TECHNISCHE UNIVERSITÄT WIEN

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Grasping Reality: Simulation to Reality Transfer of Pick-and-Place Policies

DIPLOMARBEIT

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

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. M. Vincze
Dipl.-Ing. M. Hirschmanner

submitted at the

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

by
Thomas Schönhofer
[redacted]

Vienna, May 2025

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

Danksagung

An dieser Stelle möchte ich all jenen Menschen danken, die mich auf dem Weg zu dieser Masterarbeit begleitet und unterstützt haben. Mein herzlicher Dank gilt meiner Familie, die mich während meiner schulischen und akademischen Ausbildung stets gefördert und unterstützt hat. Ohne diesen Rückhalt wäre das Erreichen dieses Meileinsteins nicht möglich gewesen.
Ich möchte mich bei Prof. Markus Vincze und der Vision for Robotics (V4R) Gruppe an der TU Wien bedanken, die mich herzlich aufgenommen und mich bei der Erstellung dieser Arbeit mit Rat und Tat unterstützt haben.
Ein großer Dank gebührt meinem Betreuer Matthias Hirschmanner. Die zahlreichen konstruktiven Diskussionen und Gespräche mit ihm haben mir stets neue Denkanstöße gegeben und wesentlich zum Fortschritt dieser Arbeit beigetragen. Seine beständige Unterstützung, über mehrere Jahre hinweg, war unerlässlich um diese Arbeit fertigzustellen.
Außerdem danke ich von Herzen meiner Partnerin Katharina. Ihr Rückhalt, Verständnis und ihre Unterstützung in dieser kräftezehrenden Zeit waren eine wichtige Quelle von Zuversicht und Kraft.

Abstract

This thesis investigates simulation-to-reality transfer of pick-and-place policies. Pick-and-place is a fundamental operation in robotics, allowing autonomous agents to interact with their environment. The Transporter Networks architecture is a recently proposed deep learning framework for robotic manipulation tasks, that combines high sample efficiency with good generalization capabilities. Due to the high sample efficiency, previous work has demonstrated the effectiveness of Transporter Networks mainly using human demonstrations. In this thesis, we extend the Transporter Networks architecture to a sim-to-real transfer setting. We enhance the Transporter Networks architecture with adaptations to the training procedure and apply state-of-the art augmentation techniques. For synthetic data generation, we utilize the Nvidia Omniverse platform, which offers a combination of photorealistic rendering capabilities with a high-performance physics simulation. Our developed synthetic data generation pipeline allows for the generation of large-scale, domain randomized, pick-and-place datasets from a set of 3D models. To evaluate the effectiveness of the developed methods, we use a simple box-picking task and a more complex shoe-sorting task, which involves generalization to unseen objects. The results indicate that augmenting depth data is crucial for successful sim-to-real transfer using RGB-pointclouds. In the evaluated tasks, augmentations and randomizations affecting color have minimal impact on performance. The trained models achieve an estimated success rate on real data of 78.6% for pick operations and 75.0% for place operations on the shoe-sorting task. Thus, we demonstrate successful generalization to unseen objects despite being trained exclusively on synthetic data with only 20 individual 3D models. To our knowledge, this thesis presents the first application of Transporter Networks in a sim-to-real setting, offering a promising approach for efficient simulation-to-reality transfer of robotic manipulation tasks.

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit dem Transfer von Pick-and-Place-Strategien für robotische Manipulationsaufgaben, von der Simulation zur Realität. Pick-and-Place ist eine grundlegende Operation in der Robotik, die es autonomen Agenten ermöglicht, mit ihrer Umgebung zu interagieren. Die Transporter-Networks-Architektur ist ein Deep-Learning-Framework für Pick-and-Place Manipulationsaufgaben in der Robotik. Im Gegensatz zu früheren Ansätzen ermöglicht diese Architektur die Generalisierung zu neuen Umgebungen, von einer geringen Anzahl an Demonstrationen. Aufgrund der geringen Anzahl benötigter Demonstrationen wurden Transporter Networks in bisherigen Arbeiten vor allem mit manuellen menschlichen Demonstrationen trainiert. In dieser Arbeit untersuchen wir den Einsatz dieser effizienten Architektur im Kontext von Simulation-zu-Realitäts-Transfer. Dazu wird die Transporter Networks-Architektur durch Anpassungen des Trainingsprozesses und Hinzufügen von Datenaugmentation erweitert. Für die Generierung von synthetischen Trainingsdaten nutzen wir die Nvidia Omniverse-Plattform. Eine entwickelte Pipeline zur Datengenerierung ermöglicht die Erzeugung randomisierter Datensätze aus 3D-Modellen mit minimalen händischen Eingriffen. Um die Effektivität der entwickelten Domain Randomization und Transformations-Methoden zu bewerten, werden zwei Aufgaben genutzt. Zum einen wird das Greifen eines Quaders getestet und zum anderen wird eine komplexere Schuh-Sortier-Aufgabe, die eine Generalisierung, auf im Training nicht vorhandene Objekte, evaluiert. Die Ergebnisse zeigen, dass die Transformation von Tiefendaten für einen erfolgreichen Simulations-zu-Realitäts-Transfer entscheidend ist. Transformationen und Randomisierungen, die die Farbe beeinflussen, haben hingegen nur minimale Auswirkungen auf die Erfolgsrate. Für die Schuh-Sortier-Aufgabe erreichen die trainierten Modelle eine Erfolgsrate von 78,6% für Pick-Operationen und 75,0% für Place-Operationen bei einer Validierung mit realen Pointclouds und unbekannten Objekten. Dies zeigt, dass die Netzwerke erfolgreich generalisieren, obwohl für das Training ausschließlich synthetische Daten und nur 20 3D-Modelle genutzt werden. Diese Arbeit demonstriert die erfolgreiche Anwendung der Transporter Networks-Architektur in einem Simulations-zu-Realitäts-Transfer Szenario. Die entwickelten Methoden bieten einen vielversprechenden Ansatz für effizientes Training von Pick-and-Place Aufgaben in Simulation.

Contents

List of Figures

List of Tables

1 Introduction

From industrial packaging to household chores, pick-and-place tasks are fundamental to many applications of robotic systems. Pick-and-place tasks generally involve a robot grasping and picking up an object from one location and placing it at a desired target location. Despite its apparent simplicity, this task requires the integration of perception, planning, and control in diverse environments. While simple cases assume fixed objects and controlled environments, the real world is often more complex. In household applications, objects often vary in appearance, and environments are cluttered and can even change dynamically. Figure 1.1 shows a robot navigating a household clean-up scenario where it must selectively interact with relevant objects, while ignoring others, as well as deal with a wide range of object appearances.
The deployment of robots in such unstructured environments is challenging, as they need to adapt to changing conditions and interact with a wide variety of objects. To operate under such variability, robots require manipulation policies that can generalize well to unseen scenes and objects. Instead of relying on rigid, hand-crafted methods, that often fail to generalize, modern approaches often use learning-based methods to develop flexible, and robust policies [1]. The ability to train generalizable manipulation policies is the foundation for the deployment of robots into many human-centric settings.

Figure 1.1: Example of a robotic pick-and-place task in an unstructured, household environment. From [2].

1.1 Challenge

Deep learning based systems have shown great success in terms of generalization performance for grasping and pick-and-place systems [1], [3]–[5]. The integration of deep learning in pick-and-place systems can be roughly categorized into two groups: 1) Methods that learn individual components of the *plan, think, act* - pipeline separately, and 2) Methods that learn end-to-end solutions that map directly from sensor inputs to robot actions. Approaches of the first category use deep learning to solve specific parts of the manipulation pipeline, such as object detection, pose estimation, or grasp planning. These methods often rely on hand-crafted control algorithms and task specific logic to execute the manipulation task [6]. While these methods can be highly effective for specific tasks, they often require a priori and domain knowledge to design and implement. Many pose estimation and grasp planning algorithms require 3D models of the objects to be manipulated [7], [8]. This is often not feasible in real-world applications, as objects may vary significantly in appearance and shape. The second category aims to learn the entire task end-to-end, mapping directly from sensors to robot actions. This often allows for better generalization abilities but typically requires immense amounts of training data and computation [3], [4]. End-to-end algorithms often require hundreds of thousands of training samples to achieve good performance [3], [5]. Even when training such systems in simulation or with synthetic data, the requirements on computational resources are often prohibitively high. Furthermore, for successful generalization of policies to unseen objects and scenes, many previous approaches require large datasets that are often difficult to obtain in a robotics context.

Ideally, one would like to combine the benefits of both approaches, creating a system that is both sample efficient and generalizable to unseen objects and environments. In a recent publication, Zeng et al. [9] introduced a new architecture, that promises to unify these properties, called *Transporter Networks*. The Transporter Networks architecture uses a combination of neural networks to predict pick-and-place poses from a top-down view of the workspace, containing color and depth information. Using a sequential approach, by first predicting a pick position and using a crop around this position to guide the search for the place position, the model is vastly more sample efficient than other end-to-end learning approaches. The Transporter Networks architecture often requires only a few hundred training samples to achieve good performance [9]. In previous work, the Transporter Networks architecture has only used human demonstrations to generate training data for real-world applications. Human demonstrations, however, are generally expensive and tedious to obtain and can be especially impractical for tasks where precise demonstrations are required. Because demonstrations are often provided in a specific environment, this also limits the generalizability of the trained policies to unseen environments. Therefore, we propose leveraging simulation-based training data generation for Transporter Networks, combining the benefits of this architecture with the scalability of synthetic data generation. Synthetic data generation allows for the generation of large amounts of training data, without the need for expensive and time-consuming human demonstrations.

Ideally, policies trained in simulation would be able to directly generalize to the real world. Due to discrepancies between the simulation environment and the real world, however, systems trained purely on simulated data often fail when deployed in the real world [10]. This is caused by what is known as the *sim-to-real* gap. The sim-to-real gap is caused by unmodeled processes, differences in the environment conditions, and other

real-world disturbances that are not present in the simulation. To overcome this problem, a possible approach is to make the simulation as close to the real world as possible. This is often referred to as *scene engineering* [11] or system identification [10]. For successful transfer from simulation to reality, this method requires precise knowledge and accurate modeling of the deployment domain. This becomes especially impractical in dynamic environments like households, where conditions can vary significantly and unpredictably. Rather than trying to precisely model the real world, an alternative approach is to make the simulation environment as diverse as possible, by randomizing all aspects that may vary during deployment. This approach, known as *domain randomization*, has been shown to be highly effective for bridging the sim-to-real gap across multiple domains and tasks [10], [12]. Domain randomization involves randomizing all aspects of the simulation that may vary in the real world, such as textures, lighting conditions, viewpoints, and camera parameters. The idea is to expose the model to a wide range of variations during training, so that the model can generalize well to unseen conditions and domains. This is especially important for robots that are deployed in unstructured environments, where the deployment environment cannot be exhaustively modeled in advance. A related approach to domain randomization is *data augmentation*, which involves applying random transformations to the training samples, to increase diversity and improve a model's generalization performance [13].

1.2 Contribution

In this thesis, we propose a novel approach for efficient sim-to-real transfer of pick-and-place policies. Our method allows for the flexible definition of desired task behavior in simulation, and leverages state-of-the-art simulation to reality transfer methods to enable the deployment of the trained policies in the real world. We leverage the strengths of the Transporter Networks architecture, which has been shown to be highly sample efficient in learning pick-and-place policies from demonstrations, and combine it with randomized synthetic data generation. Figure 1.2 shows a diagram of the developed method. We show that Transporter Networks can be trained on synthetic data and generalize to unseen objects and real world environments.

We introduce synthetic data generation pipeline for training data generation, that allows for the generation of large scale, domain randomized datasets from 3D object models, with minimal human intervention. For synthetic data generation, we use the high-fidelity rendering and physics simulation environment, *Nvidia Omniverse* [14]. *Omniverse* combines photorealistic rendering with physics simulation and robot control capabilities in a single platform, making it particularly suitable for sim-to-real transfer. We apply state-of-the-art domain randomization and data augmentation methods, to enable simulation to reality transfer of the learned policies. To the best of our knowledge, we are the first to use the Transporter Networks architecture in a simulation to reality setting.

To benchmark the sim-to-real transfer and generalization performance of the developed system, two tasks are designed. (1) A simple *Box picking* task, where the goal is to predict the pick pose of a cuboid with a fixed size. (2) A more complex *Sort shoes* task, where a pair of shoes must be aligned next to each other. The latter requires the model to generalize to unseen objects and environments simultaneously. Real-world pointcloud data is collected for both tasks and the developed randomization and augmentation methods

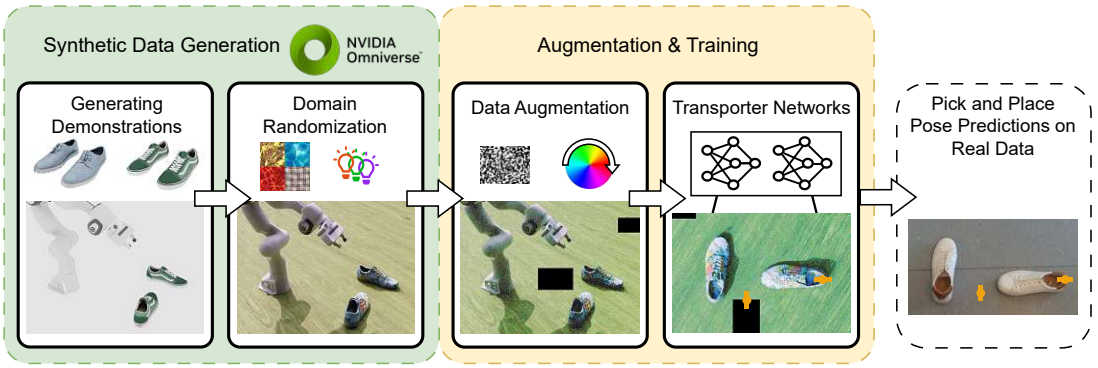


Figure 1.2: Overview of the *Grasping Reality* method developed in this thesis. The method consists of a synthetic data generation pipeline that generates domain randomized training data, used to train the Transporter Networks architecture, with various sim-to-real augmentation methods applied.

are evaluated in multiple ablation studies.
For the *Box picking* task, a real-world dataset with 40 scenes is recorded, using two cuboid-shaped tea boxes with different textures. For the *Shoe sorting* task, a dataset with 28 scenes, using four different pairs of shoes is recorded. Additionally, a dataset with distractor objects is recorded for the *Shoe sorting* task, to evaluate the robustness of the trained policies in cluttered environments. This dataset contains 20 scenes with the same four pairs of shoes, but with additional distractor objects present in the scene. The recorded pointclouds are preprocessed and manually annotated using a newly developed processing pipeline. From the annotated pointclouds we create datasets to evaluate the performance of the trained policies and to analyze the sim-to-real transfer performance of the developed methods. In extensive ablation studies, we find that the augmentation of depth data is crucial for the performance of the model on real-world data. Especially, the addition of random Gaussian noise to the depth data is essential for successful sim-to-real transfer. Randomization and augmentation of color information, on the other hand, does not significantly influence the performance.
In a final evaluation, we show successful transfer of pick-and-place policies, trained exclusively on synthetic data, to real-world data. To quantify the performance of the trained policies, we estimate the success rate using success-thresholds for the position- and angle errors (Pick: 3.00 cm and 10.00°, Place: 5.00 cm and 10.00°). On the Box picking task, we achieve an estimated pick success rate of 100.0 %, with a mean error of 6 mm. For the Shoe sorting task, we achieve an estimated pick success rate of 78.6 and an estimated place success rate of 75.0 %. The success rate for scenes where both, the pick and place operation are successful is 67.9 %.
Compared to the results of the original Transporter Networks implementation, trained without sim-to-real methods, we achieve an improvement of 57.0 % in overall success rate on the Shoe sorting task. The results show that the Transporter Networks architecture, combined with the developed randomization and augmentation systems, is able to successfully bridge the sim-to-real gap.

1.3 Thesis Structure

The remainder of this thesis is structured as follows. We first introduce the *Transporter Networks* architecture, as well as the changes applied to the networks and training process in Chapter 3. We continue with describing the synthetic data generation pipeline, the randomizations used during training data generation, and the augmentation methods employed during the training process.
In Chapter 4, we then describe the two tasks used for evaluation in detail, as well as the data collection and annotation process. In following sections, we present the results of the ablation studies and the final performance evaluation of the trained networks. Finally, we discuss the results and the limitations of the developed methods in Chapter 5 and conclude with a summary of the findings and an outlook on future work in Chapter 6.

2 Related Work

Bridging the sim-to-real gap is essential to leverage the advantages of simulation for training deep learning models for real-world use. The challenge of transferring knowledge from simulation to the real world has been a long-standing issue in robotics and computer vision. One of the first papers that uses disturbances in the simulation environment to create more robust controllers is *Evolutionary Robotics and the Radical Envelope of Noise Hypothesis* by Jakobi [15]. The author shows that algorithms trained in simulation will find ways to exploit unrealistic details and flaws in the simulators. The way to mitigate this behavior, is to add noise to all aspects of the simulation that may vary in the real world. This has laid the foundation for the idea of *domain randomization*, which has become a popular approach to bridge the sim-to-real gap. The idea of domain randomization is to expose the model to such a wide range of variations during training, that the real world appears to the model as just another variation [10].

A related approach to domain randomization is *data augmentation*. Data augmentation describes a set of techniques to artificially increase the number of training samples by applying various transformations to existing ones. It thereby enhances a given dataset by creating new samples from existing ones. In the context of sim-to-real transfer, data augmentation techniques can be used to add various perturbations to the synthetic training data, to make it more similar to the real-world data.
The main distinction between domain randomization and data augmentation is their position in the data generation and training pipeline. Domain randomization is applied to the simulation environment during data generation, while data augmentation is applied to the generated training data. More formally, domain randomization changes the mapping from a state s_t to the corresponding observation o_t ($s_t \rightarrow o_t$). Whereas data augmentation transforms an observation o_t into a new observation $o_t\prime$ ($o_t \rightarrow o_t\prime$). Data augmentation can therefore not alter aspects of the mapping from a state to an observation, such as the the viewpoint or the lighting conditions. It can however be used to impart variations to the observations that are not present in the data sample, such as noise, blurriness, or occlusions. Data augmentation is often used in conjunction with domain randomization to further increase the diversity of the training data and improve the generalization of the trained models.
This section reviews key developments and applications of these techniques.

2.1 Domain Randomization in Computer Vision

The idea of randomizing simulation parameters to improve the transfer from a simulation environment to reality has been introduced to the computer vision domain by Tobin et al. [10]. The authors showed that with enough variation in the training data, a object localization model trained exclusively in a low fidelity simulation can generalize to real-world data. To enable this transfer, the authors randomized object appearance, lighting

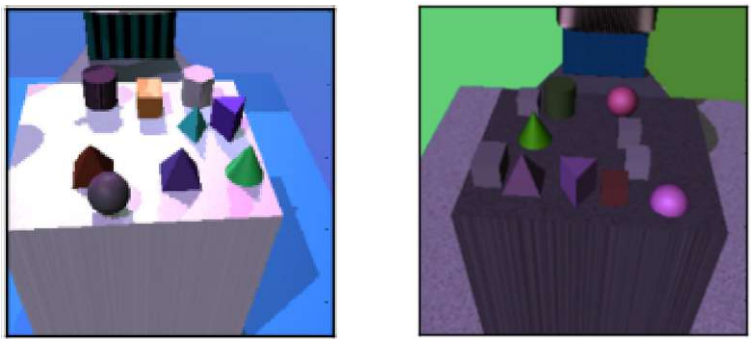


Figure 2.1: Example scenes from the publication of Tobin et al. [10].

Figure 2.2: Example scenes from the publication of Sadeghi and Levine [12].

conditions, camera position, and other simulation parameters. They introduced the term *domain randomization* for this concept. Since then, domain randomization has become a core technique to bridge the sim-to-real gap in various robotic applications.
Around the same time, Sadeghi and Levine [12], used similar techniques to train an indoor flight controller for an autonomous drone using simulated images only. Compared to the work in [10], the authors used more complex object textures during randomization. While [10] used simple textures, like gradients and checkerboards(Figure 2.1), the authors in [12] used material surface textures from the *Blender* software [16](Figure 2.1).
In a following publication, James et al. [17] trained an controller for a robotic arm to solve a pick-and-place task using visual input(Figure 2.3). The authors used a similar approach to Tobin et al. [10], but instead of using simple textures, they used textures generated from Perlin noise [18](Figure 2.3). Perlin noise is a type of procedurally generated gradient noise that is often used in computer graphics. In a following study, Perlin textures have been identified as especially effective for sim-to-real transfer [19].

With the wide availability of high-fidelity simulators, developing domain specific scenes has become a more common practice. Recent papers, combine domain randomization with scene engineering, randomizing unknown factors and modeling known ones. Eversberg and Lambrecht [11] use an object detection task for an industrial use case as a benchmark. The authors use a physics-based simulator to generate high quality synthetic data and compare modeling the target domain to domain randomization. In [11] and [20] they show that using realistic textures, combined with randomized background, lighting conditions, and camera positions achieves optimal performance on a real-world dataset. Both of these approaches, however, require that the target domain and the expected textures are known in advance. This does not hold for many real-world applications, where the textures of

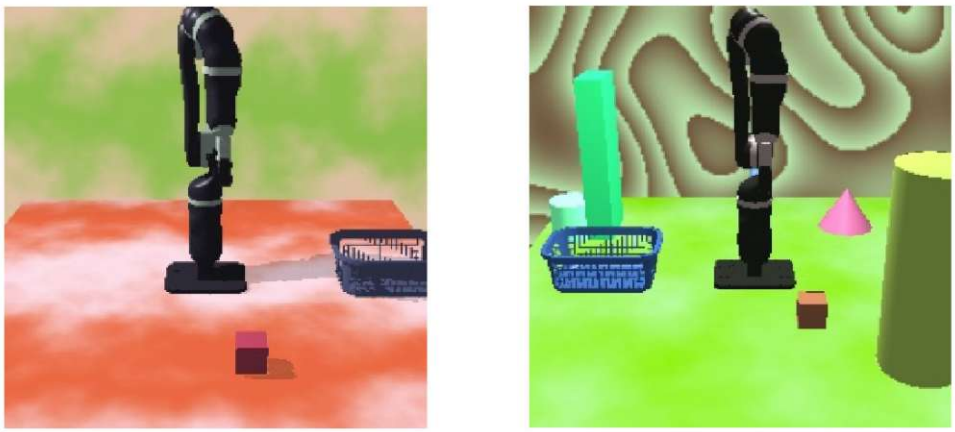


Figure 2.3: Example scenes from the publication of James et al. [17].

encountered objects are not known beforehand.
All of these methods share the idea that model robustness improves by adding variation to the training data by randomizing the simulation parameters. Domain randomization has been shown to be an essential technique for sim-to-real transfer in various applications, including object detection, grasping, and navigation tasks citesadeghiCAD2RLRealSingleImage2017, tobinDomainRandomizationTransferring2017, eversbergGeneratingImagesPhysicsBased2021. Domain randomization is often complemented by data augmentation techniques to further increase the diversity of the training data.

2.2 Data Augmentation

While domain randomization, randomizes the simulation parameters to increase the diversity of the training data, data augmentation introduces variations to the training data itself. Data augmentation is a widely used technique in computer vision to improve the performance of deep learning models, by artificially increasing the size of the training dataset. One of the first works to demonstrate the effectiveness of data augmentation on a machine learning computer vision task is the work of Lecun et al. [21]. The authors applied a combination of random translations, rotations, and scaling operations to augment training images for handwritten digit recognition. Extending previous work Krizhevsky et al. [22] introduced many state-of-the-art image augmentation techniques such as random crops, horizontal flips, and color jitter to improve the performance of an image recognition CNN. Using data augmentation techniques lead to a reduction in error rate of almost 50.00 %, compared to other state of the art models at the time.
More recently, Laskin et al. [13], applied data augmentation techniques to end-to-end reinforcement learning tasks. The authors show that augmenting the training data with random crops, color jitter, and other transformations significantly improves the performance of vision-based reinforcement learning algorithms.
In a sim-to-real context, data augmentation techniques are often used to add various real-world imperfections to the synthetic training data [23], [24]. This can include adding noise, blur, occlusions, or other artifacts that are commonly found in real-world data. The goal is to make the synthetic training data more similar to the real-world data,

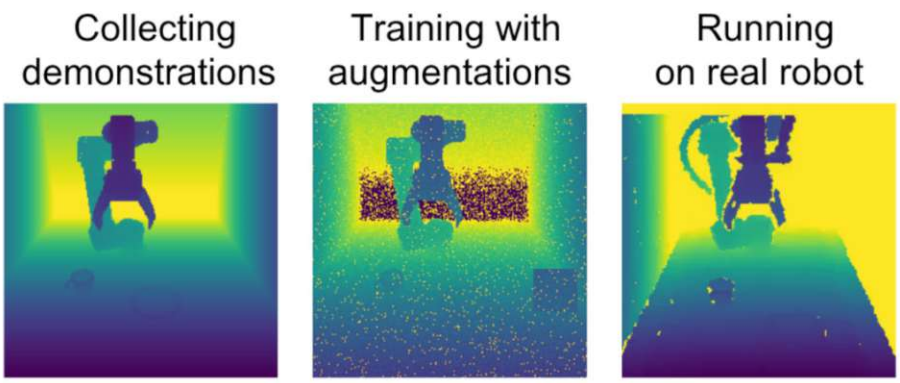


Figure 2.4: Example of depth images, from raw, over augmented to real, from the publication of Pashevich et al. [23].

thereby improving the generalization of the trained models. In contrast to traditional computer vision that mostly relies on RGB images, robotic perception often includes depth information. Depth information is an essential component for control and decision making. However, depth data is often noisy and incomplete, whereas synthetic depth data generated using optimal sensor models is generally noise-free and complete. This is a classic example of the sim-to-real gap.

Several works have explored the use of data augmentation techniques to improve the performance of models trained on synthetic depth data. Pashevich et al. [23] show that applying traditional image augmentation techniques such as transformations, cutouts, and noise to synthetic depth data to be benefitial for successful sim-to-real transfer. Figure 2.4 shows a raw depth image generated in simulation, its augmented counterpart, and a real depth image for comparison. The authors show an improvement of success rates from 15.00 % to 90.00 % when applying depth augmentations.

Other works have focused more on modeling the noise characteristics of real-world depth sensors and applying these models to synthetic depth data. For a grasp success prediction network, trained on simulated depth images, Mahler et al. [24] approximate real sensor noise by adding a distance-dependent noise component to simulated depth images. Instead of using depth images, Hagelskjaer and Buch [25], apply Gaussian noise to the xyz-positions in a RGB-pointcloud. Although this noise model, using Gaussian noise only, is far simpler than other methods [23], [26], the authors show successful simulation to reality transfer of a pose estimation network. Reaching state of the art performance on the *LINEMOD* pose estimation benchmark [27], while being trained exclusively on synthetic data. An increasingly complex noise model was introduced by Thalhammer et al. [26]. The authors combined occlusion masks, Gaussian smoothing, depth-proportional noise and Perlin noise to train a object detection network from synthetic depth images. They show an improvement of 25.00 % in detection recall score when using the developed noise augmentation pipeline.

In summary, data augmentation techniques have been shown to be especially effective in improving the simulation to reality transfer performance of models trained on synthetic depth data. These techniques can be used to add various real-world imperfections to the synthetic training data, making it more similar to real-world data.

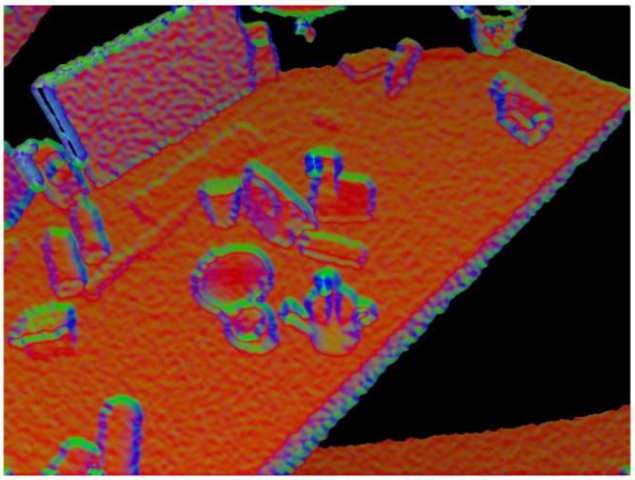


Figure 2.5: Example depth image with augmentations from the publication of Thalhammer et al. [26].

Combining domain randomization and data augmentation techniques promises to be a powerful approach for sim-to-real transfer, especially for multimodal data such as RGB-D images or pointclouds.

2.3 Simulation Environments for Sim-to-Real Transfer

The choice of simulation environment can drastically influence the sim-to-real performance, as well as the required effort to implement domain randomization techniques [28]. In the sim-to-real literature robotic simulators like PyBullet and Gazebo are often used to generate domain randomized training data [23], [29], [30]. While these simulators offer basic rendering capabilities, their focus is primarily on simulating the physics of robotic systems. They do not provide the same level of realism and fidelity as dedicated rendering engines.
Modern rendering engines, such as Unity [31], or Blender [16] on the other hand, focus on rendering quality and visual realism. They often lack the physics engines, robot models and controller support that are required for sim-to-real transfer in the robotics domain.
Nvidia Omniverse aims to offer the best of both worlds. The Omniverse platform combines the *PhsyX* [32], high performance physics engine with a ray tracing capable rendering engine. On top of this, Omniverse offers dedicated extensions for robotic control and domain randomization. *Isaac Sim* is an extension built on top of the omniverse platform that provides models, and controller support for various robotic applications [33]. With the *Replicator* extension, the Omniverse platform even provides a dedicated framework for domain randomization and synthetic data generation [34]. The combination of high-fidelity rendering and physics simulation makes the omniverse platform a powerful tool for sim-to-real transfer in robotics.

3 Grasping Reality: Sample Efficient Sim-to-Real Transfer for Robotic Pick-and-Place Tasks

Learning generalizable pick-and-place policies for robotic manipulation tasks is a challenging problem. Especially variable object shapes, appearances, and locations make it hard to generalize from limited data. Over the years, many different approaches have been proposed to solve this problem, ranging from traditional computer vision systems, requiring precise pose estimations of the task objects [35], to end-to-end deep learning approaches that use deep neural networks to directly map visual input to robot actions [17]. A recent approach to this problem are *Transporter Networks* [9]. The Transporter Networks architecture predicts pick-and-place poses from visual input. The system is trained on demonstrations of the task to solve. By reducing the problem to predicting two key poses, and leaving the planning of the movement between these poses to the robot controller, they achieve a high sample efficiency and good generalization to unseen configurations.
Another distinguishing factor of Transporter Networks is the combined use of color- and depth information in the input data. This enables the network to learn both semantic features in color information, and structural features in depth data. It is this combination of feature that motivated the use of Transporter Networks for trying to transfer pick-and-place policies from simulation to reality in thesis. We use a combination of domain randomization and data augmentation techniques to train Transporter Networks to enable sim-to-real transfer.
In contrast to most of the domain randomization literature, we use a high performance rendering engine for synthetic data generation. The intuition is that this more complex simulation can encompass a larger visual state space than simpler renderers and therefore result in better simulation to reality transfer. In this thesis we use the *Nvidia Omniverse* [14] platform for synthetic data generation. Nvidia Omniverse, provides extensive capabilities for robotic simulation and domain randomization. More details about its usage are given in Section 3.3.

To bridge the simulation-to-reality gap, we use a combination of domain randomization and data augmentation techniques. Explanations and implementation details of the used domain randomization and data augmentation methods are given in Section 3.4 and Section 3.5 respectively.

3.1 Transporter Networks

Transporter Networks are a neural network architecture designed for vision-based robotic manipulation [9]. They efficiently learn to predict spatial displacements, like pick and place poses, from visual input. By directly predicting two key poses, complex tasks such

as object rearrangement, deformable manipulation, or even multi-step problem solving can be learned very efficiently. The key features that motivate the use of the Transporter Networks architecture for this thesis are its sample efficiency, the simple input-data format, and the promise of good generalization to unseen scenes and objects. These properties make it a good candidate for transferring policies from simulation to reality. To our knowledge, this is the first usage of the Transporter Networks architecture in a sim-to-real setting.

3.1.1 Network Architecture

One of the main contributions of Transporter Networks is the use of a mechanism the authors call *spatial transformer mechanism*. This mechanism is best explained using a pick-and-place task as an example. In the first step of prediction, the Transporter Networks system uses a convolutional neural network(CNN) to predict a pick position in the scene. The predicted pick position is then used to guide the search for the place pose. A mechanism the authors call *pick-conditioned-placing*. Figure 3.1 shows the schematic of this process [9]. To predict the place-pose, the input observation is fed into two CNNs, a *key* network and a *query* network. By cropping a region of interest in the spatial feature space generated by the query network, centered at the previously determined pick position a query-feature map is created. This crop of feature-space values is then convolved over the feature map generated by the key network. The place position is predicted by finding the location of the maximum value in the result of this convolution. This mechanism is combined with a rotation of the query feature map in 10 degree increments. The convolution is done with each rotated feature map. This creates a 3D value map, where 2 dimensions represent the spatial location and the third dimension represents the orientation. The index of the maximum value in this 3D value map - the *argmax*, is used as the place pose.

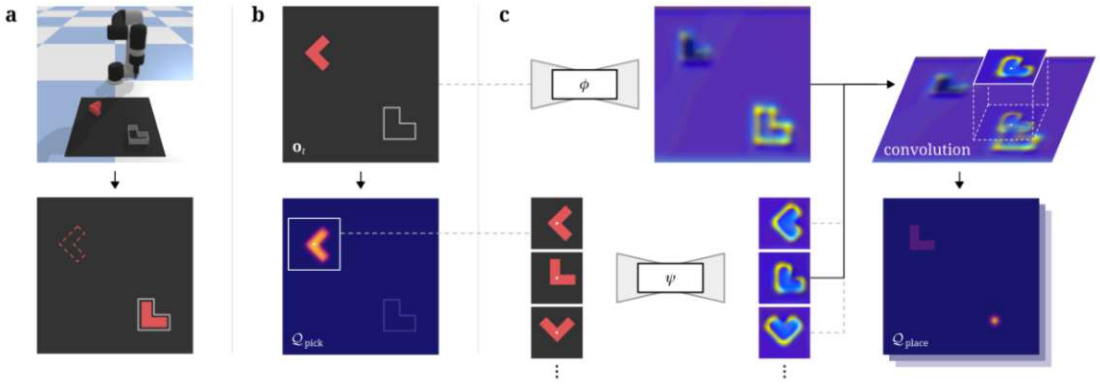


Figure 3.1: Schematic of the Transporter Network architecture from [9]. (a) The scene is projected into a top-down view. (b) The attention-network predicts the pick position. (c) The input observation is fed into the key network and a crop around the pick position is rotated and fed into the query network. The resulting feature maps are convolved to predict the place pose.

3.1.2 Sample Preprocessing

A contributing factor to the good sample efficiency of the Transporter Networks system lies in the preprocessing of observation samples. Instead of using raw RGB-D images as input to the networks directly, a top down view of the scene is created from the input observations. This is done by combining the given RGB-D images to create a RGB pointcloud. A top-down view of the scene is created from this pointcloud. Resulting in a color and a heightmap image, which are combined and used as the input to the networks. This preserves the 3D spatial structure of the scene, while at the same time making the system largely viewpoint independent.

3.1.3 Data Augmentation

An additional mechanism that contributes to the sample efficiency of the transporter networks architecture is the use of specific data augmentation methods. Random translations and rotations are applied to the input observation and the pose-labels. Thereby creating a new data sample, where the relationship between pick pose and place pose stay constant, but the absolute poses change. This way, more of the input space is covered with the same amount of labeled data.

3.1.4 Equivariant Transporter Networks

A recent extension to the Transporter Networks architecture is the *Equivariant Transporter Networks* system [36]. The authors have adapted the network to use a parallel-jaw gripper, by adding a pick angle prediction network. Being able to use a parallel-jaw gripper allows for a far broader range of objects to be manipulated. Therefore, we decided to use *Equivariant Transporter Network* for this thesis.
Additionally, the use of a special type of neural network, allows this system to be equivariant to orientation of the target object. This means, that the network, by design, is able to generalize to unseen orientation of the object to pick. This enables even higher sample efficiency than the baseline Transporter Networks system.

3.2 Adapting Transporter Networks for Sim-to-Real Transfer

As noted before, to the best of our knowledge, we are the first to use a variation of the Transporter Networks architecture in a sim-to-real setting. In previous publications the networks were either trained and evaluated on simulated data, or on real world data exclusively. No transfer from one domain to the other was done.
Despite that, we assume that the benefits of sample efficiency and generalizable policies, also offer a promising basis for bridging the sim-to-real gap. To enable the use of the *Equivariant Transporter Networks* architecture in a sim-to-real setting, we extend the original implementation. The original implementation was obtained from the GitHub repository of the authors [37].
Multiple modifications to the training code and network architecture were done and are explained below.

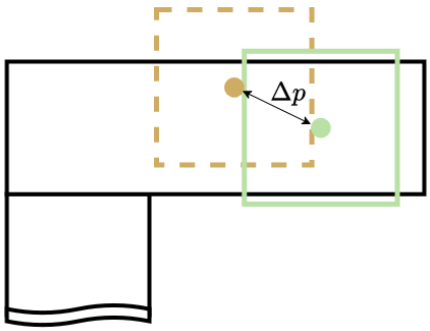


Figure 3.2: A difference in crop position Δp between ground-truth(green) and prediction(orange) leads to a difference in crop position.

3.2.1 Decoupling of Networks

During training we observed that the networks used to predict pick position, pick orientation and place pose, require different amounts of training iterations to converge to satisfactory results. This is likely due to the different complexity of the individual parts of the task. It was therefore favorable to be able to train each of the networks(pick position, pick angle and place pose) separately and only combine them for inference if necessary. This also allowed to test different learning rates and other hyperparameters for each of the networks individually. To achieve this separation, the common implementations used by all networks were left in the original class, while the network specific implementations were moved to separate classes, that could be individually trained and evaluated.

3.2.2 Augmentation of Crop Position

The crop input, used by the pick angle prediction network and the transporter network is centered around the pick position. During deployment, the position where the crop is taken, is therefore predicted by the Attention(pick position) network. Due to imperfections in the performance of the attention network, the crops will not be centered precisely at the ground-truth pick position. In the original implementation's training process however, crops are always centered at the ground-truth pick position. This leads to a difference between the crops encountered in training and the crops encountered during deployment. Figure 3.2 illustrates the difference in crop positions caused by a difference between ground-truth pick position and predicted pick position.
To account for the expected differences during deployment, we introduce a method, that adds a random offset to the crop position during training. With this perturbation, the network learns to be more robust to errors in the pick position prediction.

3.2.3 Batched Processing

The original implementation of the *Equivariant Transporter Networks* uses a batch size of 1 for training. This leads to high variability in the loss and error during training, since network weight updates are based on the loss of only one sample. It has been shown, that

higher batch sizes can lead to better generalization [38]. Batched training also allows to use modern techniques such as batch normalization, which can improve the training process [39]. Furthermore, batched training takes advantage of the parallel processing capabilities of modern GPUs. To address this, the training process and the networks are adapted to use a configurable batch size.

3.2.4 Dataset Format

The dataset from the original *Equivariant Transporter Networks* publication uses individual *pickle* files[40] to store the poses, color- and depth observations, and info fields for each sample. While this is a simple and efficient way to store the data, it is not ideal for the comparison of different randomizations of the same scene. Comparing different randomizations is drastically easier, if the storage of labels and orbservations is separated, so that the same scene can be used with different randomizations. To address this, the dataset is modified to only store the poses for each scene, and store a reference to a folder that contains the observation data. The connection between observation data and the poses is established using the index of each scene as the folder name for the observation data.

3.2.5 Input Data Format

The original *Equivariant Transporter Network* [36] uses RGB-D images as observation input. The depth images are backprojected during each training step, to create a pointcloud and from that, a top-down image of the scene. It is computationally inefficient, to do this at each training step, rather than beforehand. Furthermore, many real-world depth cameras already provide pointclouds as output.
For reasons of efficiency and simplicity, the network was adapted to use RGB-pointclouds as input. This allows for easier integration of captured real-world pointclouds during evaluation. The only requirement is that the pointclouds need to be roughly aligned to the xy-plane, and the workspace in xy-coordinates is visible in the pointcloud. This however can be done in a preprocessing step. The requirements for the alignment of the pointclouds can be somewhat relaxed, when introducing perturbations to the simulation data, as the network can learn to ignore the perturbations. This is describe in more detail in Section 3.5. We use the *Polygon File Format - PLY* [41] for the pointclouds, which is a simple file format for storing polygon objects or pointclouds. It can be easily read by most pointcloud processing libraries, as well as be viewed in most pointcloud viewers, such as *Meshlab* [42]. This is very valuable for validation of input data during development of the training process.

3.2.6 Data Loading and Preprocessing

The aforementioned increase in batch size as well as the augmentations applied to the input data, introduced in this thesis, increases the required processing time for each sample. We parallelize the data loading and preprocessing using the *Python multiprocessing* library, to alleviate this slowdown. The data loading and all augmentations for a training batch can be moved to a separate process and computed in parallel.

3.2.7 Dropout Layers

In an attempt to reduce overfitting and improve generalization, dropout layers are added to the networks. Previous research has shown that dropout can be beneficial for training deep neural networks, especially with limited dataset sizes [43]. Transporter Networks uses a variation of a *Residual Network - ResNet*, a popular architecture for computer vision tasks[44]. Dropout layers are added in the residual blocks, between the first activation output and the second convolutional layer, as has been proposed in [45]. Since the effects of dropout is compounding when using multiple encoder and decoder layers, the initial dropout rate is reduced by 0.0125 for each layer, with the highest dropout at the bottleneck of the encoder-decoder structure.

3.3 Synthetic Data Generation

Most deep learning approaches, rely on large amounts of training data to achieve good performance. Especially in the field of robotics, acquiring training data in the real-world can be costly and time-consuming. Using synthetic data generation is one of the most common approaches to overcome this limitation. This section describes the implementation details of the synthetic data generation pipeline used in this thesis.
Previous publications in domain randomization research often used low fidelity simulators for data generation [12], [29], [46]. The used, simple physics- and rendering engines, are not able to simulate complex surface materials or realistic lighting conditions. This limits the visual state space of the generated data, which can be especially problematic for challenging materials, such as transparent or reflective surfaces. In this thesis, we use the high fidelity, ray-tracing capable simulator *Nvidia Omniverse* [14]. We argue that high fidelity simulators are able to simulate more complex lighting conditions and material properties, and the simulated data therefore covers parts of the visual state space that cannot be reproduced by simpler rendering systems. Therefore, creating a more diversely randomized dataset that is more likely to encompass the visual state space of the real world.
The *Omniverse* platform contains a 3D rendering and physics simulation engine, and offers various extensions for animation and design, synthetic data generation, as well as robotics simulations and reinforcement learning. For scene and dataset generation in this thesis we use the *Isaac Sim* [33] extension, and *Omniverse Replicator* [34] for domain randomization and rendering. The *Isaac Sim* extension is a robotics simulation environment that allows for the creation of complex 3D scenes, simulation of robot interactions, and development of robotic control algorithms. Figure 3.3 shows an example of a scene in *Isaac Sim*. It offers support for physics simulation as well as physically accurate lighting and material simulation, which is especially beneficial for developing vision-based algorithms. A *python* API is offered, that allows access to the full simulation environment, as well as various extensions for control of robots and various environment conditions. This thesis uses *Isaac Sim* in version 4.2.0, which was the latest version during development of the programs for this thesis. The used extension for synthetic data generation, *Nvidia Replicator* offers a range of capabilities for the randomization of lighting conditions, material properties, physical properties of objects, camera parameters and many more. It also offers tools for rendering of synthetic data in various formats, such as color images, depth images, segmentation masks, point clouds, et cetera.

Figure 3.3: Example scene in *Omniverse Isaac Sim*.

3.3.1 Data Generation Pipeline

The data generation pipeline(Figure 3.4) consists of two main parts, (1)*Scene and Dataset Generation* and (2)*domain randomization and Rendering*. By doing so, we can randomize the exact same scenes with different randomization parameters. Therefore, all observed performance differences can be attributed to the choice of the applied randomizations. In the *Scene Generation* step, the simulator ground truth information is used to create training labels, i.e., the ground truth pick and place poses of the objects of interest. This information, together with all required information to recreate each scene is saved to the dataset *json*-file. In the *domain randomization and rendering* step the object poses and scales for each scene are loaded from the dataset file and various randomizations are applied to to scene.

3.3.2 Scene Generation

The scene generation step creates task-specific simulation scenes that serve as the basis for training data generation. During scene generation, the objects of interest are placed in a defined workspace and their poses are recorded. After the initial placement of the objects in the workspace, the world pose of a pre-annotated grasp point on the object to pick is recorded and stored as the *pick-pose* label for this scene. The *place-pose* is calculated using task-specific logic. Optionally, the pick-and-place operation is executed to validate the calculated *place-pose*.
The individual steps during scene generation are explained in more detail in the following sections.

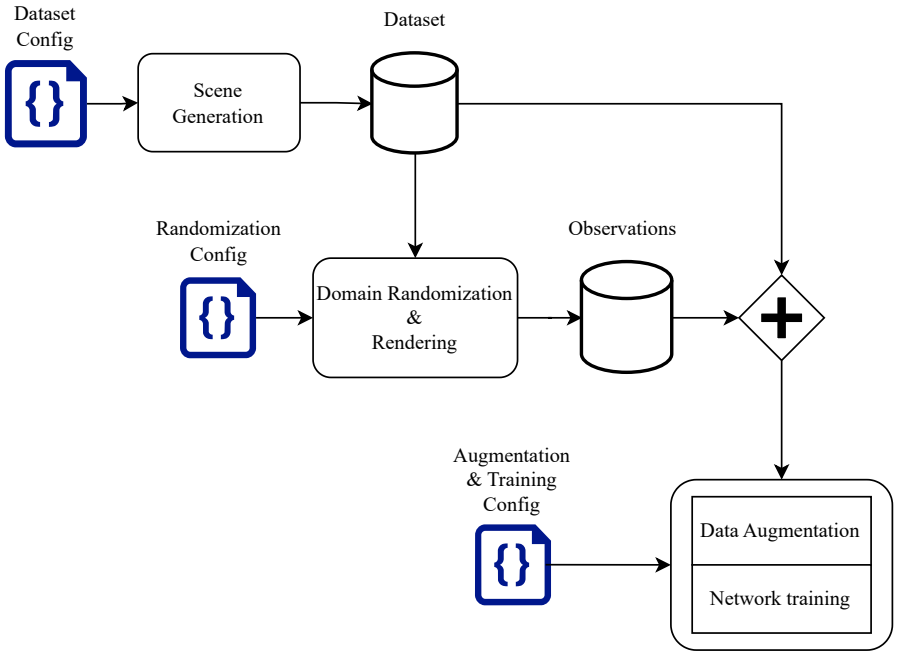


Figure 3.4: Data generation pipeline

Model Import

The first step in scene generation is the import of the desired object models into the object file format used by *Omniverse*. *Omniverse* uses the *Universal Scene Description (USD)* format as a format to save individual objects and complex scenes. The conversion to *USD* from various 3D object file formats is done using a provided importer in *Isaac Sim*. After the import of the object model, the model is scaled, so that all models of a dataset are initially the same size, regardless of the original size of the object mesh. This is done to ensure that the same randomization parameters can be used for all objects in the dataset.

Grasp Point Annotation

To determine the pick position of the objects of interest, we use a pre-annotated grasp point on the object. To explain how the desired grasp point is annotated on the object, it is first necessary to discuss how objects are structured in the simulator. In *Isaac Sim*, Scenes and objects are hierarchically structured, with the Scene being the top level object and all other objects being children of the Scene-object in the hierarchy tree. Figure 3.5 shows an example of the scene hierarchy of a grasp-experiment object in *Isaac Sim*.
The most basic object type in *Isaac Sim* is the *XFormable* object, which is a transformable object that has no physics properties or activated APIs. More complex object types, with object meshes, physics properties, etc. are derived from the *XFormable* object. The *XFormable* object is most often used as a container to group multiple objects, so that they can be manipulated together. The hierarchical design allows for inheritance of properties

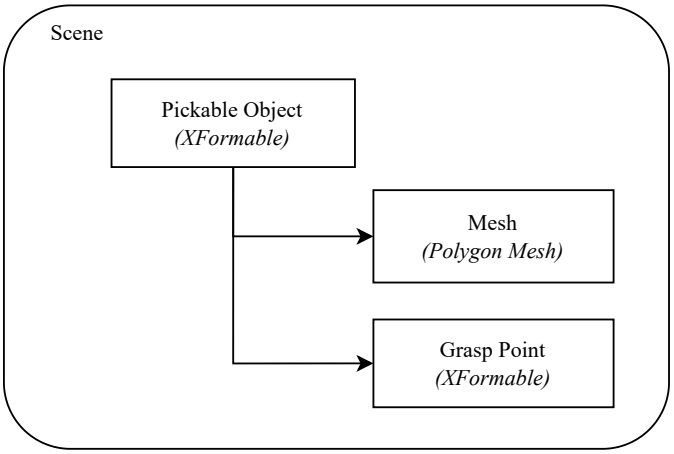


Figure 3.5: Scene hierarchy of an grasp-experiment object in *Isaac Sim*.

in the tree and also to define local transformations and scales for each object in the tree, with respect to its parent object. This parent-child inheritance model is used to define the grasp point on the object.
The grasp point is defined in a way that transforms and scales together with the base object. To implement this, the grasp point is defined as a child object of the object to pick, with a fixed translation and rotation offset from the base object reference frame. The annotation of the grasp points is done manually in the *Isaac Sim* user interface. An example of a grasp point annotation is shown in Figure 3.6.

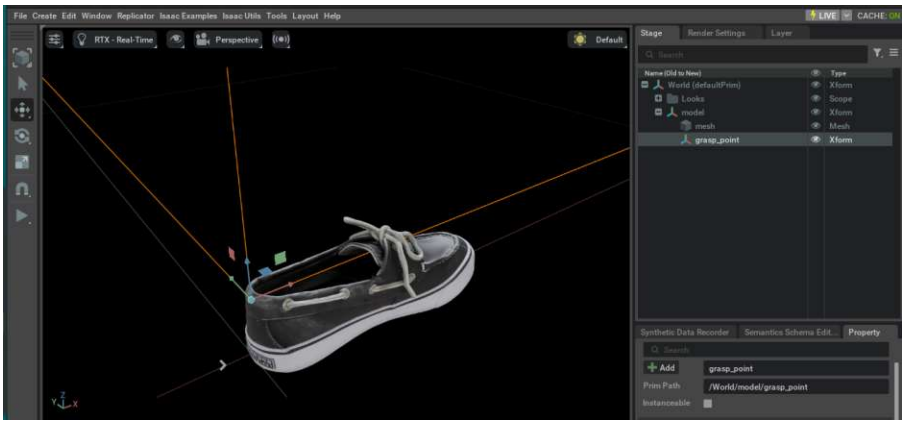


Figure 3.6: A shoe model with annotated grasp point in *Isaac Sim*.

Scene Generation

Using the annotated models, the scenes for the dataset are generated by placing the objects in the workspace and calculating the pick and place poses. The workspace is defined as a rectangular area with a size given by the dataset configuration. For each sample, the objects can be optionally scaled by random factors in each dimension. While

object scaling is a form of domain randomization, it must be implemented during the *Scene Generation* step since it directly affects the pick and place poses of the objects. After the scaling step, the objects are placed in the workspace in a random, collision-free configuration. From this initial configuration, the pick pose is determined by saving the world pose of the grasp point on the object of interest. The place-pose is calculated, from the ground-truth simulator state, using a task-specific logic that must be implemented for the task at hand.
The output of the generation procedure is a dataset-*json* file, that contains the labels for each scene, i.e., the pick and place poses, as well as all necessary information to be able to recreate the scenes in the domain randomization step. A *json* configuration file is used to specify the parameters for scene generation and the dataset.

3.3.3 Domain Randomization and Rendering

In the *domain randomization and rendering* step, the observations for a given dataset are generated by recreating the scenes from the dataset file, applying various randomizations, and rendering a RGB-pointcloud to use as an observation. RGB-pointclouds contain the xyz-coordinates, as well as the color information for each recorded point. For randomization and rendering the *Omniverse Replicator* extension is used. It provides a *Python API* to create complex randomization and rendering pipelines. The pipelines are created as *Omniverse Graphs*, which is a node-based system within *Omniverse*, to interact with simulation scenes. Figure 3.7 shows an example of a *Omniverse Graph* for randomizing a light source. *Omniverse Graphs* can either be created via a graphical user interface or programmatically using the *Python API*. In this thesis, the *Python API* is used to create the randomization and rendering graphs. Details about the used randomizations and their implementation details are given in chapter Section 3.4.

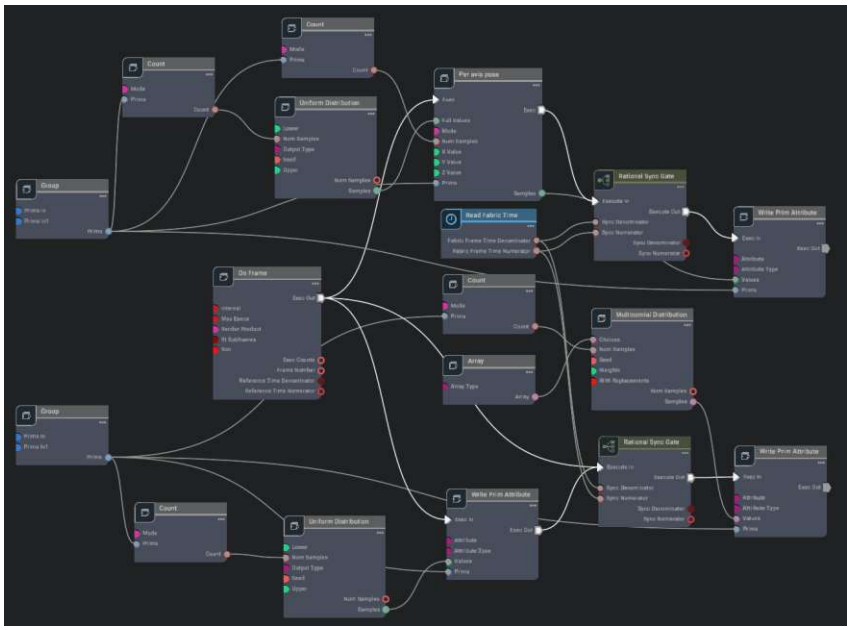


Figure 3.7: Example of a *Omniverse Graph* for randomizing a light source

Rendering

As explained in Section 3.2, the observations for the adapted *Transporter Network* are RGB-pointclouds, in the *Polygon File Format (PLY)* format. To generate observations for the dataset, the pointcloud annotator from the *Omniverse Replicator* extension is used. The pointcloud annotator is a node in the *Omniverse Graph* that is a attached to a camera and produces a pointcloud based on the depth and color information given from the simulated camera. The output of the pointcloud annotator is a *numpy*-array, containing the coordinates and color values of each recorded point in the scene. The *open3d* library [47] is then used to write the obtained *numpy*-arrays to *PLY* files for the training process.
As mentioned above, the pointcloud renderer is attached to a camera in the rendering graph. Although the calculations for creating a pointcloud from the simulated depth images is handled internally by *Replicator*, the created pointclouds are still influenced by the camera parameters of the attached camera. For example, different opening angle and distortion parameters of the camera change the created blind spots and potentially also the recorded shapes of objects.

The real-world datasets for this thesis are recorded using a *Intel RealSense D435* depth camera. A model of this camera is already included in *Isaac Sim*, that approximates the camera parameters of the real camera. More details on how the camera is modeled can be found online at the *Nvidia Omniverse* documentation [48].

3.4 Domain Randomization

Using simulation develop algorithms and to collect data for training robotic systems has many advantages over developing them with data collected in the real world only. The environment can be controlled, large-scale data collection is cheap and easy, and no harm can be done to the robot or the environment. In an ideal world, the resulting algorithms, trained with simulated data, would be directly usable in the real world. Due to discrepancies between the simulation and the real world, this is usually not the case and the developed controllers fail to generalize to real-world data [17]. This is known as the *sim-to-real gap*. It is caused by imperfect modeling of real-world processes, sensor noise, and other factors that are not present in the simulation.
Reducing the sim-to-real gap is crucial to be able to successfully deploy algorithms trained on synthetic data, in real-world settings. One of the most popular methods for reducing the sim-to-real gap is by using domain randomization methods. Domain randomization describes a set of techniques that aim to make the trained network invariant to domain-specific features and learn task-relevant features that transfer between domains. By varying all task-irrelevant features in simulation, such as object textures, lighting conditions, camera positions, the trained network will ideally be agnostic to changes in these factors in the real-world data. The appearances encountered in real-world data will ideally be a subset of the appearances and scenes encountered during training.
Especially in the sim-to-real context, domain randomization and data augmentation are closely related concepts, as both aim to make the training data contain variations that enable transfer to the real-world domain. While domain randomization modifies the mapping from a scene s_t to an observation o_t by varying properties in the rendering engine,

data augmentation transforms an already rendered observation o_t to a new observation o'_t. Data augmentation is described in more detail in Section 3.5.
The following sections describe the domain randomization techniques used in this thesis and the reasoning behind their selection.

3.4.1 Object Texture Randomization

One of the most used domain randomization techniques is the randomization of visual appearances. For many applications, it is desirable that the trained algorithm is agnostic to the color or texture of the objects of interest. The shape of an object is often the same in different domains, but the color or texture can vary significantly, even from one instance to another. For example, two cups might have the same shape, but could have different colors and surface textures. By randomizing the objects texture and color, the network learns to focus on the invariant features of the object rather than the potentially varying appearance.

State of the Art

Randomizing object appearances by applying different textures has been a core randomization technique in many publications. One of the first publications to apply texture randomization for sim-to-real transfer was by Tobin et al. [10]. They trained a CNN to predict the position of an object of interest in a 2D RGB image. By varying the textures of the objects, the background, as well as the lighting conditions, they were able to successfully use a model trained exclusively in simulation to predict the position of the object in real-world observations. They used a random choice of solid RGB colors, a gradient between two RGB colors, and a checkerboard pattern with two colors as textures for the objects. In the same year Sadeghi and Levine [12] developed a method to train a controller for an autonomous indoor drone by simulating corridors with random obstacles. Instead of relying on manually defined textures, they used a set of predefined material textures included in the *Blender* rendering engine, such as wood, metal, and stone. They achieved successful transfer to real-world domains while exclusively training with simulated data.
Texture randomization has since been identified as a key component for sim-to-real transfer and has been used in many publications [19], [28], [29], [49]. It has been shown that more complex textures lead to better generalization to unseen textures and into the real world. Borrego et al. [29] compared the performance of different randomization textures used for training an object category detection model. They evaluated four types of textures: flat colors, gradients of colors, chess pattern, and Perlin noise. Perlin noise is a type of gradient noise that is often used in computer graphics to generate textures and realistic terrains. It is generated by summing up multiple octaves of gradient noise, where each octave has a different frequency and amplitude [18]. In an ablation study on different textures [29] found that removing Perlin noise from the training set resulted in a performance decrease by a factor of 3. They found the following order of importance for the textures: Perlin noise, chess pattern, gradients, and flat colors. This is in line with the intuition that more complex object textures lead to better generalization. Instead of relying on self-generated textures, several publications have used publicly available textures to train their models to successfully bridge the sim-to-real gap [11], [30], [49].

Implementation

Based on the results of previous research, object texture randomization in this thesis is implemented using a combination of Perlin noise textures, a diverse set of textures from public websites, as well as a random selection of images from the *Common Objects in Context* (COCO) dataset [50]. To increase the variety of the appearances, the used texture-images are randomly scaled, shifted and rotated during the randomization.

- Perlin Noise textures: The Perlin noise textures are implemented using images of black and white Perlin noise generated with the online generator *Noise Maker* [51]. The generator allows to set the size of the image, as well as various parameters for the noise generation. Although it would be potentially beneficial to have the Perlin noise scale between two random RGB colors, due to limitations in *Omniverse Replicator*, only a single color could be used. To create colored version of the Perlin textures, they are multiplied with a random color during the scene randomization step. Examples of Perlin noise textures can be seen in Figure 3.8.

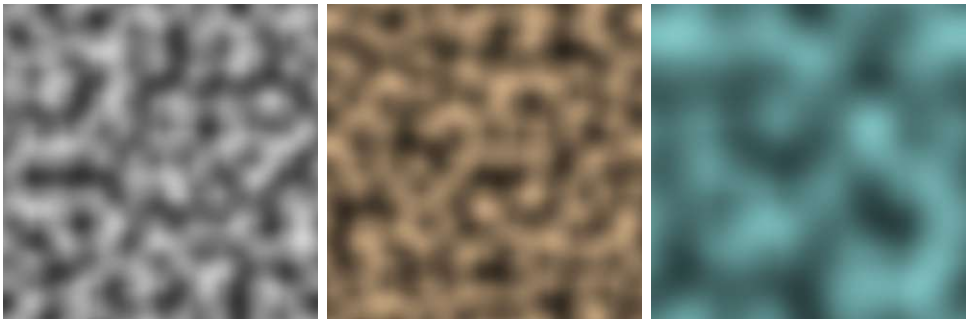


Figure 3.8: Examples of colored Perlin noise textures used for object texture randomization.

- Material textures: For the set of material textures, 90 diverse, openly available textures were obtained from various websites. The textures were selected to cover a wide range of materials, surfaces and colors. Compared to the alternatively used COCO images, it is evident that because they are intended for the use as textures, they offer superior tiling properties when scaled. Figure 3.9 shows examples of the material textures used. The used textures were obtained from the following websites: *Free Stock Textures* [52], *Texture Ninja* [53], *3D TEXTURES* [54], *Poly Haven* [55].

Figure 3.9: Examples of material textures used for object texture randomization.

- COCO textures: The COCO dataset is a large scale image dataset containing over 200.000 images of 80 different object categories in a variety of scenes [50]. The dataset is widely used for object detection and segmentation tasks. The high variety of objects and real-world scenes also makes it attractive to use in a sim-to-real setting. Eversberg and Lambrecht [11] have shown the effectiveness of using COCO images as background images in a domain randomization setting. This idea is extended in this thesis by using COCO images as textures for objects as well. A random subset of 5000 images from the COCO dataset are used as textures for object materials. Figure 3.10 shows examples of COCO images.

Figure 3.10: Examples of COCO images used for object texture randomization.

Texture Sampling

Having a large variety of different textures is important for good generalization ability of the trained model. In our case, the number of available textures in the three different categories is largely different. While there are only 90 material textures, there are 5000 COCO images and 20 pre-generated Perlin noise textures. If we were to sample from all categories in equal quantities, the textures of categories with fewer available samples would be re-used far more often. To ensure a wider range of different textures in the dataset, the textures are sampled in a way that textures with more available samples are more likely to be selected.
Because weighted sampling of textures is not directly supported by the used version of *Nvidia Replicator*, a method resembling a weighted sampling scheme is implemented. For each texture category multiple materials are created and sampled uniformly. By varying the number of materials for each category, the probability of a texture being selected can be controlled. Uniform sampling from a set of materials for which the relationship $n_{text} < n_{noise} < n_{COCO}$ holds, will result in a distribution of textures that is close to the desired weighted sampling scheme. Because the texture and COCO materials can be treated similarly, they are grouped into one category. The distribution within the texture category is controlled by sampling a subset of the available COCO textures, so the relationship between texture and COCO files is $n_{text}/n_{COCO} = 1/5$. The ratio between the Perlin noise textures and the image texture category is set to $n_{noise}/n_{img} = 2/3$. Resulting in a relationship of $n_{text}/n_{noise}/n_{COCO} \approx 1/1.5/5$.
Figure 3.11 shows a cuboid with material textures from each of the used categories. From left to right: noise, material textures, and a COCO image.

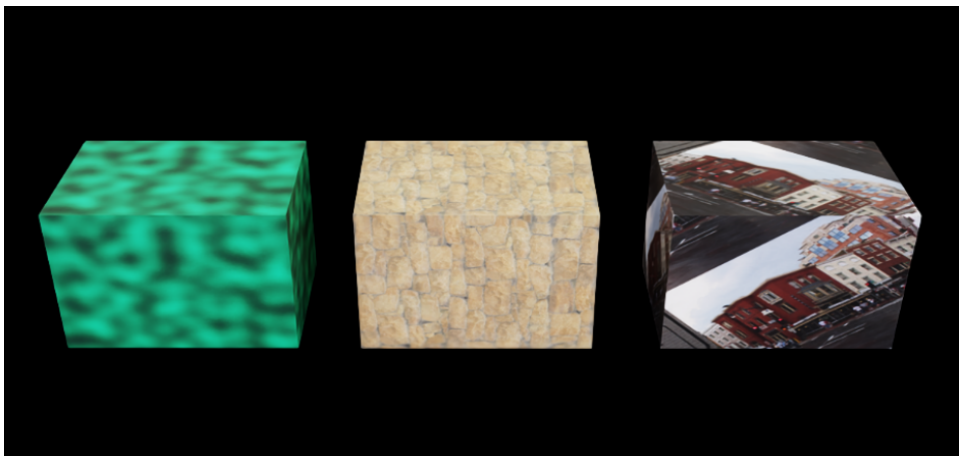


Figure 3.11: A cuboid with different material textures.

3.4.2 Surface Material Randomization

Besides its texture, there are many other properties of the surface material that define the appearance of an object. These include its surface roughness, reflectivity, and specularity. In the context of simulation, material properties refer to parameters in the rendering engine that define how the surface of a material interacts with light, such as how shiny, rough, or metallic it appears. By randomizing the material properties of the objects of interest, the goal is to make the covered state space in the recorded scenes more diverse and increase the generalization ability of the trained model.

State of the Art

When it comes to randomizing material properties, most publications give no information on whether randomization of material properties is used or the authors only mention that they used a random selection of materials, without giving more details. This could be due to the fact that the used simulators in the domain randomization literature, are typically of lower fidelity. This often makes simulating complex surface material properties impossible. It is argued that when randomizing the visual properties of the scene, high accuracy rendering, like ray-tracing, is not required to achieve successful domain transfer. We would argue however, that a more complex rendering system might be able to capture a broader visual-state space and therefore lead to better generalization.

Implementation

In this thesis, the used *Omniverse Replicator* plugin, allows access to many properties of the created materials in the scene. The following material properties are randomized during data generation:

- Roughness: Defines how smooth or rough the surface of an object appears. A roughness of 0 creates a very reflective, mirror-like surface. The higher the roughness value of a material, the less reflective the surface will get.

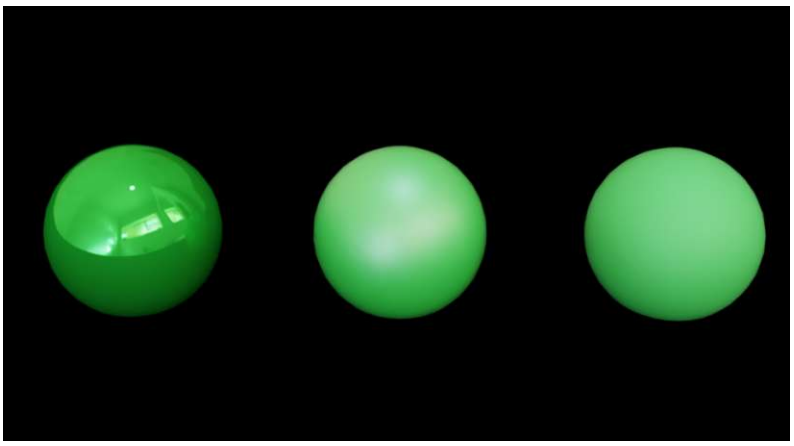


Figure 3.12: A sphere with different material properties. Left: Metallic, low roughness and high specularity. Middle: Non-metallic, medium roughness, high specularity. Right: Non-metallic, high roughness, low specularity.

- Specularity: The specularity property of a material defines how the light is reflected from the surface. A specular value of 0 means that light is scattered in all directions, while a value of 1 means that reflections are highly directional.
- Metallic: Defines the dielectric constant of a material. A value of 0 means that the material is not conductive, while a value of 1 means that the material is metallic. This property modifies the way light interacts with the surface of the object. Metallic materials reflect light differently than non-metallic materials and are rendered differently.

Figure 3.12 shows a sphere with different material properties. It can be seen, that different values for roughness, specularity and the dielectric constant significantly influence the appearance of the object.

3.4.3 Background Randomization

The background of a scene is one of the prime examples of an attribute that is not task relevant and is not constant in most real world applications. Since the credo of domain randomization is to randomize every aspect of the simulation that has no grounding in the real world, randomization of the background is often used in the literature [10], [17], [56].

State of the Art

Randomization of the background can be found in most vision based domain randomization publications. There are generally four types of background randomization techniques found in the literature:

- Solid color backgrounds: The simplest form of background randomization is to use a randomly selected solid color to fill the background plane [10], [28].
- Noise textures: Different noise textures, similar to the textures used for objects of interest were also investigated as possible scene background [17], [56].

- Random images: The most widely used method for background randomization is to use random real-world images from large-scale datasets, such as the COCO dataset [50] or the NYU-depth dataset [57]. This has been used with great success in many publications [11], [19], [49].
- Domain-specific images: In some cases, domain-specific images are used as background images. This can be reasonable in cases where the scene background can be estimated beforehand, e.g. in an industrial workplace scenario. It has been shown however, that this not necessarily leads to better transfer compared to random images [11], [58].

Implementation

Due to the great success in previous work and the simplicity of the implementation, random images from the COCO dataset are used as background images in this thesis. The images are randomly scaled, shifted, and rotated during data generation to increase the variety of the backgrounds. Additionally, the material properties, such as roughness and specularity of the floor material are randomized. Figure 3.13 shows examples of two backgrounds with different textures from the COCO dataset and different surface properties.

Figure 3.13: Examples of backgrounds with different textures and material properties. Left: high roughness, low specularity. Right: Low roughness, high specularity.

3.4.4 Lighting Randomization

The lighting conditions in a scene can have a significant impact on the appearance of objects in the scene. Different lighting conditions can lead to different shadows, reflections, and highlights on the objects. This can make generalizing to unseen scenes challenging. Randomizing the lighting conditions in the scene is another widely used technique in domain randomization literature [10], [12].

State of the Art

Previously lighting randomization has often been implemented by randomizing the number of light sources, their position, intensity, and color [10], [12], [49]. The light sources are mostly modeled as point lights and surface lights, which are simple to implement and often the only light sources available in the used rendering engines. Lighting conditions in the real world, however, can be much more complex than what can be modeled using

Figure 3.14: A scene lit using a HDRI dome light. The image is used as a light source and the reflections of the light sources are visible on the objects in the scene.

such basic light sources. Additionally, modern rendering engines are capable of simulating much more complex lighting conditions without a significant increase in computation time. Image-based lighting (IBL) is a popular method for simulating complex lighting conditions, that has its roots in modern visual effects production and game design [59]. In an IBL system, 360 degree images of an environment are used to light scenes. The images are typically High Dynamic Range images, that are captured in real world scenes. Those images naturally capture various types of light sources, their position, intensity, color, and direction, as well as the reflections of the light sources on the objects in the scene. Figure 3.14 shows an example of a scene lit using an HDRI dome light. This lighting technique has been used successfully for domain randomization in previous publications [11], [60]. It seems to be a promising technique for domain randomization and sim-to-real transfer since the lighting conditions resemble real-world lighting conditions much more closely than a random arrangement of point-light sources. Figure 3.15 shows a simple scene with different lighting conditions.

Implementation

To randomize lighting conditions the *Dome Light* feature of the *Omniverse* rendering engine is used. It allows to use HDR images as light sources in the scene, that are simulated as a dome in infinite distance around the scene.
For this thesis a set of 40 publicly available HDR images is used as light source textures. Since the use case of the developed algorithm is indoors only, *HDRI* images of indoor scenes are used. The used HDR images were obtained from *Poly Haven* [55] and *HDRI Skies* [61]. During scene generation, the HDR images are randomly sampled and applied to the dome light. Additionally, the intensity of the light source and the rotation of the dome light are randomized.

3.4.5 Camera Randomization

The camera is the device that renders the scene s_t into an observation o_t. The parameters of this mapping have a significant influence on the appearance of objects in the observation.

(a) Scene with individual lights (b) Scene with HDR dome light

Figure 3.15: A simple scene using different lighting methods.

This mapping is determined by the camera position, orientation, and intrinsic camera parameters.
The camera position and orientation are randomized to ensure the trained model is independent of the viewpoint of the camera during evaluation. This follows from the fact that viewpoint invariance is a key property of a vision model that shall be used in real-world scenarios. The randomization of intrinsic camera parameters, such as focal length, field of view, and aperture, is done to account for differences between the modeled camera in simulation and the camera used for real-world data collection.
No camera noise is added during the data generation process. This is instead done for each sample during the data augmentation during training.

State of the Art

Randomization of camera positions is a widely used technique in domain randomization with visual observations [10], [11], [17], [24], [30], [49]. The camera position is usually randomized by sampling a random position within a box or sphere in front of the workspace and the camera is oriented at the workspace center.
Randomization of intrinsic camera parameters is less common in the literature. This is likely because intrinsic camera parameters are usually assumed to be fixed and constant between simulation and reality. Nevertheless, publications have shown that randomizing intrinsic camera parameters can lead to better generalization to real-world data and unseen camera types [10], [62].

Implementation

The camera randomizations are implemented as follows:

- Camera position: The camera position is randomly sampled within a box of configurable size, in front of the workspace. The camera orientation is calculated using a direction vector that is directed to a position sampled uniformly around the center of the workspace. This additional randomization of the look-at position is done because during deployment in the real world, the camera may also not be aimed directly at the center of the workspace. This is especially true for applications in which the camera is mounted on a mobile robot, that moves through its environment. While the workspace viewed from the robot system is fixed in front of the robot, the viewpoint of the camera on the observed scene can vary significantly.

- Intrinsic camera parameters: Although the camera model used in simulation mimics the camera used for data acquisition, a discrepancy between the real active-stereo acquisition system and the modeled camera is to be expected. To account for these differences, the intrinsic camera parameters are varied during data generation. The following parameters are randomized to ±10% of the original value used by the provided model of the *D435* camera:
  - Focal length: The focal length of the camera lens.
  - Focus distance: The distance between the focus plane and the camera.
  - F stop: The ratio of the focal length to the diameter of the aperture. Influences the depth of field.
  - Aperture: The size of the aperture of the camera lens.
  - Fisheye polynomial parameters: The parameters of the polynomial that describe the fisheye distortion of the camera lens.

3.4.6 Distractor Objects

In many real-world scenarios, the objects of interest are not the only objects present in the scene. In household environments, for example, there are often many objects that are not relevant to the task at hand. Previous research has shown that if a model ought to be resilient against unrelated objects in the scene, it must be trained with distractor objects [10], [11], [30].

State of the Art

While the necessity of training with distractor objects, to enable resilience against unrelated objects in the scene, is widely accepted, different types of distractor objects are used in the literature:

- Basic geometric objects: The simplest form of distractor objects are basic geometric shapes, such as cubes, spheres, and cylinders. Previous publications have shown, that this simple types of distractor objects provide good transfer to real-world settings [10], [11], [30].
- Random household objects: Other publications have used objects from the YCB object dataset [63] as distractor objects. The YCB dataset contains a large variety of household objects, such as cups, boxes, and toys. Altough this would be a good fit for household scenarios, no performance benefit has been shown compared to using random geometric shapes [11].
- Domain-specific distractor objects: For cases where the deployment domain is known beforehand, domain-specific distractor objects can be used. It has been shown however, that this also not necessarily leads to better transfer compared to random shapes [11], [56].

Implementation

For incorporating distractor objects into the synthetic data generation process, we use basic geometric shapes. We chose this simple method of adding distractor objects, because

there are no additional 3D objects to be acquired and previous research has not shown any performance benefit of using more complex distractor objects.
The distractor objects are randomly sampled from a set of basic geometric shapes, which are provided as shape primitives in *Omniverse Isaac Sim*, such as cuboids, spheres, cylinders, and toroids. To texturize the distractor objects, a random set of textures is sampled from the COCO dataset, similar to the objects of interest. Because, in our case, the texture of the object of interest is not relevant for the tasks to solve, the potential overlap between the textures of the distractor objects and the object of interest should not be a problem.
During each scene generation step, the size and position of the distractor objects are randomized. The distractors are scattered on a plane, so that they do not overlap with the task relevant objects, and also do not overlap with the area where the place action is performed.
The size of the distractor objects is randomized between 0.05 m to 0.25 m. In the used version of *Omniverse Replicator*, it is not possible to dynamically change the number of distractor objects for each scene. As a workaround, the number of distractor objects is fixed to 8 and the size and position of the plane on which they are placed is varied. The size of the plane is randomized between 1 and 2.5 times the size of the workspace. The z-position of the plane is randomized between −0.05 m to 0.05 m. This creates scenes with different visible heights of the distractor objects, as well as scenes where no distractor objects are visible at all, because the plane is moved below the workspace. Figure 3.16 shows two empty scenes with distractor objects. The gray area is the captured workspace, the scatter plane of the distractor objects is not visible.

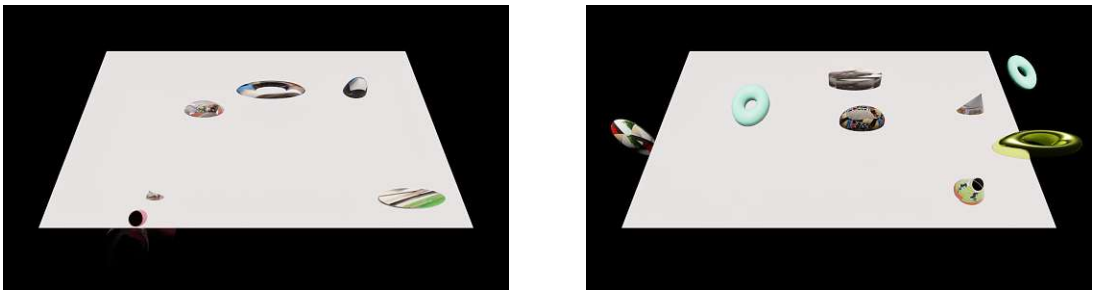


Figure 3.16: An empty scene with distractor objects. The grey area is the captured workspace.

Summary

In conclusion, the domain randomization techniques outlined in this chapter provide ways to overcome different aspects of the sim-to-real gap. By systematically introducing variability in different facets of the simulation, these methods should enable the trained models to generalize effectively to real-world scenarios. The combination of texture, material, lighting, background, and camera randomizations ensures coverage of a large portion of the visual state space.

3.5 Data Augmentation

Data augmentation describes methods that modify samples during training of algorithms, to artificially increase the size of a given dataset. During training, the samples are modified in a way that does not change the semantics of the sample but changes properties of the sample that are not task relevant. For example, consider a training sample consisting of an image of a cat and a label, that identifies the images as *"cat"*. By applying data augmentation techniques to the image, such as flipping the image, adding random noise, or changing the brightness, multiple variations of the same image can be created. The label of the image, *"cat"*, remains unchanged, as the semantic meaning of the image is still the same. Thereby, it enriches a smaller dataset with new labeled elements, created from the existing ones, improving the networks robustness and generalization. Some data augmentation techniques popular in computer vision tasks are color noise, rotation, blurring, and many others. All of which do not alter the semantic label of the image. For example, a dog is still a dog even if it is flipped horizontally or if only part of the dog is visible in a crop.

Data Augmentation to Bridge the Sim-to-Real Gap

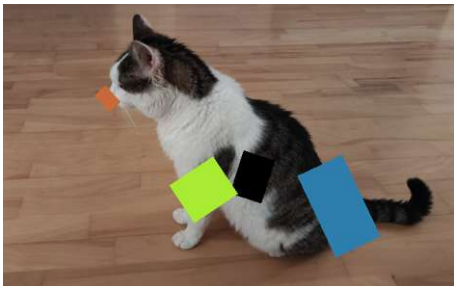
Data augmentation techniques can not only be used to improve the generalization of general-purpose computer vision models but have also shown to be essential in bridging the sim-to-real gap[25].
As discussed above, the goal of domain randomization is to facilitate generalization to real-world data by varying the environment environment properties, that may vary during a model's deployment. Data augmentation can be seen as an extension of this concept applied to each sample during model training, rather than during data generation. Just like domain randomization, data augmentation techniques can be used to modify aspects of the training data that are not constant in the real world. While domain randomization modifies the mapping from a scene state s_t to an observation o_t, data augmentation transforms an already rendered observation o_t to a new observation o'_t. Naturally, this limits the applicability of data augmentation to the properties of the observations, which can be altered after a scene has been rendered. For example, the color of the whole observation can be changed, noise can be added, or the observation can be rotated, but the position of individual objects or light sources in the observation cannot be altered after rendering.
Figure 3.17 shows examples of popular data augmentation techniques, including the addition of color noise, Gaussian blur, and cutouts.

Augmentation of Depth Data

In contrast to traditional computer vision models that often work with color images, robotic perception often includes depth information. Depth is an essential component for control and decision making in the robotic context, where spatial relationships are crucial [64]. Depth data, acquired by real-world sensors, is often noisy and contains holes where the depth measurement system failed to capture data, e.g. for transparent objects or due to reflections. Synthetic depth data in comparison is generated using idealized sensor models, is generally noise free and offers perfect capturing of the viewed area. This

(a) Unaltered image

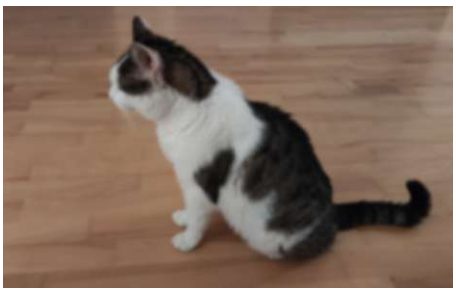
(b) Image with Gaussian color noise



(c) Image with random cutouts

(d) Image with random color transformation

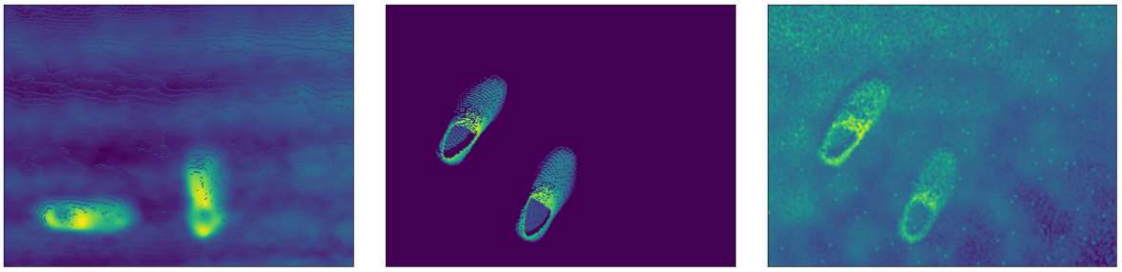
(e) Image with salt and pepper noise



(f) Image with Gaussian blur

Figure 3.17: Image of a cat with various augmentations applied.

is a good example of the sim-to-real gap that can significantly hinder the performance of models when transferring from synthetic depth data to real-world data.
Adding perturbations to simulated depth data can help to make it more similar to real-world data and thereby improve transfer performance of trained models. Many publications show that augmenting synthetic depth data with various forms of noise is essential to the performance of models trained on synthetic data in the real world [24]–[26]. To illustrate this point, Figure 3.18 shows an example of a real-world heightmap, computed from a captured pointcloud and a heightmap from a simulated pointcloud, with and without augmentations. It can be seen that the simulated heightmap(middle) is much more uniform and lacks the noise and angular offset present in the real-world data(left). The right image shows the same simulated heightmap with *Gaussian noise* and *Perlin noise* added to the depth values, as well as a transformation around the x- and y-axis applied. In addition to these noise based augmentations, Pashevich et al. [23] show that applying traditional image augmentation techniques such as transformations, cutout, and white noise to synthetic depth data can significantly improve real-world performance of



(a) Real world data (b) Simulated data (c) Simulated data with augmentations

Figure 3.18: Heightmaps generated from real-world and simulated pointclouds. For the simulated data, the same pointcloud is shown, left is unaltered, whereas the right has *Gaussian noise*, *Perlin noise* and a plane offset applied.

models. In this thesis, a combination of color and depth augmentations are applied to expose the trained models to a diverse set of variations, improving their generalization capabilities. The following sections provide a detailed description of all augmentation techniques applied in this thesis. Some augmentations only modify one modality, like *Color Noise* or *Plane Offset*, while others are applied to both modalities in conjunction or separately. The following sections describe the augmentations applied in this thesis and their implementation.

3.5.1 Color Noise

As one of the two augmentations already present in the original Transporter Networks publication [9], color noise is applied to the training images. Zero-mean Gaussian noise is added to the RGB channels of the input observations for each sample. Applying noise during training can help to make the model more robust against noise in the real-world data, caused by sensor noise or other physical imperfections.
The standard deviation of the noise σ_{rgb} is configurable via the training configuration *json* file. With the given sigma, one sample for each channel of each pixel is drawn from a normal distribution with zero-mean and standard deviation σ_{rgb}. Figure 3.17b shows an example of the *Color Noise* augmentation applied to an image.
The modified pixel intensities I_{rgb}, one for each channel, are calculated as follows:

$$\Delta_{rgb} \sim \mathcal{N}(0, \sigma_{rgb}^2)$$
$$I_{rgb} = I_{rgb} + \Delta_{rgb}$$

3.5.2 Color Transformation

Color space transformations are a common augmentation technique in image-based deep learning. In the context of sim-to-real transfer, the intuition is that these transformations would make the model more robust to changes in lighting conditions and object color. For this thesis the *ColorJitter* and *RandomGrayscale* augmentation modules from the publicly available *Torchvision* package [65] are used. The input to this augmentation module is the RGB image that is created from the top-down view of the input pointcloud, as described

in Section 3.1.
The *ColorJitter* module applies random brightness, contrast, saturation, and hue variations to the input image. The *RandomGrayscale* module converts the input image to grayscale with a probability of p_{gray}. An example of the *ColorJitter* augmentation is shown in Figure 3.17d.
The following parameters can be configured for the color transformation augmentation. The given values define the limits in which the corresponding factors are sampled from a uniform distribution. For hue, the sampled value represents the change of hue on a normalized scale, where 0.5 represents a rotation of 180° around the *Hue, Saturation, Value - HSV* color circle.

- Brightness variation
- Contrast variation
- Saturation variation
- Hue variation

3.5.3 Gaussian Blur

Smoothing of images by convolution with a Gaussian filter kernel is a popular technique in computer vision. In data augmentation pipelines, *Gaussian blur* can be used to simulate the effect of out-of-focus camera images, which is a form of data degradation that may often occur in real images. A study by Dodge and Karam [66] shows that many classification networs are especially sensitive to blur and color noise.
In this thesis, the *GaussianBlur* augmentation module from the publicly available *Torchvision* package [65] is used. Smoothing is applied to the stacked observation image that contains color, as well as depth channels. The color and depth information need to be blurred in unison to avoid misalignment of the color and depth information. Since Gaussian smoothing results in a loss of information, which can be especially problematic with depth data, the standard deviation and kernel size of the Gaussian filter are kept relatively small. Furthermore the application probability is set to below 50%, so that the augmentation is not applied to every training sample.
It is also important to note that the *Gaussian Blur* augmentation is applied before augmentations like *Salt-and-Pepper noise*, because smoothing the image after adding noise would reduce the impact of noise to the image.
The parameters for the *Gaussian Blur* augmentation are:

- Kernel size: The kernel size of the Gaussian filter.
- Sigma: The standard deviation of the Gaussian filter. This value can either be set as a constant or as a range. If given as a range, the actual value for each sample is drawn from a uniform distribution in the range.
- Apply probability: The probability with which to apply the augmentation.

3.5.4 Salt-and-Pepper Noise

Salt-and-Pepper noise is a form of noise where the values of randomly selected pixels are set to either the maximum or minimum value of the available range. The effectiveness of *Salt-and-Pepper noise* in the sim-to-real context has been shown for color images [30] as well as for depth images [23]. It is therefore added to the set of augmentation techniques

in this thesis.
Since the color- and depth channels can be treated independently with this augmentation, the parameters for the *Salt-and-Pepper noise* are also set independently for the color- and depth channels.
The configurable parameters are: the probability of a pixel being set to the minimum or maximum value p_{sp}, the ratio of salt to pepper pixels r_{sp}, and the probability with which to apply the augmentation p_{apply}.

3.5.5 Cutouts

Cutouts are a form of data augmentation where the pixel-values in a region of the image are set to a constant value. The technique has initially been used as a regularization technique applied in feature layers of deep convolutional neural networks. Research has shown that masking the network input can have an equal regularization effect and improve generalization [67].
In the field of sim-to-real transfer research, *Cutouts* have been shown to have mixed results. In the work of Pashevich et al. [23] *Cutouts* did improve the performance of models trained on synthetic depth data in real world test data. While Horváth et al. [30] showed negative effects of *Cutouts* on the model performance.
Because the evaluation data in this thesis is obtained using a stereo depth camera, where occlusions and missing parts of the scene are common, cutouts are included in the set of evaluated augmentation techniques.
The augmentation is implemented as follows.

- Between 2 and n rectangles are sampled.
- The size and aspect ratio of the rectangles are sampled uniformly from the given range.
- The rectangles are placed in a random location in the workspace, parallel to the xy-plane. Locations where the *Cutout* rectangle would overlap with the pick-and-place positions are not considered valid.
- With a probability of $p_{drop} = 0.5$ the points are cut out from the pointcloud and removed. With a probability of $1 - p_{drop}$ the rectangles are filled with a random color and set to a random height between the minimum and maximum workspace-height.

Additionally, the apply probability p_{apply} can be set to control the probability with which the augmentation is applied. Previous research showed that cutouts have a far better effect if they are not applied to every sample during training [67].

3.5.6 Depth Noise

Regardless of the technology used to capture depth data, real-world depth measurements will have a significant amount of measurement noise and depth uncertainty. In the sim-to-real literature, the consensus is that adding noise to simulated depth data is essential for successful transfer to real-world data [24], [26]. The kind of noise varies between publications, but they all show that training on noise-free depth data can greatly hinder transfer performance.

- Mahler et al. [24] use a combination of depth-proportional noise and Gaussian noise to simulate measurement noise in depth data.

- Thalhammer et al. [26] developed a sophisticated noise model, using a combination of occlusion masks, Gaussian smoothing, depth-proportional noise, and Perlin noise. They also show that only using Perlin noise already is a significant improvement over noise-free depth data.
- Pashevich et al. [23] have found that a combination of augmentations including white noise, salt noise, and depth scaling leads to the best performance in their evaluation with real-world data.
- Hagelskjaer and Buch [25] use a empirically derived, distance dependent Gaussian noise model and point dropouts to simulate noise in depth data.

In this thesis, a combination of the above methods is used to add noise to pointclouds generated in simulation.

- Gaussian noise: Gaussian noise is added to the xyz-point locations in the pointcloud. It is sampled from a normal distribution with zero mean and standard deviation σ_{depth}.
- Perlin noise: Perlin noise is added to the depth values of the pixels in the projected depth image. The Perlin noise is generated using the *Perlin-numpy* python package [68]. The number of octaves for noise generation is fixed to 2 and the persistence is set to 0.5. The number of periods for the Perlin noise generation is randomly sampled to be 4, 8 or 10. These values are fixed because they must be compatible with the desired output shape in this library. The only configurable parameters for the Perlin noise is the scaling factor s_{perlin} which is multiplied with the amplitude of the generated noise, as well as the application probability p_{apply}. Perlin noise is applied separately to the color and the depth channel.

3.5.7 Depth Scaling

Depth scaling is an augmentation technique, where the depth values in a height map or depth image are multiplied by a random factor, close to 1. This can help to make the trained model more robust to changes in the depth values, i.e. when the depth sensor is moved further away from the object or when the object is moved closer to the sensor. This augmentation has been shown to be a useful augmentation technique in the sim-to-real setting with depth images [23]. In this thesis, the depth scaling augmentation is applied to the values of the depth image, created by projecting the input pointcloud. The multiplicative scaling factor is sampled uniformly from the range $[1 - \delta_{scale}, 1 + \delta_{scale}]$.

3.5.8 Plane Offset

The plane offset augmentation is a simple augmentation technique that rotates the input pointcloud by a random amount around the x- and y-axis and adds a random z-offset to all points. This augmentation is not found in the literature, but is added to account for the fact that the real-world pointclouds are not perfectly aligned with the xy-plane, due to imperfections in the pointcloud registration process. The transformation is applied to all pointcloud points using a transformation matrix, that contains the rotation and translation components. The rotation angles θ_x and θ_y are sampled uniformly from the range $[-\delta_{angle}, \delta_{angle}]$ and the z-offset is sampled uniformly from the range $[-z_{max}, z_{max}]$.

It is important to note that this augmentation changes the label of the input pointcloud, since the objects of interest are moved in world coordinates, which is usually a problem in data augmentation. In this case, however, the target labels are world-poses, which can be transformed with the same transformation and thereby remain valid.

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_y = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The combined transformation matrix T is obtained by multiplying R_x and R_y and adding the translation component z_{offset}:

$$\mathbf{T}_z = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ \sin(\theta_x)\sin(\theta_y) & \cos(\theta_x) & \cos(\theta_x)\sin(\theta_y) & 0 \\ -\cos(\theta_x)\sin(\theta_y) & \sin(\theta_x) & \cos(\theta_x)\cos(\theta_y) & z_{\text{offset}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.5.9 Pointcloud Downsampling

Pointcloud resolution can vary significantly between different sensors and between different configurations of the same sensor. To be able to vary all important aspects of the training data that may vary during deployment, pointcloud downsampling is added as an augmentation technique. The downsampling is implemented using the *random_downsample* function from the *Open3D* python package [47], that randomly removes points from the input pointcloud, so that the resulting pointcloud has fewer points by a factor of r_{ds}. The downsampling factor, used for each training sample, is sampled uniformly between a minimum and maximum downsampling factor. The maximum value is usually 1, which means that no downsampling is applied.

4 Experiments and Results

To evaluate the effectiveness of the methods developed in this thesis, two experiments are designed and evaluated using real-world data. This chapter describes the experiments, the metrics used to evaluate the performance of the networks, and the results of the experiments.

4.1 Experiments

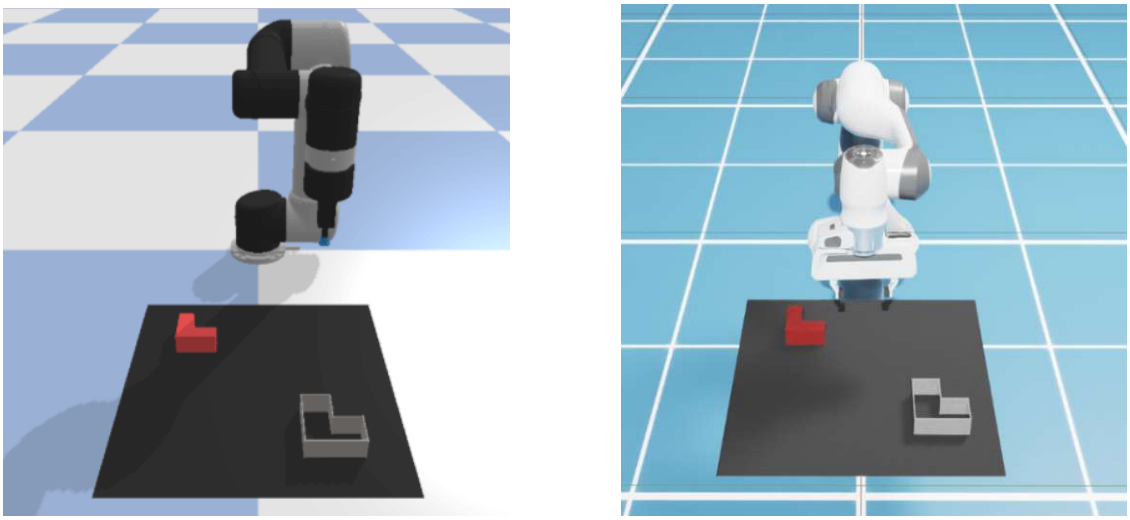
In household- and assistance robotics, the task of picking up and placing objects in unknown environments is one of the most important tasks for robots to perform [2]. What makes this especially challenging, is that environment conditions as well as the appearance of the objects to pick can vary significantly, even within the same object category. With this challenging environment in mind, we developed two tasks for evaluation. The focus was on creating tasks that are of real-world significance, while requiring only one pick-and-place step. We further focused on tasks, that do not require high levels of precision, but rather require the robot to be able to generalize to new objects and environments. In addition to the two novel tasks, the *Block insertion* task from the original Transporter Networks publication was recreated in *Nvidia Omniverse*. This is to validate that the applied changes to the networks and the training procedure do not negatively effect the performance.

4.1.1 Block Insertion Task

This task is one of the evaluation tasks used in the original Transporter Networks publication [9]. The goal of the task is for the network to predict the pick-and-place poses that move an L-shaped block into a matching fixture on a virtual table. This task was recreated in the simulation environment used in this thesis(*Nvidia Isaac Sim*) and compared with the results of the original publication. For the recreation of the task, the assets found in the Github repository of the original publication [37] are imported into *Nvidia Isaac Sim*. The annotations for the pick-and-place poses are created using the same logic as in the original publication. For this, the ground-truth pose of the block is used to calculate the pick-pose by simply adding an offset to the geometrical center. The place-pose is calculated by adding the same offset to the center of the fixture.
Figure 4.1 shows the block insertion task from the original publication side by side with the recreated task in *Nvidia Isaac Sim*.

4.1.2 Shoe Sorting

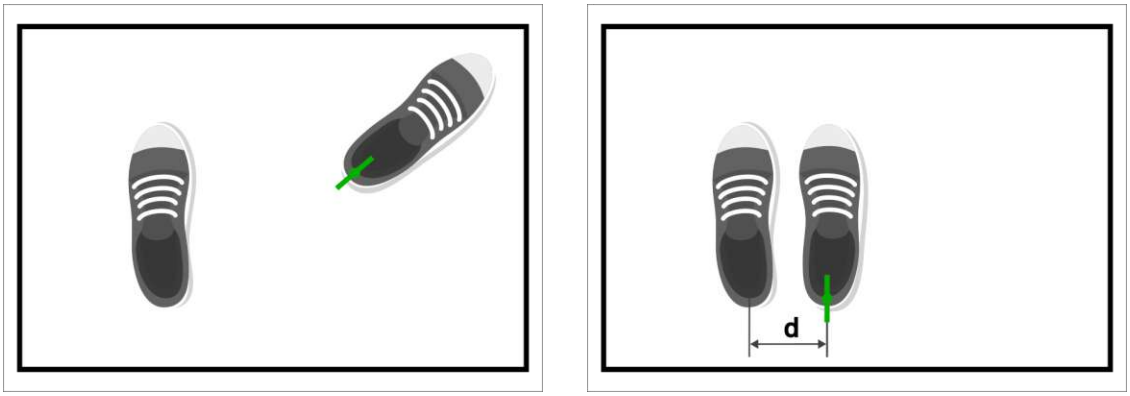
With the goal of developing a task, that is of real world significance for robots at home, we chose the task of sorting a pair of shoes. The task consists of a pair of shoes that are placed in a random orientation in the workspace. The goal of the task is to pick up one of



(a) Block insertion task from the original publication, modeled in *PyBullet* [9].

(b) Block insertion task recreated in *Nvidia Isaac Sim*.

Figure 4.1: Comparison of the block insertion task from the original publication and the recreated task in *Nvidia Isaac Sim*.



(a) Pick of the right shoe, at the rear center, inline with the shoe

(b) Placement of the right shoe next to the left shoe

Figure 4.2: Illustration of the shoe sorting task.

the shoes and place it in a predefined location next to the other shoe. This provides a relatively simple one-step task that can be used to validate the generalization abilities and transfer performance of the trained networks. The task dataset was created using freely available 3D models of shoes that are imported into the *Nvidia Isaac Sim* environment and rescaled to be of uniform size.
To create deterministic task behavior, we decided that only the right shoe of the pair would be picked up and placed in a predefined location next to the left shoe. The pick-location on the shoe is defined as the rear-center of the shoe, with the pick-angle being in-line with the shoe axis. This pose is chosen so that the shoe could be picked up using a two-finger gripper. The placement is done so that the shoes are placed next to each other, in parallel, with a distance d between the middle of the shoes. Figure 4.2 shows an illustration of the shoe sorting task.
This task poses several challenges to the network:

- Dataset size: Since the dataset was self-curated and created from freely available 3D models, the dataset size is relatively small, containing only 20 models in the training set.
- Generalization to unknown objects: The network has to generalize to new objects that are not part of the training dataset. This is especially challenging, due to the small dataset size.
- Imperfections of real-world data: Besides the generalization to unknown objects, the task requires the networks to successfully differentiate between the left and right shoe of the pair. Imperfections in the real-world depth data, like noise and occlusions, can make this differentiation difficult.

Model Import and Preparation

For creation of the task dataset, a set of freely available 3D models of shoes was sourced and downloaded from the internet. This was done manually, since no large-scale dataset of high quality 3D models of shoes could be located. A mixture of different types of shoes was selected, focusing mainly on sneakers, running shoes, and dress shoes. A combination of 3D scans and 3D models is used to create the dataset. The models have been sourced and downloaded from the following websites: *Sketchfab* [69], *Fab* [70], *CGTrader* [71] and *Scanned Objects by Google Research* [72]. In total, 22 models are used to create the training- and test-set. After the download, the meshes have been cleaned and aligned using *Meshlab* [42]. The aligned models have then been imported into *Nvidia Isaac Sim*, where they were rescaled to a unified size and a grasp point was manually annotated.

Dataset Generation

For creating datasets for this task, the set of available models is split into a training and a test set, with 20 and 2 models respectively. The test set contains only two models, to keep the training set as large as possible, since validation performance on the real world dataset is the main focus of this thesis. The models are randomly scaled during dataset generation. A new scale factor is sampled for each axis at every scene generation step. This leads to randomized, non-uniform scaling of the models, which is intended to increase the variety of shapes in the dataset. To generate the datasets, each pair of models is loaded into the

Figure 4.3: Examples of rendered, randomized scenes for the training dataset of the *Shoe sorting* task.

scene, the scale of the shoes is randomized, and the shoes are placed in a random position and orientation in the workspace. The workspace is defined as a rectangular area with size 1.00 m × 0.75 m at height 0 m.
The annotations for the dataset are created as follows: For the pick position, the position of the manually annotated grasp point on the right shoe is recorded. For the place position, the expected position of the grasp-point is calculated when the shoe is placed in the desired location next to the stationary, left shoe. The distance d between the grasp points of the shoes is set to $d = 15.00$ cm. The distance between the grasp points, instead of the inner sides of the shoes was chosen, to make the distance more consistent across different shoe models and scaling factors.
The final dataset contains a total of 6000 pick-and-place poses. 300 demonstrations for each shoe model. The test dataset contains 50 demonstrations for each of the two shoe models in the test set, resulting in a total of 100 demonstrations. For each scene in the training dataset, five differently randomized scenes are rendered, resulting in a total of 30000 rendered scenes in the training dataset. During each rendering step, various randomizations, described in Section 3.4 are applied to the scene. For the test dataset, 10 scenes are rendered for each demonstration, resulting in a total of 1000 rendered scenes in the test dataset. Figure 4.3 shows some examples of the rendered scenes used for the training dataset of the *Shoe sorting* task.

Parameter Selection for the Dataset Generation

As mentioned above, the shoe models are randomly scaled during dataset generation. Initially, the range of the scale factors was set to a range that was assumed to result in a realistic size range for the shoes. However, during the ablation study for the augmentations, it was found that the network performance was not particularly good on the real-world data.
Analysis as to why this is the case indicated that it might be due to the selected values for the scale factors. The previously selected values for the scale factors lead to a large distribution difference between the sizes present in the training dataset and the real world sizes of the shoes. Based on the analysis, the scale factors were adjusted to a range that is closer to the shoes used in the real world dataset. To illustrate this, Figure 4.4 shows the histogram of the highest 100 z-values of each pointcloud for the real world pointclouds and the two versions of the training dataset. Before analyzing the pointclouds, they were downsampled to 2.00 cm voxel size and the highest 100 z-values were selected. This was done to reduce the influence of noise in the real-world pointclouds and to focus on the shoes.
The scale ranges for the two versions of the training dataset are given in Table 4.1. Looking at the histograms, it can be seen that the second version of the training dataset is much more evenly distributed and covers the range of the real-world dataset better than the first version. Nevertheless, the distribution of the real-world dataset is still shifted to larger values than the training dataset. This is likely due to the fact that the real-world pointclouds are not perfectly aligned and are noisy observations.

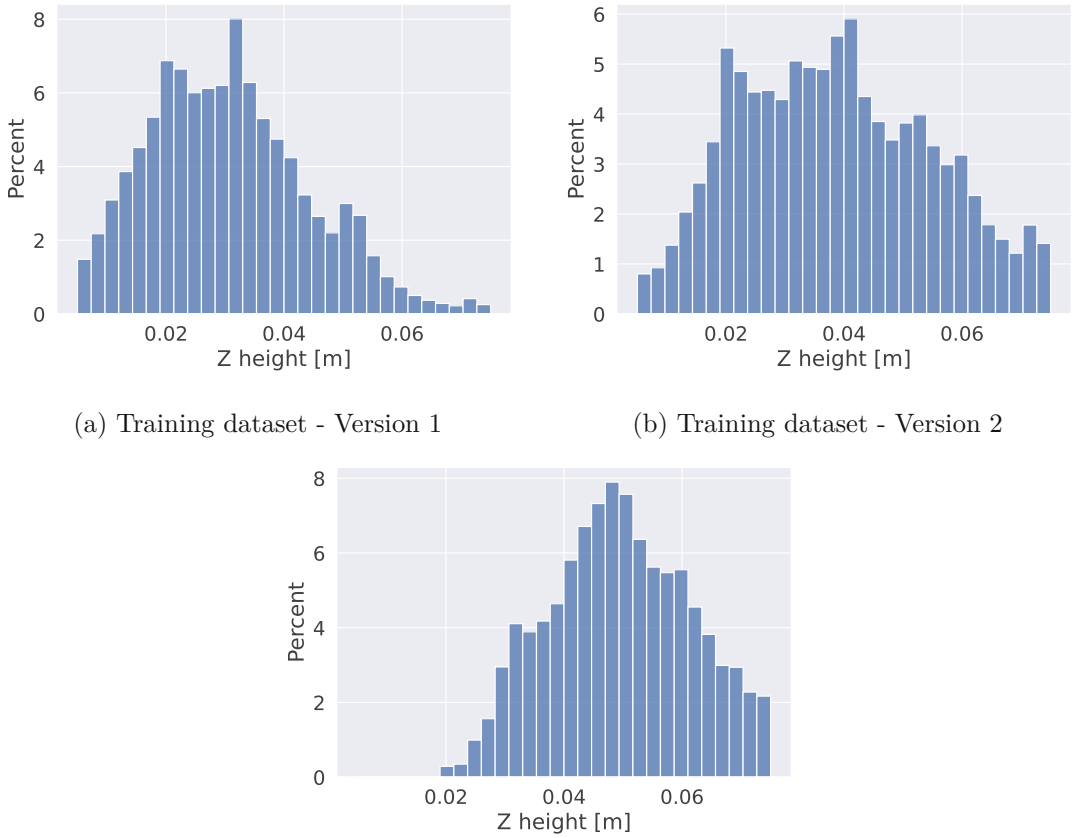
Since the distribution shift was only discovered during the ablation study, a set of the experiments was done using the original (Version 1) dataset, while later experiments were done using the adjusted dataset (Version 2). Each ablation study, however, was done using the same dataset version, so the results within one study still are comparable.

Table 4.1: Scale ranges for the shoe dataset

Dataset iteration	Scale range		
	x	y	z
Version 1	20.00 cm to 33.00 cm	11.25 cm to 17.25 cm	4.75 cm to 10.00 cm
Version 2	15.00 cm to 33.00 cm	8.25 cm to 15.00 cm	6.00 cm to 12.80 cm

Validation dataset

To validate the simulation-to-reality performance for this task, a real-world validation dataset with 4 different shoes was recorded. The types of shoes range from sneakers to dress shoes, with different colors and sizes. The sizes of the used shoes range from 37-42 (EU). Figure 4.5 shows some example recordings of scenes used for the validation dataset. The shoes were placed in a random position and orientation on the floor in an area marked by a tape on the floor that had the same size as the workspace in the simulation environment.



(a) Training dataset - Version 1

(b) Training dataset - Version 2

(c) Real-world dataset

Figure 4.4: Height histograms of the training datasets and the real-world dataset.

Figure 4.5: Examples of recorded scenes for the real-world dataset of the *Shoe sorting* task.

Figure 4.6: Example recording of a scene with distractor objects.

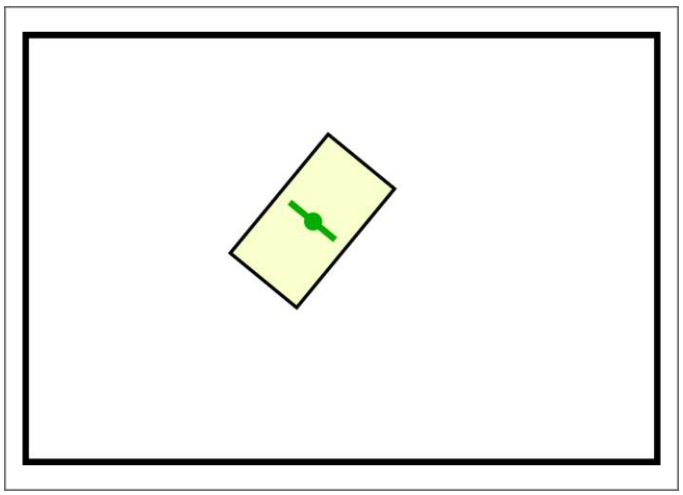


Figure 4.7: Illustration of the box picking task

Distractors dataset

In real-world scenarios, especially in household environments, the objects of interest are often surrounded by other objects. To evaluate the performance of the *Transporter Agent* in such scenarios, we additionally train and evaluate with datasets that contain distractor objects. The training dataset is generated the same way as before, just with the addition of distractor objects, as described in Section 3.4.6. The validation dataset is a second recorded dataset, using the same shoes as the previous *Sort shoes* dataset, but with distractor objects added to the scene. Figure 4.6 shows an example of a scene from the distractor dataset. Due to the recordings being separate, the poses in this distractor validation set are not the same as in the original validation set. The distractor dataset contains 20 scenes, with 4 different shoes and between 2 and 4 distractor objects per scene. The distractor objects are randomly scattered in the workspace.

4.1.3 Box Picking

While the aforementioned shoe sorting task poses an interesting challenge, it requires to simultaneously bridge the sim-to-real gap and generalize to unseen objects. This makes it difficult to interpret the influence of individual interventions on network performance and leads to a high variance in the results. In an attempt to select a second task, that does not require generalization to unseen objects, we selected the simple box picking task. The task consists of a box, with fixed size, that is placed in a random location and orientation in the workspace. For this task, we only recorded pick poses, because the ablation studies are only done for the pick pose networks. This was done, because the pick position and pick angle networks are comparably simple networks and show better convergence properties than the *Transport* network. The workspace for this task is the same as above.

Dataset Generation

To generate the dataset for the box picking task, a box of desired shape was created in the simulation environment, instead of using existing 3D models. The box is placed in

a random position and orientation in the workspace. The pick pose is annotated as the center of the top face of the box, with the angle being perpendicular to the long side of the box. An illustration of the pick configuration is given in Figure 4.7 The size of the box is varied only slightly from the size of the boxes used for real-world data collection, to ensure that the network generalizes to the real-world data. The size of the training boxes is varied by ±5% from the dimensions of the boxes used in the validation dataset. The limits for the size of the training boxes are:

- Length: 13.30 cm to 14.70 cm
- Width: 5.90 cm to 6.50 cm
- Height: 7.00 cm to 7.80 cm

Because data generation for this task is comparably fast, the dataset was created with 1000 demonstrations and 5 different randomized scenes for each demonstration, resulting in a total of 5000 rendered scenes in the training dataset. The test dataset consists of 100 demonstrations and 10 scenes for each demonstration, resulting in a total of 1000 rendered scenes in the test dataset.

Validation Dataset

To validate the trained networks on real-world data, a dataset was collected using two tea-boxes of the same size but with different print and colors. The size of the boxes used for validation is 14.00 cm × 6.20 cm × 7.40 cm. During data collection the boxes were placed in a random position and orientation on the floor in an area marked by a tape on the floor that had the same size as the workspace in the simulation environment. For each box, 20 scenes, were recorded, resulting in a total of 40 scenes in this dataset. A simple circular marker was placed on the top-center of each box, to simplify dataset annotation (described in Section 4.1.4). A picture of the boxes is shown in Figure 4.8.

4.1.4 Real World Data Collection

To evaluate the performance of the trained networks on real-world data, two datasets are collected. The details of the data collection and the annotation process is described in the following sections.

Pointcloud Recording

Data collection was done using a *RealSense D435* active-stereo depth camera [73], mounted on a table, facing downwards towards the workspace. Figure 4.9 shows the setup used for real-world data collection. The camera was mounted at an approximate height of 1.00 m above the workspace and in a distance of 75.00 cm from the workspace. Between individual captures, the position of the camera and its orientation are varied slightly from this initial configuration to increase the variety of viewpoints. Adjustments are done in a way that the workspace was always fully visible and approximately centered in the preview of the rendered pointcloud. For capturing the pointclouds, the *RealSense Viewer* software[74] was used. Besides preview and capture of the pointclouds, the software was also used to configure the camera settings. The camera is set to a resolution of 848.00 px × 480.00 px. All other configurations are set to the recommended values in the *Preset high density*, provided by *Intel* on their GitHub repository [75]. In the integrated postprocessing

Figure 4.8: Tea boxes used for real-world data collection

pipeline of *RealSense Viewer*, only a temporal filter with values $alpha = 0.1$ and $delta = 11$ was applied to reduce noise and smooth the pointclouds. For the *Shoe sorting* task, two pointclouds are recorded for each scene, one for the pick pose and another with the second shoe placed in the desired location. This was done to simplify the annotation of place poses in the next step.

Postprocessing and Annotation

To generate a dataset from the recorded pointclouds, a semi-automatic annotation pipeline was developed. The pipeline consists of the following steps. For the *Box-pick-up*-task, all steps mentioning a place pointcloud are omitted.

1. Registration

Using the *Open3D* library [47], the pointclouds are roughly cropped and aligned to the xy-plane. The alignment is done using the *Random Sample Consensus - RANSAC* algorithm [76] to find the ground plane and transform the pointclouds, so that the ground plane is aligned with the xy-plane. Only the pick pointcloud is used to find the alignment parameters. The same parameters are then applied to the place pointcloud. The registration process is visualized in Figures 4.10a and 4.10b.

2. Workspace alignment

The aligned pointclouds are projected to a top-down color image, using a fixed pixel size of 2.00 mm. In an interactive Python script, a rectangle with the size of the workspace is drawn as an overlay on the projected scene (Figure 4.10c). Using manual input, the underlying pointcloud is then iteratively translated along the x and y axis and rotated around the z axis, so that the marked workspace

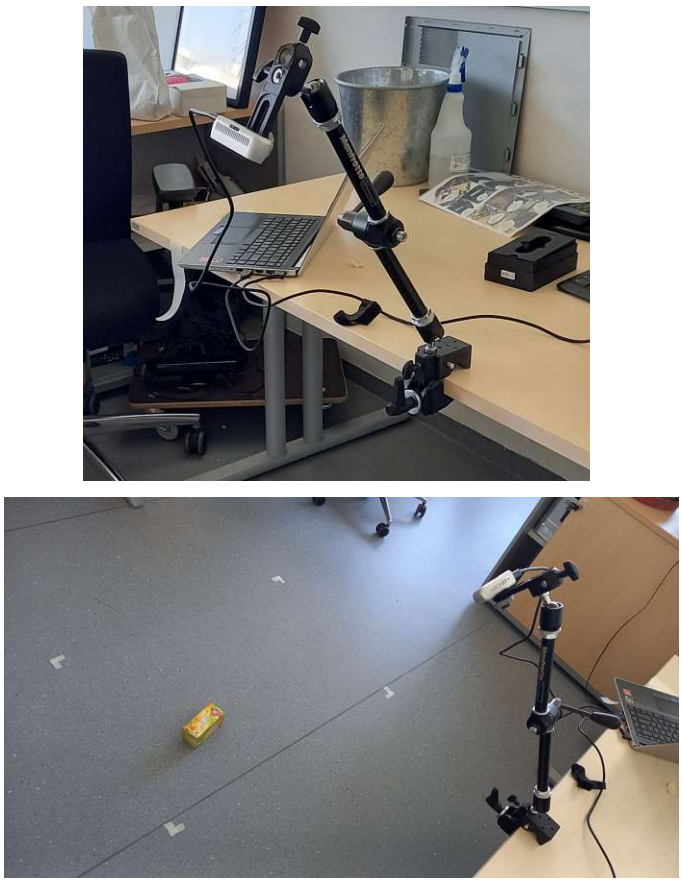
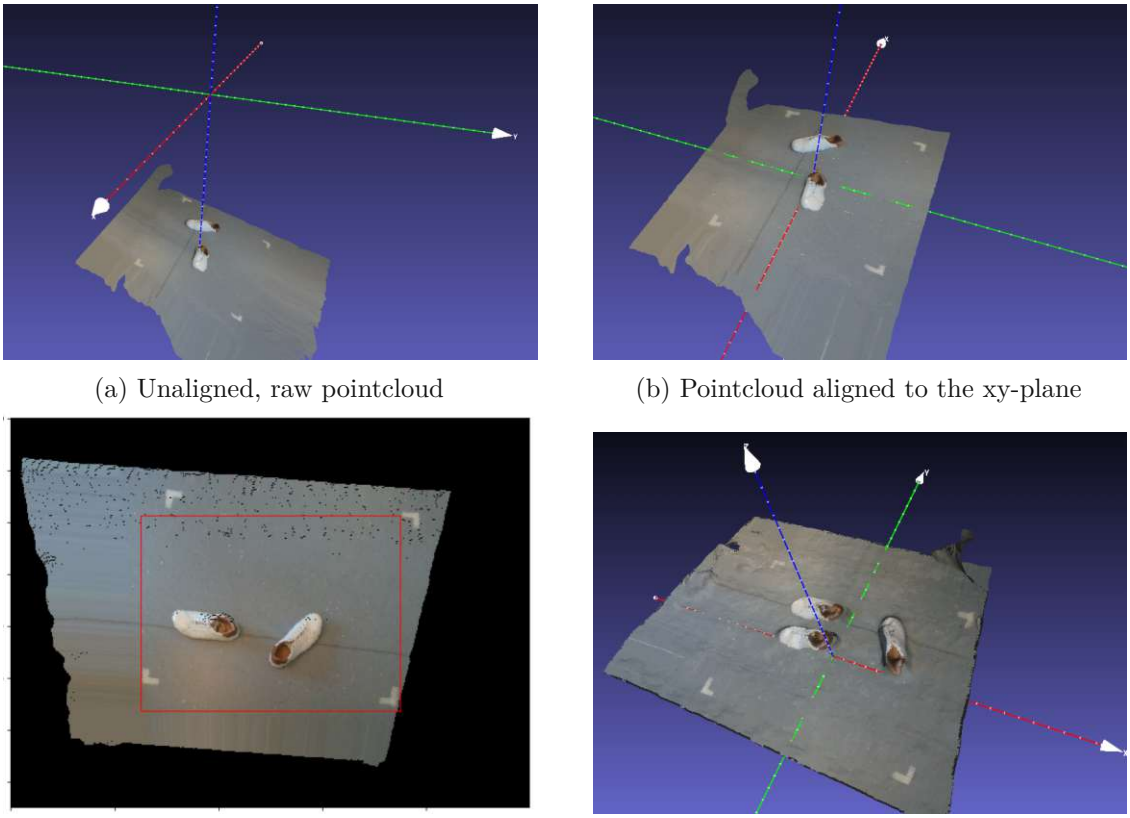


Figure 4.9: Setup for real-world pointcloud recording, consisting of a *D435i* camera mounted to a table using an adjustable arm. For recording the *RealSense Viewer* software was used.

in the recording is aligned with the overlay. The determined transformation is then applied to the pick- and the place-pointcloud, so that the workspace origin is approximately at $(0, 0, 0)$ and the workspace is aligned in parallel to the x and y axis. Additionally, the *Iterative Closest Points - ICP* registration method [77] is applied to fine-tune the alignment between the pointclouds of the pick scene and the place scene. Figure 4.10d shows the pick and the place pointclouds centered around the workspace, aligned with each other.

3. Annotation

For annotating the pick and place poses, the pointclouds are again projected to a top-down color image and a height map for each scene. The color images and heightmaps from the pick and the place scene are then visualized side by side. Annotation is done by clicking on the point in the image, that corresponds to the pick- or place-position. The position of the click is then visualized using a marker at the clicked location in the color and the depth image. Since the pixel size for the pointcloud-to-image transformation is known and the pointcloud was previously centered at the workspace, the clicked location in the image can be transformed into a 2D coordinate in the world coordinate system. To annotate the angles a line is



(a) Unaligned, raw pointcloud

(b) Pointcloud aligned to the xy-plane

(c) Workspace alignment

(d) Pick- and place-pointclouds aligned and overlayed

Figure 4.10: Illustration of the pointcloud preprocessing pipeline.

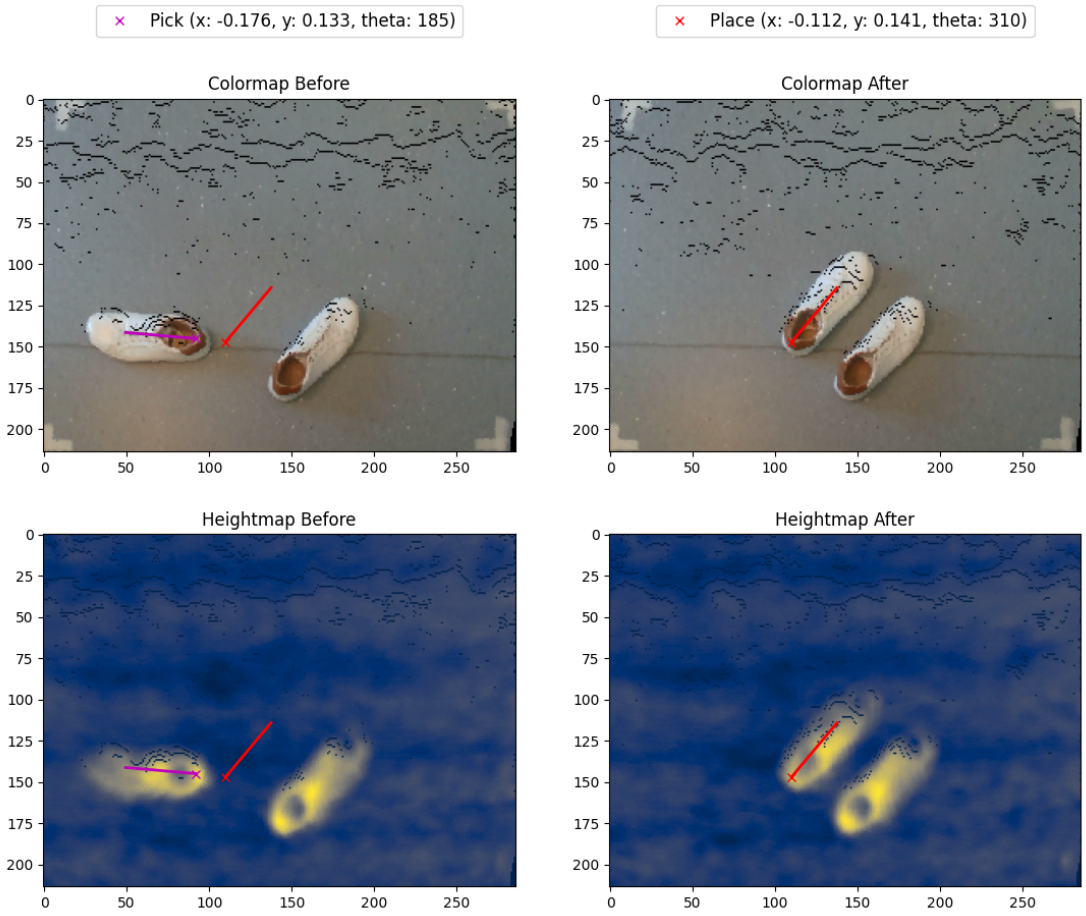


Figure 4.11: Screenshot of the built annotation tool.

drawn from the chosen points in the image, that is then rotated in 5.00 deg steps, until the desired pick or place angle is reached, using manual input. Figure 4.11 shows the annotation process for the shoe sorting task.

4. Dataset generation

Using the previously aligned pick-scene pointclouds and the annotations, a dataset in the same format as the training data is generated. This way, it can be integrated seamlessly into the training process for validation.

The whole process is configured and parameterized using a *json* configuration file, which is passed to the scripts at each step. It contains all required file paths, the pixel size for the pointcloud-to-image transformation and the size and location of the workspace in the pointcloud.

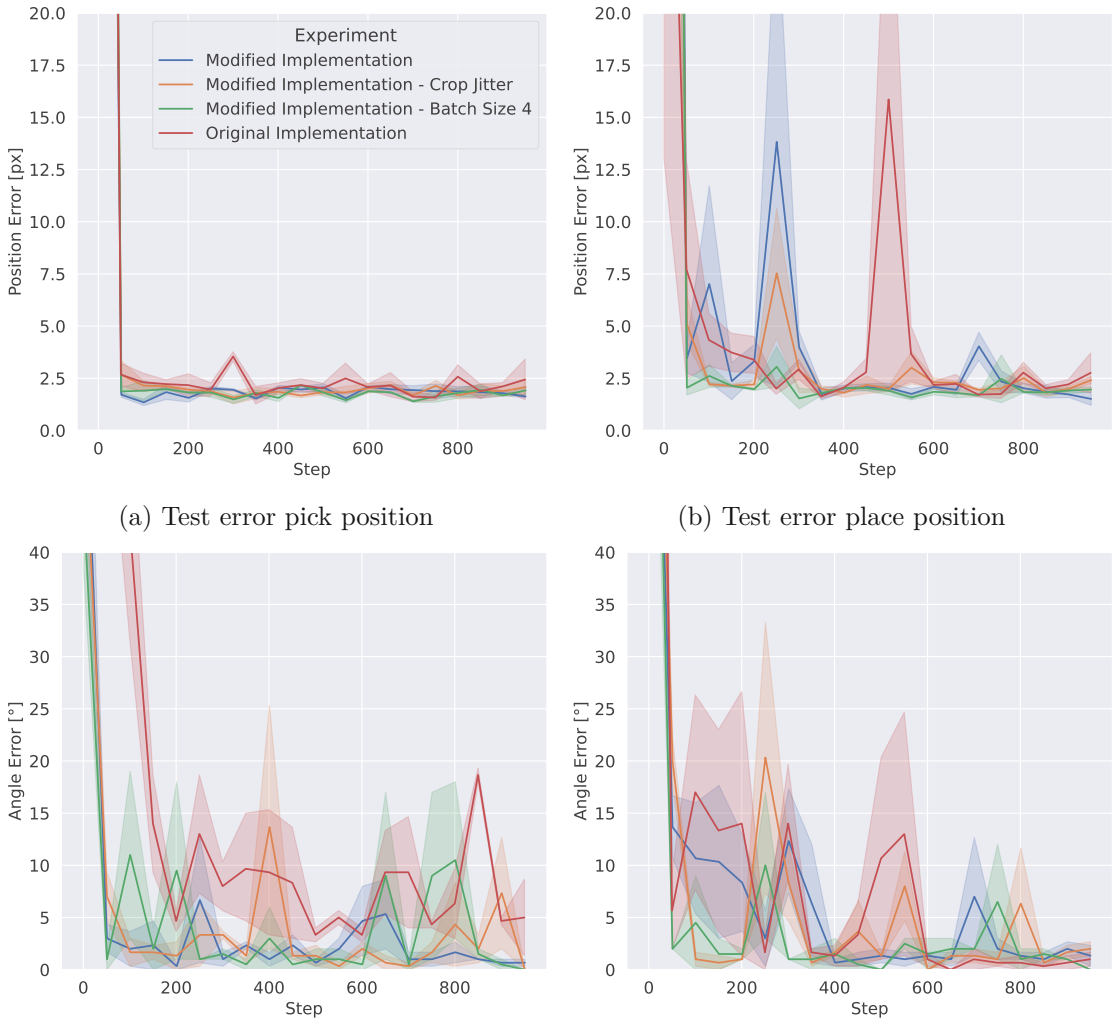
4.2 Performance Metrics and Evaluation

Two different metrics are used to evaluate the observed performance in the ablation studies. The first metric is the mean and standard deviation of the pick position or angle error. The errors are calculated as the Euclidean distance or the shortest angle distance between prediction and ground-truth label.
The mean error is calculated as the mean over multiple evaluations, done at 100 step intervals. For each evaluation, 10 predictions are compared to the ground truth labels and the mean of the errors is reported. The reported values therefore give an average over multiple evaluations during one or more training runs.
The values are given in the format $\mu \pm \sigma$, where μ is the mean error and σ is the standard deviation of the error. Since the variance of the validation error is relatively high between individual evaluation steps, an additional metric is used to evaluate the performance of the network. The second metric is the percentage of evaluations where the mean validation error is below a given success-threshold, denoted as aSR. The success-thresholds are the estimated upper limits for the pick position and angle errors, below which a pick is considered successful. The used thresholds are 3.00 cm for the pick position and 10.00° for the pick angle. This is more lenient than the thresholds used in the original Transporter Networks publication [36], where the thresholds are 1.00 cm for the pick position and 5.00° for the pick angle. They reported, however, that the chosen thresholds were too strict for real-world experiments. Furthermore, the used RGB-D camera (Intel RealSense D435) is not a high precision depth camera, as was used in the original publication.
The statistics are calculated from data of 3 training runs for each configuration. The first n iterations are excluded from the evaluations of the mean and standard deviation values to remove the influence of the initial convergence phase. Due to the different convergence properties of the networks and the different complexities of the tasks, the number of excluded iterations n is different for each network-task combination. The chosen number is given at each evaluation. In all angle network evaluations, the input crop for the angle prediction is done at the position of the ground truth pick-label. This is to isolate the performance of the angle prediction network. Unless otherwise noted, all experiments are done with a batch size of 4 and a learning rate of $1e^{-4}$. The test set is a simulation dataset while the validation set is the real-world dataset.
The resolution of the images used for training and evaluation is 320 px × 240 px. With the

workspace size of 1.00 m × 0.75 m, this results in a pixel size of 3.125 mm per pixel. The used randomization and augmentation settings can be found in Table A.2 and Table B.1.

4.3 Block Insertion Task

Before using the modified networks for the *Box picking* and *Shoe sorting* task, the networks were evaluated on the *Block insertion* task, from the original Transporter Networks publication. This is done to verify that the modification applied to the networks does not lead to differences in the performance of the networks.



(a) Test error pick position

(b) Test error place position

(c) Test error pick angle

(d) Test error place angle

Figure 4.12: Results for the block insertion task using the original implementation and on data from Omniverse, using the modified Transporter Agent. Plots show test error for the different networks used in the Transporter Networks agent. The plots show the mean error and the 75% confidence interval from 3 runs.

Figure 4.12 shows plots of the test errors for the pick position(*Attention*) network, the

pick angle network and the *Transport* network, which predicts the place position and the place angle. For both trainings, a learning rate of $1e^{-4}$ was used during training and training was done for 900 steps. The analysis of the graphs reveals several important findings. The modified implementation demonstrates performance comparable to the original Transporter Networks implementation when using identical augmentation settings. A notable observation is the relatively high variation in the test error, that is observable in both implementations. In terms of convergence properties, both the *attention* and pick-angle networks exhibit better characteristics compared to the *Transport* network. The comparison also shows that utilizing a batch size of 4 can achieve a reduction in both variance and mean values of test errors, though this comes at the cost of extended training durations (which is not apparent in the graphical representations). For this simulation-only task, the introduction of random variations to the crop position for angle prediction and the *transport* network does not significantly impact the overall performance.

4.4 Augmentation Ablations

To determine the influence of each applied augmentation on the performance on real-world data, an ablation study is performed for all augmentations. The ablations are evaluated on the *Box picking* task as well as the *Shoe sorting* task with the *Attention* and the *pick-angle* networks.
For the ablations, datasets with all randomizers activated during rendering are used. An ablation study for the different randomizers is done in Section 4.8.
The parameters used for generating the dataset used for this ablation study are given in Table B.1. For each ablation, one augmentation is disabled during training. Evaluation is done every 100 steps, using 15 samples from the test dataset and the validation dataset each.

4.4.1 Box Picking Task

The first evaluated task, is the task of picking up a rectangular box, as explained in Section 4.1.1. Since the shape of the box is constant between the training and the evaluation set from real-world data, the observed errors in the evaluation performance can be attributed to the influence of the augmentations. For each experiment run, the networks are trained on 100 pick-and-place samples for 5000 iterations. The training for all of the runs used in the augmentation study took about 3 days on a single *Nvidia RTX-3060* GPU for the pick-position network. Whereas the training of the pick-angle network took about 6 days on a single *Nvidia RTX-2080 Ti* GPU.

Pick Position

The results for the *Pick-position*(*Attention*) network on the *Box picking* task show that, for this simple task, a mean error of approximately 1.00 cm can be achieved with the developed augmentation techniques. Given that the network is trained using a pixel-size of 3.125 mm, this equals an average pixel error of below 3.00 px. This is a surprisingly good result, considering that the dataset is manually annotated and the network is trained exclusively on synthetic data. However, the validation error generally shows high variance. For the *All augmentations* case for example, the mean error on the validation set is

Table 4.2: Ablation study with one augmentation disabled in each experiment, using the *Attention* network on the *Box picking* task. Trained for 5000 iterations and evaluated every 100 iterations. The last 45 evaluations are used for the statistics. aSR success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
All augmentations	0.044 ± 0.111	0.021 ± 0.034	2.56 ± 2.35	9.95 ± 3.07	88.4
No color aug.	0.045 ± 0.091	0.026 ± 0.062	2.84 ± 2.20	9.39 ± 2.42	87.0
No color noise	0.053 ± 0.102	0.025 ± 0.053	2.57 ± 1.90	10.01 ± 2.42	87.7
No cutouts	$\mathbf{0.002 \pm 0.000}$	$\mathbf{0.009 \pm 0.002}$	1.62 ± 0.50	10.19 ± 3.37	**100.0**
No depth scaling	0.051 ± 0.108	0.027 ± 0.055	2.60 ± 1.82	9.29 ± 2.37	85.5
No downsampling	0.051 ± 0.112	0.026 ± 0.049	2.84 ± 2.76	9.76 ± 2.75	87.0
No location noise	$\mathbf{0.002 \pm 0.009}$	0.054 ± 0.101	$\mathbf{1.08 \pm 0.37}$	7.45 ± 2.06	69.6
No Perlin noise	0.051 ± 0.120	0.033 ± 0.072	2.52 ± 2.18	10.30 ± 2.81	86.2
No plane offset	0.006 ± 0.021	0.227 ± 0.174	2.72 ± 0.88	19.98 ± 6.95	13.8
No salt and pepper	0.045 ± 0.092	0.042 ± 0.084	2.56 ± 2.35	9.53 ± 2.80	79.7
No smoothing	0.016 ± 0.047	0.054 ± 0.098	2.53 ± 1.09	$\mathbf{7.21 \pm 2.17}$	73.2

2.1 cm with a standard deviation of 3.4 cm. The results, presented in Table 4.2, show that the *Plane offset* augmentation is the most critical for enabling successful transfer from simulation to real-world data for this experiment. Disabling only this augmentation leads to an increase in the mean validation error by a factor of 10 and a decrease in the average success rate aSR by 75 %. This can be attributed to the fact that, unlike the simulated pointclouds, the real-world pointclouds are not perfectly aligned to the xy-plane. Introducing similar random variations during training enables the network to cope with this misalignment.

The *Location noise* augmentation technique, where random Gaussian noise is added to the position of the individual points in the point cloud, also has a significant influence on the performance on the real-world dataset. Not adding *Location noise* doubles the mean error on the validation set and reduces the aSR by 18 %.

Surprisingly, disabling the *Cutouts* augmentation during training, leads to significantly better performance on the test and the validation set. Resulting in an error of below 1.00 cm and a aSR of 100 %. This improvement however, might not be due to the network learning the shape of the object of interest, but rather learning to locate the only object present in the scene. Because, as described in Section 3.5, the cutouts share the same shape as the object of interest, and with a probability of 0.5 the points are set to a random z-height and color. This might act as an especially strong distractor for the *Box picking* task, where the object of interest is also a rectangular shape.

The following key points can be drawn from the results:

- Augmentations that modify depth channel properties have a significantly greater impact on validation performance compared to color augmentations.
- Randomized rotations around the x- and y-axes during training are essential for the network to handle misaligned point clouds in real-world recordings.
- Excluding the *Location noise* and *Plane offset* augmentations results in better

Table 4.3: Ablation study with one augmentation disabled in each experiment, using the *Pick angle* network on the *Box picking* task. Trained for 5000 iterations and evaluated every 100 iterations. The last 40 evaluations are used for calculating the error and loss statistics. aSR success limit: 10.00°. Measurements are given as mean ± std.

Experiment	Test Error [°]	Validation Error [°]	Test Loss	Validation Loss	Valid. aSR [%]
All augmentations	5.59 ± 6.19	7.71 ± 5.41	0.50 ± 0.24	1.19 ± 0.35	72.4
No color aug.	4.17 ± 4.47	7.25 ± 5.31	0.43 ± 0.19	1.19 ± 0.36	74.0
No color noise	4.19 ± 4.51	6.95 ± 5.57	0.47 ± 0.22	1.10 ± 0.34	77.2
No cutouts	4.25 ± 4.96	7.26 ± 5.56	0.48 ± 0.22	1.14 ± 0.38	73.2
No depth scaling	4.16 ± 4.81	8.66 ± 6.54	0.46 ± 0.22	1.26 ± 0.43	62.6
No downsampling	3.48 ± 4.28	7.28 ± 6.04	0.40 ± 0.21	1.03 ± 0.36	71.5
No location noise	3.10 ± 3.89	12.18 ± 7.67	$\mathbf{0.33 \pm 0.24}$	1.20 ± 0.34	46.3
No Perlin noise	4.33 ± 4.72	7.07 ± 4.93	0.53 ± 0.25	1.07 ± 0.33	71.5
No plane offset	3.84 ± 5.00	10.64 ± 6.78	0.40 ± 0.26	1.61 ± 0.63	54.5
No salt and pepper	4.20 ± 4.53	7.86 ± 5.61	0.48 ± 0.26	1.19 ± 0.31	71.5
No smoothing	3.09 ± 3.90	$\mathbf{5.07 \pm 3.92}$	0.37 ± 0.14	$\mathbf{0.89 \pm 0.32}$	**86.2**
No crop jitter	$\mathbf{2.91 \pm 3.84}$	25.61 ± 11.15	0.34 ± 0.19	2.85 ± 0.66	2.4

performance on the simulation test set but leads to higher errors on the real-world validation set, indicating poorer transfer to real data.

- The *Cutout* augmentation seems to have a negative influence on the performance of the network.

Pick Angle

The results for the pick-angle network are given in Table 4.3. It can be seen from the results that, with the developed augmentation techniques, a mean error as low as 5.0° can be achieved on the real-world validation set. Since the pick angle network has a discrete output space, with bins of 10.0° each, the achieved mean error is below one bin. However, what can also be seen, is that the standard deviation in relation to the mean of the validation error is again very high. With all augmentations activated, the mean error is 7.7° with a standard deviation of 5.4°.

Analyzing the influence of the individual augmentations shows that the *Location noise* and the *Plane offset* augmentations are the most significant for good performance on the validation set. This is in line with the results for the augmentations with the *Pick position* network (Section 4.4.1). Also, in line with previous results is that, disabling augmentations that modify the color properties of the training samples has no significant influence on the performance of the network on the real-world validation set. By far the most significant augmentation for the performance on the validation set is the introduction of *Crop jitter*. This augmentation introduces random variations in the position where the crop for the angle prediction is taken, during training. Disabling only this augmentation leads to an increase in the mean validation error by a factor of 3 and reduces the average success rate aSR by 70 %. This is most likely caused by the fact that the *Pick position*, which is used

as the crop center, is manually annotated on the real-world dataset. Due to the manual annotation process the labels and therefore the crop position is not as consistent as the ground-truth crop position in the simulation dataset. By introducing variations in the position where the crop for the angle prediction is taken, during training, the network is able to cope with crop-positions that are not perfectly aligned on the ideal pick-point. This will also be important when using the *Attention* network to predict the pick position, instead of using the annotated labels, as this will introduce even more variance to the crop position.

Interestingly, it can be seen that the *Gaussian smoothing* augmentation, reduces mean and standard deviation of the validation error and significantly improves the aSR. This is in contrast to the findings with the *Attention* network, where disabling the *Gaussian smoothing* augmentation increased the mean and standard deviation of the validation error. This might be due to the fact that the *Gaussian smoothing* augmentation smooths the edges of the depth image, which could be more beneficial to predict the rotation of the object than the position.
In summary, the following key points can be observed:

- Augmentations that modify the color properties of the training data seem to have no significant influence on the performance of the network.
- The *Location noise* and *Plane offset* augmentations are again the most significant for good transfer performance on the validation set.
- The addition of *Crop jitter* improves the performance of the network on real data by a factor of 3.
- Contrary to the *Attention* network, removing the *Gaussian smoothing* augmentation during training leads to better performance on the test and validation set.

4.4.2 Shoe Sorting Task

As a more complex task that involves generalization to unseen objects, the *Shoe sorting* task is evaluated.

Pick position

The results for the *Attention* network on the *Shoe sorting* task are given in Table 4.4. Looking at the mean error and aSR values, it immediately becomes clear that the network shows far worse performance than on the *Box picking* task. This partly can be attributed to the fact that the correct pick-position on the shoes is much harder to detect than simply the center of a rectangular box. It must be noted however that the ablation study for the *Attention* network is done using *Version 1* of the generated dataset. As explained in Section 4.1.2, this dataset shows a relatively large difference in dimensions of the simulated shoes to the real shoes in the validation dataset, leading to sub-optimal performance. Comparing the validation error, validation loss and aSR values, to the results achieved with *Version 2* of the dataset the performance difference becomes evident. Using the *All augmentations* settings as a comparison basis. The results for *Version 2* of the dataset are a mean error of 13.0 cm with a standard deviation of 6.6 cm and an aSR of 4.9 %. This is almost half the mean error, achieved with *Version 1* of the dataset.
Although the absolute errors on *Version 1* of the validation dataset are larger than for *Version 2* of the dataset, the relative errors within this ablation study are still comparable.

Table 4.4: Ablation study with one augmentation disabled in each experiment, using the *Attention* network. Trained on the *Sort shoes* task (Dataset version 1). Network trained for 6000 iterations, evaluated every 100 iterations. The last 40 evaluations are used for calculating the error and loss statistics. *aSR* success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
All augmentations	0.093 ± 0.061	0.217 ± 0.092	3.50 ± 0.57	8.81 ± 2.30	0.0
No color aug.	0.091 ± 0.062	0.217 ± 0.084	3.59 ± 0.69	9.66 ± 2.00	0.4
No color noise	0.089 ± 0.066	0.244 ± 0.081	3.69 ± 0.73	10.28 ± 2.29	0.6
No cutouts	0.076 ± 0.057	0.242 ± 0.101	3.46 ± 0.70	9.80 ± 1.98	**0.8**
No depth scaling	0.100 ± 0.066	0.199 ± 0.080	3.42 ± 0.50	$\mathbf{7.95 \pm 1.47}$	0.4
No downsampling	0.098 ± 0.060	0.180 ± 0.083	3.41 ± 0.43	8.21 ± 1.48	0.8
No location noise	$\mathbf{0.049 \pm 0.043}$	0.440 ± 0.106	$\mathbf{3.11 \pm 0.81}$	16.06 ± 4.00	0.0
No Perlin noise	0.097 ± 0.058	0.217 ± 0.087	3.70 ± 0.65	9.84 ± 2.03	0.4
No plane offset	0.058 ± 0.047	0.359 ± 0.110	3.40 ± 0.66	11.35 ± 3.11	0.0
No salt and pepper	0.091 ± 0.062	0.230 ± 0.086	3.29 ± 0.41	8.10 ± 1.40	0.4
No smoothing	0.089 ± 0.059	$\mathbf{0.174 \pm 0.070}$	3.54 ± 0.57	8.80 ± 1.31	0.4

When comparing the results for the different augmentations, it can be seen that, as before, the augmentations that modify the spatial properties have a much larger influence on the mean-error performance than the color augmentations. The largest influence on the performance has been observed for the *Location noise* augmentation, which is in-line with the previous results. By disabling this augmentation, the mean validation error increases by a factor of 2. The second largest increase in the mean validation error is observed when disabling the *Plane offset* augmentation, which also has had a significant impact in the previous ablation studies.
Contrary to the results of the attention network on the *Box picking* task, the *Gaussian smoothing* augmentation seems to have a negative influence on the performance of the network on this task. Disabling the *Cutout* and *Pointcloud downsampling* augmentations lead to a slight decrease in the mean validation error, but the observed changes are not significant, given the high standard deviations.
To summarize the following key points can be observed:

- The color augmentations seem to have no significant impact on the performance of the network on real-world data.
- The *Location noise* and *Plane offset* augmentations have the largest influence on the real-world performance of the network.
- Although the *Gaussian smoothing* augmentation is only applied with a probability of 0.5 in the augmentation configuration, removing it from the augmentation set reduces the mean validation error and its standard deviation.

Table 4.5: Ablation study with one augmentation disabled in each experiment, using the *Pick angle* network. Trained on the *Sort shoes* task (Dataset version 2). Network trained for 7000 iterations, evaluated every 100 iterations. The last 40 evaluations are used for calculating the error and loss statistics. *aSR* success limit: 10.00°. Measurements are given as mean ± std.

Experiment	Test Error [°]	Validation Error [°]	Test Loss	Validation Loss	Valid. aSR [%]
All augmentations	7.23 ± 4.72	9.49 ± 5.73	0.94 ± 0.26	1.28 ± 0.26	60.6
No crop jitter	5.38 ± 4.24	20.29 ± 8.17	0.90 ± 0.30	1.61 ± 0.28	6.5
No depth scaling	5.94 ± 5.46	20.73 ± 9.54	0.68 ± 0.25	1.18 ± 0.29	12.5
No downsampling	4.62 ± 4.24	21.06 ± 8.31	0.66 ± 0.23	1.22 ± 0.27	12.5
No location noise	5.92 ± 4.85	10.09 ± 4.54	0.77 ± 0.21	1.24 ± 0.17	44.4
No plane offset	$\mathbf{4.31 \pm 4.49}$	20.63 ± 8.29	$\mathbf{0.50 \pm 0.17}$	1.44 ± 0.31	6.9
No smoothing	7.23 ± 5.46	$\mathbf{9.12 \pm 6.13}$	0.92 ± 0.30	$\mathbf{1.11 \pm 0.19}$	**63.4**

Pick Angle

An augmentation ablation study is also done for the pick-angle network on the *Shoe sorting* task. However, due to the long training time required for the pick-angle network, the ablation study is only done for the augmentations, that showed the most significant influence on the validation performance in the previous experiments(4.4.2). The evaluated augmentations are: *Location noise*, *Plane offset*, *Gaussian smoothing*, *Pointcloud downsampling*, *Depth scaling* and *Crop jitter*. The results of the ablation study are given in Table 4.5.

This ablation study is performed using *Version 2* of the dataset of the *Shoe sorting* task, as explained in Section 4.1.2.

It can be seen from the results that the mean validation error and the average success rate are still comparable to the results of the *Box picking* task(4.4.1). This is somewhat unexpected, since the network needs to generalize to unseen objects.

Similar to the ablation study on the *Box picking* task, the removal of *Crop jitter* and *Plane offset* lead to the largest decrease in average success rate. The effect of removing the *Location noise* augmentation is less significant than in the previous ablation study, but still leads to a decrease in the average success rate of 16 %. In-line with previous results, the color augmentations have little influence on the performance of the network on the real-world validation set. Contrary to the previous results however, removing the *Pointcloud downsampling* augmentation, has strong negative effects on the performance on the real-world validation set, reducing the *aSR* from 60 % to 12 %. The same can be observed for the *Depth scaling* augmentation. While showing little effect on the *Box picking* task, disabling this augmentation on the *Shoe sorting* task, leads to an increase in the mean validation error by a factor of 2.

Another notable finding is that, again, the color augmentations seem to have little effect on the validation performance of the network.

Table 4.6: Ablation study of different learning rates and weight decay values. Experiments done using the *Attention* network on the *Sort shoes* task (Dataset version 1). Trained for 6000 iterations and evaluated every 100 iterations. The last 40 evaluations are used for calculating the statistics. aSR success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
lr:1×10^{-4} wd:0	0.087 ± 0.063	0.194 ± 0.095	$\mathbf{3.34 \pm 0.46}$	7.75 ± 1.32	0.4
lr:1×10^{-4} wd:1×10^{-4}	$\mathbf{0.084 \pm 0.065}$	$\mathbf{0.182 \pm 0.077}$	3.36 ± 0.50	$\mathbf{7.55 \pm 1.30}$	**0.6**
lr:5×10^{-4} wd:1×10^{-4}	0.088 ± 0.057	0.194 ± 0.068	3.62 ± 0.55	8.39 ± 1.56	0.0
lr:8×10^{-4} wd:1×10^{-4}	0.117 ± 0.082	0.196 ± 0.061	4.22 ± 1.77	8.93 ± 2.37	0.0

4.4.3 Summary

To summarize the results of the augmentation ablation studies, the following key points can be observed:

- Adding *Location noise* to the pointcloud observation has a significant influence on the performance on the real-world dataset.
- The *Plane offset* and *Crop jitter* augmentations are essential for the networks to handle misaligned point clouds and variations in the predicted pick-position in the real-world data.
- The augmentations that modify the color channel properties seem to have little to no influence on the performance on the real-world dataset.
- The *Gaussian smoothing* augmentation generally seems to have a negative influence on the performance of the networks.

4.5 Learning Rate and Batch Size

The changes to the network architecture and the training process, described in Section 3.2, enable the use of batched training. The motivation behind this is to reduce the variance of the error during training, as shown in [38].
Since changing the batch size influences the learning dynamics of the networks compared to the original paper, adapting the learning rate might be beneficial. In this section, the performance using different batch sizes and learning rates is analyzed. This analysis is predominantly done on the *Attention* network, because doing the ablations on the *Pick-angle* would take prohibitively long. Furthermore the *Pick-angle* network already shows satisfactory performance on the *Shoe sorting* task, even with the default settings. The learning rate and batch size ablations are only done on the *Shoe sorting* task, since the results for the *Attention* network on the *Box picking* have shown to not generalize well to the more interesting *Shoe sorting* task.

Table 4.7: Ablation study of different batch sizes for the *Attention* network on the *Sort shoes* task (Dataset version 2). Evaluated every 100 iterations in the range: Batch size 1: 3000 → 10000, Batch size 4: 2000 → 5000. *aSR* success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
Batch size 1	0.218 ± 0.068	0.127 ± 0.062	$\mathbf{4.90 \pm 0.46}$	$\mathbf{5.57 \pm 0.53}$	1.1
Batch size 4	$\mathbf{0.200 \pm 0.074}$	$\mathbf{0.119 \pm 0.056}$	5.03 ± 0.62	5.99 ± 0.56	**3.3**

Table 4.8: Ablation study of different batch sizes for the *Pick angle* network on the *Sort shoes* task (Dataset version 2). Evaluated iterations: Batch size 1: 3000 → 10000, Batch size 4: 2000 → 5000. *aSR* success limit: 10.00°. Measurements are given as mean ± std.

Experiment	Test Error [°]	Validation Error [°]	Test Loss	Validation Loss	Valid. aSR [%]
Batch size 1	$\mathbf{6.31 \pm 5.01}$	20.42 ± 7.58	$\mathbf{0.79 \pm 0.24}$	$\mathbf{1.11 \pm 0.21}$	8.8
Batch size 4	7.88 ± 5.40	$\mathbf{10.05 \pm 5.88}$	1.01 ± 0.32	1.28 ± 0.23	**59.5**

4.5.1 Learning Rate

For the learning rate ablation, the learning rate is varied between the default value of 1×10^{-4} and 8×10^{-5}. Additionally to altering the learning rate, the weight decay parameter of the used *AdamW* optimizer is changed. Weight decay is a common regularization technique that penalizes large weights in the network. This has been shown to be beneficial for the generalization performance of neural networks [78]. The weight decay is set to a value of 10^{-4} when used.
Because this was one of the first ablations done, the ablation is done on *Version 1* of the dataset for the *Shoe sorting* task. The training is carried out for 6000 iterations, excluding the first 2000 iterations in the evaluation. The results in Table 4.6 show that higher learning rates do not lead to better performance on the validation set. However, the addition of weight decay seems to have a positive effect on performance. Adding weight decay decreases the mean validation error as well as the mean test- and validation loss.

4.5.2 Batch Size

Since the change of the used batch size is partly motivated to decrease the variance of the error during training, it is interesting to see if a difference could be observed between a batch size of 4 and the default batch size of 1. The results of the batch size ablation study are shown in Table 4.7 and Table 4.8. The batch size of 4 is used, because it is the largest batch size that does not cause "out-of-memory" errors on the used GPU (*Nvidia RTX-3060*). This ablation is evaluated on *Version 2* of the dataset for the *Shoe sorting*

Table 4.9: Evaluation of using Dropout layers with different dropout rates in the *Attention* network. Trained on the *Sort shoes* task (Dataset version 2). Trained for 6000 iterations and evaluated every 100 iterations. The last 40 evaluations are used for the statistics. aSR success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
No dropout	**0.170 ± 0.078**	0.132 ± 0.063	4.03 ± 0.57	**6.92 ± 0.89**	**4.9**
Dropout rate 0.15	0.174 ± 0.064	0.122 ± 0.062	4.01 ± 0.58	7.22 ± 1.15	4.3
Dropout rate 0.3	0.175 ± 0.064	**0.109 ± 0.050**	**3.85 ± 0.52**	7.58 ± 1.10	2.5
Dropout rate 0.6	0.183 ± 0.073	0.113 ± 0.044	4.00 ± 0.58	7.57 ± 1.10	1.1

task.
Because the amount of processed samples per step is different between the two batch sizes, the range of evaluated iterations is adjusted accordingly.
The *Pick-angle* network shows a significant reduction in mean error and a 50% increase in the average success rate aSR when increasing the batch size from 1 to 4. The results for the *Attention* network show a similar trend, but the observed changes are less significant.

4.6 Dropout

Especially when smaller training sets are used, dropout is a popular regularization technique to reduce overfitting. Usually, dropout is used in cases where the network shows overfitting on the training set versus the validation set in the same domain [43]. In this case, we analyze the effects of dropout on the transfer performance of the network from simulation to real-world data. For the dropout ablation study, *Version 2* of the *Shoe sorting* dataset is used, as explained in Section 4.1.2. The training is done for 6000 iterations, excluding the first 2000 iterations in the evaluation. The results of the dropout ablation study are shown in Table 4.9. It can be seen that introducing dropout leads to a slight decrease in the mean test and validation error, indeed indicating less overfitting to the training set. However, when using dropout, the standard deviation of the validation error also decreases. Although this would usually be a good thing, this decreases the aSR. In the given use-case, dropout is therefore not a suitable regularization technique and is not used in the final system.

4.7 Augmentation Parameter Optimization

The augmentation ablation studies in 4.4 show that *Location noise* and *Plane offset* are the most significant augmentations for the transfer performance of the network. Therefore, we investigate the influence of different parameterizations of these augmentations on the performance of the network. For this parameter tuning, the amplitudes of the applied augmentations are varied. The evaluation is done on *Version 2* of the *Shoe sorting* dataset with the *Attention* network. All other augmentations are kept constant, and only the

Table 4.10: Ablation study of different standard deviations for the *Location noise* augmentation. Experiments done using the *Attention* network, on the *Sort shoes* task (Dataset version 2). Trained for 6000 iterations and evaluated every 100 iterations. The last 40 evaluations are used for calculating the statistics. *aSR* success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
$\sigma = 0.0025$	0.179 ± 0.071	0.210 ± 0.094	$\mathbf{3.37 \pm 0.58}$	10.70 ± 2.26	0.0
$\sigma = 0.005$	$\mathbf{0.167 \pm 0.067}$	0.139 ± 0.059	3.47 ± 0.42	8.72 ± 1.28	2.5
$\sigma = 0.0075$	0.180 ± 0.074	0.120 ± 0.062	4.01 ± 0.50	7.07 ± 1.04	**4.9**
$\sigma = 0.01$	0.183 ± 0.070	$\mathbf{0.115 \pm 0.052}$	4.50 ± 0.45	6.29 ± 0.67	1.2
$\sigma = 0.0125$	0.203 ± 0.068	0.118 ± 0.049	5.24 ± 0.48	$\mathbf{6.04 \pm 0.59}$	2.5

parameters of the augmentation to be evaluated are changed. For the ablations, the training is done for 6000 iterations, excluding the first 2000 iterations in the evaluation.

4.7.1 Gaussian Location Noise

Across all evaluations, the addition of Gaussian noise to the location of the points in the pointcloud has shown to have a significant influence on the transfer performance on real world data. Naturally, it is interesting to see how the amount of noise added to the point positions influences the performance. For the ablation study, the standard deviation of the Gaussian noise is varied between 2.5 mm and 12.5 mm. A couple of interesting relationships can be observed from the results, given in Table 4.10.
The higher the standard deviation of the Gaussian noise, the lower the mean error on the validation set. The performance on the validation set is inversely related to the performance on the test set. The highest average success rate is reached with a standard deviation of 7.5 mm. This is the value used for the augmentation ablation studies(4.4.2, 4.4.2) as well as for the final experiments.

4.7.2 Plane Offset

The *Plane offset* augmentation has shown to be essential for the trained networks to able to cope with the registration errors and misalignment of real-world pointclouds. The augmentation tilts the pointcloud by a random degree across the x- and y-axis. In the initial augmentation study, a maximum offset of 7.50° was used. To see how the maximum angle of the plane offset influences the performance, an ablation study is done with maximum angles of 2.50°, 5.00°, 7.50° and 10.00°. The results of the ablation study are given in Table 4.11. It can be seen that the increase of the angle variation from 2.50° to 7.50°, decreases the mean validation error by 20.00 %. What is interesting to see is that the highest average success rate is achieved with a maximum angle of 10.00°, although the mean validation errors are very similar.

Table 4.11: Ablation study of different maximum angles for the *Plane offset* augmentation. Experiments done using the *Attention* network, on the *Sort shoes* task (Dataset version 2). Trained for 6000 iterations and evaluated every 100 iterations. The last 40 evaluations are used for calculating the statistics. *aSR* success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
$\alpha = \pm 2.5°$	0.192 ± 0.075	0.131 ± 0.055	$\mathbf{4.47 \pm 0.48}$	6.82 ± 0.94	2.5
$\alpha = \pm 5°$	0.192 ± 0.074	0.121 ± 0.049	4.48 ± 0.52	6.56 ± 0.70	2.5
$\alpha = \pm 7.5°$	0.195 ± 0.067	$\mathbf{0.109 \pm 0.051}$	4.51 ± 0.40	6.41 ± 0.77	3.7
$\alpha = \pm 10°$	$\mathbf{0.181 \pm 0.064}$	0.111 ± 0.054	4.53 ± 0.52	$\mathbf{6.25 \pm 0.75}$	**6.2**

4.8 Randomization Ablations

In this section, the influence of each of the randomization methods, described in Section 3.4, on the performance of the networks is evaluated. We analyze the influence of randomization of the camera position, the camera parameters, background textures, object textures, as well as scene lighting conditions. The experiments are performed on *Version 2* of the *Sort shoes* dataset, the same as in the previous chapter, but with different observations rendered from the same scenes. During training, an optimized set of augmentations is applied to the observations. The settings of the augmentations are given in Table A.1. The following augmentations are used:

- Applied to every sample: *Color augmentation*, *Color noise*, *Plane offset*, *Location noise*, *Crop jitter*
- Applied with apply probability <50.00 %: *Downsampling*, *Perlin noise*, *Salt-and-pepper noise*, *Gaussian smoothing*

The results of the ablation study for the randomization methods are shown in Table 4.12. It can be seen that disabling individual randomization methods has generally less of an effect than disabling important augmentations, such as *Location noise*, during training. Even with no randomizations applied, the mean validation error increases only by 16.00 %. It is especially interesting to see, that not randomizing the camera position has such little impact on the results. Randomizing the camera position is reported to be one of the most important randomizations for sim-to-real transfer in the literature [10], [11], [17], [24], [30], [49]. The small effect of randomizing the camera position in this case could be explained by the top-down projection of the pointclouds used in the Transporter Networks system. This seems to be a key component for making the model largely viewpoint independent. Another contributing factor could be the employed augmentations, like *Plane offset* that potentially help compensate the missing variety in viewpoints by adding variety to the pointclouds.
Another interesting observation is that randomizing the camera parameters seems to have a negative effect on the performance on the real-world dataset. Disabling this randomization, slightly improves the observed performance on the validation dataset.

Table 4.13 shows the results of the domain randomization ablation study for the *Pick-*

Table 4.12: Ablation study with one, none, or all randomizations disabled in each experiment, using the *Attention* network. Trained on the *Sort shoes* task (Dataset version 2). Network trained for 6000 iterations and evaluated every 100 iterations. The last 40 evaluations are used for the error and loss statistics. *aSR* success limit: 3.00 cm. Measurements are given as mean ± std.

Experiment	Test Error [m]	Validation Error [m]	Test Loss	Validation Loss	Valid. aSR [%]
All randomizations	0.177 ± 0.065	0.107 ± 0.054	$\mathbf{3.98 \pm 0.44}$	6.31 ± 0.78	6.5
No background	0.194 ± 0.071	0.108 ± 0.052	4.12 ± 0.51	6.27 ± 0.81	5.3
No camera param.	0.193 ± 0.072	$\mathbf{0.102 \pm 0.056}$	4.11 ± 0.47	6.28 ± 0.90	**7.8**
No camera positions	$\mathbf{0.173 \pm 0.066}$	0.117 ± 0.055	4.00 ± 0.42	6.22 ± 0.86	3.2
No light	0.179 ± 0.067	0.107 ± 0.059	4.08 ± 0.44	$\mathbf{6.10 \pm 0.64}$	6.7
No textures	0.187 ± 0.070	0.109 ± 0.052	4.11 ± 0.48	6.19 ± 0.87	3.9
No randomizations	0.184 ± 0.072	0.127 ± 0.060	4.22 ± 0.50	6.80 ± 0.81	3.9

angle network. For the ablation study, the networks are trained for 3000 iterations and evaluated every 100 iterations. The first 500 iterations are excluded from the evaluation to remove the influence of the initial convergence phase.
It can be seen that the influence of the individual randomization methods is minimal. In every evaluated case the mean error is below 10.00° and the *aSR* is above 98.00 %. The lowest mean error is achieved with *All randomizations* activated. The results with deactivated randomizations, are all very close and the difference is not significant, due to the high standard deviation in relation to the mean value. Even with no randomization activated, the network achieves a success rate of 100.00 %.

4.9 Final Pick-and-Place Results

For a final performance evaluation, the *Attention* network, the *Pick-angle* network and the *Transport* network are trained using the best augmentation and randomization settings found in the ablation studies. Each of the networks is trained separately and they are combined into the final *Tansporter Agent* system for evaluation. This allows us to save the weights at an optimal iteration for each network. For the evaluation, the weights for each network are loaded and the error of the Transporter Agent is measured on the whole validation dataset. As before, the training is done with a batch size of 4, a learning rate of $1e^{-4}$, and a weight decay rate of $1e{-}4$. Unlike before, the networks are validated using 20 samples from the validation dataset every 100 iterations, instead of only using 10 samples. The smaller sample size during the ablation studies was chosen to reduce the overhead of the evaluation and is now increased to get a more accurate estimate of the error. For the data augmentation settings, the parameters are set to the optimal values found in the augmentation studies. The parameters for randomization settings and augmentation settings are given in Table B.1 and Table A.3.
It is important to note that the error of the pick-angle and the Transport network is dependent on the performance of the Attention network. This is because the Attention network is used to determine the position of the crop that is used as input in the pick-angle

Table 4.13: Ablation study with one, none, or all randomizations disabled in each experiment, using the *Pick angle* network. Trained on the *Sort shoes* task (Dataset version 2). Trained for 3000 iterations and evaluated every 100 iterations. The last 25 evaluations are used for the statistics. aSR success limit: 10.00°. Measurements are given as mean ± std.

Experiment	Test Error [°]	Validation Error [°]	Test Loss	Validation Loss	Valid. aSR [%]
All randomizations	3.79 ± 1.75	$\mathbf{4.70 \pm 1.09}$	0.79 ± 0.25	$\mathbf{1.06 \pm 0.17}$	100.0
No background	3.68 ± 1.91	6.17 ± 1.32	0.77 ± 0.31	1.23 ± 0.20	98.1
No camera param.	3.84 ± 1.75	4.85 ± 1.09	0.77 ± 0.26	1.07 ± 0.22	100.0
No camera positions	3.88 ± 1.96	4.96 ± 1.13	0.77 ± 0.26	$\mathbf{1.06 \pm 0.17}$	100.0
No light	3.65 ± 1.84	5.42 ± 1.03	0.76 ± 0.26	1.12 ± 0.12	100.0
No textures	$\mathbf{3.46 \pm 1.32}$	4.84 ± 1.28	0.78 ± 0.25	1.08 ± 0.13	100.0
No randomizations	3.88 ± 1.69	6.70 ± 1.09	$\mathbf{0.73 \pm 0.19}$	1.53 ± 0.27	**100.0**

and Transport networks.
The final evaluation is done on the whole validation dataset, instead of using only a subset. The error reported for the final evaluation is therefore expected to be different from the error reported in the ablation studies. Instead of the average success rate (aSR), the success rate (SR) on the whole validation dataset is now reported. The conditions for a successful grasp are the same as for the ablation studies. The conditions for successful placing are chosen to be more lenient, because for the *Sort shoes* task, the spacing between the shoes allows for a larger error in the place position and place angle. A successful grasp is counted when the following conditions are met:

- The predicted pick position is within 3.00 cm of the ground-truth pick position.
- The predicted pick angle is within 10.00° of the ground-truth pick angle.
- The predicted place position is within 5.00 cm of the ground-truth place position.
- The predicted place angle is within 15.00° of the ground-truth place angle.

4.9.1 Box Picking Task

To evaluate the *Box picking* task, we use the recorded validation dataset, containing 20 scenes with one of the two tea-boxes each. The used boxes are shown in Figure 4.13. The results for both boxes are nearly identical.

From the results of the evaluation, given in Table 4.14, it can be seen that the achieved errors are very low. We achieve a mean pick position error of 6.00 mm and a mean pick angle error of just 2.88°. Considering, we used a pixel size of 3.125 mm for all experiments, the mean error for the pick position, across the whole dataset, is below 2 px. This error is well within the margin of error of the manual pointcloud annotation.

4.9.2 Sort Shoes Task

Compared to the *Box picking* task, the *Sort shoes* task is more challenging. It requires the networks to generalize to unseen objects and bridge the sim-to-real gap. This generalization is especially challenging, because the training dataset is limited to only 20 different shoe

(a) Model 1 (b) Model 2

Figure 4.13: The two tea boxes used for evaluation of the *Box picking* task.

Table 4.14: Evaluation results of the *Attention-* and *Pick angle* networks on the complete *Box picking* validation dataset. Evaluated on the best validation iteration of 3 training runs with 5000 steps. Limits for pick success are 3.00 cm and 10.00°.

Box	**Pick Position Error [m]**	**Pick Angle Error [°]**	**SR Pick [%]**
Model 1	0.007 ± 0.004	$\mathbf{2.75 \pm 3.70}$	**100.0**
Model 2	$\mathbf{0.006 \pm 0.004}$	3.00 ± 3.67	**100.0**
Overall	0.006 ± 0.004	2.88 ± 3.69	**100.0**

models.
For training the networks in this final evaluation study, the *No Camera Parameters* dataset is used. The results of the randomization ablation study indicate a slight performance benefit for this dataset compared to the *All Randomizers* dataset. During the evaluation, it was found that the errors vary a lot between the different shoe models in the validation dataset, shown in Figure 4.14. The results for the individual shoe models from the validation dataset, along with the overall mean results, are presented in Tables 4.15–4.17. Looking at the results, it can be seen that all networks show their best performance on *Model 2* and *Model 4*. The errors on these models are even comparable to the errors on the tea-boxes. The mean pick position errors are 9 mm and 6 mm respectively. While the and the mean pick angle errors are 5.7° and 2.2° for both models. An estimated pick-and-place success rate of 85 % is achieved on both of these shoe-models. The highest error across all networks is observed on *Model 3*, resulting in a success rate of only 29 %.

(a) Model 1 (b) Model 2 (c) Model 3 (d) Model 4

Figure 4.14: The four shoe models used for evaluation of the *Shoe sorting* task.

What is also notable, is that the Transport network shows a higher error than the pick networks. Especially the angle error is significantly higher. This is likely due to the fact, that the Transport network is trained to jointly predict the place position and angle. This is expected to be a more difficult problem to solve.
The overall success rate of the *Transporter Agent* across all shoe models is 68 %.

Table 4.15: Evaluation results of the *Attention-* and *Pick angle*-network, trained on the *No Camera param* dataset, with optimized augmentations. Validated on the whole *Sort shoes* validation dataset. Evaluated on the best validation iteration of 3 training runs with 6000 steps. Limits for pick success are 3.00 cm and 10.00°.

Shoe	Pick Position Error [m]	Pick Angle Error [°]	SR Pick [%]
Model 1	0.113 ± 0.173	15.71 ± 24.56	71.4
Model 2	0.009 ± 0.004	5.71 ± 1.75	85.7
Model 3	0.220 ± 0.248	27.86 ± 36.34	57.1
Model 4	$\mathbf{0.006 \pm 0.004}$	$\mathbf{2.14 \pm 2.47}$	**100.0**
Overall	0.087 ± 0.175	12.86 ± 24.14	78.6

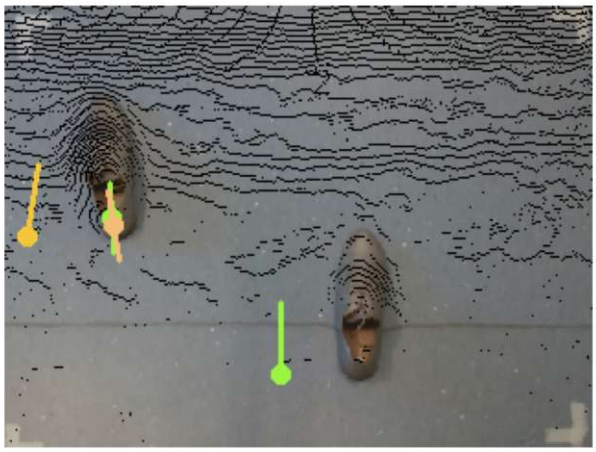
Table 4.16: Evaluation results of the *Transporter*-network, trained on the *No Camera param* dataset, with optimized augmentations. Validated on the whole *Sort shoes* validation dataset. Evaluated using the best validation iteration of 3 training runs with 10000 steps. Limits for place success are 5.00 cm and 15.00°.

Shoe	Place Position Error [m]	Place Angle Error [°]	SR Place [%]
Model 1	0.055 ± 0.103	13.57 ± 25.17	85.7
Model 2	$\mathbf{0.018 \pm 0.012}$	$\mathbf{4.29 \pm 3.19}$	**100.0**
Model 3	0.370 ± 0.233	35.00 ± 34.43	28.6
Model 4	0.064 ± 0.125	15.71 ± 28.46	85.7
Overall	0.127 ± 0.200	17.14 ± 28.01	75.0

Failure Modes

Figure 4.15 shows the most common failure modes of the *Transporter Agent* on the validation dataset. It can be seen that the most common failures are caused by the agent mistaking the left and right shoe. This most often results in the agent picking the wrong shoe, and placing it besides the shoe that should actually be picked. The second most common failure mode is the agent picking the correct shoe, but predicting the place position next to the picked shoe.
All in all, the results and failure modes indicate that the *Transporter Agent* is able to pick

(a) Pick failure: Wrong shoe, otherwise successful pick. Place predicted correctly, even with wrong pick.



(b) Place predicted to wrong shoe, otherwise would be successful.

Figure 4.15: Examples of the most common failure modes on the validation dataset of the *Pick-and-place* task. Green is the ground-truth, orange are the predicted poses.

Table 4.17: Success rates of the full *Transporter Networks* agent, trained on the *No Camera param* dataset, with optimized augmentations. Validated on the whole *Sort shoes* validation dataset. Evaluation is done using the same trained models as above. Combined success is only counted if both the pick and place are successful. Limits for pick success are 3.00 cm and 10.00°, for place success 5.00 cm and 15.00°.

Shoe	SR Pick [%]	SR Place [%]	SR Combined [%]
Model 1	71.4	85.7	71.4
Model 2	85.7	**100.0**	**85.7**
Model 3	57.1	28.6	28.6
Model 4	**100.0**	85.7	**85.7**
Overall	78.6	75.0	67.9

and place shoes with a high success rate. The failures are more often caused by the agent mistaking the left and right shoe, than by the agent failing to predict a suitable pick or place location.

The results of this final evaluation study show that the sim-to-real transfer methods, developed in this thesis, enable an agent trained only using simulated data to successfully predict pick-and-place poses on real-world data. This further shows, that the extended Transporter Networks agent is able to generalize well to unseen objects, even when the training-dataset size is limited.

4.9.3 Distractor Objects

To evaluate the performance of the *Transporter Agent* in cluttered environments, we train and evaluate the agent on datasets that contains distractor objects. We evaluate the performance of both, the agent trained with distractor objects and the agent trained without distractor objects, on both datasets. The results are given in Table 4.18. It can be seen that the addition of distractor objects to the scenes leads to a significant decrease of the overall success rate. Even when trained with distractor objects, the overall success rate is only 55 % on the distractors validation set.
It is interesting to note, that, on the distractors dataset, the pick success rate of the agent trained without distractor objects is only slighlty lower than on the no distractors validation set. The pick success rate of the agent trained without distractors is even higher than the pick success rate of the agent trained with distractor objects.
This lower pick success rate is also what limits the overall success rate of the agent trained with distractor objects. The place success rate, when trained with distractor objects, is the same as for the agent trained and evaluated without distractor objects, at 75.00 %.

4.9.4 Evaluation of Original Implementation

To validate our work against a baseline, we also evaluate the original implementation of the Equivariant Transporter Networks [36]. The networks are trained using the *No*

Table 4.18: Success rates of the *Transporter Networks* agent, trained with and without distractor objects. Validated on a validation set with and without distractors. Evaluated using the best validation iteration of 3 training runs (pick: 6000 steps, place: 10000 steps). Combined success is counted if pick and place are successful. Limits for pick success are 3.00 cm and 10.00°, for place success 5.00 cm and 15.00°.

Training set	Validation set	SR Pick [%]	SR Place [%]	SR Combined [%]
No distractors	No distractors	**78.6**	**75.0**	**67.9**
With distractors	No distractors	67.9	53.6	46.4
No distractors	With distractors	65.0	55.0	50.0
With distractors	With distractors	60.0	**75.0**	55.0

Randomizers dataset. The evaluation is done on a true baseline, as no additional data augmentation or randomization is used.
The results, given in Table 4.19–4.21, show that without methods to bridge the sim-to-real gap, the performance of the *Transporter Agent* is significantly worse. The overall success rate is only 10.7 %. It must however be noted, that the pick pose predictions are still surprisingly accurate, with a mean pick success rate of 57.1 %. This indicates good generalization capabilities of the *Attention* and *Pick-angle* networks. The *Transport* network, however, shows a very low success rate of only 21.4 %. It is interesting to note, that the shoe Model 3, now shows the best performance, with a place success rate of 42.9 %. This is in contrast to the results of the extended Transporter Agent, where Model 3 had the lowest performance.
The results of the original implementation show that the methods developed in this thesis are necessary to achieve successful sim-to-real transfer.

Table 4.19: Evaluation results of the unmodified *Attention*- and *Pick angle*-network. Trained without augmentations and without domain randomization. Evaluated using the best validation iteration of 3 training runs of 10000 steps. Batchsize: 1. *Shoe-sorting* validation dataset. Limits for pick success are 3.00 cm and 10.00°.

Shoe	**Pick Position Error [m]**	**Pick Angle Error [°]**	**SR Pick [%]**
Model 1	$\mathbf{0.125 \pm 0.171}$	15.71 ± 24.56	**71.4**
Model 2	0.189 ± 0.214	28.57 ± 28.37	57.1
Model 3	0.179 ± 0.215	$\mathbf{15.00 \pm 30.71}$	42.9
Model 4	0.158 ± 0.187	26.43 ± 30.67	57.1
Overall	0.160 ± 0.200	21.43 ± 29.33	57.1

Table 4.20: Evaluation results of the unmodified *Transporter*-network. Trained without augmentations and without domain randomization. Evaluated using the best validation iteration of 3 training runs of 14000 steps. Batchsize: 1. *Shoe-sorting* validation dataset. Limits for place success are 5.00 cm and 15.00°.

Shoe	Place Position Error [m]	Place Angle Error [°]	SR Place [%]
Model 1	0.296 ± 0.183	36.43 ± 16.63	0.0
Model 2	0.214 ± 0.128	40.71 ± 24.26	14.3
Model 3	$\mathbf{0.167 \pm 0.159}$	40.00 ± 32.40	**42.9**
Model 4	0.215 ± 0.196	$\mathbf{34.29 \pm 23.36}$	28.6
Overall	0.223 ± 0.175	37.86 ± 24.94	21.4

Table 4.21: Evaluation results of the complete, default *Transporter Networks*-agent. Trained without augmentations and without domain randomization. Evaluated using the best validation iteration of 3 training runs of 10000 steps(pick) and 12000 steps(place). *Shoe-sorting* validation dataset. Limits for pick success are 3.00 cm and 10.00°. Limits for place success are 5.00 cm and 15.00°. Combined success is only counted if both pick and place are successful.

Shoe	SR Pick [%]	SR Place [%]	SR Combined [%]
Model 1	**71.4**	0.0	0.0
Model 2	57.1	14.3	**14.3**
Model 3	42.9	**42.9**	**14.3**
Model 4	57.1	28.6	**14.3**
Overall	60.7	21.4	10.7

5 Discussion

In this chapter, the findings and their relation to previous research are discussed and the limitations of this study are highlighted.

5.1 Sim-to-Real Transfer with Transporter Networks

To the best of our knowledge, this is the first work that uses a *Transporter Network* architecture in a sim-to-real transfer setting. Although previous work, did use the Transporter Networks architecture on real-world data, the training data was then also recorded in the real-world [9], [36], so no domain gap had to be overcome. The results, presented in Section 4.9, show that the Transporter Networks architecture is able to successfully bridge the sim-to-real gap. We showed successful generalization to new objects and scenes, even when trained with a comparatively small synthetic dataset.

5.1.1 Generalization to Unseen Viewpoints

From the results of the domain randomization ablation study, it can be seen that the top-down projection mechanism employed in the Transporter Network architecture is very effective. The projection of the pointclouds to a top-down view, in conjunction with applying random transformations to the input pointclouds, enables the model to become largely viewpoint invariant. Even when recording all training samples from a fixed camera position, the model successfully transfers to unseen camera positions in the validation data. It can further be seen that randomizing camera parameters, such as the focal length or distortion parameters, has a negative effect on the performance, indicating that an approximated model of the used camera is sufficient for good transfer performance.

5.1.2 Batched Training

The main change to the original Transporter Networks architecture is the introduction of batched training and parallel data preprocessing. As shown in Section 4.5.2, especially the pick angle prediction network benefits from the introduction of batched training. Due to the high memory requirements during gradients calculation of the Equivariant networks used in the architecture, the batch size is limited to 4. Tests with larger batch sizes on a GPU with more memory could potentially lead to further performance improvements.
The introduction of parallel data preprocessing, when used with batched training, allows for a significant speedup of the training process.

5.1.3 Distractor Objects

Especially in household environments, clutter and objects unrelated to the task at hand are often present. The results show, that especially the pick pose prediction system struggles

with the presence of distractor objects. Even when trained with distractor objects, the pick pose prediction is significantly worse when distractor objects are present in the validation data. This is a significant limitation for real-world application of the trained models.

5.2 Sim-to-Real Transfer of RGB-Pointclouds

In previous work in the field of domain randomization, often either color information or depth information is used [10], [25]. In this thesis, we use both modalities with the Transporter Networks architecture. The results indicate that, for the selected range of tasks, where the visual appearance of the objects of interest are not important, the networks rely mostly on depth information for pick- and place-pose prediction. In our case, the addition of color randomizations or augmentations shows no significant benefit. This is in stark contrast to other works that only used color images for training, where texture randomizations and augmentations are shown to be crucial for sim-to-real transfer [10], [28], [49]. This could potentially change, when the task is appearance dependent, e.g., when the model needs to learn to distinguish between different objects based on color or texture information. In our experiments however, the appearance of the objects was not important, as the model only needed to learn the spatial relations between the objects. This indicates that if the task is appearance agnostic, the addition of color information to the depth data for successful operation of Transporter Networks might not be necessary. The results of the data augmentation ablation study show that augmenting the spatial information is crucial for the performance of the model on real-world data. The addition of random Gaussian noise to the depth data shows to be crucial for the performance of the model on real-world data. Additionally, the results indicate that accounting for alignment errors in the real-world data, by adding random axial rotations to the training data, is beneficial for the performance of the model on real-world pointclouds.

5.3 Domain Randomization with Nvidia Omniverse

For synthetic data generation, the *Nvidia Omniverse* platform was used in this thesis. Compared to other publications in the field of domain randomization research, that often use low fidelity simulators[10], [12], [56], the Omniverse platform enables high-quality rendering of scenes. Robotics and domain randomization extensions makes this platform a powerful tool for generating synthetic data for sim-to-real transfer.
Despite that, for the tasks used in this thesis, the rendering quality was not required to be very high, the platform is capable of generating high-quality images and pointclouds. This could be beneficial for tasks where the visual appearance of the objects is important or the objects of interest have material properties that are difficult to render, like very reflective or translucent materials. Although the Omniverse platform offers a range of beneficial features, it is still in an early stage of development. We experienced multiple breaking changes to the API during the development of this thesis and experienced several issues with the stability of the platform.

5.4 Limitations

The identified limitations of this work are discussed in the following.

5.4.1 Dataset Size

As with all machine learning approaches, the size of the training dataset is also crucial for the sim-to-real performance in a domain randomization context. Many works in the field of domain randomization report that the size of the training dataset directly influences the performance of the model on unseen data [28], [29]. Datasets in the domain randomization literature often contain multiple thousands of samples for the objects of interest. In our case, the training dataset is generated from only 20 individual 3D models.
It is unclear if the transfer performance is limited due to the small dataset size or due to limitations in the augmentations and network capabilities.

5.4.2 Manual Annotation

The manual annotation of the pick and place pose for the validation dataset is done using a top-down view of the scene. Because the annotation is done manually, by visually approximating the desired pick and place pose, the annotation is likely not perfect. This introduces an unknown error in the validation dataset.

5.4.3 Lack of Real-World Experiments

Due to time and resource constraints, the experiments and evaluations for this thesis were conducted using an annotated validation dataset only. As explained above, the annotation was done manually, which naturally introduces a small amount of error. Although we expect, that the success rates on this dataset give a good indication of the performance in real-world scenarios, it is unclear how well the calculated success rates translate to real-world experiments. To evaluate the performance of the trained model, it would be interesting to conduct experiments using a real robot.

5.4.4 Transfer to Other Tasks

The two tasks used in this thesis are chosen for their relative simplicity and relevance as a common household assistance task. What both of these tasks have in common is that they are not dependent on visual object properties like color or texture. The network is explicitly trained to ignore the visual appearance of the objects and only learn the spatial relations between them. It would be interesting to see how well the findings of this thesis translate to other tasks where the visual appearance of the objects is important to successfully solve the task.

6 Conclusion and Future Work

In this thesis, we analyzed different state-of-the-art domain randomization and data augmentation techniques for sim-to-real transfer of pick-and-place tasks. The *Transporter Network* architecture [9] is used as the framework for the pick-and-place algorithm. This is the first work that uses a *Transporter Network* architecture in a simulation-to-reality transfer setting. Specifically, we used the *Equivariant Transporter Networks* architecture [36] that improves sample efficiency from the original paper and contains a network for pick angle prediction. For this thesis, we extended the architecture with the following improvements:

- Adapting the training to use batched computations, which allows for faster training and better performance.
- Parallelization of data loading and sample augmentation in multiple threads.
- Enabling the use of RGB-Pointclouds as input data.
- Implementation of various new data augmentation techniques.

All training data for this thesis was generated using the *Omniverse Isaac Sim* simulator. Compared to previous works in the field of domain randomization, this allows for high-quality rendering of the training data, while simultaneously implementing a physics simulation. For the synthetic data generation in this thesis, a Synthetic Data Generation pipeline was developed, which allows for the generation of large datasets from imported 3D models, with a variety of randomizations. To evaluate the impact of different randomization and augmentation techniques on the simulation-to-reality transfer performance, performance analysis and ablation studies were performed on two different tasks: *Box pick-up* and *Shoe sorting*. For both tasks, a synthetic training dataset was generated from freely available 3D models. To analyze the transfer performance we recorded and annotated real-world pointclouds, using a developed annotation pipeline.

6.1 Findings

In this thesis, we found that the Transporter Networks architecture, is successfully able to generalize to unseen objects, even from a comparatively small dataset. We further developed a set of effective augmentations and randomizations that allowed the trained network to generalize to real-world pointclouds while being trained exclusively from synthetic data.
Extensive ablation studies on the developed tasks demonstrated that augmenting spatial information is the most important factor for successful simulation-to-reality transfer, when using RGB-pointclouds. Augmenting or randomizing color information does not significantly influence the performance on real-world data.
This finding also translates the domain randomization ablation study, where the randomization of textures and lighting conditions has shown to have no significant effect on the

performance of the model on real-world data.
The results of the ablation studies indicate that for tasks where the color and texture of objects is not important for solving the task, the focus should be on augmenting spatial information rather than color information. Especially the use of random Gaussian noise to augment depth data shows to be crucial for the performance of the model on real-world data. The employed mechanisms in the original publication [9] show to be effective tools for generalization to unseen scenes and viewpoints.
In a final performance evaluation, each network employed in the Transporter Networks system was individually trained and a training iteration with optimal performance was used to measure the error on a real-world dataset. The minimal achieved error was evaluated on the *Shoe sorting* task, where the network has to generalize to unseen objects and simultaneously bridge the sim-to-real gap. For the pick operation, we achieved an estimated success rate between 57.1 % and 100.0 % depending on the shoe model, with an overall success rate of 78.6 %. For the place pose operation, we achieved an estimated success rate between 28.6 % and 100.0 % depending on the shoe model, with an overall success rate of 75.0 %. The success rate for scenes where both, the pick- and place operation, are successful, across all shoe models is 67.9 %.
Failure mode analysis shows that the most common failure mode is the agent mistaking the left and right shoe, which results in the agent picking the wrong shoe and placing it next to the shoe that should actually be picked. This indicates that the model is able to learn the spatial relations between the objects and successfully transfers from simulation to reality, but fails to distinguish between the left and right shoe.

6.2 Future Work

Since the achieved error is within a margin that, as we estimated, would allow at least most of the grasps to be successful, it would be interesting to validate the estimated grasp success rates on a real robot. Integrating the developed algorithm in a *pick-and-place* pipeline and deploying it on a physical system.
Future work, could further train the network on a task with a larger dataset, to evaluate if the performance found in this thesis is limited by the dataset size or by the capabilities of the networks.
Furthermore we found that especially the pick pose prediction network struggles with the presence of distractor objects. It should be investigated if the performance can be improved by modifying the pick pose prediction network.
We would also recommend experimenting with employing the projection of pointclouds into a top-down image of the scene, for algorithms and networks that would usually be fed with RGB-D images directly. This mechanism shows to be greatly beneficial to generalization to new viewpoints. Finally, we would recommend to further investigate the use of the *Nvidia Omniverse* simulator, in conjunction with the developed randomization pipeline and augmentation techniques, for other tasks and network architectures. Data intensive deep learning applications in robotics, like *Deep Reinforcement Learning* or *Pose estimation* could benefit from using the developed methods, for simulation to reality transfer. The methods introduced in this thesis offer potential for future research in the field of simulation to reality transfer and show promising results.

A Augmentation Settings

Table A.1: Augmentation Settings used in the Randomization ablation studies.

Augmentation	**Parameter**	**Value**
Color Noise	sigma	0.012
Color Augmentation	brightness	±0.3
	contrast	±0.3
	saturation	±0.3
	hue	±0.5
	grayscale prob.	0.1
Location Noise	sigma	0.0075 m
Plane Offset	tilt angle	±10.00°
	z-offset	±0.05 m
Color Salt-and-Pepper	salt vs. pepper ratio	0.5
	salt-and-pepper prob.	0.03
	apply prob.	0.85
Depth Salt-and-Pepper	salt vs. pepper ratio	0.5
	salt-and-pepper prob.	0.04
	apply prob.	0.85
Downsample Pointcloud	downsampling factor	0.50 to 1.00
Depth Perlin Noise	scale	0.03 m
	apply prob.	0.85
Color Perlin Noise	scale	0.03 m
	apply prob.	0.85
Randomize Crop Position	max. offset	10.00 px
Gaussian Blur	sigma	0.10 to 1.50
	kernel size	3.00 px to 5.00 px
	apply prob.	0.3

Table A.2: Augmentation Settings used in the Augmentation ablation studies.

Augmentation	**Parameter**	**Value**
Color Noise	sigma	0.025
Color Augmentation	brightness	±0.3
	contrast	±0.3
	saturation	±0.3
	hue	±0.5
	grayscale prob.	0.15
Location Noise	sigma	0.0075 m
Dropout	max. dropouts	6
	size	0.01 m to 0.20 m
	apply prob.	0.33
Plane Offset	tilt angle	±7.50°
	z-offset	±0.03 m
Depth Scaling	scale	0.75 to 1.25
	apply prob.	0.33
Color Salt-and-Pepper	salt vs. pepper ratio	0.5
	salt-and-pepper prob.	0.03
	apply prob.	0.85
Depth Salt-and-Pepper	salt vs. pepper ratio	0.5
	salt-and-pepper prob.	0.04
	apply prob.	0.85
Downsample Pointcloud	downsampling factor	0.50 to 1.00
Depth Perlin Noise	scale	0.04 m
	apply prob.	0.85
Color Perlin Noise	scale	0.04 m
	apply prob.	0.85
Randomize Crop Position	max. offset	10.00 px
Gaussian Blur	sigma	0.10 to 1.50
	kernel size	3.00 px to 5.00 px
	apply prob.	0.3

Table A.3: Augmentation Settings used in the final experiments.

Augmentation	**Parameter**	**Value**
Color Noise	sigma	0.012
Color Augmentation	brightness	±0.3
	contrast	±0.3
	saturation	±0.3
	hue	±0.5
	grayscale prob.	0.1
Location Noise	sigma	0.0075 m
Plane Offset	tilt angle	±10.00°
	z-offset	±0.05 m
Color Salt-and-Pepper	salt vs. pepper ratio	0.5
	salt-and-pepper prob.	0.03
	apply prob.	0.85
Depth Salt-and-Pepper	salt vs. pepper ratio	0.5
	salt-and-pepper prob.	0.04
	apply prob.	0.85
Downsample Pointcloud	downsampling factor	0.50 to 1.00
	apply prob.	0.2
Depth Perlin Noise	scale	0.03 m
	apply prob.	0.85
Color Perlin Noise	scale	0.03 m
	apply prob.	0.85

B Randomization Settings

Table B.1: Randomization settings used for the experiments.

Randomizer	**Parameter**	**Value/Range**
Camera	Resolution	800 px × 600 px
	Focal Length Deviation	±20%
	Focus Distance Deviation	±20%
	F-Stop Deviation	±10%
	Horizontal Aperture Deviation	±10%
	Fisheye Poly Coeff Deviation	±10%
	Fisheye Dim Deviation	±10%
	Position (X)	−0.25 m to 0.25 m
	Position (Y)	0.50 m to 1.25 m
	Position (Z)	0.80 m to 1.50 m
Domelight	Intensity	1000 to 3000
	Rotation (X)	−20° to 20°
	Rotation (Y)	−20° to 20°
	Rotation (Z)	−170° to 170°
Floor material	Roughness	0.30 to 1.00
	Specularity	μ: 0.4, σ: 0.1
	Texture Scale X/Y	μ: 1, σ: 0.2
	Texture Rotation	0.00° to 360.00°
Object material	Roughness	0.20 to 1.00
	Specularity	μ: 0.2, σ: 0.1
	Texture Scale X/Y	μ: 0.05, σ: 0.075
	Noise Scale X/Y	μ: 1, σ: 0.25
	Metallic	0.00 to 1.00

Bibliography

[1] *Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching*. (visited on 01/07/2023).

[2] J. Wu, R. Antonova, A. Kan, et al., „TidyBot: Personalized Robot Assistance with Large Language Models," *Autonomous Robots*, vol. 47, no. 8, pp. 1087–1102, Dec. 2023, ISSN: 0929-5593, 1573-7527. arXiv: 2305.05658 [cs]. (visited on 04/17/2025).

[3] D. Kalashnikov, A. Irpan, P. Pastor, et al., *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*, Nov. 2018. arXiv: 1806.10293 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1806.10293 (visited on 12/11/2022).

[4] L. Pinto and A. Gupta, *Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours*, Sep. 2015. arXiv: 1509.06825 [cs]. (visited on 05/05/2025).

[5] N. Gomes, F. Martins, J. Lima, and H. Wörtche, „Deep Reinforcement Learning Applied to a Robotic Pick-and-Place Application," in Jan. 2021, pp. 251–265, ISBN: 978-3-030-91884-2.

[6] D. A. Klein, B. Illing, B. Gaspers, D. Schulz, and A. B. Cremers, *Hierarchical Salient Object Detection for Assisted Grasping*, Jan. 2017. arXiv: 1701.04284 [cs]. (visited on 05/06/2025).

[7] S. Peng, Y. Liu, Q. Huang, H. Bao, and X. Zhou, *PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation*, Dec. 2018. arXiv: 1812.11788 [cs]. (visited on 05/05/2025).

[8] K. Park, T. Patten, and M. Vincze, „Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 7667–7676. arXiv: 1908.07433 [cs]. (visited on 05/09/2025).

[9] A. Zeng, P. Florence, J. Tompson, et al., *Transporter Networks: Rearranging the Visual World for Robotic Manipulation*, Jan. 2022. arXiv: 2010.14406 [cs]. (visited on 01/06/2023).

[10] 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*, Mar. 2017. arXiv: 1703.06907 [cs]. [Online]. Available: http://arxiv.org/abs/1703.06907 (visited on 06/15/2023).

[11] L. Eversberg and J. Lambrecht, „Generating Images with Physics-Based Rendering for an Industrial Object Detection Task: Realism versus Domain Randomization," *Sensors*, vol. 21, no. 23, p. 7901, Jan. 2021, ISSN: 1424-8220. (visited on 10/22/2023).

[12] F. Sadeghi and S. Levine, *CAD2RL: Real Single-Image Flight without a Single Real Image*, Jun. 2017. arXiv: 1611.04201 [cs]. (visited on 06/21/2023).

[13] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, *Reinforcement Learning with Augmented Data*, Nov. 2020. arXiv: 2004.14990 [cs, stat]. (visited on 11/20/2023).

[14] *NVIDIA Omniverse*. [Online]. Available: https://www.nvidia.com/en-us/omniverse/ (visited on 03/23/2025).

[15] N. Jakobi, „Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis," *Adaptive Behavior*, vol. 6, no. 2, pp. 325–368, Sep. 1997, ISSN: 1059-7123. (visited on 03/11/2025).

[16] B. Foundation, *Blender.org - Home of the Blender project - Free and Open 3D Creation Software*. [Online]. Available: https://www.blender.org/ (visited on 05/05/2025).

[17] S. James, A. J. Davison, and E. Johns, *Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task*, Oct. 2017. arXiv: 1707.02267 [cs]. (visited on 06/20/2023).

[18] K. Perlin, „An image synthesizer," vol. 19, no. 3, 1985.

[19] M. Ani, H. Basevi, and A. Leonardis, „Quantifying the Use of Domain Randomization," in *2020 25th International Conference on Pattern Recognition (ICPR)*, Jan. 2021, pp. 6128–6135.

[20] M. Weisenböhler, B. Hein, and C. Wurll, „On Scene Engineering and Domain Randomization: Synthetic Data for Industrial Item Picking," *Intelligent Autonomous Systems 17 – Proceedings of the 17th International Conference IAS-17. Ed.: I. Petrovic*, p. 643, 2023, ISSN: 2367-3370. (visited on 10/22/2023).

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, „Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 1558-2256. (visited on 03/13/2025).

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, „ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782, 1557-7317. (visited on 03/13/2025).

[23] A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid, *Learning to Augment Synthetic Images for Sim2Real Policy Transfer*, Jul. 2019. arXiv: 1903.07740 [cs]. (visited on 01/04/2025).

[24] J. Mahler, J. Liang, S. Niyaz, et al., *Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics*, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1703.09312v3 (visited on 06/20/2023).

[25] F. Hagelskjaer and A. G. Buch, *Bridging the Reality Gap for Pose Estimation Networks using Sensor-Based Domain Randomization*, Aug. 2021. arXiv: 2011.08517 [cs]. (visited on 10/14/2023).

[26] S. Thalhammer, K. Park, T. Patten, M. Vincze, and W. G. Kropatsch, *SyDD: Synthetic Depth Data Randomization for Object Detection using Domain-Relevant Background*. (visited on 12/12/2024).

[27] S. Hinterstoisser, V. Lepetit, S. Ilic, et al., „Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," in *Computer Vision – ACCV 2012*, D. Hutchison, T. Kanade, J. Kittler, et al., Eds., vol. 7724, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 548–562. (visited on 05/07/2025).

[28] R. Alghonaim and E. Johns, *Benchmarking Domain Randomisation for Visual Sim-to-Real Transfer*, May 2021. arXiv: 2011.07112 [cs]. [Online]. Available: http://arxiv.org/abs/2011.07112 (visited on 06/15/2023).

[29] J. Borrego, A. Dehban, R. Figueiredo, P. Moreno, A. Bernardino, and J. Santos-Victor, *Applying Domain Randomization to Synthetic Data for Object Category Detection*, Jul. 2018. arXiv: 1807.09834 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1807.09834 (visited on 06/15/2023).

[30] D. Horváth, G. Erdős, Z. Istenes, T. Horváth, and S. Földi, „Object Detection Using Sim2Real Domain Randomization for Robotic Applications," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1225–1243, Apr. 2023, ISSN: 1552-3098, 1941-0468. arXiv: 2208.04171 [cs]. (visited on 08/08/2024).

[31] *Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine*. [Online]. Available: https://unity.com (visited on 05/07/2025).

[32] *PhysX SDK - Latest Features & Libraries | NVIDIA Developer*. [Online]. Available: https://developer.nvidia.com/physx-sdk (visited on 05/07/2025).

[33] *Isaac Sim*. [Online]. Available: https://developer.nvidia.com/isaac/sim (visited on 03/23/2025).

[34] *Replicator — Omniverse Extensions*. [Online]. Available: https://docs.omniverse.nvidia.com/extensions/latest/ext_replicator.html (visited on 03/23/2025).

[35] *Model-based 3D pose estimation for pick-and-place application | IEEE Conference Publication | IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/7986888 (visited on 03/23/2025).

[36] H. Huang, D. Wang, R. Walters, and R. Platt, *Equivariant Transporter Network*, Sep. 2022. arXiv: 2202.09400 [cs]. [Online]. Available: http://arxiv.org/abs/2202.09400 (visited on 09/14/2023).

[37] H. Huang, *HaojHuang/Equivariant-Transporter-Net*, Jan. 2025. [Online]. Available: https://github.com/HaojHuang/Equivariant-Transporter-Net (visited on 05/03/2025).

[38] P. M. Radiuk, „Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," Dec. 2017. [Online]. Available: https://elar.khmnu.edu.ua/handle/123456789/11047 (visited on 03/16/2025).

[39] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Mar. 2015. arXiv: 1502.03167 [cs]. (visited on 03/16/2025).

[40] *Pickle — Python object serialization*. [Online]. Available: https://docs.python.org/3/library/pickle.html (visited on 03/25/2025).

[41] *The PLY Polygon File Format*, Dec. 2016. [Online]. Available: https://web.archive.org/web/20161204152348/http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/ply.html (visited on 02/28/2025).

[42] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, „MeshLab: An Open-Source Mesh Processing Tool,“

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting,“

[44] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, Dec. 2015. arXiv: 1512.03385 [cs]. (visited on 03/25/2025).

[45] B. J. Kim, H. Choi, H. Jang, D. Lee, and S. W. Kim, *How to Use Dropout Correctly on Residual Networks with Batch Normalization*, Feb. 2023. arXiv: 2302.06112 [cs]. (visited on 03/25/2025).

[46] J. Tobin, L. Biewald, R. Duan, et al., *Domain Randomization and Generative Models for Robotic Grasping*, Apr. 2018. arXiv: 1710.06425 [cs]. (visited on 06/18/2023).

[47] Q.-Y. Zhou, J. Park, and V. Koltun, *Open3D: A Modern Library for 3D Data Processing*, Jan. 2018. arXiv: 1801.09847 [cs]. (visited on 02/28/2025).

[48] *Sensor Assets — Isaac Sim 4.2.0 (OLD)*. [Online]. Available: https://docs.omniverse.nvidia.com/isaacsim/latest/features/environment_setup/assets/usd_assets_sensors.html#intel-realsense-d455 (visited on 03/23/2025).

[49] J. Tremblay, A. Prakash, D. Acuna, et al., *Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization*, Apr. 2018. arXiv: 1804.06516 [cs]. (visited on 11/24/2023).

[50] T.-Y. Lin, M. Maire, S. Belongie, et al., *Microsoft COCO: Common Objects in Context*, Feb. 2015. arXiv: 1405.0312 [cs]. (visited on 01/21/2025).

[51] *Noise Maker*. [Online]. Available: http://kitfox.com/projects/perlinNoiseMaker/ (visited on 03/04/2025).

[52] *Free Stock Textures*. [Online]. Available: https://freestocktextures.com/ (visited on 03/04/2025).

[53] *Texture Ninja*. [Online]. Available: http://textureninja.com (visited on 03/04/2025).

[54] *3D TEXTURES*, Dec. 2017. [Online]. Available: https://3dtextures.me/ (visited on 03/04/2025).

[55] *Poly Haven*. [Online]. Available: https://polyhaven.com/ (visited on 03/04/2025).

[56] S. Grushko, A. Vysocký, J. Chlebek, and P. Prokop, *HaDR: Applying Domain Randomization for Generating Synthetic Multimodal Dataset for Hand Instance Segmentation in Cluttered Industrial Environments*, Apr. 2023. arXiv: 2304.05826 [cs]. (visited on 08/08/2024).

[57] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, „Indoor Segmentation and Support Inference from RGBD Images,“ in *Computer Vision – ECCV 2012*, D. Hutchison, T. Kanade, J. Kittler, et al., Eds., vol. 7576, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 746–760. (visited on 03/08/2025).

[58] D. Pasanisi, E. Rota, M. Ermidoro, and L. Fasanotti, „On Domain Randomization for Object Detection in real industrial scenarios using Synthetic Images," *Procedia Computer Science*, 4th International Conference on Industry 4.0 and Smart Manufacturing, vol. 217, pp. 816–825, Jan. 2023, ISSN: 1877-0509. (visited on 03/08/2025).

[59] P. Debevec, *Image-Based Lighting*, Mar. 2002. [Online]. Available: `https://ict.usc.edu/pubs/Image-Based%20Lighting.pdf` (visited on 03/08/2025).

[60] S. Z. Valtchev and J. Wu, „Domain randomization for neural network classification," *Journal of Big Data*, vol. 8, no. 1, pp. 1–12, Dec. 2021, ISSN: 2196-1115. (visited on 03/08/2025).

[61] *HDRI Skies*. [Online]. Available: `https://hdri-skies.com/` (visited on 03/08/2025).

[62] K. B. da Cunha, C. Brito, L. Valença, L. Figueiredo, F. Simões, and V. Teichrieb, „The impact of domain randomization on cross-device monocular deep 6DoF detection," *Pattern Recognition Letters*, vol. 159, pp. 224–231, Jul. 2022, ISSN: 0167-8655. (visited on 03/11/2025).

[63] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, „Benchmarking in Manipulation Research: The YCB Object and Model Set and Benchmarking Protocols," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, Sep. 2015, ISSN: 1070-9932. arXiv: `1502.03143 [cs]`. (visited on 05/13/2025).

[64] X. Pang, W. Xia, Z. Wang, et al., *Depth Helps: Improving Pre-trained RGB-based Policy with Depth Information Injection*, Aug. 2024. arXiv: `2408.05107 [cs]`. (visited on 04/23/2025).

[65] S. Marcel and Y. Rodriguez, „Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10, New York, NY, USA: Association for Computing Machinery, Oct. 2010, pp. 1485–1488, ISBN: 978-1-60558-933-6. (visited on 03/13/2025).

[66] S. Dodge and L. Karam, *Understanding How Image Quality Affects Deep Neural Networks*, Apr. 2016. arXiv: `1604.04004 [cs]`. (visited on 03/13/2025).

[67] T. DeVries and G. W. Taylor, *Improved Regularization of Convolutional Neural Networks with Cutout*, Nov. 2017. arXiv: `1708.04552 [cs]`. (visited on 03/13/2025).

[68] P. Vigier, *Pvigier/perlin-numpy*, Mar. 2025. [Online]. Available: `https://github.com/pvigier/perlin-numpy` (visited on 03/14/2025).

[69] *Sketchfab*. [Online]. Available: `https://sketchfab.com/feed` (visited on 03/16/2025).

[70] *Fab*. [Online]. Available: `https://www.fab.com/` (visited on 03/16/2025).

[71] *CGTrader*. [Online]. Available: `https://www.cgtrader.com/` (visited on 03/16/2025).

[72] *Scanned Objects by Google Research*. [Online]. Available: `https://fuel.gazebosim.org/1.0/GoogleResearch/fuel/collections/Scanned%20Objects%20by%20Google%20Research` (visited on 03/16/2025).

[73] *Depth Camera D435*. [Online]. Available: `https://www.intelrealsense.com/depth-camera-d435/` (visited on 03/01/2025).

[74] *IntelRealSense/librealsense/tools/realsense-viewer*. [Online]. Available: `https://github.com/IntelRealSense/librealsense/tree/master/tools/realsense-viewer` (visited on 03/18/2025).

[75] *D400 Series Visual Presets*. [Online]. Available: `https://github.com/IntelRealSense/librealsense/wiki/D400-Series-Visual-Presets` (visited on 03/18/2025).

[76] M. A. Fischler and R. C. Bolles, „Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,“ *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. (visited on 03/18/2025).

[77] P. 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, Feb. 1992, ISSN: 1939-3539. (visited on 03/18/2025).

[78] A. Krogh and J. A. Hertz, „A Simple Weight Decay Can Improve Generalization,“

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct - Regeln zur Sicherung guter wissenschaftlicher Praxis, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.
Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 19. Mai 2025

Unterschrift: [redacted]

Thomas Schönhofer