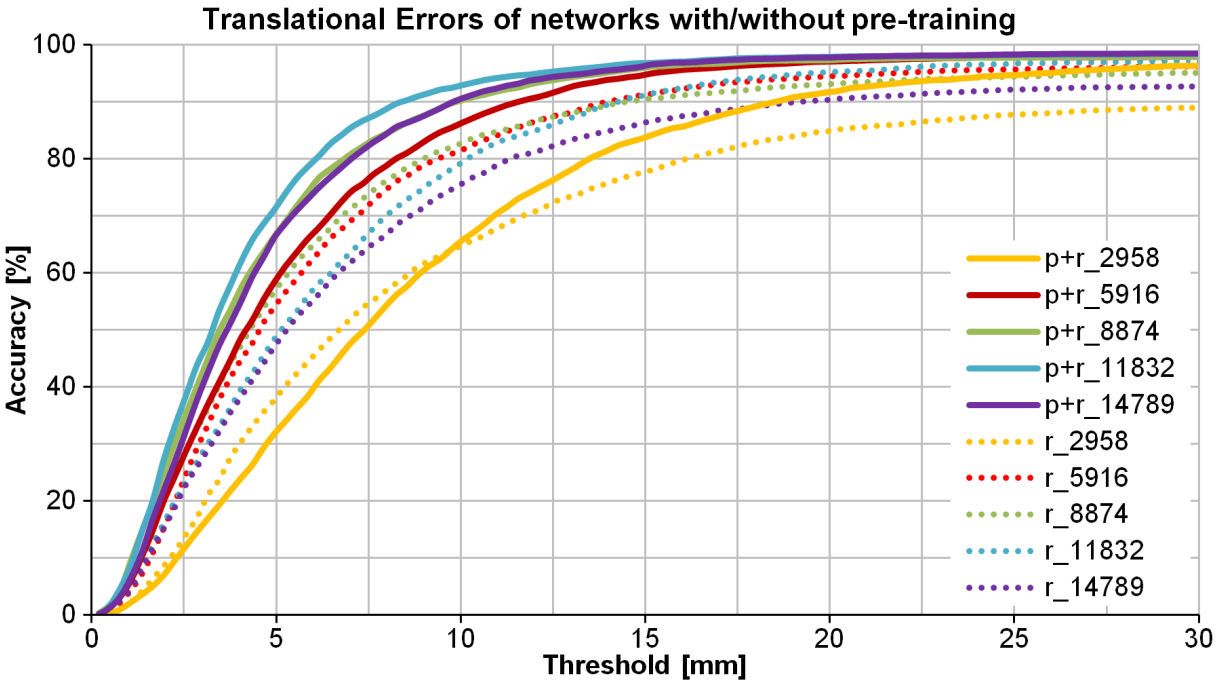


Figure 39: Translational Errors of pre-trained (p+r_ – solid lines) and randomly initialized networks (r_ – dotted lines). The colors indicate different datasets.

All pre-trained networks show better performance than the networks trained on the same datasets initialized with random weights, except the network trained on the smallest dataset. The performance of this network gets better than its not-pre-trained counterpart for thresholds bigger than 10 mm. In general, pre-training leads to significantly better performance. Even the

pre-trained network trained with only 40% of the dataset shows smaller translational errors than all networks without pre-training. Since pre-training does not influence training time negatively, using pre-trained network models is advisable in any case.

When considering the impacts of the dataset-size, a bigger dataset does not necessarily lead to better performance. For the pre-trained networks, the dataset including 80% of the samples shows best results, while the one containing 60% is equivalent to the full dataset. The biggest boost is observable between 20% and 40% of training images. For the networks initialized with random weights, the best accuracy is reached by the network trained on 60% of the samples, followed by 40%. The performance of the network trained on the full dataset is only better than the one trained on only 20% of the samples.

This means that more training data does not necessarily lead to higher accuracy. In these experiments, the best performance is observable for about 9,000 to 12,000 samples. The inferior performance of networks trained on more data can be explained by the bias of the dataset regarding scene diversity and pose-distribution, visualized in Figures 28 and 29. Since samples are not evenly distributed over lighting conditions and poses, when more data is used for training, also more samples showing specific conditions or poses are added. Data with other parameters is therefore interpreted as outliers and not detected correctly [175].

Figure 40 shows the performance of the same ten networks with and without pre-training considering rotational errors. These results are similar to the translational errors, although the pose-estimation is more accurate regarding the rotation. The accuracy stays approximately constant for the best performing networks after about 97% of all samples are reached for both

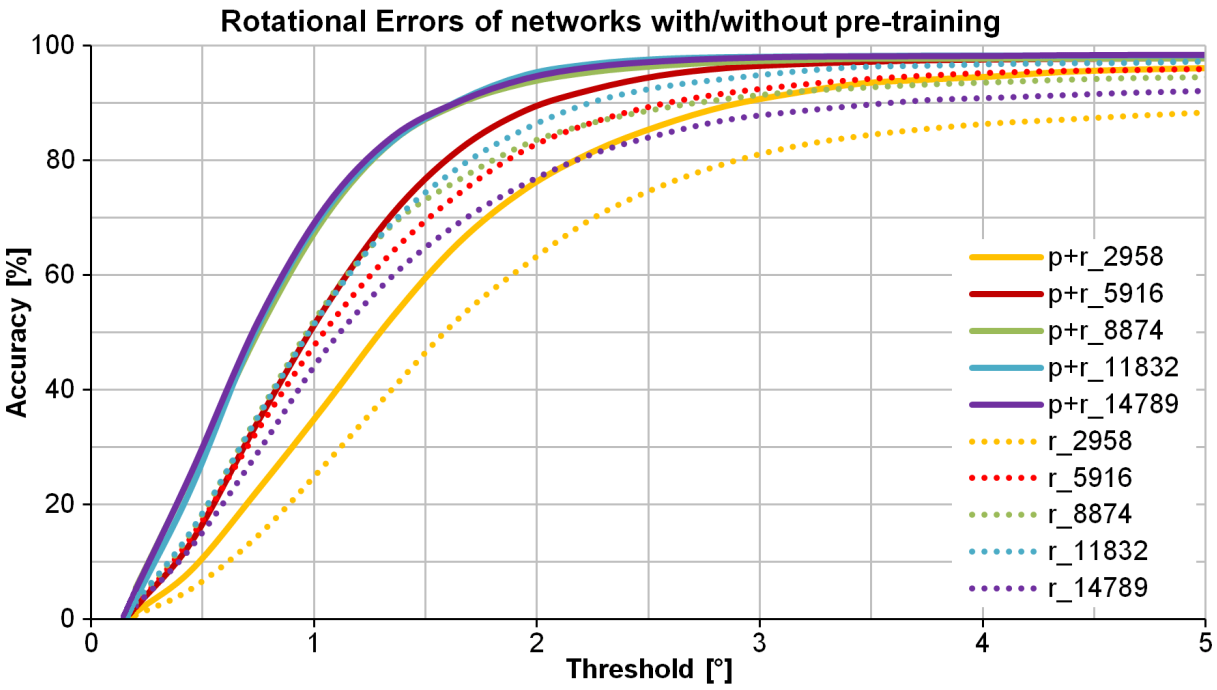


Figure 40: Rotational Errors of pre-trained (p+r_ – solid lines) and randomly initialized networks (r_ – dotted lines). The colors indicate different datasets.

metrics. This means that for approximately 3% of the samples the pose-estimation is wrong to such an extent it is not presented in the plots (bigger than 30 mm or 5°). This 97% limit is reached at a translational error of about 20 mm and a rotational error of 2.5°. When grasping an object, a wrong pose by 20 mm is considerably worse than by 2.5°.

Also for the rotational error, the networks initialized with random weights can not compete with the pre-trained systems. Again, the pre-trained network trained on the smallest dataset is the only one performing worse than all networks without pre-training, but still better than its counterpart without pre-training. Considering dataset-size, the pre-trained networks trained on 60%, 80% and 100% of the samples perform equivalent. The pose of 70% of the samples is estimated with very high accuracy of under 1°.

These results show that pre-training should be used if possible since performance is increased in any case without additional training time. Even when trained with 1 million images, pre-training significantly improves results [169]. Dataset-size also influences the performance of the networks, though more training data does not necessarily lead to better results. During these experiments, best accuracy is obtained using between 9,000 and 12,000 images (60% to 80%). This is caused by adding samples from a biased dataset, resulting in "confusion" of the pose-estimation system. The network gets more used to specific poses and lighting conditions since more training data showing these parameters are included, resulting in bad detections of other poses and conditions [175]. In general, the poses estimated by this system are more accurate concerning rotations. Approximately 97% of all poses show an error smaller than 2.5° and 20 mm.

6.2.3 Number of Epochs

All trained networks are trained for 120 epochs, as suggested by the developers of the DOPE network [29]. At the end of each epoch, the weights are stored. An evaluation has been performed for 30, 60, 90 and 120 epochs for each trained network to analyze impacts of the training time on the detection performance. The results are visualized in Figure 41. The blue bars show the number of samples the object has been detected in. Green and red bars present the Translation-20 and Translation-15 error-metric respectively. These values specify the percentages of samples where the error of the estimated translation is smaller than 20 or 15 mm.

Better overall performance of the pre-trained networks is also visible in this chart. For most of the networks, the metrics are better for a higher number of epochs. In some cases, the networks trained only for 90 epochs show better results, which is caused by overfitting. Overfitting means that because of the same training data being presented to the network multiple times, it corresponds too closely to this data. The loss while training keeps reducing since the network is still learning better estimations, but it fails to generalize and learns special features only visible in the training data [14]. This means that after a specific number of epochs the network in some cases overfits to the training data and therefore fails to interpret the test data correctly.

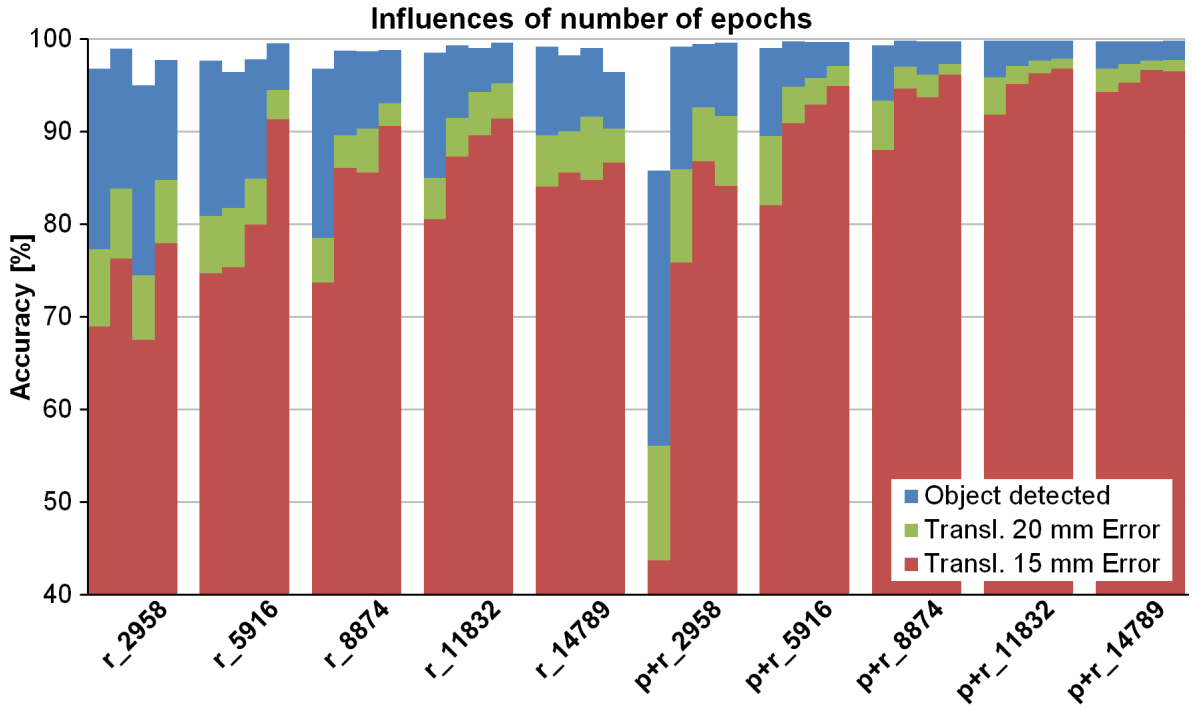


Figure 41: Influences of the number of epochs for networks trained on different datasets with and without pre-training: Overall detection performance (blue), translational-20 (green) and translational-15-error-metric (red). The bars show the results for 30, 60, 90 and 120 epochs.

Figure 42 shows the accuracy-threshold-curves for the networks trained on 80% and 100% of the training data. The networks trained on the smaller dataset (left) show increasing performances from 30 to 120 epochs. For the networks trained on the full dataset (right), the best results are achieved by the network trained only for 90 epochs.

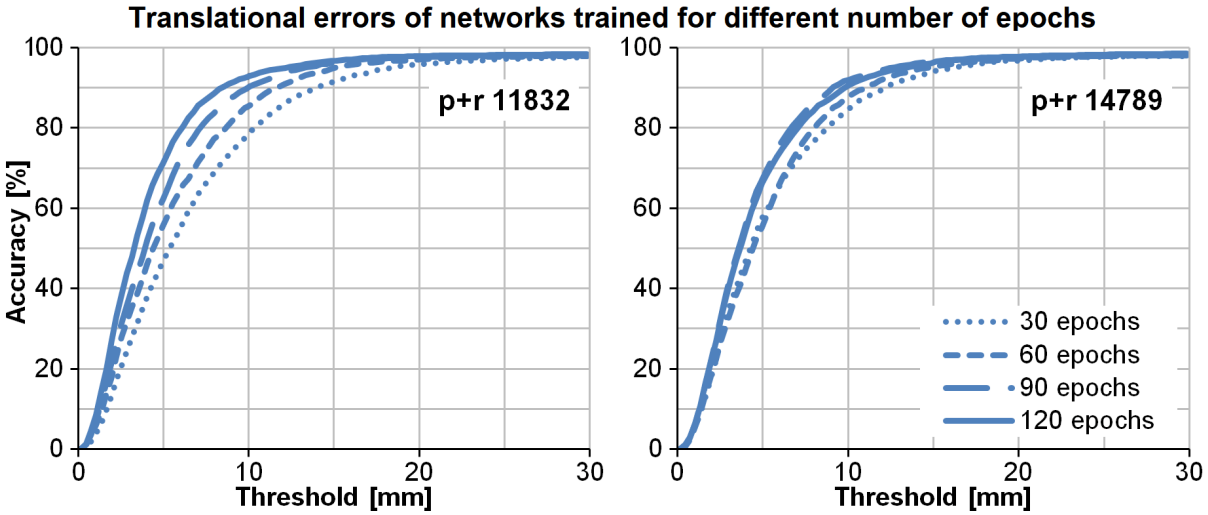


Figure 42: Influences of the number of epochs: Rising accuracy for the networks trained on 80% of the data (left) over increasing number of epochs. Overfitting of the networks trained on the bigger dataset (right) for more epochs.

These results show that a higher number of epochs and accompanying higher training time does not necessarily lead to better accuracy due to overfitting. Similar results have also been obtained by Tremblay et al. [29], who were able to reach the peak performance at a dataset containing 300,000 of their 1,000,000 synthetic images. In general, it is reasonable to perform training for a high number of epochs. The performance should always be evaluated for a different number of epochs to obtain best results. It is also possible to test the network periodically during training and stop when the lowest error metrics on the validation data are reached.

6.2.4 Synthetic and Real Data

In [29], the network has been trained on synthetic data only. The authors performed experiments using domain randomized and photo-realistically rendered images. In this work, a dataset containing 15,000 domain randomized images is used to train networks that are also evaluated on the test-dataset containing only real-world views. All networks trained on synthetic data are initialized pre-trained. One network is trained for 120 epochs on the synthetic images only (p+s_15k_120). This network is used as basis for a second network which is additionally trained on the full dataset containing about 15,000 real samples (p+s_15k_120+r_14789_120). The third network is trained on a dataset containing all images in one dataset featuring about 30,000 samples (p+s_15k+r_14789_120). Figure 43 visualizes the results for these networks.

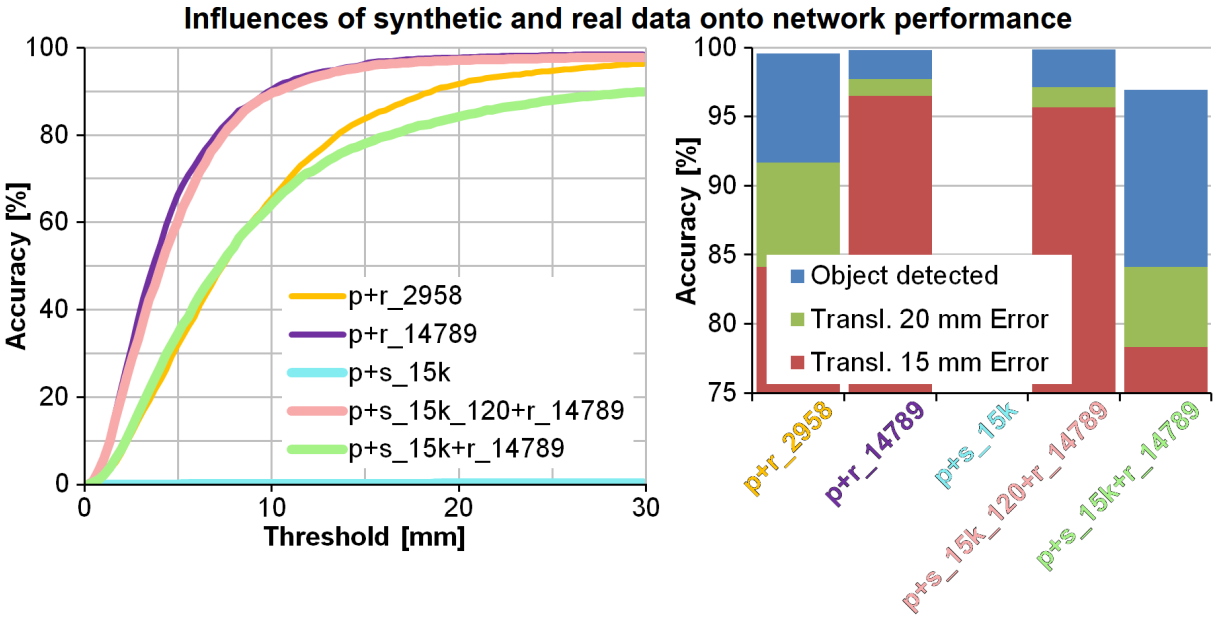


Figure 43: Influences of synthetic and real data: Networks trained on synthetic data compared to the models trained on the smallest (yellow) and biggest (violet) real dataset. 15k synthetic samples (cyan), 15k synthetic samples for 120 epochs and 14,789 real samples for 120 epochs (rose), 15k synthetic and 14,789 real samples mixed (green). Translational error accuracy-threshold-curve (left), Percentage of samples showing a specific error metric (right). Overall-detection-performance (blue), translation-20 (green) and translation-15 error-metric (red). Note that the vertical axis of the bar-chart starts at 75%.

The network trained on domain randomized images only is unable to detect the object in a single image. For the network trained additionally on the biggest real-world dataset, a similar performance as when using the real data only could be achieved. This means that doubling the training time does not provide better results. The weights which are fitted to the domain randomized data during the first run are completely changed to fit the real data in the second run. The network trained on a 50:50 mixed real and domain randomized dataset shows poorer performance than the one trained on about 3,000 real samples only.

To gather results similar to those achieved by Tremblay et al. [29], a much bigger synthetic dataset would be necessary. Also Tremblay et al. do not report useful performances for domain randomized data only when trained on a dataset including 60,000 samples. Results similar to those from the mixed dataset are achieved by using 120,000 samples from mixing domain randomized and photorealistic rendered images. Usable accuracy values are reached using domain randomized data only at a dataset size of 100,000 samples, which results in immense training times. Since the used test dataset in this work is biased similarly to the training dataset, the performance of real-data can not be reached with the mixed real-synthetic-data. Though, it is possible that the networks trained on the mixed data do generalize better and may work for a wider variety of data. It is also possible that the network struggles to generalize to the texture-less object in the domain randomized samples since only few features are extractable. However, such investigations would go beyond the scope of this work.

These results show that if no data from the target domain is available, domain randomized mixed with photorealistic training data is useful to train a network for high generalization. If data can be recorded in the target domain, it is possible to obtain good results using a small dataset.

6.2.5 Data Augmentation

As described in section 5.3, the smallest real-world dataset containing about 3,000 images has been augmented twice to gain 9,000 training images. A network has been trained using this dataset and the performance is compared in Figure 44 to the original dataset, the dataset containing 9,000 unique real images and to the network trained on 15,000 samples. A performance increase of 8% and 4% at a threshold of 15 mm and 20 mm respectively is observable for the translational error when the augmented dataset is compared to the original dataset. Compared to networks trained with datasets containing 9,000 or 15,000 real images, the network trained on the augmented dataset shows values lower by only 4% and 1% at thresholds of 15 mm and 20 mm respectively. Similar results are observable for the rotational errors. It can furthermore be assumed that the network trained on augmented data generalizes better to new data and overfitting is reduced due to more variation in the training dataset [14, 122, 130].

Due to these observations, when only a limited number of training data is available, data augmentation for introducing a higher number of different scene conditions and noise can boost the performance of networks. However, throughout the experiments, the accuracy obtained by a dataset containing the same number of uniquely recorded images could not be reached.

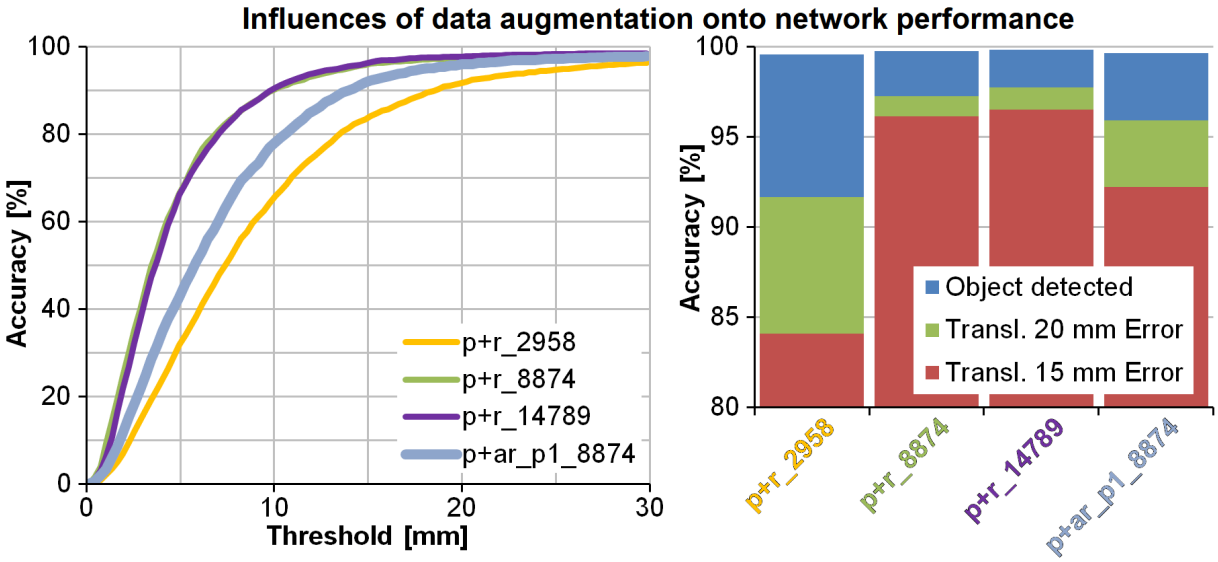


Figure 44: Influences of data augmentation: Different networks trained on 3,000 unique (yellow), 9,000 unique (green), 15,000 unique (violet) and 9,000 augmented samples (blue). Translational error accuracy-threshold-curve (left), Percentage of samples showing a specific error metric (right). Overall-detection-performance (blue), translation-20 (green) and translation-15 error-metric (red). Note that the vertical axis of the bar-chart starts at 80%.

6.2.6 Evaluation of Holder

The results for the carrier show that good performance can be reached using a dataset containing between 9,000 and 15,000 samples. Since gathering an annotated training dataset is a complex and time-consuming task, augmentations can be used to boost detection-rate. Generating a relatively small synthetic training dataset has not shown to improve results. Therefore,

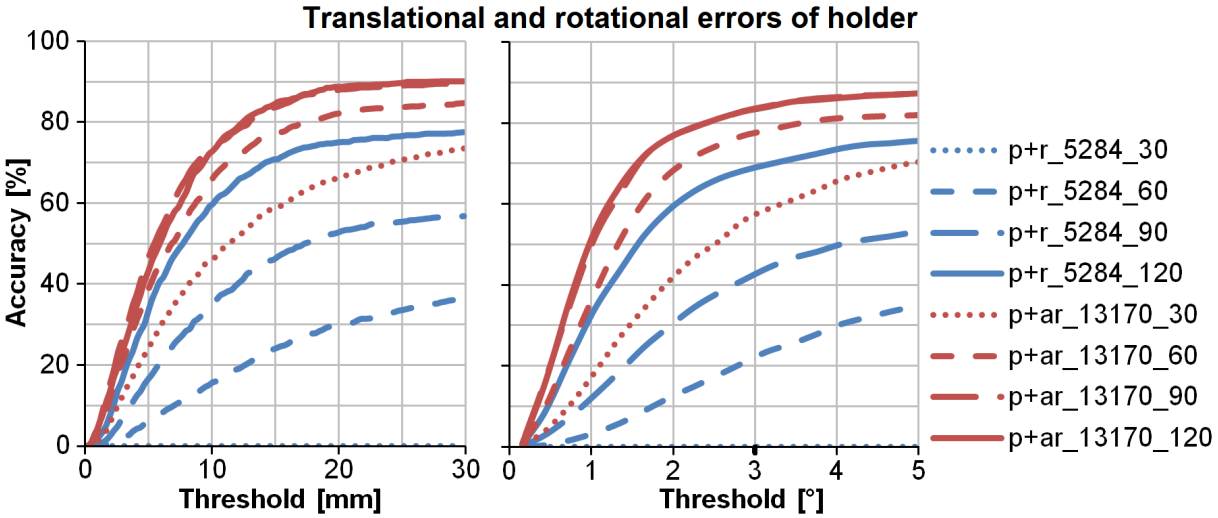


Figure 45: Translational (left) and rotational errors (right) of the holder, trained on two different datasets (blue: 5284 real samples, red: 13170 augmented samples) for 30, 60, 90 and 120 epochs.

for training a network for pose-estimation of the holder, a dataset containing about 5,300 individual images has been recorded. Each sample of this dataset has been augmented once or twice to generate a training dataset of about 13,000 images.

Figure 45 illustrates the accuracy-threshold-curves of translational and rotational errors for both datasets for networks trained for 30 to 120 epochs. All networks trained on augmented data (red) perform better than the networks trained on the original dataset when compared after 30 epochs. An overall accuracy boost of 14% is observable between the networks trained for 120 epochs on the original and augmented dataset. Rotations show similar behavior, although rotational errors of all trained models are higher than for the carrier (see Figure 40 on page 65). For the best performing model of the holder, the pose of about 87% of all samples has been detected with an accuracy smaller than 5°. The worst network trained on the smallest dataset of the carrier shows this limit at approximately 2.7°, indicating much better rotational detection performance. This is possibly caused by poorer contrast between the grey object and the background, but such investigations would go beyond the scope of this work.

This again shows the benefits of data augmentations when limited training data is available. Since the network trained on augmented data for 90 epochs performs slightly better than after 120 epochs, overfitting is present again. Nonetheless, satisfactory results could be achieved for the second object, confirming the results determined for the carrier.

6.2.7 Performance and Training Time

The presented quantitative results show that pre-training, dataset-size and type, data augmentation and number of epochs influence results. Pre-training strongly improves results while not increasing training time. Networks trained on reasonably sized datasets containing between 9,000 and 12,000 samples perform best while training time is slightly raised with every sample by approximately 110ms on the used hardware. Real data from the target domain usually performs better than much bigger datasets containing synthetic data and augmenting a small real dataset, which increases training time only for every sample, can also lead to performance boosts. The number of trained epochs influences training time most. Training on the same dataset for too long can lead to overfitting on the training dataset, why results on the test-dataset decrease for networks trained for more epochs.

To visualize these relationships, Table 5 provides an overview of all trained networks of the carrier and holder and their performances. It lists the percentage of samples evaluated with a translational error smaller than 15 mm and 20 mm. Furthermore, the mean rotational error for each network, which is calculated by averaging the rotational errors of all samples, as well as the total training time, are given.

The listed performance values in the table confirm the results discussed above. The comparison also shows that results do not increase to the extent the training time does. This is also visible in the bubble charts in Figure 46. The bubble size visualizes the size of the dataset, which is an indirect measure of training time. The colors indicate the data and training-type:

Synthetic data (green), augmented data (cyan), pre-trained networks (blue) and non-pre-trained networks (red). On the x-axis, the performance of the networks regarding translational-error-15mm-metric (left) and the mean rotational error (right) are plotted. The y-axis shows the training time in hours.

Considering the translational error, network IX shows the best performance to training-time ratio. Also networks VIII and X, which are trained on similar datasets, show comparable results. The performance of network XIII is in the same range, but the training time and dataset-size are immensely higher. When considering the mean rotational error, networks VIII and X show slightly better results than network IX, meaning that this network is superior regarding prediction

Table 5: Comparison of network performances and training times: Each trained network has a unique ID and network name. An *X* in the second column indicates if the model has been initialized with pre-trained weights. *Dataset ID* and *# Samples* identify the used dataset for training the networks. The columns for the translational error gives the percentages of test-samples where the estimated pose deviates less then 15 and 20 mm from ground-truth. The last columns show the mean rotational error and the training time. Background-colors indicate performance for each column (green: best, red: worst)

ID	Pretrained	Dataset ID	# Samples	Network Name	Transl. Err. ≤15 [%]	Transl. Err. ≤20 [%]	Rot. Err. Mean [°]	Training Time [h]
I		1	2958	r_2958_120	78.0	84.8	6.6	10.6
II		2	5916	r_5916_120	91.4	94.5	4.8	21.1
III		3	8874	r_8874_120	90.6	93.1	4.6	31.8
IV		4	11832	r_11832_120	91.4	95.2	4.5	44.1
V		5	14789	r_14789_120	86.6	90.3	5.2	56.3
VI	x	1	2958	p+r_2958_120	86.8	92.6	5.8	10.6
VII	x	2	5916	p+r_5916_120	94.9	97.1	4.4	21.1
VIII	x	3	8874	p+r_8874_120	96.1	97.3	3.4	31.8
IX	x	4	11832	p+r_11832_120	96.8	97.9	3.5	44.1
X	x	5	14789	p+r_14789_120	96.7	97.6	3.4	56.3
XI	x	6	8874	p+ar_p1_8874_120	92.2	95.9	4.6	31.8
XII	x	8	15000	p+s_15k_120	-	-	-	57.1
XIII	x	5.8	29789	p+s_15k_120+r_14789_120	96.7	97.5	3.8	113.5
XIV	x	9	29789	p+s_15k+r_14789_120	78.3	84.1	7.0	113.5
XV	x	10	5284	holder_p+ar_13170_120	69.5	73.8	5.9	20.1
XVI	x	11	13170	holder_p+r_5284_120	83.8	87.5	4.7	50.2

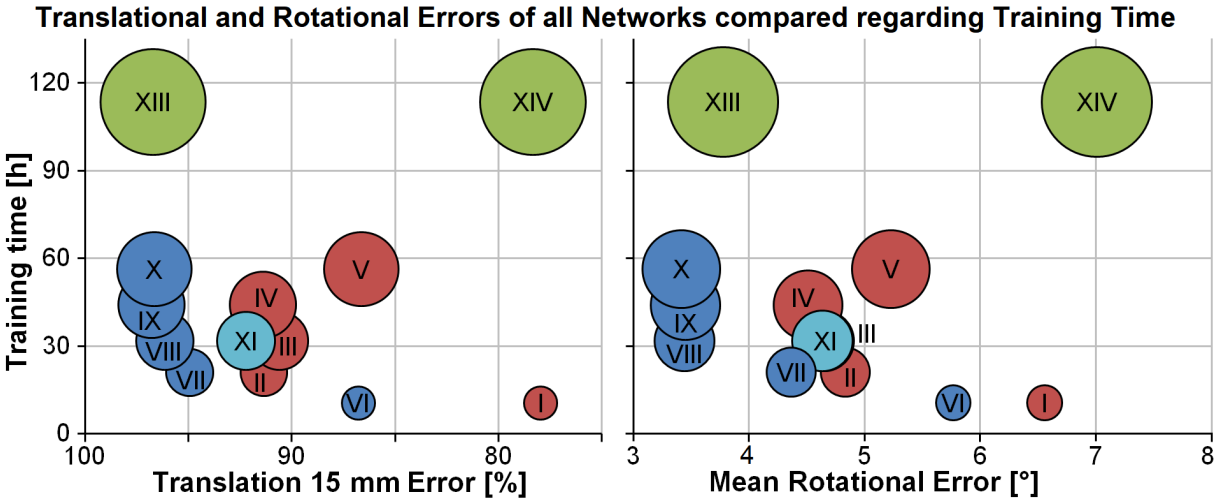


Figure 46: Translational (left) and rotational errors (right) compared regarding training time, dataset type / size and pre-training: Networks with the best performance-to-training-time ratios are located in the lower left corner. Synthetic data (green), augmented data (cyan), pre-trained networks (blue) and non-pre-trained networks (red). Bubble-size visualizes dataset-size.

of position but not orientation. This shows that a single error metric does not provide absolute validity for the overall network performance since good translational-error metrics do not always correspond with good rotational-error metrics. These charts also visualize that network XI trained on the augmented dataset shows a performance between the original dataset VI and a dataset of the same size containing uniquely recorded samples VIII.

6.2.8 Reasons for Unsuccessful Detections

To analyze factors that negatively influence detection performance, all networks trained for 120 epochs are assessed with regard to images where the object could not be detected. This shows that for some images, no network or only a few networks are able to detect the object successfully, which indicates factors that negatively influence performance.

Figure 47 (a) shows a color-shift towards the blue channel possibly caused by the camera settings. All evaluated networks fail to detect the object in this image correctly. If lighting and contrast are better, some networks are able to detect the object despite color-shift. In Figure 47 (b), a cable is hanging in front of the camera, occluding big parts of the object of interest. Only seven of the 14 evaluated networks are able to still detect the object correctly. For some similar images, where the distance between camera and object is smaller and therefore the object is not occluded this strong, more networks perform positively. Figure 47 (c, d) features extreme lighting conditions. Although similar images are included in the training dataset, all networks without pre-training fail to detect the object in these images.

These results show that unusual color-shifts, extreme lighting conditions and occlusions can negatively influence detection results, due to too low generalization. If objects are to be detected in such conditions – especially occlusions are a common problem in real-life applications –

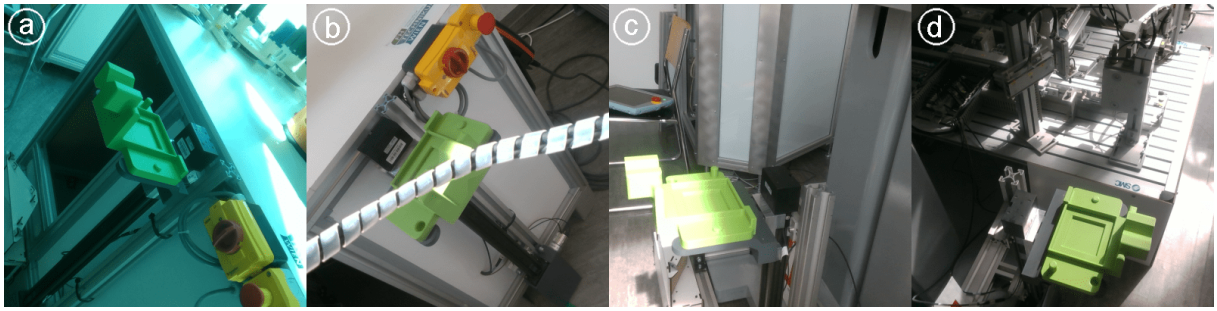


Figure 47: Images, which multiple networks are unable to detect the object in due to a color-shift (a), occlusions (b) and extreme lighting conditions (c, d).

images containing such factors have to be included in the training dataset in sufficient number or special techniques have to be chosen [66, 29, 71]. Pre-training leads to more generalization, enabling better detection results for unusual scene-conditions.

6.3 Qualitative Analysis of Pose-Estimation

In the qualitative evaluation, a real robot equipped with a camera is used to test if the accuracy and robustness of the pose-estimation system are sufficient for use in a real-life application. The software presented in Section 5.5.1 is deployed on the mobile manipulator, which has also been used for recording data.

6.3.1 Pick-and-Place Use Case

Six different goals for the mobile robot and related angles for the articulated robot to search for the object at three pick-and-place-stations are defined. The mobile manipulator is sent to a goal to search for the carrier and transport it to a different station. Then the holder is searched and the carrier put down at this position. The mobile robot is sent to the same station oriented differently and searches again for the carrier, to transport it to another station. This procedure is repeated 30 times to test the system on a total of 60 pose-estimations (30 pick- and 30 place-movements). For object detection of the carrier, the network trained on the biggest dataset with real images (p+r_14789_120) is used. For detecting the holder, the system trained on the augmented dataset (h_p+ar_13170_120) is used. Both networks have been initialized with pre-trained weights and have been trained for 120 epochs.

The carrier could be grasped successfully in 25 of 30 tries (83%). In four cases, extreme sunlight has lead to overexposure of the images, causing the object detection to fail. The other unsuccessful grasp has been caused by a wrong initial pose estimation, as visualized in Figure 49 (e) on page 76. The implemented pose refinement algorithm captures the object multiple times, originating from a given search pose. Since typically this pose-detection provides more reliable poses than the images captured for refinement (in other images objects are sometimes

cropped), other data is only used for averaging if the poses do not vary heavily. Though, if the initial pose is estimated falsely to a large degree, the object's pose is not improved or corrected using the other estimations.

Placing the carrier onto the holder has been successful 19 times (63%). The object could not be detected in five cases due to strong sunlight. Six times, the carrier has been placed onto the holder slightly rotated, which is not considered a successful place-movement (see Figure 51 (a, b) on page 51). Also quantitative results show lower rotational accuracy for the holder. This is possibly caused by the smaller dataset size and poorer contrast between the grey object and the background. The rotational error for the carrier trained on small datasets is not as large as for the holder on comparable datasets, which suggests that poor contrast has bigger impacts.

When two networks perform simultaneous detection on the used hardware, a frame-rate of 1.17 fps is possible. To increase the accuracy of the detected poses, an ICP algorithm could be used for aligning 3D models to the captured pointclouds. Though, such algorithms usually require multiple seconds for finding a solution [69, 70].

Summed up, unsuccessful pick- and place-movements are mainly caused by extreme lighting conditions. Such training samples are included in the datasets but apparently not in sufficient number. Furthermore, higher reliability and accuracy would be possible by using a more sophisticated algorithm for pose-refinement. This should include capturing the object from a higher number of stronger varying poses and rejecting poses largely deviating from the others. Nonetheless, mobile manipulators using the proposed object pose-estimation system could be deployed in real-life logistics applications. Robustness could be increased by controlling lighting conditions and refining the poses in a multi-view way, as suggested by Sock et al. [85].

6.3.2 Additional Experiments

Additional experiments have been performed for evaluating factors that are not present in the training datasets such as strongly cropped object views, differently colored objects, extreme poses and items placed on the objects of interest (occlusions). The belief maps, which indicate the positions of the cuboids (corners of the 3D bounding box around the object) and centroid are also analyzed in the following images.

Figure 48 (a-c) shows grey and blue-orange colored carriers. These objects, colored differently from the training object, could not be detected correctly. Interestingly, the neural network is able to correctly identify two cuboids of the grey carrier (c top) and furthermore the pose of the grey holder could be estimated correctly, although being strongly occluded by the grey carrier (c bottom). The right images (d, e) show the object in poses that are not included in the training dataset. As visualized by the belief maps (top), the cuboids are detected at wrong positions, why the PnP algorithm fails to retrieve the object's pose correctly.

In Figure 49 (a-c), the carrier is captured in approximately the same pose with different lighting conditions. In the left image (a), the object's pose is estimated nearly 10 mm wrong. Changed illumination (b) leads to the pose estimation being more accurate and also the pose of the

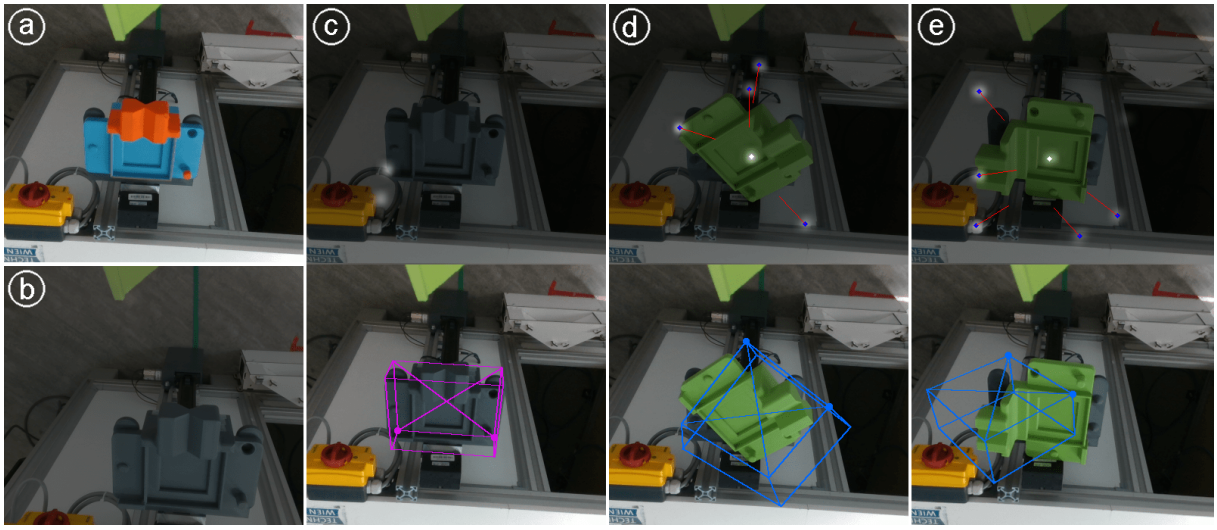


Figure 48: Detection of differently colored carriers (a-c) and extreme poses, not included in training data (d, e): Top images show belief maps and vector fields: Estimated cuboid corners (blue) and centroid (white) with confidence (white glow). Predicted vectors from cuboids to centroid (red). Bottom images visualize the estimated pose of the 3D bounding box.

holder being detected correctly. The central image (c) shows the carrier directly illuminated by the sun, which does also not influence object detection negatively. The deviating object pose in the left image (a) is caused by slightly wrong detected cuboid points.

The carrier is strongly cropped in Figure 49 (d). Nevertheless, correct pose estimation is possible because at least four cuboids and the centroid are detected. It is interesting to note that in this example, the green jaw of the gripper – which is visible at the top of all images – is also

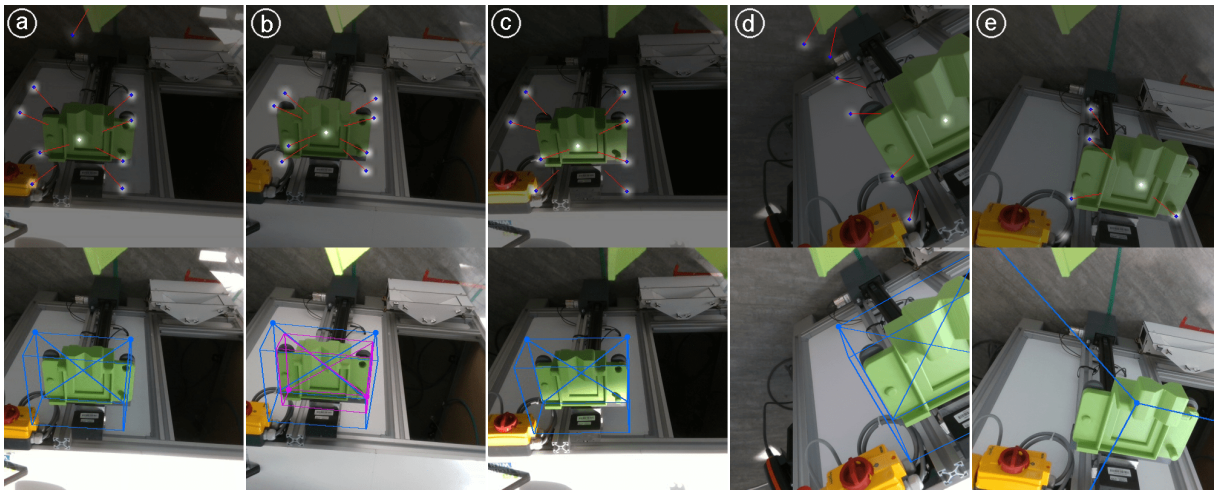


Figure 49: Carrier captured from the same perspective with varying lighting conditions, leading to different accuracies (a-c). Correct pose estimation despite heavily cropped image (d) and failure of PnP algorithm (e). Top images show belief maps and vector fields: Estimated cuboid corners (blue) and centroid (white) with confidence (white glow). Predicted vectors from cuboids to centroid (red). Bottom images visualize the estimated pose of the 3D bounding box.

considered being a carrier. The right image (e) shows a failure of the PnP algorithm possibly caused by cuboid positions being predicted inaccurately. It would be possible to increase the minimum number of corners that have to be detected correctly, to prevent such wrong estimations. On the other hand, this would make the detection of strongly cropped objects impossible.

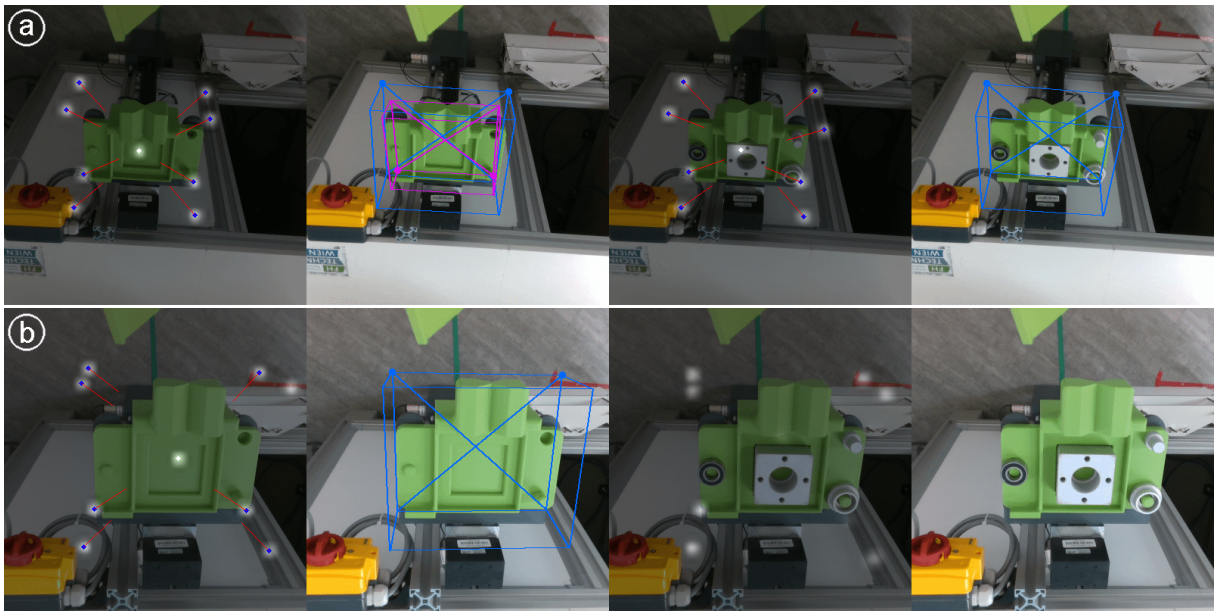


Figure 50: Detection of the carrier with and without items placed on it. Successful (a) and failed attempt (b). Belief maps and vector fields: Estimated cuboids (blue) and centroid (white) with confidence (white glow). Predicted vectors from cuboids to centroid (red).

Further experiments investigate if detection of the carrier with items placed on it is also possible, although only images showing the empty carrier have been included in the training dataset. For this purpose, pictures are captured from the same position, showing the object with and without items placed on it. In the top images of Figure 50 (a), the carrier is detected with and without being occluded. If no items are placed on it, also the pose of the holder, occluded by the carrier, could be detected successfully. Showing the object in a slightly different perspective in the bottom images (b), the system is not able to detect the carrier with items placed on. This might be caused by the centroid being detected with very low probability, indicated by the light white dot in the center of the belief map.

Similar experiments have also been performed for the holder in Figure 51. Images (a, b) show the higher rotational errors for this object, which have also been discussed in Section 6.2.6. Wrongly detected cuboid positions mainly cause these inaccuracies. Image (c) illustrates that the rotational errors are not systematically and therefore, only occur in some poses and lighting conditions. The right image (d) shows that also the pose of the holder can be estimated correctly, although big parts are cropped.

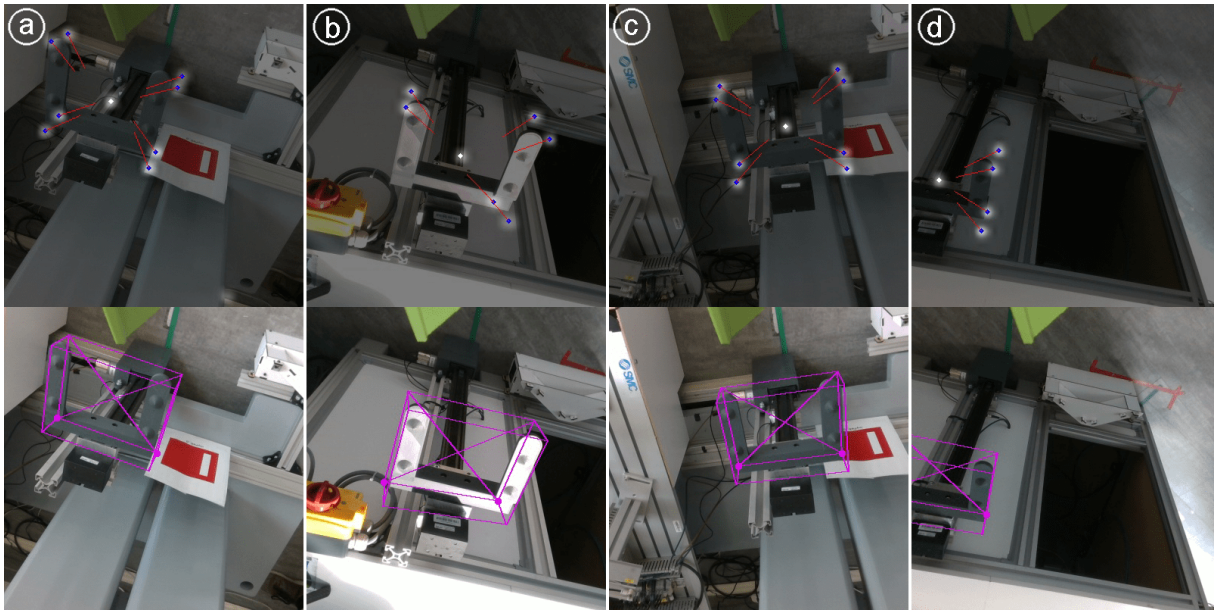


Figure 51: Qualitative analysis of the holder: Rotational errors are present for some poses and lighting conditions (a, b), although not being systematically (c). Also the pose of the holder is detectable despite cropping (d). Top images show the visualized belief maps and vector fields: Estimated cuboids (blue) and centroid (white) with confidence (white glow). Predicted vectors from cuboids to centroid (red).

6.4 Conclusion

The evaluation of the accuracy of the ArUco marker-detection shows that maximum errors of approximately 1 mm for the three directions and up to 2° for the rotations are possible. These results do include errors from marker detection along with imprecisions of the robot. To gather higher accuracy, multiple images from different perspectives are captured for computing the mean object pose, which reduces the error of the ground-truth object pose. Qualitative results indicate that the accuracy of the ground-truth is sufficient, though it could be increased using marker-boards or ICP algorithms.

As the quantitative results show, no general answer regarding optimal dataset and training parameters can be given. The best performance to training-time ratio has been possible in these experiments using a dataset including about 11,000 images from the target domain. Some general guidelines can be summarized as followed:

- Using a pre-trained model leads to better performance than initializing networks with random weights
- Bigger datasets do not necessarily improve the output of neural networks since biased datasets lead to wrong generalizations
- Increasing the number of training epochs leads to higher training time, but accuracy does not always become better due to overfitting

- If networks are to be trained on synthetic data only, large training datasets including around 100,000 samples are necessary [29]
- A relatively small dataset recorded in the target domain can achieve better results than a several times larger synthetic dataset
- Data augmentation can increase network performance, especially if only a small set of training examples is available
- The pose of objects with poor contrast is more difficult to detect accurately

The qualitative results show that mobile manipulators using the proposed pose-estimation system could be deployed in real-life logistics applications. Unsuccessful pick- and place-movements could be prevented by including more images of extreme lighting conditions into the training datasets or using a more sophisticated multi-view pose-refinement algorithm. Experiments also show that for detecting differently colored objects or objects in extreme poses, such training samples would be necessary and that occlusions play a minor role. The carrier could sometimes be detected despite items being placed on it and the holder has sometimes been detected when the carrier has been placed on it – although such samples have not been included in the training datasets. The PnP algorithm, which calculates poses from given feature points, sometimes fails if only a few cuboids are detected. Though, increasing the minimum number of detected cuboids necessary to allow pose estimation would lead to not being able to detect cropped objects.

7 Summary and Future Work

In this work, an approach for semi-automatic generation of 6D-pose-annotated training data is presented, using an articulated robot equipped with an RGB-D camera. The images of the camera are used to determine the pose of the object of interest with respect to the robot's base, using a marker placed at a defined position on the object. After removing the marker, the robot moves hemispherically around the object and captures images of it at random poses. Using the transformation between the robot's base and the camera, the 6D pose of the object can be computed for each image.

The presented approach can only be used for generating annotated datasets of objects being big enough and shaped such that a marker can be placed at a defined position on it. Using this semi-automatic data generation procedure, cumbersome post-processing for removing markers in images or computationally expensive fitting of CAD models are avoided. In contrast to other 6D pose annotation tools, only very little human labor is involved, no CAD models of the objects are necessary and diverse datasets are created because single images are captured in random poses.

Experiments show maximum deviations of a single marker-pose estimation of about 1 mm for the three directions and up to approximately 2° for the rotations, considering imprecisions of the robot and the ArUco marker detection. Multiple images from different perspectives are captured and the computed poses are merged, to gather ground-truth poses with sufficient accuracy, which could be further increased in future work using marker-boards or ICP algorithms.

It would be possible to decrease the time needed to capture annotated data by adopting the software to capture images while the robotic arm is in motion. This is not possible using the implemented robot-driver and would include ensuring synchronization of pose-information and images.

Multiple annotated datasets of two different objects have been created using the proposed method, complemented with augmented data and synthetic (domain randomized) images. Furthermore, differently sized subsets of the datasets have been generated for training. The deep-learning-based 6D pose estimation method DOPE by Tremblay et al. [29] has been chosen for the evaluation of these datasets.

Quantitative evaluation on an evenly distributed test dataset shows that no general answer regarding optimal dataset and training parameters can be given. Results verify that using pre-trained deep models leads to better performance than initializing them with random weights. Bigger datasets and a higher number of epochs do not necessarily improve the output of the neural network due to wrong generalizations and overfitting. A small dataset recorded in the

target domain usually achieves better results than several times larger synthetic datasets, especially when extended using data augmentation. The results for the carrier could be confirmed using the holder, although comparison shows that lower accuracy of the estimated poses is observable due to poorer contrast between the object and background. In future work, it would be interesting to investigate if similar results can be achieved for different deep network structures, trained on the same datasets.

The qualitative analysis shows that mobile manipulators using the proposed pose-estimation system could be deployed in real-life logistics applications. In future work, a more sophisticated multi-view pose-refinement algorithm could be used to increase reliability and enable deployment in real shop floors.

Bibliography

[1] J. Wallén, "The history of the industrial robot," *Technical report from Automatic Control at Linköpings universitet*, 2008.

[2] M. A. Roa, D. Berenson, and W. Huang, "Mobile manipulation: Toward smart manufacturing," *IEEE Robotics Automation Magazine*, vol. 22, no. 4, pp. 14–15, 2015.

[3] J. Sturm, *Approaches to Probabilistic Model Learning for Mobile Manipulation Robots*, ser. Springer Tracts in Advanced Robotics (STAR). Springer Verlag, 2013.

[4] H. Hirsch-Kreinsen, "Digitization of industrial work: development paths and prospects," *Journal for Labour Market Research*, vol. 49, no. 1, pp. 1–14, 2016.

[5] D. Holz and S. Behnke, "Fast edge-based detection and localization of transport boxes and pallets in RGB-D images for mobile robot bin picking," in *47st International Symposium on Robotics – ISR*, 2016, pp. 1–8.

[6] D. Wurhofer, T. Meneweger, V. Fuchsberger, and M. Tscheligi, "Reflections on operators' and maintenance engineers' experiences of smart factories," in *ACM Conference on Supporting Groupwork*, 2018, pp. 284–296.

[7] D. Pavlichenko, G. M. García, S. Koo, and S. Behnke, "Kittingbot: A mobile manipulation robot for collaborative kitting in automotive logistics," in *15th International Conference on Intelligent Autonomous Systems – IAS*, 2018, pp. 849–864.

[8] Z.-E. Chebab, J.-C. Fauroux, N. Bouton, Y. Mezouar, and L. Sabourin, "Autonomous collaborative mobile manipulators: State of the art," in *TrC-IFToMM Symposium on Theory of Machines and Mechanisms*, 2015.

[9] U. Asif, M. Bennamoun, and F. Sohel, "Real-time pose estimation of rigid objects using RGB-D imagery," in *IEEE 8th Conference on Industrial Electronics and Applications – ICIEA*, 2013, pp. 1692–1699.

[10] Mobile Industrial Robots, *MiR100 User Guide*, Mobile Industrial Robots, 2019.

[11] H. Xu, G. Chen, Z. Wang, L. Sun, and F. Su, "RGB-D-based pose estimation of workpieces with semantic segmentation and point cloud registration," *Sensors*, vol. 19, no. 8, 2019.

[12] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 57, 2018.

[13] G. Du, K. Wang, and S. Lian, “Vision-based robotic grasping from object localization, pose estimation, grasp detection to motion planning: A review,” *CoRR*, 2019.

[14] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[15] 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,” in *IEEE Winter Conference on Applications of Computer Vision – WACV*, 2017, pp. 880–888.

[16] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, K. Konolige, G. Bradski, and N. Navab, “Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes,” in *11th Asian Conference on Computer Vision – ACCV*, 2012, pp. 593–596.

[17] J. Tremblay, T. To, and S. Birchfield, “Falling things: A synthetic dataset for 3D object detection and pose estimation,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops – CVPR Workshops*, 2018, pp. 2038–2041.

[18] Cornell University, “Cornell grasping dataset,” http://pr.cs.cornell.edu/grasping/rect_data/data.php, 2009, accessed: 2019-08-26.

[19] R. Kaskman, S. Zakharov, I. Shugurov, and S. Ilic, “HomebrewedDB: RGB-D dataset for 6D pose estimation of 3D objects,” *CoRR*, 2019.

[20] P. Marion, P. R. Florence, L. Manuelli, and R. Tedrake, “Label Fusion: A pipeline for generating ground truth labels for real RGBD data of cluttered scenes,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2017, pp. 1–8.

[21] T. Sølund, A. G. Buch, N. Krüger, and H. Aanæs, “A large-scale 3D object recognition dataset,” in *4th International Conference on 3D Vision – 3DV*, 2016, pp. 73–82.

[22] P. Wohlhart and V. Lepetit, “Learning descriptors for object recognition and 3D pose estimation,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2015, pp. 3109–3118.

[23] M. Garon, D. Laurendeau, and J. F. Lalonde, “A framework for evaluating 6-DOF object trackers,” in *15th European Conference on Computer Vision – ECCV*, 2018, pp. 608–623.

[24] A. Tejani, D. Tang, R. Kouskouridas, and T. K. Kim, “Latent-class hough forests for 3D object detection and pose estimation,” in *13th European Conference on Computer Vision – ECCV*, 2014, pp. 462–477.

[25] J. M. Wong, V. Kee, T. Le, S. Wagner, G. L. Mariottini, A. Schneider, L. Hamilton, R. Chipalkatty, M. Hebert, D. M. Johnson, J. Wu, B. Zhou, and A. Torralba, “SegICP: Integrated deep semantic segmentation and pose estimation,” in *IEEE International Conference on Intelligent Robots and Systems – IROS*, 2017, pp. 5784–5789.

[26] P. C. Wu, Y. Y. Lee, H. Y. Tseng, H. I. Ho, M. H. Yang, and S. Y. Chien, “A benchmark dataset for 6DoF object pose tracking,” in *IEEE International Symposium on Mixed and Augmented Reality – ISMAR*, 2017, pp. 186–191.

[27] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” *International journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[28] L. Pinto and A. Gupta, “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2016, pp. 3406–3413.

[29] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, “Deep object pose estimation for semantic robotic grasping of household objects,” in *2nd Annual Conference on Robot Learning – CoRL*, 2018, pp. 306–316.

[30] S. Kumra and C. Kanan, “Robotic grasp detection using deep convolutional neural networks,” in *IEEE International Conference on Intelligent Robots and Systems – IROS*, 2017, pp. 769–776.

[31] J. Tobin, L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, J. Schneider, P. Welinder, W. Zaremba, and P. Abbeel, “Domain randomization and generative models for robotic grasping,” *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS*, pp. 3482–3489, 2017.

[32] U. Asif, M. Bennamoun, and F. A. Sohel, “RGB-D object recognition and grasp detection using hierarchical cascaded forests,” *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 547–564, 2017.

[33] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, “Dex-Net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics,” in *Robotics: Science and Systems XIII*, 2017.

[34] I. Lenz, H. Lee, and A. Saxena, “Deep learning for detecting robotic grasps,” *The International journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[35] J. Bohg, A. Morales, T. Asfour, and D. Kragic, “Data-driven grasp synthesis: A survey,” *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2014.

[36] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, “Using simulation and domain adaptation to improve efficiency of deep robotic grasping,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2017, pp. 4243–4250.

[37] U. Viereck, A. ten Pas, K. Saenko, and R. Platt, “Learning a visuomotor controller for real world robotic grasping using simulated depth images,” in *1st Annual Conference on Robot Learning – CoRL*, 2017, pp. 291–300.

[38] J. Redmon and A. Angelova, “Real-time grasp detection using convolutional neural networks,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2015, pp. 1316–1322.

[39] A. V. Patil and P. Rabha, “A survey on joint object detection and pose estimation using monocular vision,” *CoRR*, 2018.

[40] T. Hodaň, J. Matas, and Š. Obdržálek, “On evaluation of 6D object pose estimation,” in *Computer Vision - ECCV Workshops*, 2016, pp. 609–619.

[41] S. Peng, Y. Liu, Q. Huang, H. Bao, and X. Zhou, “PVNet: Pixel-wise voting network for 6DoF pose estimation,” *CoRR*, 2018.

[42] M. Sundermeyer, Z. C. Marton, M. Durner, M. Brucker, and R. Triebel, “Implicit 3D orientation learning for 6D object detection from RGB images,” in *15th European Conference on Computer Vision – ECCV*, 2018, pp. 712–729.

[43] J. Wu, B. Zhou, R. Russell, V. Kee, S. Wagner, M. Hebert, A. Torralba, and D. M. S. Johnson, “Real-time object pose estimation with pose interpreter networks,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems –IROS*, 2018, pp. 6798–6805.

[44] T.-T. Do, T. Pham, M. Cai, and I. Reid, “LieNet: Real-time monocular object instance 6D pose estimation,” in *British Machine Vision Conference – BMVC*, 2018.

[45] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, “SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again,” in *IEEE International Conference on Computer Vision – ICCV*, 2017, pp. 1530–1538.

[46] D. Oñoro-Rubio, R. J. López-Sastre, C. Redondo-Cabrera, and P. Gil-Jiménez, “The challenge of simultaneous object detection and pose estimation: A comparative study,” *Image and Vision Computing*, vol. 79, pp. 109–122, 2018.

[47] P. J. Besl and N. D. McKay, “A method for registration of 3-d shapes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[48] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm,” in *3rd International Conference on 3-D Digital Imaging and Modeling – 3DIM*, 2001, pp. 145–152.

[49] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, “Learning 6D object pose estimation using 3D object coordinates,” in *13th European Conference on Computer Vision – ECCV*, 2014, pp. 536–551.

[50] F. Liu, P. Fang, Z. Yao, R. Fan, Z. Pan, W. Sheng, and H. Yang, “Recovering 6D object pose from RGB indoor image based on two-stage detection network with multi-task loss,” *Neurocomputing*, vol. 337, pp. 15–23, 2019.

[51] Z. Cao, Y. Sheikh, and N. K. Banerjee, “Real-time scalable 6DOF pose estimation for textureless objects,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2016, pp. 2441–2448.

[52] T.-T. Do, M. Cai, T. Pham, and I. Reid, “Deep-6DPose: Recovering 6D object pose from a single RGB image,” *CoRR*, 2018.

[53] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2005, pp. 886–893.

[54] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab, “Dominant orientation templates for real-time detection of texture-less objects,” in *23rd IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2010, pp. 2257–2264.

[55] R. Rios-Cabrera and T. Tuytelaars, “Discriminatively trained templates for 3D object detection: A real time scalable approach,” in *IEEE International Conference on Computer Vision – ICCV*, 2013, pp. 2048–2055.

[56] D. M. Gavrila, “A bayesian, exemplar-based approach to hierarchical shape matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 2, pp. 1408–1421, 2007.

[57] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, “Multimodal templates for real-time detection of textureless objects in heavily cluttered scenes,” in *IEEE International Conference on Computer Vision – ICCV*, 2011, pp. 858–865.

[58] V. Lepetit, F. Moreno-Noguer, and P. Fua, “EPnP: An accurate o(n) solution to the PnP problem,” *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, 2009.

[59] B. Drost, M. Ulrich, N. Navab, and S. Ilic, “Model globally, match locally: Efficient and robust 3D object recognition,” in *23th IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2010, pp. 998–1005.

[60] J. Vidal, C. Y. Lin, and R. Marti, “6D pose estimation using an improved method based on point pair features,” in *4th International Conference on Control, Automation and Robotics – ICCAR*, 2018, pp. 405–409.

[61] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, “DenseFusion: 6D object pose estimation by iterative dense fusion,” *CoRR*, 2019.

[62] F.-e. Ababsa and M. Mallem, “Robust camera pose estimation using 2d fiducials tracking for real-time augmented reality systems,” in *International conference on Virtual Reality Continuum and its applications in industry – VRCAI*, 2004, pp. 431–435.

[63] R. Muñoz-Salinas, M. J. Marín-Jimenez, E. Yeguas-Bolivar, and R. Medina-Carnicer, “Mapping and localization from planar markers,” *Pattern Recognition*, vol. 73, pp. 158–171, 2018.

[64] R. Muñoz-Salinas, “ArUco : An efficient library for detection of planar markers and camera pose estimation,” https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit, 2019, accessed: 2019-08-26.

[65] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[66] D. Hu, D. Detone, V. Chauhan, I. Spivak, and T. Malisiewicz, “Deep ChArUco: Dark ChArUco marker pose estimation,” *CoRR*, 2018.

[67] M. Fu and W. Zhou, “DeepHMap++: Combined projection grouping and correspondence learning for full DoF pose estimation,” *Sensors*, vol. 19, no. 5, 2019.

[68] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[69] J. Liu and S. He, “6D object pose estimation without PnP,” *CoRR*, 2019.

[70] G. Billings and M. Johnson-Roberson, “SilhoNet: An RGB method for 3D object pose estimation and grasp planning,” *CoRR*, 2018.

[71] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes,” in *Robotics: Science and Systems XIV*, 2018.

[72] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, “Mask R-CNN,” in *IEEE International Conference on Computer Vision – ICCV*, 2017, pp. 2980–2988.

[73] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2016, pp. 779–788.

[74] M. Rad, M. Oberweger, and V. Lepetit, “Domain transfer for 3D pose estimation from color images without manual annotations,” in *14th Asian Conference on Computer Vision – ACCV*, 2018.

[75] B. Tekin, S. N. Sinha, and P. Fua, “Real-time seamless single shot 6D object pose prediction,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2018, pp. 292–301.

[76] 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,” in *IEEE International Conference on Computer Vision – ICCV*, 2017, pp. 3848–3856.

[77] Z. Zhao, G. Peng, H. Wang, H.-S. Fang, C. Li, and C. Lu, “Estimating 6D pose from localizing designated surface keypoints,” *CoRR*, 2018.

[78] Y. Hu, J. Hugonot, P. Fua, and M. Salzmann, “Segmentation-driven 6D object pose estimation,” *CoRR*, 2018.

[79] S. Zakharov, I. Shugurov, and S. Ilic, “DPOD: 6D pose object detector and refiner,” *CoRR*, 2019.

[80] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, “DeepIM: Deep iterative matching for 6D pose estimation,” in *15th European Conference on Computer Vision – ECCV*, 2018, pp. 695–711.

[81] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, “Deep learning of local RGB-D patches for 3D object detection and 6D pose estimation,” in *14th European Conference on Computer Vision – ECCV*, 2016, pp. 205–220.

[82] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, “6-DoF object pose from semantic keypoints,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2017, pp. 2011–2018.

[83] C. Mitash, A. Boularias, and K. Bekris, “Robust 6D object pose estimation with stochastic congruent sets,” in *British Machine Vision Conference – BMVC*, 2018.

[84] A. Zeng, K. T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, “Multi-view self-supervised deep learning for 6D pose estimation in the amazon picking challenge,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2017, pp. 1386–1393.

[85] J. Sock, S. Hamidreza Kasaei, L. S. Lopes, and T. K. Kim, “Multi-view 6D object pose estimation and camera motion planning using RGBD images,” in *IEEE International Conference on Computer Vision Workshops – ICCVW*, 2018, pp. 2228–2235.

[86] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, “Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2016, pp. 3364–3372.

[87] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations – ICLR*, 2014.

[88] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2009, pp. 248–255.

[89] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, “Scene coordinate regression forests for camera relocalization in RGB-D images,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2013, pp. 2930–2937.

[90] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[91] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, “Deep learning for generic object detection: A survey,” *CoRR*, 2018.

[92] F. Manhardt, D. M. Arroyo, C. Rupprecht, B. Busam, N. Navab, and F. Tombari, “Explaining the ambiguity of object detection and 6D pose from visual data,” *CoRR*, 2018.

[93] C. M. Bishop, *Pattern recognition and machine learning*. Berlin, Heidelberg: Springer-Verlag, 2006.

[94] Q. Abbas, M. E. Ibrahim, and M. A. Jaffar, “A comprehensive review of recent advances on deep vision systems,” *Artificial Intelligence Review*, vol. 52, no. 1, pp. 39–76, 2019.

[95] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *IEEE International Conference on Intelligent Robots and Systems – IROS*, 2017, pp. 23–30.

[96] S. James, A. J. Davison, and E. Johns, “Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task,” in *1st Annual Conference on Robot Learning – CoRL*, 2017.

[97] E. Santana, K. Dockendorf, and J. C. Principe, “Learning joint features for color and depth images with convolutional neural networks for object classification,” in *IEEE International Conference on Acoustics, Speech and Signal Processing – ICASSP*, 2015, pp. 1320–1323.

[98] S. Krig, *Computer Vision Metrics - Survey, Tyonomy, and Analysis*, 1st ed. Berkely, CA, USA: Apress, 2014.

[99] S. Giancola, M. Valenti, and R. Sala, *A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies*, ser. SpringerBriefs in Computer Science. Springer International Publishing, 2018.

[100] A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-Escolano, J. García-Rodríguez, and A. Jover-Alvarez, "The RobotriX: An extremely photorealistic and very-large-scale indoor dataset of sequences with robot trajectories and interactions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS*, 2018, pp. 6790–6797.

[101] P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez, "UnrealROX: An extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation," *CoRR*, 2018.

[102] J.-P. Mercier, C. Mitash, P. Giguère, and A. Boularias, "Learning object localization and 6D pose estimation from simulation and weakly labeled real images," in *International Conference on Robotics and Automation – ICRA*, 2019.

[103] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison, "SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation?" in *IEEE International Conference on Computer Visio – ICCV*, 2017, pp. 2697–2706.

[104] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" in *IEEE International Conference on Robotics and Automation – ICRA*, 2017, pp. 746–753.

[105] W. Qiu and A. Yuille, "UnrealCV: Connecting computer vision to unreal engine," in *Computer Vision – ECCV Workshops*, 2016, pp. 909–916.

[106] A. Tsirikoglou, J. Kronander, M. Wrenninge, and J. Unger, "Procedural modeling and physically based rendering for synthetic data generation in automotive applications," *CoRR*, 2017.

[107] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *14th European Conference on Computer Vision – ECCV*, 2016, pp. 102–118.

[108] A. Dundar, M.-Y. Liu, T.-C. Wang, J. Zedlewski, and J. Kautz, "Domain stylization: A strong, simple baseline for synthetic to real image domain adaptation," *CoRR*, 2018.

[109] T. Hodaň, V. Vineet, R. Gal, E. Shalev, J. Hanzelka, T. Connell, P. Urbina, S. N. Sinha, and B. Guenter, "Photorealistic image synthesis for object instance detection," *CoRR*, 2019.

[110] R. Khirodkar, D. Yoo, and K. M. Kitani, "VADRA: Visual adversarial domain randomization and augmentation," *CoRR*, 2018.

[111] Y. Zhang, S. Song, E. Yumer, M. Savva, J. Y. Lee, H. Jin, and T. Funkhouser, "Physically-based rendering for indoor scene understanding using convolutional neural networks," in

30th IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR, 2017, pp. 5057–5065.

[112] Y. Movshovitz-Attias, T. Kanade, and Y. Sheikh, "How useful is photo-realistic rendering for visual learning?" in *Computer Vision – ECCV Workshops*, 2016, pp. 202–217.

[113] A. Carlson, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, "Sensor transfer: Learning optimal sensor effect image augmentation for sim-to-real domain adaptation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2431–2438, 2019.

[114] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation policies from data," *CoRR*, 2018.

[115] A. Dosovitskiy, J. T. Springenberg, and T. Brox, "Unsupervised feature learning by augmenting single images," in *2nd International Conference on Learning Representations – ICLR*, 2014.

[116] A. Carlson, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, "Modeling camera effects to improve deep vision for real and synthetic data," *CoRR*, 2018.

[117] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *8th International Conference on Quality of Multimedia Experience – QoMEX*, 2016, pp. 1–6.

[118] A. D'Innocente, F. M. Carlucci, M. Colosi, and B. Caputo, "Bridging between computer and robot vision through data augmentation: A case study on object recognition," in *11th International Conference on Computer Vision Systems – ICVS*, 2017, pp. 384–393.

[119] D. Dwibedi, I. Misra, and M. Hebert, "Cut, paste and learn: Surprisingly easy synthesis for instance detection," in *IEEE International Conference on Computer Vision – ICCV*, 2017, pp. 1310–1319.

[120] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas, "Normalized object coordinate space for category-level 6D object pose and size estimation," *CoRR*, 2019.

[121] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield, "Structured domain randomization: Bridging the reality gap by context-aware synthetic data," in *International Conference on Robotics and Automation – ICRA*, 2019.

[122] A. Buslaev, A. Parinov, E. Khvedchenya, V. I. Iglovikov, and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," *CoRR*, 2018.

[123] J. Rambach, C. Deng, A. Pagani, and D. Stricker, "Learning 6DoF object poses from synthetic single channel images," in *IEEE International Symposium on Mixed and Augmented Reality – ISMAR*, 2018.

[124] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla, “SceneNet: Understanding real world indoor scenes with synthetic data,” *CoRR*, 2015.

[125] B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, O. Lehmann, T. Chen, A. Hutter, S. Zakharov, H. Kosch, and J. Ernst, “DepthSynth: Real-time realistic synthetic data generation from CAD models for 2.5D recognition,” in *International Conference on 3D Vision – 3DV*, 2018, pp. 1–10.

[126] C. Sweeney, G. Izatt, and R. Tedrake, “A supervised approach to predicting noise in depth images,” in *International Conference on Robotics and Automation – ICRA*, 2019, pp. 796–802.

[127] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *CoRR*, 2014.

[128] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, “Learning from simulated and unsupervised images through adversarial training,” in *30th IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2017, pp. 2242–2251.

[129] L. Sixt, B. Wild, and T. Landgraf, “RenderGAN: Generating realistic labeled data,” in *5th International Conference on Learning Representations – ICLR*, 2017.

[130] J. Lemley, S. Bazrafkan, and P. Corcoran, “Smart augmentation learning an optimal data augmentation strategy,” *IEEE Access*, vol. 5, pp. 5858–5869, 2017.

[131] B. Planche, S. Zakharov, Z. Wu, A. Hutter, H. Kosch, and S. Ilic, “Seeing beyond appearance - mapping real images into geometrical domains for unsupervised CAD-based recognition,” *CoRR*, 2018.

[132] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition – CVPR*, 2014, pp. 580–587.

[133] N. Dvornik, K. Shmelkov, J. Mairal, and C. Schmid, “BlitzNet: A real-time deep network for scene understanding,” in *IEEE International Conference on Computer Vision – ICCV*, 2017, pp. 4174–4182.

[134] C. Mitash, K. Bekris, and A. Boularias, “A self-supervised learning system for object detection using physics simulation and multi-view pose estimation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS*, 2017, pp. 545–551.

[135] Y. Jiang, S. Moseson, and A. Saxena, “Efficient grasping from RGBD images: Learning using a new rectangle representation,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2011, pp. 3304–3311.

[136] M. Firman, “RGBD datasets: Past, present and future,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops – CVPR*, 2016, pp. 661–673.

[137] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, “LabelMe: A database and web-based tool for image annotation,” *International journal of Computer Vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[138] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, “LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop,” *CoRR*, 2015.

[139] M. Suchi, T. Patten, D. Fischinger, and M. Vincze, “EasyLabel: A semi-automatic pixel-wise object annotation tool for creating robotic RGB-D datasets,” in *International Conference on Robotics and Automation – ICRA*, 2019, pp. 6678–6684.

[140] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, “ScanNet: Richly-annotated 3D reconstructions of indoor scenes,” in *30th IEEE Computer Society Conference on Computer Vision and Pattern Recognition – CVPR*, 2017, pp. 2432–2443.

[141] B. S. Hua, Q. H. Pham, D. T. Nguyen, M. K. Tran, L. F. Yu, and S. K. Yeung, “SceneNN: A scene meshes dataset with aNNotations,” in *4th International Conference on 3D Vision – 3DV*, 2016, pp. 92–101.

[142] Y. Wang, X. Tan, Y. Yang, Z. Li, X. Liu, F. Zhou, and L. S. Davis, “Improving annotation for 3D pose dataset of fine-grained object categories,” *CoRR*, 2018.

[143] A. Aldoma, T. Faulhammer, and M. Vincze, “Automation of ’ground truth’ annotation for multi-view RGB-D object instance recognition datasets,” in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 5016–5023.

[144] T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, W. Hodge, and S. Birchfield, “NDDS: NVIDIA deep learning dataset synthesizer,” 2018.

[145] ECMA, *ECMA-404: The JSON Data Interchange Format*, 2nd ed. Geneva, Switzerland: ECMA (European Association for Standardizing Information and Communication Systems), 2017.

[146] T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. G. 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,” in *15th European Conference on Computer Vision – ECCV*, 2018, pp. 19–35.

[147] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source robot operating system,” in *International Conference on Robotics and Automation, Workshop on Open Source Software – ICRA*, 2009.

[148] J. N. Rauer, W. Wöber, and M. Aburaia, “An autonomous mobile handling robot using object recognition,” in *Proceedings of the Joint ARW & OAGM Workshop*, 2019, pp. 38–43.

[149] W. Meeussen, “Coordinate frames for mobile platforms,” http://www.ros.org/reps/rep-0105.html, 2010, accessed: 2019-08-26.

[150] Z. Zhang, “Flexible camera calibration by viewing a plane from unknown orientations,” in *7th IEEE International Conference on Computer Vision*, 1999, pp. 666–673.

[151] R. Hartley and A. Zisserman, *Multiple View Geometry in computer vision*, 2nd ed. New York, NY, USA: Cambridge University Press, 2004.

[152] Intel Corporation, “Intel ® RealSense ™ D400 series calibration tools,” https://www.intel.com/content/www/us/en/support/articles/000026723/emerging-technologies/intel-realsense-technology.html, 2018, accessed: 2019-08-26.

[153] R. Y. Tsai and R. K. Lenz, “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration,” *IEEE Transactions on Robotics and Automation*, vol. 5, no. 3, pp. 345–358, 1989.

[154] Y. C. Shiu and S. Ahmad, “Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form ax = xb,” *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 16–29, 1989.

[155] X. Zhi and S. Schwertfeger, “Simultaneous hand-eye calibration and reconstruction,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS*, 2017, pp. 1470–1477.

[156] W. Li, M. Dong, N. Lu, X. Lou, and P. Sun, “Simultaneous robot–world and hand–eye calibration without a calibration object,” *Sensors*, vol. 18, no. 11, 2018.

[157] T. T. Andersen, *Optimizing the Universal Robots ROS driver*. Technical University of Denmark, Department of Electrical Engineering, 2015.

[158] T. Foote, “tf: The transform library,” in *2013 IEEE International Conference on Technologies for Practical Robot Applications – TePRA*, 2013, pp. 1–6.

[159] P. Bourke, “Calculating the area and centroid of a polygon,” http://paulbourke.net/geometry/polygonmesh/, 1997, accessed: 2019-08-26.

[160] International Organization for Standardization, *ISO/IEC 15948:2004 – Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification*. Geneva: International Organization for Standardization, 2004.

[161] P. Azad, T. Gockel, and R. Dillmann, *Computer Vision: Das Praxisbuch*, 2nd ed. Elektor-Verlag, Aachen, 2009.

[162] Blender Online Community, “Blender - a 3D modelling and rendering package,” http://www.blender.org, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2019, accessed: 2019-08-26.

[163] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *13th European Conference on Computer Vision – ECCV*, 2014, pp. 740–755.

[164] A. Quattoni and A. Torralba, “Recognizing indoor scenes,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops – CVPR Workshops*, 2009, pp. 413–420.

[165] B. D. Ripley and N. L. Hjort, *Pattern Recognition and Neural Networks*, 1st ed. New York, NY, USA: Cambridge University Press, 1995.

[166] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: A modern approach*, 3rd ed. Prentice Hall, 2010.

[167] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[168] M. Schwarz, H. Schulz, and S. Behnke, “RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features,” in *IEEE International Conference on Robotics and Automation – ICRA*, 2015, pp. 1329–1335.

[169] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, “Training deep networks with synthetic data: Bridging the reality gap by domain randomization,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops – CVPR Workshops*, 2018, pp. 969–977.

[170] Universal Robots A/S, “UR5 technical specifications item no. 110105,” https://www.universal-robots.com/media/50588/ur5_en.pdf, Universal Robots A/S, Universal Robots A/S, Denmark, 2016, accessed: 2019-08-26.

[171] R. S. Xavier, B. M. F. da Silva, and L. M. G. Goncalves, “Accuracy analysis of augmented reality markers for visual mapping and localization,” in *Workshop of Computer Vision – WVC*, 2017, pp. 73–77.

[172] V. Agnus, S. Nicolau, and L. Soler, “Illumination independent and accurate marker tracking using cross-ratio invariance,” *IEEE Computer Graphics and Applications*, vol. 35, no. 5, pp. 22–33, 2015.

[173] J.-H. Yoon, J.-S. Park, and C. Kim, “Increasing camera pose estimation accuracy using multiple markers,” in *16th International Conference on Artificial Reality and Teleexistence - ICAT*, 2006, pp. 239–248.

[174] Intel Corporation, “Intel ® RealSense ™ D400 depth camera series fact sheet,” https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/realsense-d415-d435-fact-sheet.pdf, Intel Corporation, Intel Corporation, California, 2018, accessed: 2019-08-26.

[175] J. Liu, Y. Chen, K. Liu, and J. Zhao, “Attention-based event relevance model for stock price movement prediction,” in *Second China Conference on Knowledge Graph and Semantic Computing. Language, Knowledge, and Intelligence – CCKS*, 2017, pp. 37–49.

List of Figures

List of Tables

List of Code

List of Abbreviations

CAD	Computer-Aided Design
CNN	Convolutional Neural Network
DA	Data Augmentation
DoF	Degree of freedom
DR	Domain Randomization
GAN	Generative Adversarial Nets
GT	Ground-Truth
ICP	Iterative Closest Point
JSON	JavaScript Object Notation
NDDS	NVIDIA Dataset Synthesizer
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
ROS	Robot Operating System
TCP	Tool Center Point
w.r.t.	with respect to

A Links to code of approaches for pose estimation

In Table 6, the links to the code of the analyzed deep learning-based approaches for object pose estimation (see Table 1) are given.

Table 6: Links to the code of deep learning-based approaches for object pose estimation: First column shows reference to paper and author-name (method-name). Second column gives links to the code corresponding to the approaches.

Approach	Link to Code
[61] Wang et al. (DenseFusion)	https://github.com/j96w/DenseFusion
[86] Brachmann et al.	https://cloudstore.zih.tu-dresden.de/index.php/s/3b532493918350b17ddec91cb317aea4/download
[70] Billings et al. (SilhoNet)	https://github.com/gidobot/SilhoNet
[71] Xiang et al. (PoseCNN)	https://github.com/yuxng/PoseCNN
[43] Wu et al.	https://github.com/jimmyyhwu/pose-interpreter-networks
[29] Tremblay et al. (DOPE)	https://github.com/NVlabs/Deep_Object_Pose
[75] Tekin et al.	https://github.com/Microsoft/singleshotpose
[77] Zhao et al.	https://github.com/sjtuytc/betapose
[41] Peng et al. (PVNet)	https://github.com/zju3dv/pvnet
[42] Sundermeyer et al.	https://github.com/DLR-RM/AugmentedAutoencoder
[84] Zeng et al.	https://github.com/andyzeng/apc-vision-toolbox