Faculty of Electrical Engineering, Computer Science
and Information Technology Osijek,
J. J. Strossmayer University of Osijek, Croatia

MATEJ DŽIJAN

Furniture detection in indoor scenes trained on synthetic 3D data and classification of their opening mechanisms

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Science, scientific area of Technical Sciences,
scientific field of Computer Science at J. J. Strossmayer University of Osijek

Osijek, Croatia, August 16, 2024

Furniture detection in indoor scenes trained on synthetic 3D data and classification of their opening mechanisms

Prepoznavanje namještaja u unutarnjem prostoru učenjem na sintetičkim 3D podacima i klasifikacija načina njegovog otvaranja

Matej Džijan, © August, 2024.

MENTOR

prof. Robert Cupec, Ph.D., Faculty of Electrical Engineering, Computer Science and Information Technology Osijek, J. J. Strossmayer University of Osijek, Croatia

Acknowledgements

"It is important to draw wisdom from many different places. If you take it from only one place, it becomes rigid and stale."

—Iroh

No man is an island, and neither am I. This thesis is not only the product of my work, it was possible thanks to the help of many people. I would like to try to show my gratitude by acknowledging some of them here.

First of all, I would like to thank my great mentor professor Robert Cupec. Thank you for guiding me through this process and for teaching me how to approach research. And thank you for your patience, I know I have been tardy at times, but you stuck with me and now we're here. I hope we can continue to work together on future research.

I would also like to thank some of my colleagues who are more than just that. My greatest friend and office mate Valentin Šimundić, thank you for all the good times. Thank you for boosting my confidence and reminding me of my capabilities when I doubted myself. Thank you, Petra Pejić, for being kind of a big sister of our work group, guiding me through the intricacies of the academia and making me do all the extracurricular stuff I often didn't feel like doing. Thank you, Marin Benčević, for sanity-checking my ideas and allowing me to do the same for yours; you often made me feel valued and useful. And thank you to Ratko Grbić, Ivan Vidović and Mateja Pul for being great colleagues and collaborators. It's been a pleasure working with you all and I hope we continue to do so.

I would like to acknowledge FERIT for this opportunity and the Croatian Science Foundation for supporting my work under the project IP-2019-04-6819.

And thank you to my non-work friends that have provided support when I needed it and for hanging out with me when I just needed to relax and take my mind off work. Thank you, Filip Akrap, for all the distractions, even though we sometimes went overboard with those.

Last, but certainly not least, I would like to thank my amazing parents, Slavica and Tunjo, and my siblings, Magdalena, Perica and Marko. You provided me with everything I ever needed, and your sacrifices have laid the foundation for all my achievements. You've instilled curiosity and thirst for knowledge in me and those make life so much more fun.

To everyone mentioned here and the many others who supported me along the way, your contributions have made this journey more enjoyable and fun.

Matej Džijan
August, 2024.

Abstract

Intelligent service robots are increasingly being deployed in households, hospitals, warehouses, and other environments. These robots are capable of performing complex tasks, largely relying on their vision systems for object detection. As these robots operate and interact in 3D space, data obtained with 3D sensors is crucial for object detection, as it offers a detailed representation of the environment in the form of point clouds. These point clouds allow for precise extraction of information about the position, size, and geometry of objects within the 3D environment. Examples of such sensors include LiDARs and RGB-D cameras. State-of-the-art 3D object detection methods are predominantly deep learning-based and, as such, require large amounts of annotated data to perform effectively. However, acquiring such data can be both expensive and time-consuming. As a result, synthetic data generation presents a promising alternative, but current object detectors trained on synthetic data still underperform compared to those trained on real data. This suggests that current synthetic data generation methods do not yet achieve sufficient realism.

This thesis aims to address this issue by exploring the field of realistic synthetic 3D data generation for indoor environments. The primary goal is to analyze the impact of five key factors of realism: presence of background objects, camera noise, positioning of objects, context of scenes, and object sizes. To investigate these factors, a modular method for generating realistic synthetic single-view point clouds was developed. This method allows for the generation of large, customizable datasets with varying levels of realism, specifically controlling for the aforementioned factors.

These datasets are used to train state-of-the-art object detectors, and the impact of each realism factor is evaluated based on the detection performance. Furthermore, the experiments show that the performance of an object detector can be improved by pre-training it on a baseline synthetic dataset and fine-tuning it on real data. Notably, the model trained on geometric data only using this approach outperforms the same object detector trained solely on real data, which uses both geometric and color data.

In addition to detecting objects, a service robot needs the ability to interact with objects in their environment. One particular challenge arises with openable objects such as cabinets, nightstands, and closets. While traditional openable objects often feature simple handles that can be grasped to open them, modern design increasingly incorporates handleless doors and drawers. This presents a significant challenge for service robots, as most existing methods for the robotic manipulation of articulated objects focus on objects with handles, and there is limited research on handling objects without them.

This thesis addresses the first crucial step in managing such objects: the classification of the opening mechanism. Three categories are proposed for this classification: objects with regular handles, objects that can be grasped by their surface to be opened, and objects with a push latch mechanism. Given that the latter two categories often appear visually similar and can be difficult even for humans to distinguish based solely on appearance, this work explores methods that utilize both images of the objects and images of a human demonstrating the approach to opening them.

Experiments conducted using a new dataset indicate that combining images without demonstration and images with demonstration significantly improves the performance

of CNN classifiers compared to using either type of image alone. Additionally, experiments conducted with a modern object detector show that such classifiers can be applied to automatically detected regions providing evidence that the proposed methods could be used in real-world environments as part of fully autonomous systems.

Sažetak

Inteligentni servisni roboti sve se češće koriste u kućanstvima, bolnicama, skladištima i drugim okruženjima. Ovi roboti sposobni su obavljati složene zadatke, uvelike se oslanjajući na svoje vizualne sustave za prepoznavanje objekata. Budući da djeluju u trodimenzionalnom prostoru, za prepoznavanje objekata koriste se podaci dobiveni 3D senzorima jer nude detaljan prikaz okoline u obliku oblaka točaka. Oblaci točaka omogućuju precizno izdvajanje informacija o položaju, veličini i geometriji objekata unutar 3D okoline. Primjeri takvih senzora uključuju LiDAR-e i RGB-D kamere. Moderne metode prepoznavanja 3D objekata uglavnom se temelje na dubokom učenju i, kao takve, zahtijevaju velike količine označenih podataka za učinkovito funkcioniranje. Međutim, pribavljanje takvih podataka može biti skupo i dugotrajno. Generiranje sintetičkih podataka predstavlja obećavajuću alternativu, ali trenutni detektori objekata učeni na sintetičkim podacima i dalje pokazuju slabije rezultate u usporedbi s onima koji su učeni na stvarnim podacima. To sugerira da trenutne metode generiranja sintetičkih podataka još uvijek ne postižu dovoljnu razinu realizma.

Cilj ovog rada je doprinijeti rješavanju ovog problema istraživanjem područja generiranja realističnih sintetičkih 3D podataka unutarnjih prostora. Glavni cilj je analizirati utjecaj pet ključnih faktora realizma: prisutnosti pozadinskih objekata, šuma kamere, pozicioniranja objekata, konteksta scene i veličine objekata. Kako bi se istražili ovi faktori, razvijena je modularna metoda za generiranje realističnih sintetičkih oblaka točaka iz jedne točke promatranja. Ova metoda omogućava generiranje velikih, prilagodljivih skupova podataka s različitim razinama realizma, posebno kontrolirajući svaki od navedenih faktora.

Ti skupovi podataka se koriste za treniranje modernih detektora objekata, a utjecaj svakog faktora realizma ocjenjuje se na temelju uspješnosti prepoznavanja. Nadalje, eksperimenti pokazuju da se performanse detektora objekata mogu poboljšati treniranjem istog na sintetičkom skupu podataka i dodatnom treniranju na stvarnim podacima. Zanimljivo je da model treniran samo na geometrijskim podacima nadmašuje isti detektor objekata obučen isključivo na stvarnim podacima, koji koristi i geometrijske podatke i podatke o boji.

Pored sposobnosti detekcije objekata, servisni robot treba imati i sposobnost interakcije s objektima u svom okruženju. Poseban izazov javlja se kod objekata koji se mogu otvoriti kao što su ormarići, noćni ormarići i ormari. Dok tradicionalni objekti koji se mogu otvoriti često imaju jednostavne ručke koje se mogu uhvatiti za otvaranje, moderni dizajn sve više uključuje vrata i ladice bez ručki. To predstavlja značajan izazov za servisne robote jer se većina postojećih metoda robotskog otvaranja vrata fokusira na objekte s ručkama, a postoji malo istraživanja o interakciji s objektima bez ručki.

Ovaj rad bavi se prvim ključnim korakom u interakciji s takvim objektima: klasifikacijom mehanizma otvaranja. Predložene su tri kategorije za ovu klasifikaciju: objekti s ručkama, objekti koji se mogu uhvatiti za površinu kako bi se otvorili te objekti s mehanizmom za otvaranje na dodir. S obzirom na to da objekti posljednje dvije kategorije često izgledaju slično, pa čak i ljudima može biti teško razlikovati ih samo na temelju izgleda, ovaj rad istražuje metode koje koriste i slike objekata i slike na kojima ljudi demonstriraju pristup otvaranju.

Pokusi provedeni s novim skupom podataka pokazuju da kombiniranje slika bez demonstracije i slika s demonstracijom značajno poboljšava performanse CNN klasifikatora u usporedbi s korištenjem samo jedne vrste slike. Dodatno, pokusi provedeni s modernim detektorom objekata pokazuju da se takvi klasifikatori mogu primijeniti na automatski detektirane regije, što sugerira da se predložene metode mogu koristiti u stvarnim okruženjima kao dio potpuno autonomnih sustava.

Contents

List of Figures

List of Tables

1 Introduction

Intelligent service robots are no longer a thing of the distant future, service robots are already being used for various purposes. They are being utilized both in private and public sectors. Service robots are helping out in household environments and improving people's lives by assisting with various tasks. They are also being used in professional settings like hospitals and warehouses where they help out by transporting goods, entertaining patients, cleaning and so on. In most of these tasks, object detection is a very important component.

Object detection entails localization and classification of objects in a scene. Since service robots often need to traverse and interact with objects in 3D space, it is important that they work with 3D data such as point clouds and detect objects in that 3D space. There's a plethora of research about 3D object detection, with both indoor and outdoor environments. State-of-the-art methods for 3D object detection are based on supervised deep learning and as such require large amounts of annotated 3D data for training.

Such data is usually acquired and annotated manually so it can be a time-consuming and very expensive process. One way to combat this issue is to generate synthetic data for the purpose of training object detectors. Annotated synthetic data can be fast and cheap to generate. Not any synthetic data can be used for training, it is often important that the data is realistic and diverse. There are many factors of realism that can be considered when generating synthetic data such as realistic sizes and camera noise. While there are various proposed methods for synthetic 3D data generation, there is not much research done on the topic of the importance of various factors of realism in synthetic data generation and their effect on the performance of object detectors trained on synthetic data.

A modular method for generating synthetic 3D data of indoor spaces is proposed in this thesis. Its modularity allows for control of realism for the following five factors: presence of background objects, camera noise, positioning of objects, context of scenes, and object sizes. Using this method, different datasets are created which can then be used to train object detectors and perform an ablation study to examine the importance of those factors of realism. Furthermore, the method is used for generating a synthetic dataset which is used for pretraining an object detector to improve its performance when finetuned on real data.

Some of the objects in indoor spaces can be interacted with. These include small objects which can be moved, objects with horizontal surfaces on which other objects can be placed on, and objects with enclosed volumes such as kitchen cabinets, closets and nightstands which have doors and drawers that can be opened. In the case of these objects which have moving parts that can be opened, it is not enough to just localize the objects, it is also important to detect the way in which these can be opened. This includes determining if the object needs to be pulled out like a drawer or opened like a door, the parameters of this motion, and how to grasp the object to open it.

Traditionally, doors and drawers simply had handles which needed to be grasped when opening. However, openable furniture without handles has gained popularity in households. These include doors and drawers which have the so-called push latch mechanism or those that lack a handle and need to be grabbed by the surface of the object.

Because of this, it is important to first determine the type of opening before anything can be done with such objects. However, there is limited research on this subject and that is precisely why it is one of the focuses of this thesis.

Because these handleless objects can look similar to one another, it is sometimes hard to classify them even for a human. They are also very often just design choices and not necessarily the best choice based on objective parameters. Consequently, it can be helpful to determine the opening type with the help of human demonstration. The methods presented in this thesis classify the opening types of openable furniture based on human demonstration. The proposed methods are tested on a new dataset to determine the importance of using human demonstration for this task.

1.1 Contributions

In this thesis, a modular method for synthetic 3D data generation is proposed and studied. Additionally, a method for classification of openable objects based on human demonstration is proposed. The following original contributions to the scientific literature are proposed:

- **Modular method for generating synthetic 3D indoor scenes based on real 3D scenes and a database of 3D furniture models,**
- **Analysis of the effects of scene realism factors and camera model in synthetic 3D scene generation on the performance of object detectors in indoor spaces,**
- **Method for classifying furniture opening methods based on an image and human demonstration of furniture opening type.**

1.2 Organization of the thesis

This thesis is organized in six chapters. The first one is this chapter, Introduction. In Chapter 2, a brief overview of object detection is given. 2D object detection is described and various methods are presented. After this, the field of 3D object detection is covered. 3D data collection methods are described, followed up by common feature extractors for 3D data and object detectors which use these feature extractors. Finally, an overview of the evaluation of 3D object detectors is given. In Chapter 3, the aforementioned modular method for synthetic 3D data generation is covered. It briefly touches on existing methods for synthetic 3D data generation, followed up by in-depth description of the method and all its modules. This is followed up by experiments using datasets generated by the method and a discussion about the results. Chapter 4 focuses on furniture opening methods. A brief introduction to the subject is given and it's followed up by an in-depth analysis of the dataset created for this research. Various methods for classification are then proposed and described. Finally, the proposed methods are tested using the dataset and the results are discussed. Key findings of this thesis and potential directions of future research are described in the Conclusion.

2 Object detection

Object detection is one of the most important tasks in the field of computer vision. It entails the identification and localization of objects in images and videos. It's a fundamental task for various applications, including autonomous driving, medical imaging, intelligent service robots, augmented reality, and many others. Object detection can be divided into two main subtasks:

- **Localization:** Localization entails determining the precise location of an object within an image or some other medium. This is typically represented by a 2D or 3D bounding box.
- **Classification:** Classification is the task of categorizing an object into a discrete, finite set of categories.

To illustrate the importance and functionality of object detection, consider the example of an autonomous vehicle navigating through a busy urban environment. The vehicle must detect various objects such as pedestrians, other vehicles, traffic signs, and obstacles to make informed driving decisions. This real-time object detection capability enables the vehicle to avoid collisions, adhere to traffic rules, and ensure passenger safety.

Figure 2.1: An example of object detection in an autonomous driving scenario (Lin et al., 2014). The vehicle detects and localizes a pedestrian, a dog, and several cars.

The evolution of object detection methods has been characterized by significant advances, moving from traditional approaches relying on hand-crafted features to modern techniques using deep learning. This progression has led to substantial improvements

in detection accuracy and robustness, enabling more complex and dynamic real-world applications.

This chapter explores the development of object detection methods, starting with 2D object detection techniques and extending to the realm of 3D object detection. Various methodologies are described, highlighting key milestones and innovations that have shaped the field.

2.1 2D Object Detection

2D object detection aims to identify and locate objects within a two-dimensional image. The primary goal is to predict bounding boxes around objects and classify them into predefined categories. The bounding box is uniquely defined by four values. It is most often described by the coordinates of its center or top-left corner, and by its width and height. Over the years, various approaches have been proposed for 2D object detection, which can be broadly categorized into traditional methods and deep learning-based methods. Today, an extension of this task is also very common: instance segmentation. In addition to the tasks of localization and classification, instance segmentation includes per-pixel annotation of each detected object.

2.1.1 Traditional Methods

Traditional methods for 2D object detection often rely on handcrafted features and classical machine learning techniques. Popular approaches include the use of Haar-like features (Viola and Jones, 2001), Histogram of Oriented Gradients (Dalal and Triggs, 2005), and Deformable Part Models (Felzenszwalb et al., 2009). These methods typically involve:

- **Feature Extraction:** Extracting meaningful features from the image that can represent the objects of interest.
- **Classification:** Using machine learning classifiers such as Support Vector Machines (SVM) to classify regions of the image as containing objects or not.
- **Post-Processing:** Refining the detection results through non-maximum suppression (NMS) to remove redundant bounding boxes.

Furthermore, traditional object detectors, while not explicitly generating region proposals in the way modern methods do, utilize mechanisms to focus on promising regions within the image. The Viola-Jones detector, for example, employs a sliding window approach that scans the image at multiple scales and positions, using a cascade of classifiers to quickly discard non-object regions.

2.1.2 Deep Learning-Based Object Detection Methods

As deep learning has developed, object detection has seen significant improvements in performance. Convolutional neural networks (CNNs) have become the foundation of modern 2D object detection methods. However, the general strategy remains similar to traditional methods, involving feature extraction, region proposals, and classification, as shown in Figure 2.2.

The breakthrough of deep CNNs, starting with LeNet-5 (LeCun et al., 1998) and later AlexNet (Krizhevsky, Sutskever, and Hinton, 2012), marked a significant step towards the application of deep learning in object detection. These networks demonstrated the

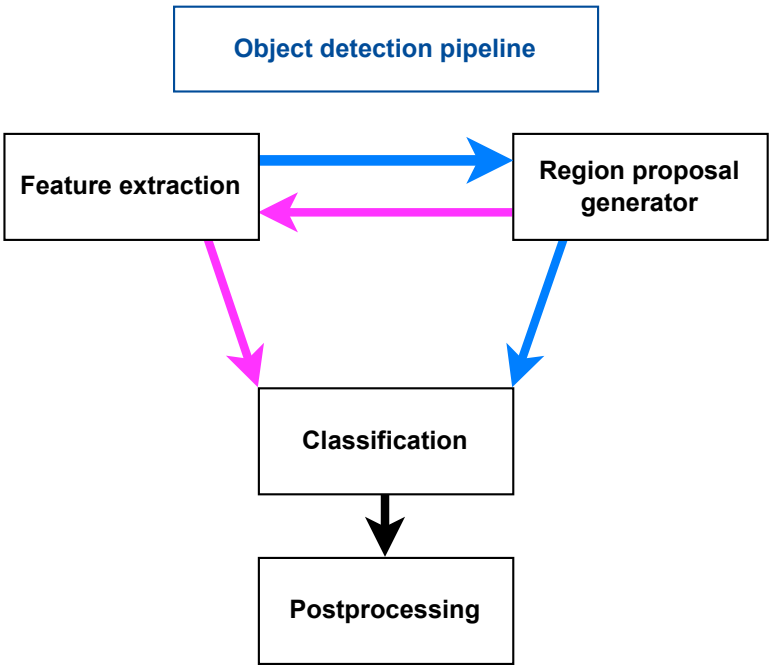

Figure 2.2: The pipeline of the general approach to object detection. More traditional approach is shown in magenta where the region proposal precedes feature extraction, and shown in blue is the more modern approach where feature extraction is done first. Some approaches don't have separate region proposal generators, instead they predict bounding boxes and classes at once.

power of deep learning for feature extraction and classification, setting the stage for modern object detection frameworks. After AlexNet, various architectures for feature extraction from RGB images have been proposed: VGG (Simonyan and Zisserman, 2014), ResNet (He et al., 2016), EfficientNet (Koonce and Koonce, 2021), etc.

R-CNN and its Derivatives

Region-based Convolutional Neural Networks (R-CNN) (Girshick et al., 2014) was a novel approach that integrated deep learning with object detection. It involved a multi-stage pipeline that included region proposal generation, feature extraction using CNNs, and classification with Support Vector Machines (SVMs).

- **Region Proposal:** Selective search generates around 2000 region proposals from the input image.
- **Feature Extraction:** Each region proposal is resized to a fixed size and passed through a CNN (e.g., AlexNet) to extract features.
- **Classification and Bounding Box Regression:** Features are fed into SVM classifiers for classification and a linear regressor for bounding box refinement.

While it was a big step in deep learning-based object detection, R-CNN is very computationally expensive and has a slow detection time. Various improvements to this network were later proposed, which boosted both its speed and detection performance.

Fast R-CNN (Girshick, 2015) improved upon R-CNN by improving the detection pipeline, making it faster and more efficient. Instead of running the CNN for each region proposal, Fast R-CNN processes the entire image with a CNN to produce a feature map. Region proposals are then projected onto this feature map, and RoI pooling extracts fixed-size feature vectors from each region. These features are fed into fully connected layers that predict class scores and bounding box coordinates. This approach significantly reduces computation by sharing CNN features across proposals.

Faster R-CNN (Ren et al., 2016) introduced the Region Proposal Network (RPN) to generate region proposals, combining proposal generation and detection into a single network. The entire image is passed through a CNN to produce a feature map, and the RPN generates region proposals by predicting objectness scores and bounding box parameters. These proposals are used to extract features using RoI pooling, followed by classification and bounding box regression. The whole architecture is shown in Figure 2.3. The addition of the RPN makes the framework more efficient and allows for end-to-end training, leading to better performance.

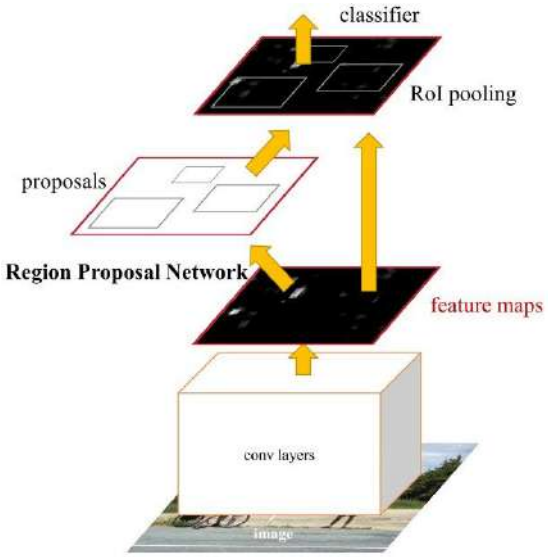

Figure 2.3: Faster R-CNN architecture (Ren et al., 2016).

You Only Look Once (YOLO)

Unlike traditional methods that use a pipeline of separate components for region proposal and classification, YOLO (Redmon et al., 2016) predicts bounding boxes and class probabilities directly from full images in one evaluation, allowing for real-time processing.

YOLO v1 (Redmon et al., 2016) divides the input image into an SxS grid. A grid cell is responsible for detecting an object if that object's center falls into that grid cell. Each grid cell predicts a fixed number, two in the original proposal, of bounding boxes and confidence scores for those boxes. Additionally, class probabilities are predicted for each grid cell. These are then used to obtain the final predictions. This design makes YOLO very fast and suitable for real-time applications. The network, however, struggles with smaller objects and complex scenes. An illustration of the detection process is shown in Figure 2.4.

YOLO v2 (Redmon and Farhadi, 2016), also known as **YOLO9000**, introduces batch normalization, a high-resolution classifier, and anchor boxes, improving accuracy and

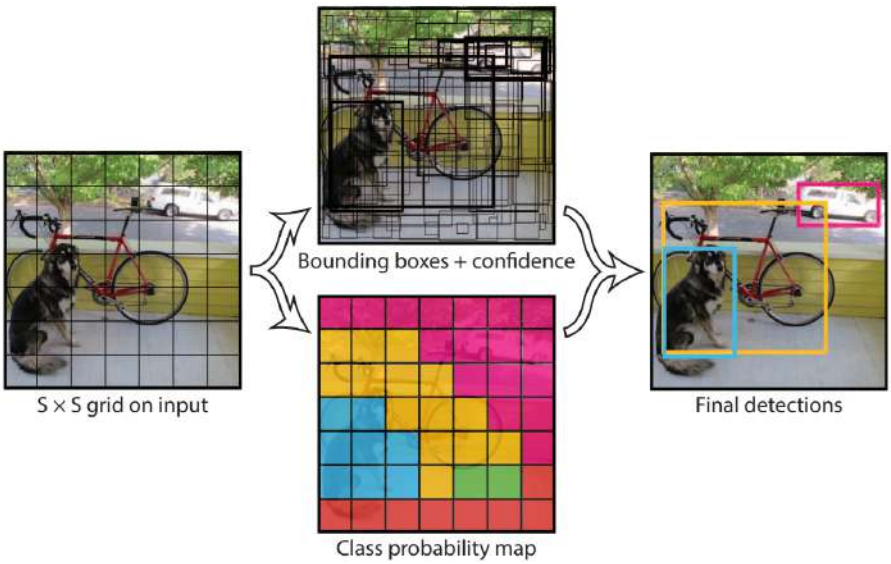

Figure 2.4: An illustration of the detection process of YOLO v1 (Redmon et al., 2016).

speed over YOLO v1. **YOLO v3** (Redmon and Farhadi, 2018) uses a deeper network, Darknet-53 with residual connections, and predicts bounding boxes at three different scales, which improves small object detection and overall performance.

YOLO v4 (Bochkovskiy, Wang, and Liao, 2020) further enhances the architecture by integrating several techniques to boost both speed and accuracy. It uses the CSP-Darknet53 backbone with Cross-Stage Partial connections (Wang et al., 2020) to reduce computation while maintaining accuracy. The Path Aggregation Network (PANet) (Liu et al., 2018) is utilized for better feature fusion. Improved anchor generation and prediction across multiple scales, along with training enhancements like Bag of Freebies and Bag of Specials, are incorporated to enhance training without increasing inference cost. YOLO v4 maintains real-time performance while improving detection accuracy.

YOLOv7 (Wang, Bochkovskiy, and Liao, 2022) is based on the YOLOv4 algorithm and improves accuracy and inference times by introducing several structural and optimization modifications. One of which is the Extended Efficient Layer Aggregation Network (E-ELAN) (Wang, Liao, and Yeh, 2022) backbone, which is an architecture in the backbone of the framework that allows the framework to continuously learn from the data while not disturbing the existing flow of gradients during training. Additionally, the compound scaling method was proposed, which subsequently scales the width of the block with the same amount of change to the transition layers. This results in preserving the properties of the initial model design and the optimal structure. YOLO v7 offers improved detection accuracy while maintaining high speed, though it requires more computational resources for training.

Single Shot MultiBox Detector (SSD)

SSD (Liu et al., 2016) is based on a feed-forward CNN but with three additional key features. The first of which is multi-scale feature maps. As the input image is passed through a CNN, feature maps at different layers are produced. Each of these feature maps is responsible for detecting objects at different scales. Additionally, it uses convolutional predictors for detection instead of using fully connected layers for this purpose. Each aforementioned feature layer produces a set of detections using only convolutional

filters. Finally, the method uses pre-defined default boxes, or anchors, with different scales and aspect ratios for each cell in the feature maps. For each of these boxes in each of the cells, an offset of the bounding box and class scores are predicted. At the end NMS is applied as usual to obtain the final predictions. SSD outperformed the state-of-the-art methods at the time, both in accuracy of detection and speed.

2.1.3 Detection Transformers (DETR)

Detection Transformers (DETR) (Carion et al., 2020) represent a novel approach to object detection by leveraging transformers (Vaswani et al., 2017) for both object detection and bounding box prediction. Unlike traditional convolutional approaches, DETR uses an encoder-decoder architecture based on transformers. The input image is processed by a CNN backbone to produce a feature map, which is then flattened and passed through a transformer encoder. The encoder captures the global context of the image. In the decoder, a fixed set of learned object queries is used to decode the encoded features. Each query attends to the entire feature map and predicts an object's bounding box and class.

One of the key advantages of DETR is its ability to simplify the detection pipeline by removing the need for region proposal networks or non-maximum suppression. DETR is trained end-to-end, and the loss function combines classification and bounding box regression, along with a bipartite matching loss to ensure one-to-one matching between predicted and ground truth boxes. This end-to-end approach, combined with the power of transformers to capture long-range dependencies and global context, allows DETR to achieve competitive performance with traditional methods. However, it requires large amounts of data and computational resources for training and initially exhibits slower convergence compared to CNN-based methods.

2.2 3D object detection

Similar to 2D object detection, the task of 3D object detection is classification and localization of objects, but instead of detecting bounding boxes on images, bounding boxes in three-dimensional space are predicted. 3D object detection is crucial for applications such as autonomous driving, robotics, and augmented reality, where understanding the spatial relationships is essential. Various approaches have been developed to handle 3D data, often leveraging point clouds, RGB-D images, and multi-view images. Understanding how 3D data is collected and the different types of data available is essential for developing effective 3D object detection methods.

2.2.1 Data Collection Methods

3D data can be collected using a variety of sensors and techniques. Two of the most common methods are LiDAR and RGB-D cameras.

LiDAR (Light Detection and Ranging): LiDAR sensors emit laser pulses and measure the time it takes for the pulses to return after hitting an object. This time-of-flight measurement allows the sensor to calculate the distance to the object, creating a 3D point cloud that represents the environment. LiDAR is widely used in autonomous vehicles due to its high accuracy and ability to capture detailed spatial information over large distances.

RGB-D Cameras: RGB-D cameras, such as the Microsoft Kinect, capture both color (RGB) images and depth (D) information. Depth is typically measured using structured light or time-of-flight techniques. RGB-D cameras are popular in robotics and indoor

mapping applications because they provide rich color information along with depth data, enabling a comprehensive understanding of the scene.

2.2.2 Types of 3D Data

3D data can be represented in various ways, some of which are easily converted in one another.

Point Clouds: Point clouds are sets of data points defined in a three-dimensional coordinate system. Each point represents a sample on the surface of an object or within the environment. Point clouds are commonly generated by LiDAR sensors or with RGB-D cameras, and are widely used in 3D object detection and mapping due to their detailed spatial representation. An example of a point cloud captured with a LiDAR is shown in Figure 2.5.

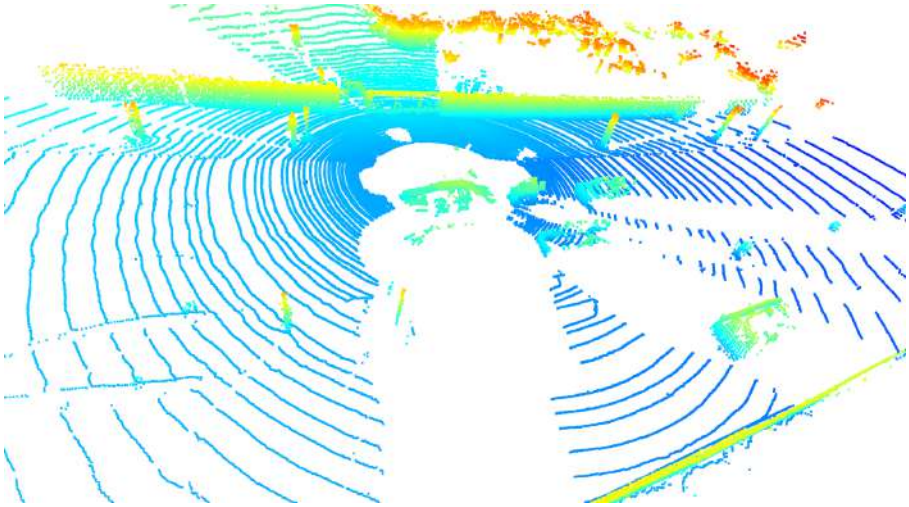

Figure 2.5: An example (Geiger, Lenz, and Urtasun, 2012) of a point cloud of an outdoor scene captured with a LiDAR.

Depth Images: Depth images, or depth maps, are 2D images where each pixel value represents the distance from the sensor to the object in the scene in the direction of the camera optical axis. Depth images can be generated by RGB-D cameras and are useful for applications requiring both visual and spatial information. Due to some limitations and challenges of depth sensing technologies, including reflective and absorptive surfaces, geometric occlusions and other, depth images can have some missing values or gaps. Given the camera parameters, depth images can easily be converted into point clouds, or coloured point clouds given the corresponding RGB image. An examples of an RGB-D image, a corresponding coloured point cloud and the point cloud with 3D bounding boxes of objects is shown in Figure 2.6.

Voxel Grids: Voxel grids are a structured representation of three-dimensional space, dividing it into a regular grid of small, cube-shaped volumes called voxels (volumetric pixels). Each voxel in this grid represents a discrete unit of space, much like a pixel represents a unit of space in a 2D image. Voxel grids provide a way to discretize 3D space, allowing for efficient processing and analysis of volumetric data. They are commonly used in 3D object detection, medical imaging and 3D reconstruction. By converting point cloud data into a structured grid format, voxel grids enable the use of 3D convolutional neural networks (CNNs) to extract spatial features directly from volumetric data. An example of a point cloud with its corresponding voxel grid is shown in Figure 2.7.

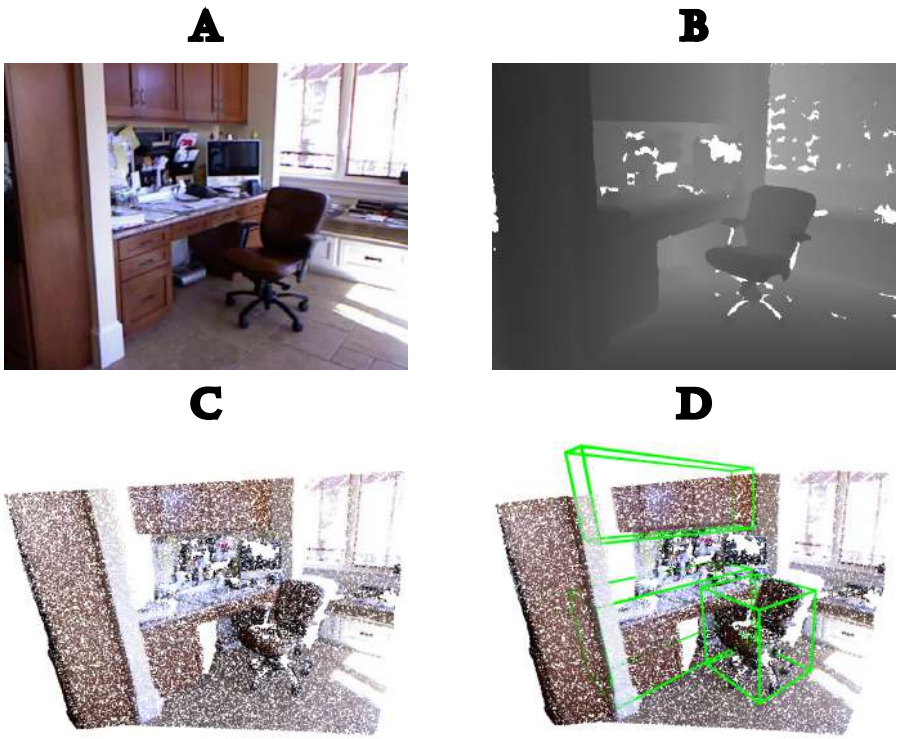

Figure 2.6: An example (Song, Lichtenberg, and Xiao, 2015) of a 3D indoor scene: A) RGB image, B) Depth image, C) Coloured point cloud, D) Coloured point cloud with 3D bounding boxes of objects. White pixels in the depth image are missing values.

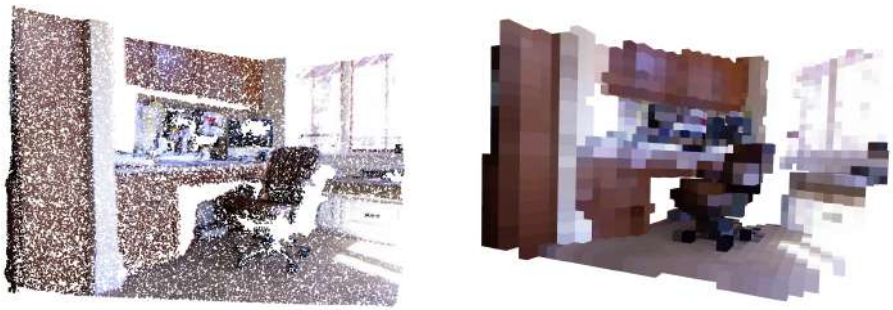

Figure 2.7: An example (Song, Lichtenberg, and Xiao, 2015) of a point cloud (left) with the corresponding voxel grid (right).

Multi-View Point Clouds: Multi-view point clouds are created by combining point clouds or depth images from multiple viewpoints. This approach enhances the completeness and accuracy of the 3D representation by capturing the scene from different angles. Multi-view point clouds are often used in applications like 3D reconstruction and augmented reality.

3D Scans of Space: 3D scans involve capturing detailed spatial information of an environment or object using techniques like laser scanning or photogrammetry. These scans can produce highly accurate 3D models used in various fields, including architecture, cultural heritage preservation, and industrial inspection.

Different types of 3D data offer various advantages, depending on the application. Point clouds are ideal for detailed spatial analysis and object detection in environments

with significant depth variations. Depth images are beneficial for integrating depth information with traditional 2D image processing techniques. While voxel-grids usually discard some information when generated from point clouds, they are structured data and allow for use of 3D CNNs. Multi-view point clouds and 3D scans provide comprehensive and accurate representations of complex scenes, making them valuable for reconstruction and detailed analysis.

2.2.3 Feature Extractors for 3D Data

Point clouds, a common form of 3D data, are unstructured and irregular, making them more challenging to process compared to structured 2D images. Efficient feature extraction from point clouds is essential for accurate 3D object detection. Several methods have been developed to address these challenges, including 3D convolutional networks, PointNet and its variants, and transformers.

3D Convolutional Networks

3D convolutional networks extend the concept of 2D convolutions to three dimensions, allowing the network to learn spatial features directly from volumetric data. This approach involves voxelizing the point cloud, i.e., dividing the space into a regular grid of voxels, and applying 3D convolutions. Voxel-based methods have shown good performances, but have high memory consumption and can have artifacts due to the voxelization operation.

PointNet

PointNet Qi et al., 2017a was a breakthrough in processing point clouds directly without voxelization. There are several issues when working with point clouds directly. The method must be invariant to point cloud permutations, i.e., the order of the points mustn't impact the results of the method. The method also needs to be invariant to rotations and translations of the point cloud, and it must be sensitive to local structure and geometry; it cannot look at each point in isolation.

PointNet first transforms the input point cloud in order to make it invariant to geometric transformations. An affine transformation is predicted by a T-Net and directly applied to the coordinates of the input points. It is followed up by a multilayer perceptron (MLP), another T-Net and transformation, and a second MLP. After this, a feature vector is assigned to each point in the point cloud which represents which area of the observed volume the point belongs to. Max-pooling is then applied on these feature vectors and the resulting global feature vector encodes which areas of the observed volume are filled with points. Such a vector is invariant to permutations of the input point cloud. This approach is efficient and deals with the other previously mentioned issues. However, PointNet primarily captures global features and may not effectively capture local structures.

PointNet++

PointNet++ Qi et al., 2017b extends PointNet by introducing a hierarchical structure that captures both local and global features. It applies PointNet recursively on nested partitions of the input point set, enabling the network to learn local features at multiple scales. This hierarchical approach improves the ability to capture fine-grained geometric details, enhancing the performance for tasks requiring precise localization.

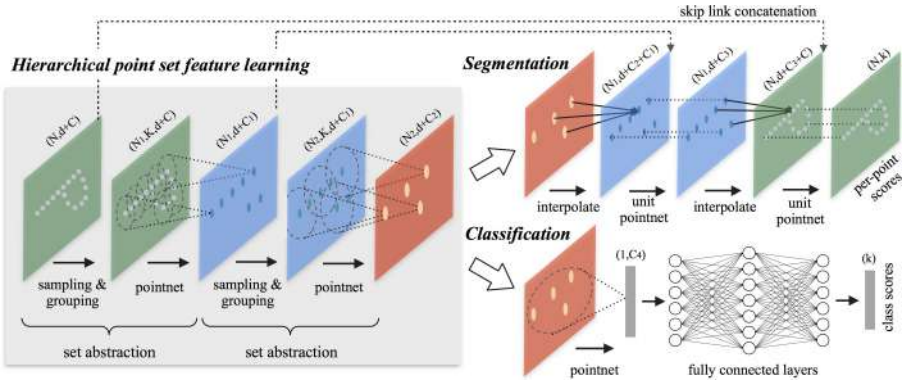

Figure 2.8: Illustration of the PointNet++ architecture Qi et al., 2017b. Pictured here are points in 2D space for better visualization. Also shown on the image are applications of segmentation and classification based on the extracted features (shown in gray rectangle).

Transformers

Transformers (Vaswani et al., 2017) have recently been adapted for 3D point cloud processing. They leverage self-attention mechanisms to capture relationships between points, allowing the network to focus on important regions of the point cloud. Transformers can handle irregular and unordered point sets efficiently and have shown promising results in various 3D tasks. The flexibility and effectiveness of transformers make them a powerful tool for 3D feature extraction.

2.2.4 PointNet-Based Methods

PointNet and its variants have been pivotal in advancing 3D object detection by directly processing point clouds without the need for voxelization. **Frustum PointNets** (Qi et al., 2018) leverage 2D object detection to propose 2D regions and classify them. These 2D regions are then transformed into frustum proposals. A frustum is a portion of a pyramid that is left after the upper part is cut off. Each frustum proposal is processed by a PointNet (Qi et al., 2017a) to obtain binary segmentation of the point cloud in the frustum. The segmented point cloud is further processed by another PointNet to obtain the final bounding box. An illustration of the detection process is shown in Figure 2.9. **VoteNet** (Qi et al., 2019) introduces a novel approach where the input point cloud is processed by PointNet++ (Qi et al., 2017b), which outputs a subsampled subset of points with features. These so-called seed points are then used to generate votes, which are later clustered and processed to obtain the final predictions (illustrated in Figure 2.10). VoteNet outperformed the state-of-the-art methods with only geometric data, i.e., without using RGB data. **H3DNet** (Zhang et al., 2020) enhances the VoteNet method by predicting three geometric primitives instead of just one. In addition to bounding box centers, it also predicts bounding box face centers and bounding box edge centers. All of these are then used to generate proposals. By combining different geometric cues, such as planes and lines, H3DNet improves the detection of objects in complex scenes with varying structures. **Back-tracing Representative Points Network (BRNet)** (Cheng et al., 2021) upgrades the VoteNet architecture in a different fashion. After the voted clusters have been generated, representative points are back-traced, which imply the possible area of the object. Features of nearby seed points are then aggregated to these representative points and fused with clustered vote features. These can then be used to produce refined representative points and classes for final predictions. This method reduces the negative

effects that partial coverage of the objects and outliers from the cluttered background can have on the performance of a detector.

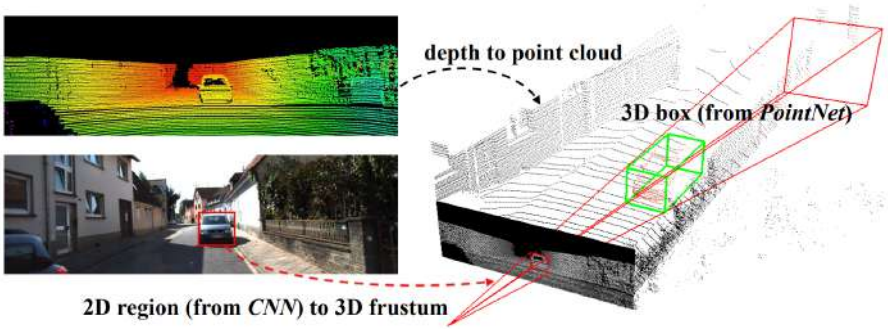

Figure 2.9: An illustration of the detection process used in Frustum PointNets (Qi et al., 2018). A 2D region is proposed based on the RGB image which is then transformed into a frustum proposal. This proposal is then processed to obtain the final prediction.

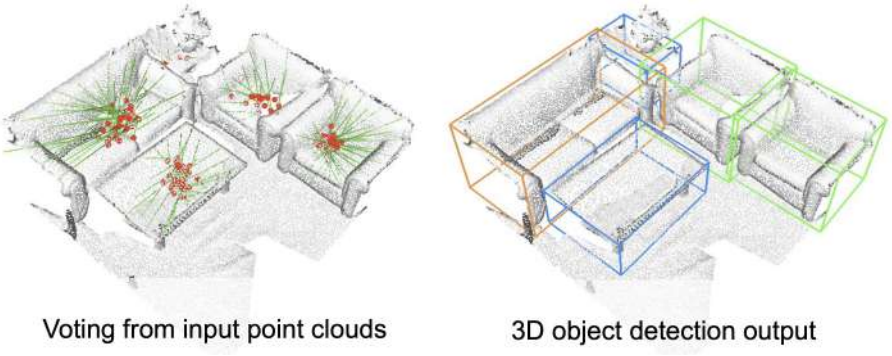

Figure 2.10: Voting mechanism used in VoteNet (Qi et al., 2019) and other methods (Zhang et al., 2020; Cheng et al., 2021). Each point votes for a center, or a different geometric primitive. These votes can then be grouped and used as proposals for final predictions.

2.2.5 3D Convolution-Based Methods

3D convolution-based methods leverage the power of 3D convolutions to process voxelized point clouds, enabling the extraction of spatial features directly from volumetric data. There are many notable convolution-based methods (Zhou and Tuzel, 2018; Rukhovich, Vorontsova, and Konushin, 2021; Shi et al., 2020; Yan, Mao, and Li, 2018; Yang et al., 2020). Some of them are explained here to give an insight into the basic principles of the convolution-based methods. **VoxelNet** (Zhou and Tuzel, 2018) was one of the first methods to directly operate on raw point clouds by dividing the point cloud into equally spaced 3D voxels and applying 3D CNNs to extract features. It first partitions the 3D space into equally spaced voxels. Points are then grouped into corresponding voxels and subsampled inside those voxels. Each group of points is then passed through a stack of Voxel Feature Encoding (VFE) layers. A VFE layer takes a point-wise input and outputs a point-wise feature map. This stack of VFEs outputs a voxel-wise feature map which is then passed through a 3D CNN and finally an RPN to obtain predictions. **Fully Convolutional Anchor-Free 3D (FCAF3D)** (Rukhovich, Vorontsova, and Konushin, 2021) is another recent development that simplifies the 3D object detection

pipeline by eliminating the need for anchor boxes. FCAF3D leverages sparse 3D convolutions to directly predict object locations and classifications from 3D feature maps. It uses so called HDResNets (Xiang, Sun, and Tu, 2023) as backbone, followed up by a simplified GSDN (Generative Sparse Detection Networks) decoder (Gwak, Choy, and Savarese, 2020) and, finally, an anchor-free sparse convolution head is used for predictions. **Point-Voxel Region-based Convolutional Neural Network (PV-RCNN)** (Shi et al., 2020) combines the strengths of point-based and voxel-based methods by utilizing both point features and voxel features in its detection pipeline. It first applies voxelization and sparse 3D convolutions to extract voxel features, then refines these features using a PointNet-based (Qi et al., 2017a) module to incorporate fine-grained point details.

2.2.6 Other Methods

Besides these two categories, there are many other methods for 3D object detection. Some utilize transformers (Liu et al., 2021; Misra, Girdhar, and Joulin, 2021). Attention mechanisms allow these networks to focus on relevant parts of the point cloud effectively. On the other hand, Point Graph Neural Network (Point-GNN) (Shi and Rajkumar, 2020) utilizes graph neural networks (GNNs) to model the relationships between points in a point cloud.

2.3 Evaluating 3D Object Detectors

Evaluating the performance of 3D object detectors is crucial to understanding their effectiveness and identifying areas for improvement. To understand the evaluation of object detectors, the output of an object detector has to be understood first. A detector outputs a finite set of predictions. Each prediction consists of 3 things:

- Parameters of the bounding box. In 3D object detection, this is usually a bounding box which is aligned with the upright axis, i.e. aligned with the floor of a scene. In such a case, the bounding box is uniquely defined by the position of its center, its dimensions, and the rotation around the axis with which it is aligned.
- Predicted class of the detected object.
- Confidence score. The higher the confidence score, the higher the confidence of the detector that the prediction is correct and more weight is given to that particular prediction when evaluating the detector.

When evaluating a detector, it is important to define the confidence threshold which determines the predictions which will be kept as true predictions and which will be discarded as unimportant. Confidence score is used when calculating some evaluation metrics.

The most common metric of evaluation is mean average precision or mAP. Another popular one is average recall or AR. To understand these metrics, however, one must first understand a few other important terms. These include precision, recall, and Intersection over Union (IoU). It's important to note that object detectors cannot realistically identify all objects on a given scene, instead they have a finite set of categories of objects which they identify, referred to as objects of interest in this thesis.

Precision measures the ability of a detector to correctly identify present objects. It is defined as follows:

$$P = \frac{TP}{TP + FP}, \tag{2.1}$$

where TP stands for True positive, i.e. number of correctly identified objects, and FP stands for False positive, i.e. number of detections which do not correspond to actual objects.

Similarly, recall measures the detector's ability to identify all objects of interest present on the scene. It is defined by the following equation:

$$R = \frac{TP}{TP + FN}, \tag{2.2}$$

where FN stand for False negative, i.e. number of objects of interest which have not been identified by the detector.

2.3.1 Intersection over Union (IoU)

To understand these metrics, it is important to define what is a correct prediction (TP) and what is not (FP). Intersection over Union (IoU) is used for this purpose. IoU measures the overlap between a predicted bounding box and the ground truth bounding box. It is defined as the ratio of volume of intersection of the two bounding boxes and the union of those two bounding boxes:

$$\text{IoU (A, B)} = \frac{\text{Volume of Intersection } (A \cap B)}{\text{Volume of Union } (A \cup B)} \tag{2.3}$$

A visual representation of this equation for 2D bounding boxes is shown in Figure 2.11. This representation is analogous to 3D IoU. In the case of 2D IoU, volumes of intersection and union are simply replaced with areas in the above formula.

An IoU value ranges from 0 to 1, where 0 stands for no overlap and 1 for a perfect overlap. A higher IoU value means a better prediction. When evaluating a detector, so-called IoU threshold is defined which determines the minimum IoU required for a predicted bounding box to be considered correct. In 3D object detection, IoU threshold of 0.25 or 0.5 are most commonly used.

2.3.2 Mean Average Precision (mAP)

Mean average precision (mAP) is widely used as a benchmark for the evaluation of 2D and 3D object detectors. Mean average precision is simply the average of average precisions (AP) over all classes. To define the average precision, precision-recall curve first needs to be defined.

Precision-recall curve

Ideal detector would have a high precision and a high recall, i.e., all detections would be correct and all objects of interest would be detected. As previously mentioned, the model outputs confidence score for each predicted bounding box and that score needs to be higher than some confidence threshold for a prediction to not be discarded. The lower this confidence threshold, the higher the number of detections made by the detector and the lower the probability that objects were missed. This *generally* results in a higher recall. On the other hand, the higher the confidence threshold is, the lower is the chance that a false detection is made which *generally* results in higher precision. As it is important for a detector to have both high precision and recall, there is a trade-off between these two values based on the confidence threshold.

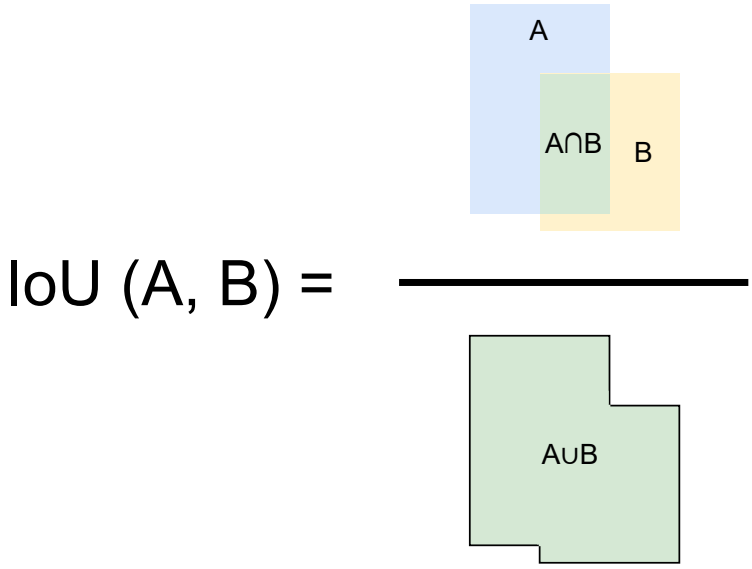

Figure 2.11: Visual representation of Intersection over Union (IoU) for two 2D bounding boxes, A and B. In 3D, these bounding boxes represent 3D volumes and, accordingly, the intersection and the union are also volumes.

A precision-recall curve which plots the value of precision against recall for different confidence threshold values can be defined. The specific threshold values are not important for the curve as the curve only plots precision and recall. This is best demonstrated with an example.

Given a dataset which contains 4 objects which need to be detected, a detector has made 10 predictions. For each of these predictions, it is determined if the prediction is correct or not based on the IoU of the prediction and the ground truth bounding boxes. Additionally, the detector has also output the confidence score for each of the predictions. These predictions are then sorted based on their confidence scores. The predictions are shown in Table 2.1. The first column shows the rank of each prediction (based on confidence score), the second shows whether the prediction is correct or not, and the last two show precision and recall for the confidence threshold equal to the confidence score of that prediction, i.e., precision and recall when considering all the predictions with higher or equal confidence score to that prediction. Precision and recall are calculated by Equations 2.1 and 2.2, respectively.

Rank	Correct	Precision	Recall
1	True	1.0	0.25
2	False	0.5	0.25
3	True	0.67	0.5
4	False	0.5	0.5
5	False	0.4	0.5
6	False	0.33	0.5
7	True	0.43	0.75
8	False	0.38	0.75
9	False	0.33	0.75
10	True	0.4	1.0

Table 2.1: An example of predicted bounding boxes sorted by their confidence score and the precision and recall for each confidence threshold.

Taking the fifth row as example, there are 2 TP, 3 FP, and 2 FN, since there are 4 objects in total which need to be detected. Based on Equation 2.1, precision is equal to 2/5, or 0.4. Similarly, based on Equation 2.2, recall is equal to 2/4, or 0.5.

As the confidence threshold decreases (going down the table), the recall increases or stays the same. On the other hand, precision decreases with each false prediction and then spikes back up with each correct prediction. This is best visualized with a graph which is shown in Figure 2.12.

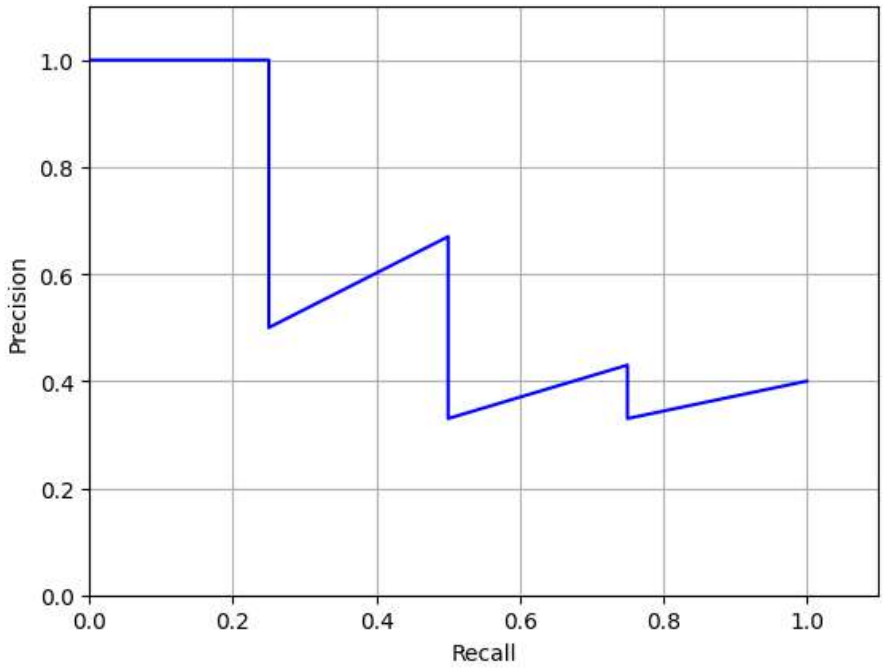

Figure 2.12: Precision-recall curve for the example given in Table 2.1.

Average Precision (AP)

Average precision is defined as the area under the precision-recall curve:

$$AP = \int_0^1 p(r)dr, \tag{2.4}$$

where p is precision, and r is recall. It is desirable that this area is as high as possible, and when both precision and recall are high, the area under the curve will also be high. However, when either of these is low across a large range of confidence threshold values, the area will be low. As both precision and recall are real numbers between 0 and 1, AP will also be a real number between 0 and 1.

Since AP is defined as an integral, it is approximated in practice. This is done because a continuous function is needed for integration, but in practice, there is a finite number of recall-precision pairs. Continuous representation of the model's performance at all possible threshold levels would be computationally intensive and impractical. There are two common approaches to this approximation. One is interpolation and averaging across a definite set of points. For example, an 11-point average can be taken by taking 11 values of recall between 0.0 and 1.0 with 0.1 increment. Precision can then be taken for these 11 values of recall. In some cases, precision-recall pairs for these exact values of recall might not be available so precision might need to be interpolated from the curve. In the general case of N values, AP is calculated as follows:

$$AP = \frac{1}{N} \sum_{\substack{r=0.0 \\ \text{step s}}}^{1.0} p(r), \tag{2.5}$$

where $s = \frac{1}{N-1}$.

Another approximation approach, common in the field of 3D object detection is using a monotonic approximation of the precision-recall curve. Each precision value $p(r)$ is replaced with the maximum precision value for recalls higher than r:

$$\tilde{p}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}). \tag{2.6}$$

AP is then calculated as the area under this approximation of the precision-recall curve which is equal to the sum of the rectangles defined by the marginal values of precision and recall. This is best illustrated by an example. Approximation of the curve shown in Figure 2.12 is shown in Figure 2.13. The figure shows the monotonic approximation of the curve and the rectangles whose areas can easily be calculated and summed to obtain the final AP value.

After AP has been calculated for each of the classes of interest, mean average precision is calculated as follows:

$$mAP = \frac{1}{k} \sum_{i}^{k} AP_i, \tag{2.7}$$

where k is the number of classes, and AP_i is the average precision of class i.

2.3.3 Additional Evaluation Metrics

While mAP is the primary metric, other evaluation criteria may be important for a comprehensive assessment of 3D object detectors:

Average Recall

Average recall is used to evaluate the recall capabilities of an object detector. The definition of average recall varies depending on use. One definition is similar to the definition of average precision in that it averages recall over an interval of IoU values, typically $[0.5, 1.0]$, and it is equal to the area under the recall-IoU curve:

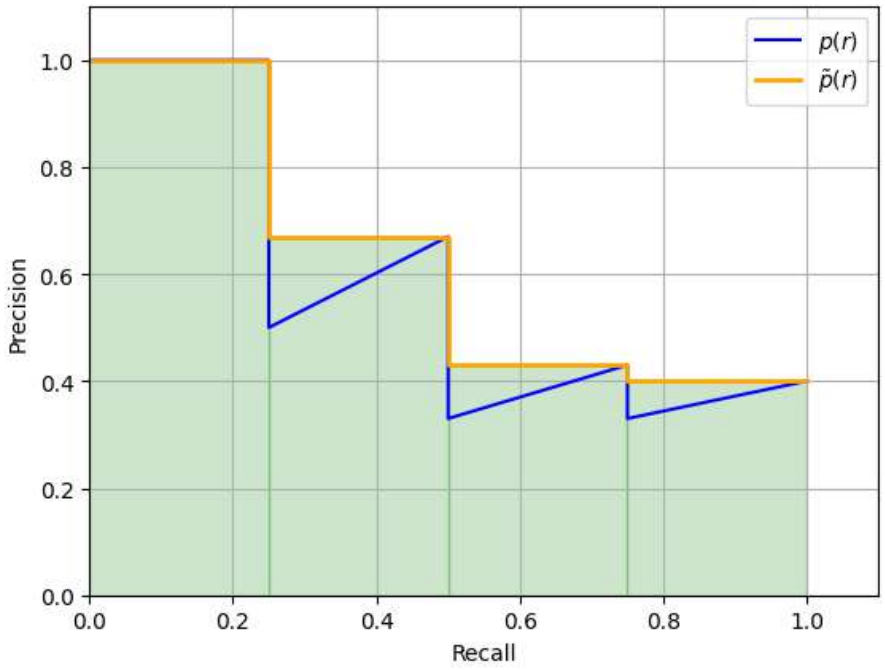

Figure 2.13: Approximation (orange) of the precision-recall curve (blue). AP is the sum of the areas of the green rectangles under the monotonic approximation of the precision-recall curve.

$$AR = 2 \int_{0.5}^{1} r(o)do, \tag{2.8}$$

where o is IoU and $r(o)$ is the corresponding recall. This can be calculated separately for each class and the mean average recall can be calculated as the mean value of AR across all classes.

Another definition of AR is AR at specific IoU threshold. It is often noted as $AR@o$, where o is the IoU threshold value. E.g. $AR@0.25$ represents the average recall for all classes with IoU threshold of 0.25. This is the definition which will be used throughout this thesis.

Accuracy

In multiclass classification, accuracy is often used to measure the performance of a classifier and it is defined as follows:

$$A = \frac{CP}{All}, \tag{2.9}$$

where CP is the number of correct predictions and All is the total number of samples. Accuracy is, however, not a suitable measure for the task of object detection since there is a discrepancy between predicted bounding boxes and ground truth ones, and mAP is a more appropriate metric for this task.

F1 score

Another measure used in classification but not often used in object detection is the F1 score and it is defined as follows:

$$F_1 = 2\frac{P \cdot R}{P + R}, \tag{2.10}$$

where P and R are precision and recall, respectively. F1 score is a valuable metric in classification tasks with imbalanced data. When data is imbalanced, a classifier which favors overclassifying into the most numerous class will have high accuracy even though it might not predict the less represented classes. Moreover, if it is not clear which of the precision and recall is more important for a specific task, F1 can be a good metric because it values both equally. F1 is not used often in object detection because mAP captures both precision and recall across different IoU thresholds and object classes more effectively.

Inference Time

Inference time measures the speed of the detector, which is critical for real-time applications such as autonomous driving. It is usually measured in milliseconds per frame (ms/frame) or frames per second (fps). A detector should provide a good balance between accuracy and speed for real-time applications.

Memory Usage

Memory usage is an important metric, especially for deployment on resource-constrained devices. It includes the amount of GPU memory and RAM required during inference. Efficient memory usage ensures that the detector can be deployed on a wider range of hardware.

2.3.4 Benchmark Datasets

Evaluating 3D object detectors requires large, annotated datasets that represent various scenarios and environments. Common benchmark datasets include:

- **KITTI** (Geiger et al., 2013): A dataset for autonomous driving that includes LiDAR point clouds, RGB images, and annotations for cars, pedestrians, and cyclists.
- **Waymo Open Dataset** (Sun et al., 2020): A large-scale dataset containing LiDAR and camera data from autonomous vehicles, with annotations for various object classes.
- **NuScenes** (Caesar et al., 2020): A dataset providing 360-degree sensor coverage with LiDAR, radar, and camera data, including annotations for diverse object classes.
- **SUN RGB-D** (Song, Lichtenberg, and Xiao, 2015): An indoor dataset with RGB-D images and 3D annotations for various objects in indoor environments.
- **ScanNet** (Dai et al., 2017): A large-scale dataset of 3D scans of indoor scenes, captured using RGB-D sensors.

These datasets offer a wide range of environments and scenarios, from urban driving to indoor navigation, making them essential resources for developing and evaluating 3D object detection algorithms.

3 Generating synthetic 3D indoor scenes

One of the biggest issues with deep learning-based methods in general is that they usually require large quantities of data. This is no different when it comes to 3D object detection. Furthermore, since the state-of-the-art methods for 3D object detection require supervised learning, the data also has to be annotated. Collecting and annotating data can be a time-consuming and costly process, and sometimes it's just not feasible to collect adequate amounts of data for a specific problem. While there are various methods for training robust models with limited data, this chapter deals with one way of doing this, using synthetic data.

To detect and correctly classify objects in complex scenes, neural networks rely not only on the shape of the object, but also on its size and position relative to other objects in the scene. The presence of an object can help classify other objects that may not be clearly visible in a scene, for example, due to occlusion or limited field of view of the camera. In this thesis, this concept is referred to as the *context* of a scene. In addition, in real life there are often objects that have a similar shape to the objects of interest, but belong to different classes. If these objects, dubbed background objects, are not present in the training data, this can lead to a high number of false positive detections. Finally, neural networks can distinguish important details from unimportant ones if both are present in the training data. Therefore, local distortions of object surfaces due to measurement noise in images taken by a real camera can confuse a detector if it has not learned to ignore them, which it can only learn to do if such distortions are present in the training data.

This chapter focuses on 3D indoor scenes, in particular single-view depth images that can be easily converted into point clouds, with annotations in the form of oriented 3D bounding boxes of objects and per-point semantic segmentation. Various factors can be considered when creating realistic synthetic depth images for machine learning: presence of background objects, camera noise, positioning of objects, context of scenes, object sizes, texture, occlusion, environment layout, reflectance properties etc. The goal in this chapter is to gain insight into the importance of the first five of the listed factors. The approach used in this chapter is to generate synthetic depth images using the co-occurrence, placement, and size of objects from real scenes and train an object detector using these images. In this way, the baseline object detector is obtained, which is tested on real test scenes. Furthermore, additional synthetic depth image datasets with random object classes, arrangements and sizes are also generated. These additional datasets are used to perform an ablation study by comparing the performance of the same object detector trained on these additional datasets with the results of the baseline object detector. Analogously, the importance of the presence of background objects in synthetic scenes and realistic camera simulation are also investigated.

For this to be possible, a method for synthetic dataset generation that controls for realism of these factors is necessary. In this chapter, a modular method for generating synthetic 3D data of indoor scenes is proposed. The method consists of object selection, scaling, and positioning modules that can be configured to use parameters from real

scenes or random values. In addition, it includes modules for introducing background objects and simulating camera noise that can be turned on and off. The method allows for creation of various datasets, including a dataset with an object arrangement similar to that of a real dataset, but with different object instances, which can be used as a baseline. By systematically eliminating the aforementioned five factors of realism one by one, an ablation study can be conducted to determine the importance of each of those factors.

3.1 Existing methods for synthetic 3D data generation

3D synthetic data generation. Most methods for 3D synthetic data generation are designed for outdoor environments. Ros et al., 2016 capture scenes of a man-made model of a city, providing a highly controlled setting for generating consistent data for various tasks. Similarly, Johnson-Roberson et al., 2017, Saleh et al., 2018, and Hu et al., 2019 use models of cities from the video game GTA V to generate synthetic data. These methods leverage the rich, detailed environments available in GTA V to simulate diverse urban scenes, which are useful for tasks like object detection, semantic segmentation, and autonomous driving research.

When it comes to indoor scenes, Handa et al., 2016 manually create indoor environments, placing furniture and objects to simulate realistic room layouts. These scenes are then captured from various angles to generate a comprehensive dataset, dubbed SceneNet. Furthermore, they propose an automatic furniture arrangement method which allows them to augment these scenes and obtain even more variety in their dataset. McCormac et al., 2017 build upon SceneNet by sampling random layouts from it and random objects from ShapeNet (Chang et al., 2015). By using a physics engine to generate random positions, a practically unlimited number of scene configurations can be created. A few examples from their dataset, called SceneNet RGB-D, are shown in Figure 3.1.

Figure 3.1: A few examples from the SceneNet RGB-D synthetic dataset McCormac et al., 2017.

The synthetic scenes in Tremblay, To, and Birchfield, 2018 are generated using the NVIDIA Deep Learning Dataset Synthesizer (To et al., 2018). This tool allows for the creation of highly customizable environments. In their study, the authors selected three different settings — a kitchen, a sun temple, and a forest — and manually chose 15 locations within these environments to cover a variety of terrain and lighting conditions. During scene generation, models were randomly placed and oriented within a vertical cylinder at the selected locations, introducing variability and complexity into the dataset.

Leonardi et al., 2022 present a pipeline and tool in Blender (Community, 2018) for generating synthetic 3D data. This tool allows users to adjust various parameters, including camera position, lighting, and object colors, providing a high degree of control over the generated scenes. This flexibility makes it possible to create a wide range of scenarios tailored to specific research needs.

Although there are a number of approaches to generating synthetic datasets, none of them is suitable for performing the ablation study presented in this chapter. In the approaches presented in McCormac et al., 2017, Tremblay, To, and Birchfield, 2018, and Leonardi et al., 2022, the objects are randomly positioned, which does not allow for a study of the influence of realistic object arrangement. Moreover, these approaches assume that the models of the objects inserted in the scene have realistic sizes, while the method described in this chapter can use models with normalized sizes and retrieves realistic object sizes from a distribution obtained from a real dataset. Finally, a method that reconstructs walls from real images is proposed, ensuring a wall layout similar to that of a real dataset, which is not done in any of the related research considered in this section.

Imperfect camera simulation. There are two main approaches to generate camera noise: algorithm-based and deep learning-based methods. The method in Gschwandtner et al., 2011 models realistic cameras by simulating light rays to obtain precise distances from points to the camera. It uses 3D models to directly generate noisy images. Similarly, the methods in Landau, Choo, and Beling, 2016 and Planche et al., 2017 use 3D CAD data and render noisy images by modeling real sensors. Considering deep learning methods, two GAN-based methods are proposed in Shrivastava et al., 2017 and Bousmalis et al., 2017, while the method proposed in Sweeney, Izatt, and Tedrake, 2019 is a CNN which predicts noise in simulated depth images. There is no universally accepted method for comparing camera simulators, apart from the qualitative assessment of the images produced by a particular method.

3.2 Method

In this chapter, a method for generating realistic synthetic depth images for training 3D object detectors for indoor scenes is proposed. The method is suitable for studying the importance of various factors involved in this process. It requires two datasets, a dataset containing real scenes with bounding boxes of objects described by the object's class, size, and position, and a dataset with 3D CAD models of objects commonly found in indoor environments.These datasets are referred to as the scene dataset and the model dataset, respectively.

In the proposed data generation method, there's a distinction between foreground classes and background classes including walls. Foreground classes are the classes of interest, which an object detector is tasked with recognizing, while background classes are the remaining objects which can appear in certain indoor scenes.

The method has a modular structure (see Figure 3.2), and takes into account the following five factors, which are important for the creation of realistic synthetic scenes: camera noise, presence of background objects, positioning, context, and size. A separate module is created for each of these five factors. While camera noise and the presence of background objects can simply be turned off, there are two modes of operation for the other three modules, ground truth based mode and random mode. The positioning, context, and size modules are described in subsection 3.2.1. More information about inserting walls into the scenes can be found in subsection 3.2.1, while the camera noise module is described in subsection 3.2.2.

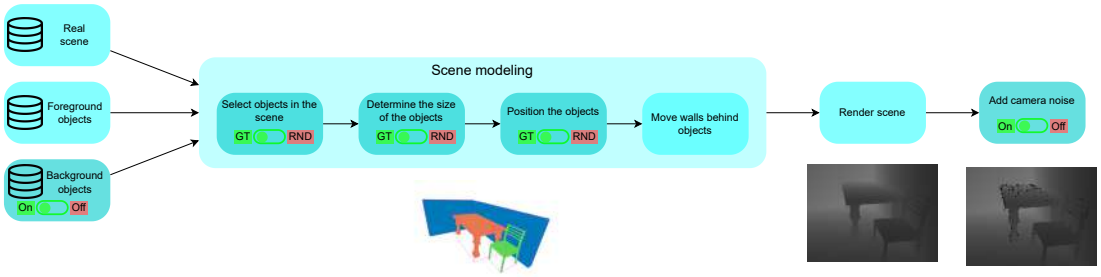

Figure 3.2: Illustration of the modular synthetic data generation method. The five modules are shown in dark cyan. The background objects module and the camera noise module can simply be turned on or off, while the other three have two modes of operation, ground truth based mode (GT) and a random mode (RND). To generate data, a scene is first modeled by selecting, scaling and positioning objects. Walls are then moved backwards in order to avoid occlusion of objects in the scene. The final model of the scene is comprised of 3D mesh models, where each object is assigned its bounding box. After this, the scene is rendered with an ideal camera, and, finally, the noise is (optionally) added to the image. The point cloud is then generated from the final depth image.

3.2.1 Scene modeling

A typical indoor scene consists of objects on a horizontal floor surface and walls. Therefore, to create a scene, the classes of objects that should appear in the scene, need to be determined first. Next, sizes and specific instances of the objects must be selected. These objects are then placed in the scene and finally the positions of the walls are adjusted to avoid obscuring other objects. However, these modules are not independent of each other. When ground truth context is used, ground truth sizes are also used. Furthermore, when ground truth positioning is used, the ground truth context and sizes are also used.

Context

In the ground truth based mode, a synthetic scene is generated from a real scene from the scene dataset using the models from the model dataset. Each object in the real scene is represented in the synthetic scene by a model of the same class, providing the context for the scene. The camera is positioned at the same height above the floor as in the real scene.

To evaluate the contribution of context, in the random mode synthetic scenes are generated without realistic context by drawing a random number of objects (n_{obj}) of random classes and a random number of walls (n_w) drawn from uniform distributions on the intervals $[n_{obj,min}, n_{obj,max}]$ and $[n_{w,min}, n_{w,max}]$, respectively. Wall dimensions in meters are drawn from uniform distributions on intervals $[1, 2]$ for width and height, and $[0.4, 0.5]$ for depth. Furthermore, the camera height is randomly selected from a normal distribution with mean μ_c and standard deviation σ_c obtained by fitting a normal distribution to the camera height values from the scene dataset. The values of the parameters used in the experiments are shown in Table 3.4.

Object size

There are many datasets with 3D CAD models which do not have realistic sizes and instead consist of normalized models, e.g., ShapeNet (Chang et al., 2015). To use these

datasets, a size needs to be assigned to each object and the object model scaled to that size before placing it in a scene.

The object size is defined with three values: h, w_p, and l_p. These values are defined as the height of the object, the width divided by the height, and the length divided by the height, respectively. In the ground truth based mode, the height h of the object in a synthetic scene is equal to the height of the bounding box of the corresponding object in the real scene. When generating synthetic scenes without a realistic context, the height and proportions of an object are determined by taking a random bounding box of the desired class from the scene dataset.

A suitable model for the given size is determined by randomly selecting models of the desired class from the model dataset and comparing the w_p and l_p values of the models to the w_p and l_p values of the desired real bounding box. Once a model is found where both values are within a threshold p_{th} of the desired values, that model is selected as a suitable one. If no suitable model is found within a maximum number of trials $n_{try,max}$, the desired values w_p and l_p are swapped and the process is repeated. The repeated matching with the swapped w_p and l_p is performed because some models may not be rotated in the same way as other models of this class and because some of the bounding boxes in the scene dataset may not be oriented correctly. If a suitable model is found for swapped w_p and l_p, it is rotated 90° and used. However, if no suitable model is found, a model with the best w_p and l_p values is selected, i.e. the mean squared error of these values is minimized. After a model is selected, it is scaled to the desired height h and placed on the scene.

To evaluate the contribution of the correct object size, synthetic scenes are generated without using realistic object sizes, but by drawing the object height randomly from a uniform distribution on the interval $[h_{min}, h_{max}]$. In this case, a random model of the desired class is selected and scaled to the selected height.

Object positioning

In the ground truth based mode, the models from the model dataset are positioned in a synthetic scene based on the positions of the bounding boxes of the corresponding objects in a real scene.

Since the bottom of some bounding boxes in the scene dataset is not exactly on the floor plane, the bottoms of the models are aligned with the lowest bounding box. This step is skipped for objects whose bottom is at a predefined height b_h or more from the floor, since some objects are not supposed to be on the floor at all, e.g. bookshelves on the wall.

To evaluate the contribution of realistic object positioning, an alternative positioning method is used in which objects are randomly positioned. In this case, the bounding boxes of the objects from a given scene are taken, but their positions are disregarded and instead the objects are randomly placed on the scene, ensuring that no object is in collision with another object. Next, an efficient method for randomly positioning objects is described.

The *object plane* is defined as the horizontal plane at a certain vertical distance s relative to the camera (see Figure 3.3). Furthermore, the object *reference point* is defined as the orthogonal projection of the object center onto the object plane. The approach used for random object positioning is based on determining a set F of points that satisfy the following conditions:

1. The point is visible in the image.

2. If the point is the reference point of an object, then the depths of all points of this object are within a predefined distance range $[z_{min}, z_{max}]$ from the camera.

3. If the point is the reference point of an object, then the object is not in collision with other objects in the scene.

To explain the random object positioning approach, the *scene reference frame* is defined with the origin at the camera viewpoint, the z-axis antiparallel to the gravity axis, and the x- and y-axes parallel to the floor plane. Additionally, the *camera reference frame* is defined with the origin at the camera viewpoint and the z-axis identical to the optical axis of the camera. The relative orientation between these two reference frames is determined by the extrinsic parameters of the camera, which are taken from a random scene in the scene dataset.

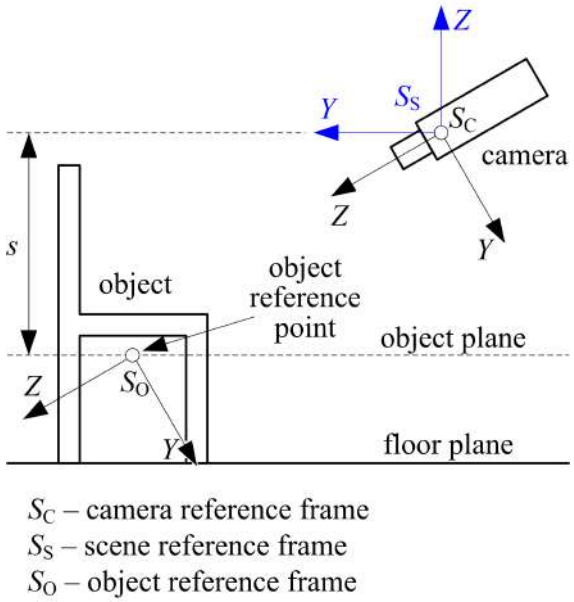

Figure 3.3: An example of the reference frames: scene reference frame (S_s), camera reference frame (S_c), and object reference frame (S_o).

The *object reference frame* is centered at the object reference point with its axes aligned with the axes of the camera reference frame. Let O be the set of all points of an object, $z_{O,min}$ and $z_{O,max}$ be the minimum and maximum z-coordinate of all points $q \in O$ with respect to the object reference frame. Then, if the z-coordinate of the object reference point with respect to the camera reference frame is within the interval $[z_{min} - z_{O,min}, z_{max} - z_{O,max}]$, depths of all object points are within the interval $[z_{min}, z_{max}]$.

Let $p \in \mathbb{R}^3$ be a vector representing the 3D coordinates of a scene point with respect to the camera reference frame. The z-coordinate of a point, i.e. the third component of the vector p, represents the *depth* of that point. The image projection of a point p can be computed by

$$m = f_{pr}(Kp), \tag{3.1}$$

where $f_{pr} : \mathbb{R}^3 \to \mathbb{R}^2$ is a function defined by

$$f_{pr} = [x_1/x_3, \;\; x_2/x_3],$$

K is the camera intrinsic matrix and $m \in \mathbb{R}^2$ is the coordinate vector of the point representing the image projection of the 3D point p, where $m = (u, v)$.

Let A be the set of all image points computed by Equation 3.1 from 3D points p that lie on the object plane and whose z-coordinates are within the interval $[z_{min} - z_{O,min}, z_{max} - z_{O,max}]$. Then, each image point $m \in A$ corresponds to a 3D point which satisfies the first two conditions in the definition of the set F.

A is the set of all image points whose u coordinate lies in the interval $[u_{min}, u_{max}]$, and v coordinate lies in the interval $[v_{min}, v_{max}]$. Since the x-axis of S_c is parallel to the x-axis of S_s, u_{min} and u_{max} are constrained by the width of the image. v_{max} is the projection of the maximum value of the y-coordinate, i.e. the second component of the vector p, onto the image. The maximum value of the y-coordinate is the y-coordinate of the line on the object plane whose z-coordinate is equal to $z_{min} - z_{O,min}$. Analogously, v_{min} is constrained by the minimum value of the y-coordinate.

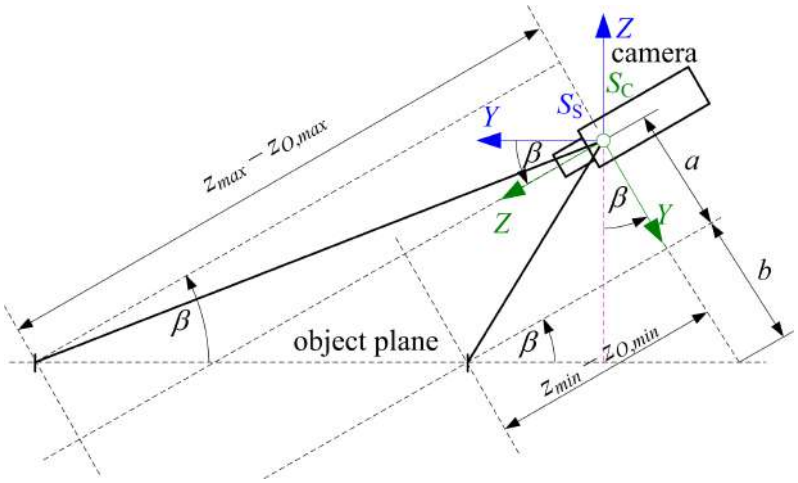

Figure 3.4: A schematic representation of calculation of the set A. β is the tilt of the camera, determined by the extrinsic parameters. s, the distance from the camera reference frame to the object plane, is shown in magenta.

A schematic representation of calculation of the set A is shown in Figure 3.4. a represents the maximum y-coordinate which can be projected using the pinhole camera model to determine the value v_{max}:

$$v_{max} = f_y \frac{a}{z_{min} - z_{O,min}} + v_c, \tag{3.2}$$

where f_y is y-value of the focal length of the camera and v_c is the second component of the principal point. As per Figure 3.4, a can be calculated from the following equations:

$$a + b = \frac{s}{\cos\beta},$$

$$\frac{b}{z_{min} - z_{O,min}} = -\tan\beta,$$

where s is the distance from the camera reference frame to the object plane and β is determined by the extrinsic parameters of the camera, i.e. it is the tilt of the camera.

Plugging these into Equation 3.2, v_{max} can be calculated as follows:

$$v_{max} = f_y \frac{\frac{s}{z_{min} - z_{O,min}} + \sin\beta}{\cos\beta} + v_c. \tag{3.3}$$

Analogously, v_{min} can be calculated as follows:

$$v_{min} = f_y \frac{\frac{s}{z_{max} - z_{O,max}} + \sin\beta}{\cos\beta} + v_c. \tag{3.4}$$

Consider the third condition. If the intersection of the convex hulls of the orthogonal projections of two objects onto any plane is an empty set, then there is no collision between these two objects. Let O_i, $i = 1, 2, ..., n_O$ be the set of objects positioned on a scene and O_0 be the new object that needs to be positioned on the scene. Also, let O_0^o and O_i^o be the orthogonal projections of O_0 and O_i onto the object plane respectively. Furthermore, let H_0 and H_i be the convex hulls of O_0^o and O_i^o respectively. To determine a collision-free set of positions of O_0, an approach based on the Minkowski difference is used, which is common in robot path planning. The alternative definition of the Minkowski difference is used in this context. For vectors $A, B \in \mathbb{R}^2$, it is defined similar to the Minkowski sum but with vector subtraction instead of addition:

$$A - B = \{a - b \mid a \in A, b \in B\} = A + (-B),$$

The convex polygon S_i obtained by the Minkowski difference of H_i and H_0 is the set of all centers of H_0 for which the intersection of H_i and H_0 is a nonempty set. S_i is then a set of points in which the orthogonal projection of the center of O_0, C_0^o, should not be placed, i.e. if $C_0^o \notin S_i$, objects O_0 and O_i will not be in collision. An illustration of this process is shown in Figure 3.5.

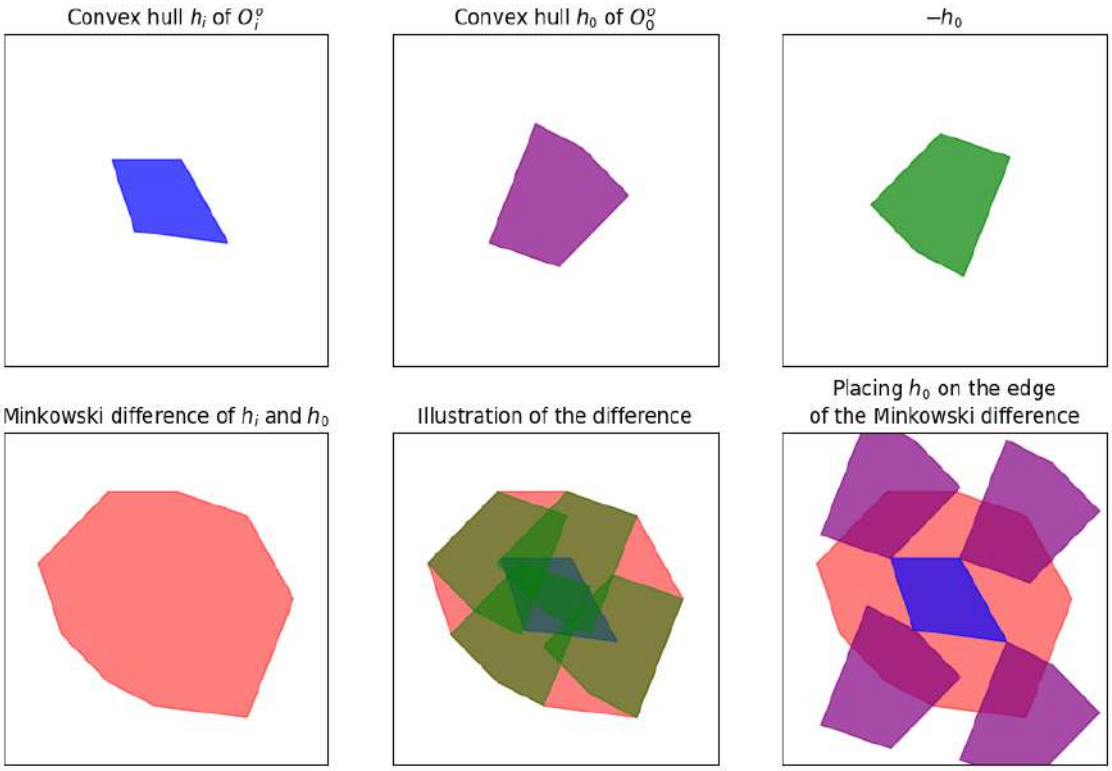

Figure 3.5: An illustration of the Minkowski difference of the convex hulls of O_i^o and O_0^o. The bottom middle image shows an illustration of the Minkowski difference of h_i and h_0, i.e., it shows that the vertices of the sum can be obtained by placing the center of $-h_0$ on vertices of h_i. The bottom right image shows that placing the convex hull h_0 on the edge of the Minkowski difference doesn't lead to a collision between h_i and h_0.

By computing S_i for each i, and subtracting the image projections of all polygons

S_i from A, the set of image points A_{free} is obtained. Each image point $m \in A_{free}$ corresponds to a 3D point which satisfies all three conditions in the definition of the set F. Consequently, F can be determined as the set of all points p on the object plane for which there is a point $m \in A_{free}$ that satisfies (3.1).

The positioning of a new object O_0 on the scene can be performed using the following algorithm.

1. Chose a model as described in 3.2.1 and give it a random rotation about the z-axis of the scene reference frame.
2. Compute F based on the object to be inserted (O_0) and the objects already positioned.
3. Select a random point c_c in F and transform this point into the scene reference frame using the extrinsic parameters of the camera to obtain c_s.
4. Position the object O_0 in the scene so that its reference point is identical to c_s.

An example of the described process is shown in Figure 3.6. It shows a few steps of the process where A_{free} is calculated for each step and a random point from A_{free} is selected as the orthogonal projection of the center of the object O_0 onto the object plane. Object arrangements in each step are also shown. The final object arrangement is the result of this process and the shown depth image is a rendering of the generated scene.

Walls

Walls represent important background objects that are always present in indoor scenes. Therefore, it can be beneficial to include them when generating synthetic scenes for 3D object detection training. For this purpose, the information about the positions of the walls is taken from the scene dataset and the walls are placed in the synthetic scenes in the same position and orientation as the detected walls. However, most datasets do not have labelled bounding boxes of walls, as is the case in SUN RGB-D (Song, Lichtenberg, and Xiao, 2015), and the walls must be detected before being placed in a corresponding synthetic scene. The scene generation approach used in this chapter involves the detection of walls in depth images of real scenes from the scene dataset. The proposed approach requires semantic annotation of the scene dataset, where all pixels representing walls are annotated. Since in some scenes a semantic region may represent multiple walls, each individual wall must be detected and extracted from this region.

Detecting walls in depth images. The preprocessing step of the wall detection method consists of extracting wall points based on the annotation contained in the scene dataset. Walls are almost always perpendicular to the floor plane, so the detection of walls in 3D can be limited to the planes that are perpendicular to the floor plane. Therefore, after the preprocessing step, a 2D *wall occupancy grid* is created by projecting the 3D wall points orthogonally onto the xy-plane of the scene reference frame. This grid covers a 10 m $\times$ 10 m square area of the scene in front of the camera, which is divided into $c_w \times c_w$ square cells. Each cell is assigned the number of wall points projected onto that cell. Then, the *wall map* is created by detecting local maxima in the wall occupancy grid. The wall map is a binary image where each point corresponds to a wall occupancy grid cell. The cells with the largest value in the 5 $\times$ 5 neighborhood are assigned the value 1 in the wall map if their value is $\geq \tau_{w,1}$, while the other points in the wall map have the value 0. An example of a wall occupancy grid and a corresponding wall map is shown in Fig. 3.7.

The wall map is used as input to the Hough transform for straight line detection. Let n be the total number of points of the wall map with value 1 and let m_k, $k = 1, 2, ..., n$

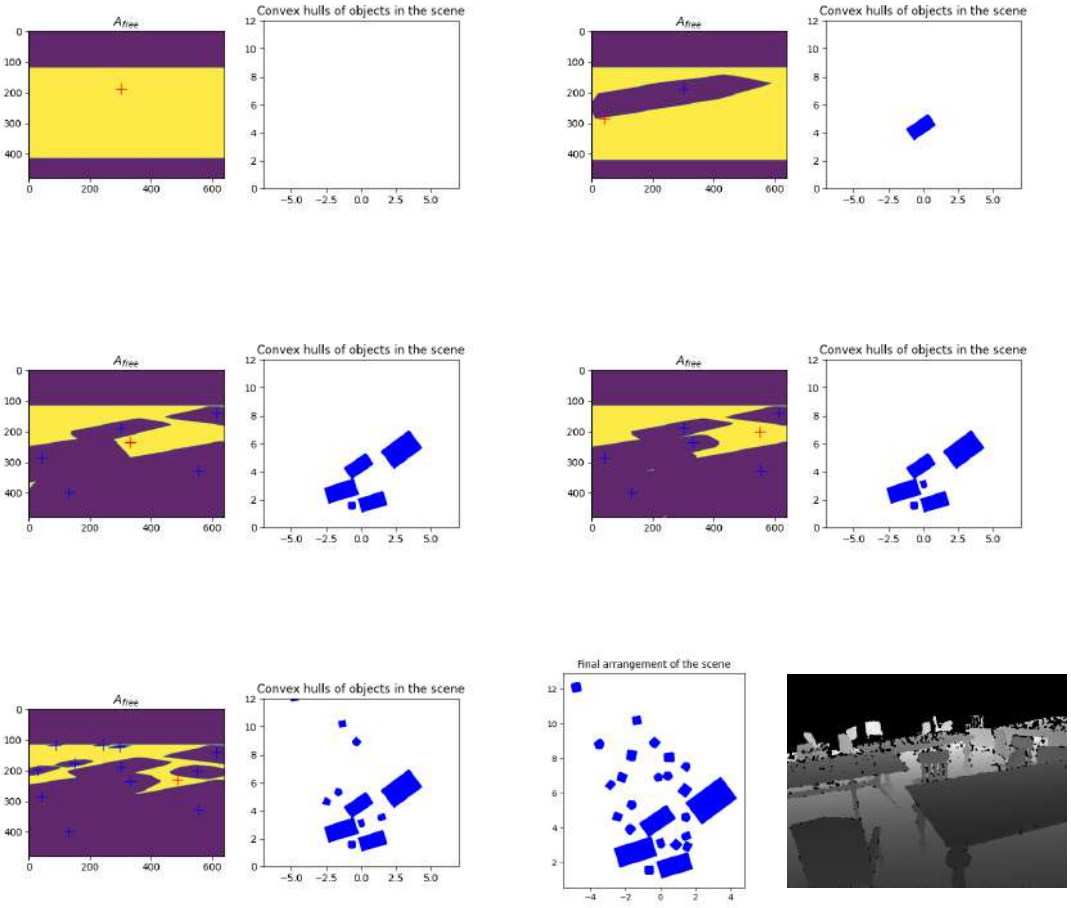

Figure 3.6: An example of positioning objects and A_{free}. Several steps of calculating A_{free} are shown. Each pair of images, except the bottom right, represents one step. Yellow region represents A_{free}, while the purple region represents the pixels in the image which do not satisfy the three conditions. Red cross is the selected center of O_0^o which will be added to the scene, it has to be in A_{free}. Blue crosses represent the centers of orthogonal projections of objects previously placed in the scene. The right image in each pair shows the arrangement of the objects currently on the scene. The last pair shows the final arrangement, and the depth image generated from that arrangement.

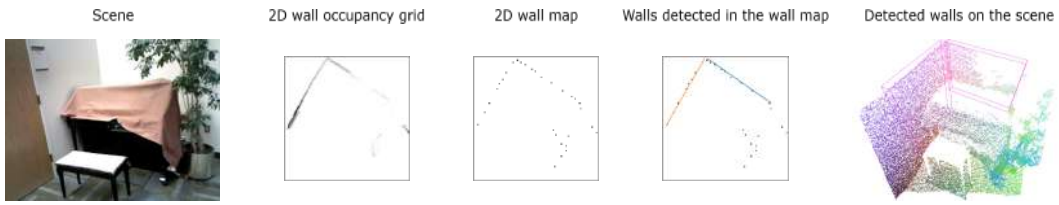

Figure 3.7: An example of the wall detection process.

be the coordinate vectors defining the positions of these points. For each line L_i, $i = 1, 2, ..., m$ detected by the Hough transform, all points satisfying

$$d_L(m_k, L_i) \leq \delta_{w,1}, \tag{3.5}$$

where d_L denotes the point-to-line distance and $\delta_{w,1}$ is an experimentally determined threshold, are identified. The orthogonal projections of these points onto L_i are referred to in this chapter as *line occupancy points*. The obtained set of line occupancy points is segmented into the minimum number of line segments Λ_{ij} such that two line occupancy points with a distance $\leq \delta_{w,2}$ belong to the same segment. Any line segment consisting of at least $\tau_{w,2}$ points is considered a wall candidate.

Wall candidates are pruned using the following non-maximum suppression (NMS) approach. Each wall candidate Λ_{ij} is assigned a score γ_{ij} defined by

$$\gamma_{ij} = \sum_{m_k \in \Lambda_{ij}} exp\left(-\left(\frac{d_L(m_k, L_i)}{\sigma_w}\right)^2\right), \tag{3.6}$$

where σ_w is an experimentally determined parameter. When two wall candidates overlap, the one of them with the lower score γ_{ij} is discarded. The overlap between two wall candidates Λ_{ij} and Λ_{uv} is measured by the similarity measure Intersection-over-Union (IoU) defined by

$$\text{IoU}(\Lambda_{ij}, \Lambda_{uv}) = \frac{\Lambda_{ij} \cap \Lambda_{uv}}{\Lambda_{ij} \cup \Lambda_{uv}}. \tag{3.7}$$

Two wall candidates Λ_{ij} and Λ_{uv} are considered overlapping if

$$\text{IoU}(\Lambda_{ij}, \Lambda_{uv}) \geq \tau_{w_3}. \tag{3.8}$$

Finally, synthetic walls are created from the wall candidates that remain after the described NMS procedure. The points in the sets Λ_{ij} used to create walls are removed from the rest of the procedure, a new wall map is created from the remaining points and the procedure is repeated until no more walls are created.

A synthetic wall created from a wall candidate is a cuboid that represents the bounding box of the scene points assigned to it. Each wall is assigned a set of scene points using a criterion analogous to that used to assign wall map points to line segments representing wall candidates. In this step, however, all scene points are considered, not only the points of the wall map. Four edges of this cuboid are parallel to the gravity axis and four edges are parallel to the line segment from which the wall is created. The above parameters used for wall detection are listed in Table 3.1.

Moving walls behind objects. Although the described wall detection method gives mostly satisfactory results, it is not perfect. It relies on the semantic segmentation of the scene dataset, which in some cases may be erroneous. Therefore, it can sometimes

Symbol	Value
τ_{w_1}	10
τ_{w_2}	8
τ_{w_3}	0.1
δ_{w_1}	3
δ_{w_2}	20
σ_w	0.8
w_w	0.2 m
c_w	0.02 m

Table 3.1: Wall Detection Parameters

produce false walls, which in some cases can obscure a large part of the scene. Furthermore, in a synthetic scene generated from a real scene in which an object touches a wall, the synthetic object representing that real object may sink into the wall due to the inaccuracy of the wall detection method and the differences between the sizes of the real objects and their representatives in the synthetic scenes. To avoid these problems, the walls are moved behind objects in generated scenes, so that they do not overlap or occlude any object in the scene. The algorithm, that corrects the wall positions, is described below.

Each wall is assigned a reference frame whose origin is at the center of the wall's front surface, i.e. the wall's surface facing the camera. The x-axis of a wall reference frame is orthogonal to the wall's front surface and its y-axis lies on that surface. Both x- and y-axes are horizontal, while the z-axis is vertical, i.e. antiparallel to the gravity axis. A top view of a simple scene is shown in Figure 3.8.

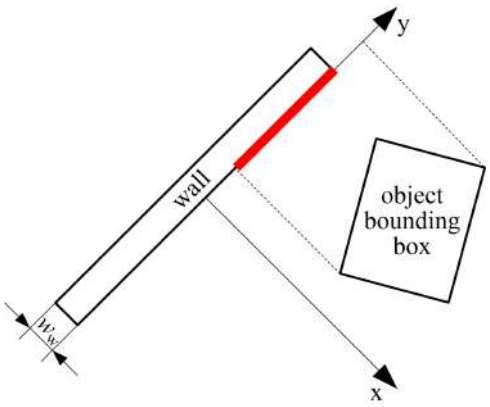

Figure 3.8: Top-down view of a simple scene consisting of one wall with assigned reference frame and one object represented by its bounding box. The orthogonal projection of the object bounding box onto the front surface of the wall is indicated by a thick red line.

Let W_i be the set of indices of all object bounding boxes whose orthogonal projection onto the ith wall front surface is not an empty set. An example is shown in Figure 3.8. Furthermore, let the distance x_{ij} between the object and the wall be defined as the minimum x-coordinate of all vertices of the jth object bounding box with respect to the ith wall reference frame. The wall is displaced in the direction of the x-axis of its reference frame by the distance μ_i which is computed as follows

$$\mu_i = \min\{\min_{j \in W_i} x_{ij}, 0\}. \tag{3.9}$$

By moving the i-th wall by the distance μ_i, situations where walls intersect or occlude objects are avoided. If $x_{ij} \geq 0$, for all $j \in W_i$, then $\mu_i = 0$, which means that the ith wall does not occlude or intersect any object and, therefore, does not need to be moved.

3.2.2 Rendering a scene

To render a scene, in all the considered cases, the extrinsic parameters of the camera are taken from the scenes in the scene dataset. In the cases where context is used, the synthetic scene is generated with the extrinsic parameters of the corresponding real scene, while in the cases where context is not used, the camera extrinsic parameters of a random scene are used. Moreover, the same intrinsic camera parameters are used for each scene.

For rendering scenes the code provided by Stutz and Geiger, 2020 is used. No color data is rendered, only depth. Objects are rendered one at a time using the z-buffering technique. In addition, each pixel is labelled with the class of the object it belongs to. After all objects are rendered, the bounding boxes of objects with fewer than $n_{min,vp}$ visible pixels are discarded, since some objects in the scene may be mostly or completely occluded. If the scene does not contain bounding boxes of the foreground classes, the scene is discarded and the process is repeated. The floor is then rendered as a planar surface whose orientation with respect to the camera is defined by the extrinsic parameters of the camera and whose distance from the camera is defined by the lowest bounding box. Finally, camera noise is optionally added as described in subsection 3.2.2 and the point cloud is generated from the final depth image.

Simulating imperfect camera

To create realistic depth images, the measurement model of RGB-D cameras is used, such as Microsoft Kinect or PrimeSense. These cameras project a pattern in the infrared spectrum, captured by an infrared camera. Some ideas from Gschwandtner et al., 2011 in a simplified form are used in order to create an efficient camera noise simulation module, which operates on depth images without a need for the original mesh from which the depth image is created. There are four main types of errors caused by imperfections of real depth cameras:

1. *depth noise* - the error in depth measurement assigned to each pixel,
2. *position noise* – the effect that occurs at the depth discontinuities, where the sensor assigns foreground value to background pixels and opposite,
3. *holes* – image regions to which no depth value is assigned due to absorption or specular reflection of the camera projector beam,
4. *shadows* – image regions to which no depth value is assigned because they represent surfaces that are visible from the camera viewpoint but occluded from the projector viewpoint.

Depth noise. The depth measurement model of the considered cameras is analogous to the model of stereo vision systems. The approach proposed in Demirdjian and Darrell, 2001 is used. The camera measurements are represented in disparity space, where the noise has isotropic and homogeneous behavior. The depth camera model proposed in Khoshelham and Elberink, 2012 is used, which computes the depth z from the disparity d according to the following equation

$$z = \frac{z_0}{1 + \gamma d}, \tag{3.10}$$

where z_0 and γ are camera constants. The depth noise is simulated by white Gaussian noise with the standard deviation σ_z superimposed on the disparity value of each image pixel with a defined depth.

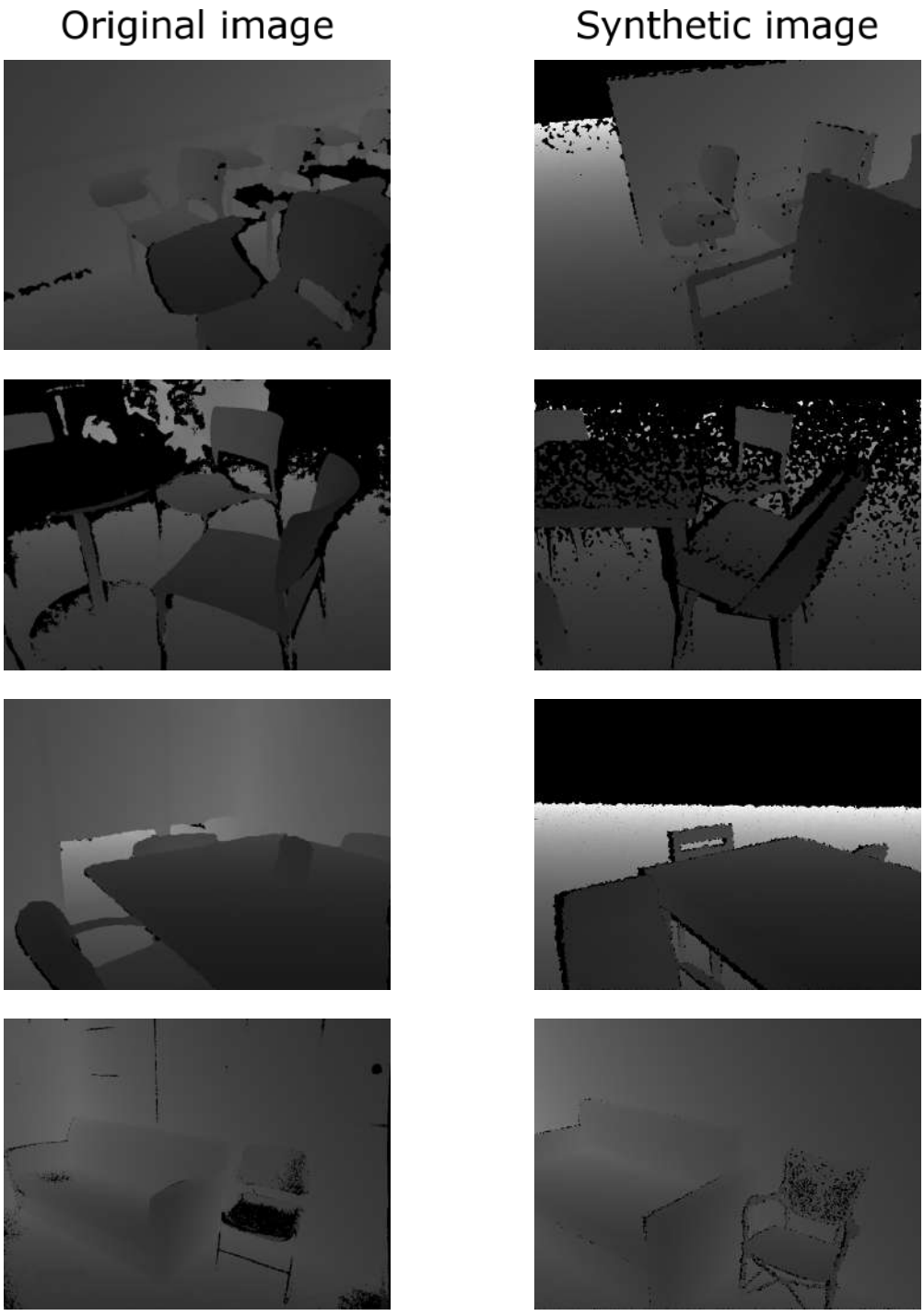

Figure 3.9: Examples of depth images from the SUN RGB-D dataset (Song, Lichtenberg, and Xiao, 2015) taken with different cameras and synthetic noisy images created based on these images using different configurations of the camera noise module.

Position noise. The position noise is simulated by creating a *displacement map* that assigns a small random displacement to each pixel m in the image. This displacement is a 2D vector $\delta(m)$ whose components are drawn from a Gaussian distribution with standard deviation $\sigma_{xy,1}$ and filtered with a Gaussian low-pass filter with standard deviation $\sigma_{xy,2}$. The displacement map is used to create a noisy depth image $z'(m)$ from the original depth image $z(m)$ by

$$z'(m) = z(m + \delta(m)). \tag{3.11}$$

Holes. The probability that the camera will detect a projector beam reflected from a given point on a surface decreases as the angle of incidence increases. Therefore, the occurrence of holes can be viewed as a stochastic process in which the probability that a given pixel is a hole increases monotonically with the angle of incidence. This process is simulated by assigning the *hole probability* to each pixel m, which is computed as follows

$$P_h(m) = P_{h,max} exp\left(-\left(\frac{\cos\theta(m)}{\sigma_{h,1}}\right)^2\right), \tag{3.12}$$

where $\theta(m)$ is the angle of incidence of the projector beam reflected from a scene point and projected onto pixel m. The parameters $P_{h,max}$ and $\sigma_{h,1}$ are experimentally determined constants. If the process of hole occurrence was modeled as white noise, i.e. assuming that the hole probability of a particular pixel is not correlated with the hole probabilities of neighboring pixels, then the holes could be simulated by making each pixel m a hole with probability $P_h(m)$ independently. However, by examining depth images acquired by real RGB-D cameras, it could be noticed that the holes mostly appear as patches rather than isolated pixels with undefined depth. Therefore, the holes are simulated by a stochastic process that produces holes in a form of small patches. First, the algorithm selects a set of pixels, which are referred to as *hole seeds*. Each pixel m can be selected as a *hole seed* with probability $P_h(m)$. The hole seeds are assigned a random value $g(m)$ from the interval $[0, 1]$. For the remaining image pixels, $g(m) = 0$. In this way, the *hole map* is created. This hole map is then filtered with a Gaussian low-pass filter with standard deviation $\sigma_{h,2}$, to produce the *filtered hole map* $g_f(m)$. Finally, all pixels with $g_f(m) > \tau_h$ are declared as holes, where τ_h is an experimentally determined threshold.

Shadows. Shadows are illustrated by Figure 3.10 which shows a top view of a scene observed by a depth camera. The depth value assigned to each image pixel represents its z-coordinate with respect to the camera reference frame. The points on the object surfaces, indicated by green lines, are visible from both the camera and the projector viewpoint. Therefore, their depth can be measured. On the other hand, the points on the object surfaces marked by red lines are visible only from the camera viewpoint, which prevents measurement of their depth. Shadows are simulated by using an algorithm that scans each image line from right to left and computes the projection ray for each image point m based on its 3D coordinates p. If the scene point p' corresponding to a subsequent image point m' has a greater depth than the depth of the point q' representing the intersection of the projection ray of p and the camera optical ray of p', then p' is occluded by p from the projector viewpoint. Therefore, the depth of p' cannot be measured and no depth value is assigned to the image point m'. An opposite example is the point p'' in Figure 3.10, whose depth can be measured because there is no scene point that occludes p'' from the projector viewpoint.

The average execution time for the camera noise module using an AMD Ryzen 7 5800X processor is 158 ms. Some examples of synthetic noisy images created with different configurations of the camera noise module are given in Figure 3.9.

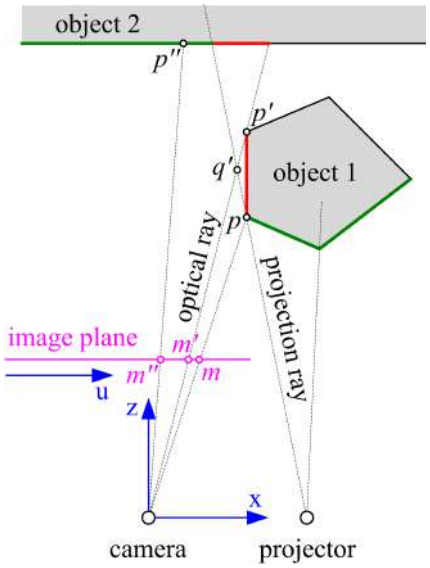

Figure 3.10: The red scene points are occluded by the green scene points from the projector viewpoint.

class	bed	table	sofa	chair	toilet	desk	drser	nigtsd	bkshf	bathtub	# of scenes	total #
bed	155	83	24	218	0	107	258	436	34	0	1096	1279
table	83	989	338	2533	2	163	9	20	132	2	3271	4843
sofa	24	338	240	343	0	53	10	13	66	0	993	1321
chair	218	2533	343	3347	2	1302	37	45	206	0	4931	18974
toilet	0	2	0	2	5	0	0	0	0	51	275	280
desk	107	163	53	1302	0	523	23	16	75	0	1549	2871
drser	258	9	10	37	0	23	49	147	13	0	330	391
nigtsd	436	20	13	45	0	16	147	73	4	0	460	540
bkshf	34	132	66	206	0	75	13	4	81	1	387	494
bathtub	0	2	0	0	51	0	0	0	1	0	105	105

Table 3.2: Co-occurrences of the foreground classes in the SUN RGB-D dataset (Song, Lichtenberg, and Xiao, 2015). Each row shows the number of scenes an object of that particular class co-occurs with an object of another class. An object co-occurs with an object of the same class if two or more objects of that class are present in a scene. Additionally, the last two columns show how many scenes contain an object of that class and the total number of objects of that class in the dataset, respectively.

3.3 Experiments

For data generation the SUN RGB-D dataset (Song, Lichtenberg, and Xiao, 2015; Silberman et al., 2012; Karayev et al., 2011; Xiao, Owens, and Torralba, 2013) is used as the scene dataset and models from the ShapeNet and ShapeNetSem (Chang et al., 2015; Savva, Chang, and Hanrahan, 2015) are used for the model dataset. To determine the effectiveness of the data generation method, VoteNet (Qi et al., 2019) and FCAF3D (Rukhovich, Vorontsova, and Konushin, 2021) are trained on the generated data and evaluated on the SUN RGB-D dataset.

3.3.1 Datasets

The **SUN RGB-D** dataset consists of single-view RGB-D images with pointwise semantic segmentation and 2D and 3D bounding box annotations with class labels. It is divided into a training and a test dataset. The training and test dataset consist of ∼5k scenes each. Only the training partition is used for data generation, the same partition used in Qi et al., 2019, and only the following 10 classes are used as foreground classes: bed, table,

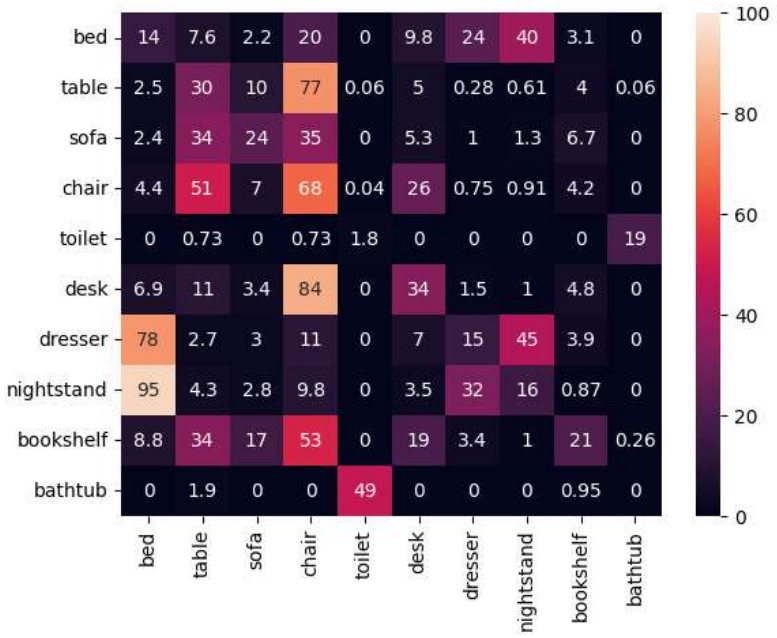

Figure 3.11: Visual representation of the co-occurrences in the SUN RGB-D dataset. The values shown are in percentages relative to the number of scenes in which objects of that class occur, i.e., number of co-occurrences for each pair is divided by the number of scenes containing an object of the class in the row.

sofa, chair, toilet, desk, dresser, nightstand, bookshelf, and bathtub. These 10 classes are used to allow comparison of the results with those of Qi et al., 2019 and Rukhovich, Vorontsova, and Konushin, 2021. In addition, the following three background classes (along with the walls) are used: lamp, bag, cabinet. Mean Average Precision (mAP) is used as the performance measure.

Table 3.2 shows the co-occurrence of the foreground classes in the whole SUN RGB-D dataset. Each row shows in how many scenes an object of that class occurs with an object of another class. An object can co-occur with an object of the same class if two or more objects of that class are present on the scene. Figure 3.11 visualizes the relative co-occurrences of the foreground classes. Each row shows the co-occurrence of the objects of that row's class with objects of other classes divided by the total number of scenes that contain an object of the row's class. The values are shown as percentages. For example, the value 77 in the second row and fourth column shows that in 77% of the scenes where a table shows up, at least one chair also shows up. Some expected values show up in these co-occurrence statistics. Dressers and nightstands rarely show up without a bed, desks are rarely in a scene where there is no chair, beds and toilets do not show up together on any scenes and so on.

ShapeNet is a 3D model dataset with about 51k unique models distributed across 55 common object classes. Since not all classes used for data generation are found in ShapeNet, the ShapeNetSem dataset is also used. The **ShapeNetSem** dataset consists of 12k models distributed over 270 classes. A model database is created by selecting 3D models from the toilet and nightstand classes in the ShapeNetSem dataset and the remaining eleven classes in the ShapeNet dataset.

Parameter	Value
Focal length x	570.34
Focal length y	570.34
Principal point x	320
Principal point y	240
Image width	640
Image height	480
Minimum distance	0.7 m
Maximum distance	9.7 m

Table 3.3: Camera intrinsic parameters

Parameter	Value
$n_{obj,min}{}^{3,4}$	1
$n_{obj,max}{}^{3,4}$	10
$n_{w,min}{}^{3,4}$	0
$n_{w,max}{}^{3,4}$	4
$\mu_c{}^{3,4}$	1.23 m
$\sigma_c{}^{3,4}$	0.27 m
$s^{2,3,4}$	1 m
$n_{min,vp}{}^{*}$	1000
$h_{min}{}^{4}$	0.5 m
$h_{max}{}^{4}$	2 m
$p_{th}{}^{B,1,2,3,5}$	0.2
$n_{try,max}{}^{B,1,2,3,5}$	500
$b_h{}^{B,1,5}$	0.4 m
Imperfect camera simulation^{B,2,3,4,5}	
$\sigma_{h,1}$	0.2
$\sigma_{h,2}$	2
$P_{h,max}$	0.3
τ_h	0.1
σ_z	1
$\sigma_{xy,1}$	6
$\sigma_{xy,2}$	3

Table 3.4: Parameters used for data generation. Parameters denoted by a * are used in all ablations, while parameters used in specific ablations are denoted by the ablation numbers in superscript (B represents the baseline case).

3.3.2 Ablation studies

For all ablation studies, the same intrinsic camera parameters were used, as shown in Table 3.3, while the other parameters are listed in Table 3.4. For each ablation, 20000 scenes were generated. Scenes were generated using an AMD Ryzen 7 5800X processor with 16 threads and the average execution time per scene for each ablation is shown in Table 3.5. Specifically, the average time per scene for generating the baseline data is 221 ms.

For VoteNet, the network architecture, training, and inference steps are the same as in Qi et al., 2019, except that the networks were trained with a batch size of 12 on a single GeForce RTX 2080 Ti GPU. FCAF3D is also trained with the same parameters as in Rukhovich, Vorontsova, and Konushin, 2021, except with a batch size of 4 on a single GeForce RTX 3080 GPU. Since the data generation method does not generate RGB data, the RGB data is, instead, filled with the value 0.5. The trained networks are tested with IoU thresholds of 0.25 and 0.5. The results of all ablations are shown in Table 3.5

Baseline. For the baseline case, all the modules described in Section 3.2 are used to obtain as realistic images as possible, creating a baseline to which a series of ablations is applied by changing the mode of individual modules or removing them. Using this

		Baseline	Baseline†	Ablation 1	Ablation 2	Ablation 3	Ablation 4	Ablation 5
Noise		+	+	-	+	+	+	+
Position		+	+	+	-	-	-	+
Context		+	+	+	+	-	-	+
Size		+	+	+	+	+	-	+
Background		+	+	+	+	+	+	-
Time(ms)		221	221	173	249	295	299	209
mAP@0.25	VoteNet	37.0	51.7	25.0	21.9	16.6	4.1	20.7
	FCAF3D	42.2	64.3	33.0	23.5	25.4	19.6	27.7
AR@0.25	VoteNet	78.7	86.9	70.6	69.3	77.0	69.9	67.9
	FCAF3D	88.0	92.9	79.7	72.6	76.6	71.5	79.1
mAP@0.5	VoteNet	17.9	26.9	8.7	8.6	4.8	0.0	7.1
	FCAF3D	27.5	43.0	18.8	13.3	13.8	11.2	15.6
AR@0.5	VoteNet	38.3	47.9	25.2	25.1	27.1	5.6	22.4
	FCAF3D	54.9	65.0	42.9	37.6	39.5	36.8	41.1

Table 3.5: Average computation time per scene and mean average precision scores of each ablation with IoU thresholds 0.25 and 0.5 on the SUN RGB-D test set. Ablations which use ground truth based modes of the size, context and position modules are marked with +, while ablations which use the random modes are marked with -. Ablations are marked with + or - for the noise modules depending on whether they use them or not. The background module is marked with + if an ablation uses background objects and with - if it does not. Baseline† uses the same model as Baseline but the model is tested on scenes with background objects removed.

baseline method for generating annotated images, mAP@0.25 of 37.0 and 42.2 is achieved with VoteNet and FCAF3D, respectively. These networks achieve mAP@0.25 of 59.1 and 64.2 when trained with real data. While these results are not as good as the results achieved with real data, they are still suitable results considering they are achieved using only synthetic data. This shows that this method can be considered an adequate method for synthetic data generation.

Ablation 1. With the first ablation, the importance of simulating camera imperfections is investigated. To generate data for this ablation, no camera noise is added to the generated depth images. Table 3.5 shows a significant drop in the performance of both networks with this ablation, from 37.0 to 25.0 mAP@0.25 or 32.4% (VoteNet) and from 42.2 to 33.0 mAP@0.25 or 21.8% (FCAF3D), indicating that the network does not learn well with point clouds generated using the model of an ideal camera. This is expected since the networks are trained using perfect data and it is hard for them to perform well when realistic, noisy data is used as input. Since FCAF3D works with voxels instead of points, this drop in performance is somewhat smaller for this network.

Ablation 2. This ablation does not use the ground truth object positions from the SUN RGB-D dataset. The significant reduction of mAP@0.25 from 37.0 to 21.9 or 40.8% (VoteNet) and from 42.2 to 23.5 or 44.3% (FCAF3D) suggests that object positioning is very important in complex scenes. For example, if a chair is heavily occluded by a table and only a small part of the chair is visible, it can still be concluded that this small visible part is the chair just because it is next to the table, which is a common arrangement of these objects in real scenes. In the case of VoteNet, it is to be expected that the positioning is important due to the underlying feature extraction network, PointNet++ (Qi et al., 2017b). This network generates feature vectors at the highest level of abstraction by collecting information from nearby object points in the range of 1.2m. Thus, when two objects are positioned at a greater distance, they provide no context to each other even though they appear in the same scene.

Ablation 3. The purpose of this ablation is to examine the importance of the context. Since it makes little sense to use the positioning of the objects without the context, neither the context nor the object position from the ground truth scenes are used in this ablation.

However, the correct size of the objects is used, which is extracted from the ground truth data as explained in subsection 3.2.1. This ablation results in a decrease in mAP@0.25 from 21.9 to 16.6 or 24.2% for VoteNet. As shown in Figure 3.11, objects of some classes are rarely present without objects of some other classes and not learning this context from the training dataset can have a negative impact on the final model when tested on real data. This performance degradation is significant, but smaller than that resulting from neglecting object position information (Ablation 2). Interestingly, there is a slight increase in mAP@0.25 from 23.5 to 25.4 or 8.1% for FCAF3D. This suggests that FCAF3D relies less on the context of the scene when information about the real object arrangement is not used.

Ablation 4. The purpose of this ablation is to investigate the importance of the realistic object size. From the obtained results, it is evident that the correct size plays an important role for VoteNet. The mAP@0.25 decreases from 16.6 to only 4.1 from Ablation 3 to Ablation 4, which is a difference of 75.3%. The mAP@0.5 even decreases to 0. The drop for FCAF3D is smaller, going from 25.4 to 19.6, which is only 22.8% drop, indicating that this network is not as affected by object size as VoteNet.

Ablation 5. For this ablation, data is generated in the same way as in the baseline case, but the background classes are not included in the data, so the generated scenes consist only of the foreground objects. The drops in mAP@0.25 are still high in this case, dropping to 20.7 or 44.1% for VoteNet and to 27.7 or 34.4% for FCAF3D. This indicates that the presence of background objects in the training data can help with detection of foreground objects as well. This is supported by the fact that, in addition to mAP, average recall drops in this ablation as well.

Baseline tested without background objects. To test the influence of background objects, an experiment is conducted in which the point clouds from the test dataset are processed by removing all points that were neither within one of the ground truth bounding boxes of foreground objects nor belonged to the floor. The floor plane was set as the upper bound of the lowest 1% of the points, and the points located at most 10 cm above the floor plane were considered to belong to the floor. The baseline models were tested with these modified point clouds and they achieved mAP@0.25 of 51.7 for VoteNet and 64.3 for FCAF3D. This is a large increase compared to the tests with the original data, indicating that the presence of background objects in the test data has a large impact on detector performance.

The AR@0.25 remains relatively high for most ablations, slightly below 70 for Ablations 2, 4, and 5 for VoteNet and never below 70 for FCAF3D. The relative changes of mAP@0.5 and AR@0.5 are higher than those of mAP@0.25 and AR@0.25 for all ablations for VoteNet and for most ablations for FCAF3D, indicating that the object localization precision achieved by the networks is strongly influenced by the factors considered since the high IoU threshold requires the model to localize objects much better.

Examples of scenes generated for each ablation are given in Figure 3.12. The original scene consists of a table, three chairs, and the background. In the baseline synthetic scene, the original objects are replaced by randomly selected models of the same class. The corresponding example for Ablation 1 is the depth image and point cloud of the same scene as captured by an ideal camera. The example for Ablation 2 consists of objects of the same classes as the baseline scene, but randomly positioned. In the examples for Ablation 3 and 4, objects of random classes are placed in the scene. Since these ablations do not use the context of the scene, they only share the extrinsic parameters of the camera with other ablations. In the example of Ablation 4, the sizes of all objects are random. For example, a large chair appears next to a small sofa. The Ablation 5 example has no wall or cabinet in the background like the baseline scene. Depth images of five more examples are shown in Figure 3.13.

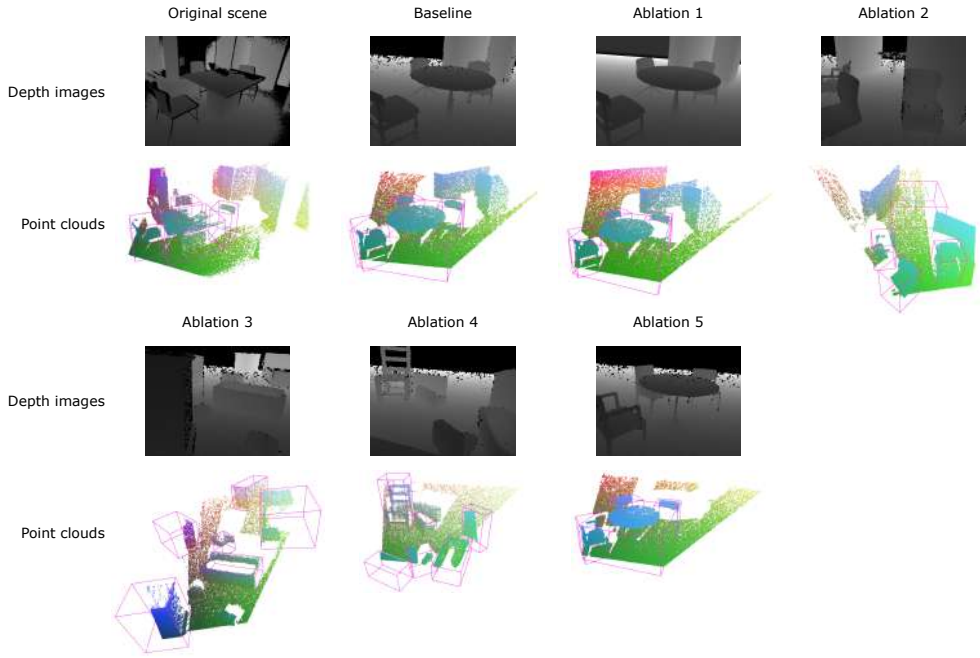

Figure 3.12: An example of a scene from the SUN RGB-D dataset (Song, Lichtenberg, and Xiao, 2015) (Original scene) and the scenes generated from it using the baseline method and the methods obtained by the considered ablations. The depth image and its corresponding point cloud with bounding boxes of foreground objects are shown for each scene.

3.3.3 Pretraining with synthetic data

Synthetic datasets are often used for pretraining as a way of improving performance of a neural network. In this section, the results of pre-training FCAF3D (Rukhovich, Vorontsova, and Konushin, 2021) on a synthetic dataset generated by the proposed method and subsequent fine-tuning on real data from the SUN RGB-D dataset (Song, Lichtenberg, and Xiao, 2015) are reported. FCAF3D trained on real RGB-D data achieved mAP@0.25 of 64.2 and mAP@0.5 of 48.9. Since the data generation method produces only depth data, FCAF3D is also trained without the RGB data to better compare the fine-tuning with regular training. With such training, mAP@0.25 of 62.6 and mAP@0.5 of 46.9 is achieved. Although the results obtained by training FCAF3D with only synthetic images generated by the presented method are not as good as the results of this network trained with real images, it is shown in this subsection that pretraining this 3D object detector with synthetic data improves its performance. In addition, this is tested with different amounts of synthetic scenes and with different amounts of real data. All variants are trained on synthetic data for 12 epochs without learning rate decay and then fine-tuned for another 12 epochs with the same parameters as described above. The results of these experiments are shown in Tables 3.6 and 3.7.

First, consider the case where 20,000 synthetic scenes are generated and all real data is used for both data generation and fine-tuning. This approach increases the mAP@0.25 by 0.9 and mAP@0.5 by 3.0 compared to training with only the real data. Additionally, using less real data for both data generation and fine-tuning is examined. When training with only half of the real data and 20,000 synthetic scenes, the results are only slightly worse than the baseline, with decreases of 1.8 and 0.4 in mAP@0.25 and mAP@0.5, respectively. Compared to training with the same real data alone, mAP@0.25 increases by 2.5 and

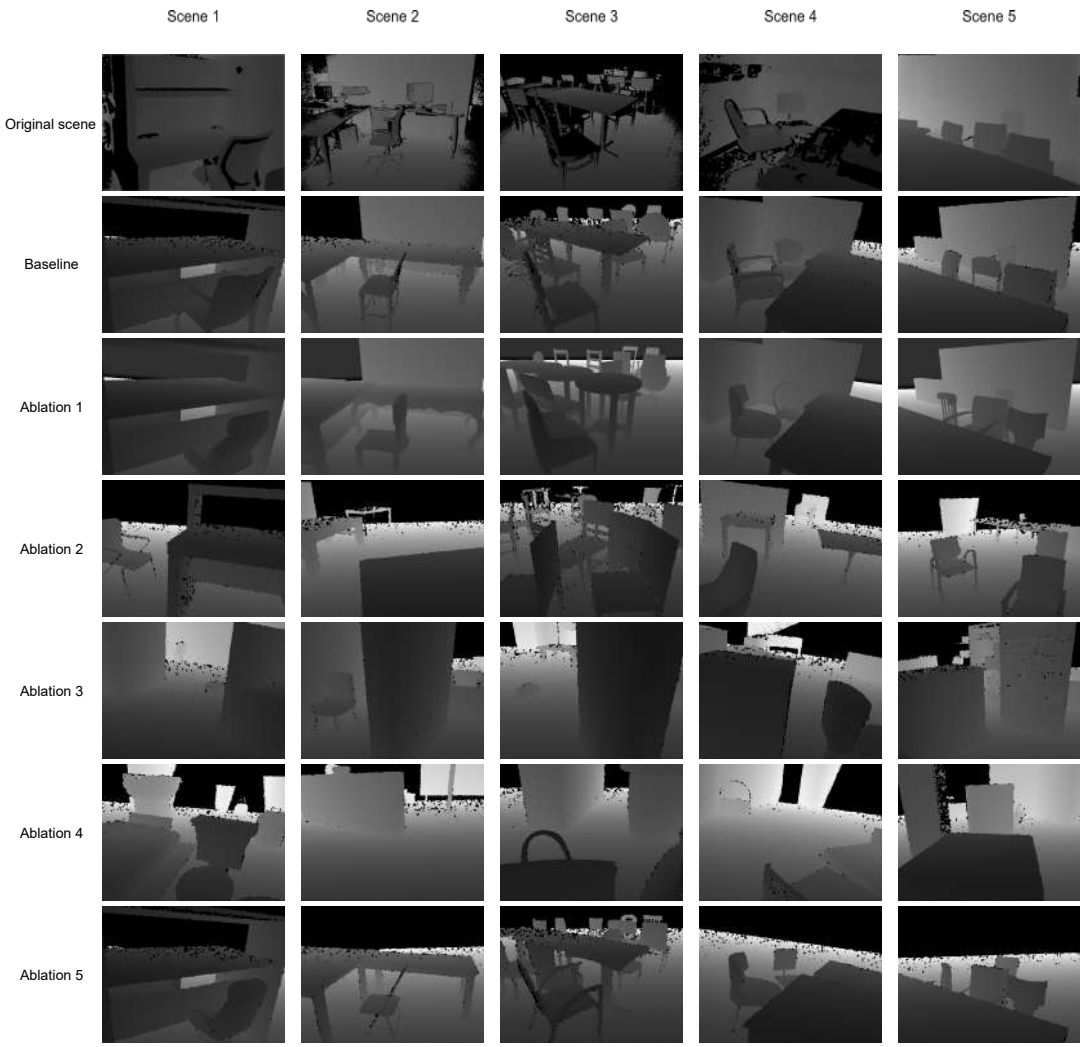

Figure 3.13: A few examples of scenes from the SUN RGB-D dataset (Original scene) and the scenes generated from them using the baseline method and the methods obtained by the considered ablations.

real data %	# of synthetic scenes	bathtub	bed	bkshf	chair	desk	drser	nigtstd	sofa	table	toilet	mAP
100%	0	75.8	87.6	31.2	80.6	32.1	36.8	70.7	69.0	51.7	90.8	62.6
100%	20k	78.3	87.3	29.6	82.0	38.4	33.7	69.6	71.5	54.3	90.4	63.5
50%	0	69.2	86.6	26.0	78.8	25.6	29.1	66.3	65.1	48.5	88.0	58.3
50%	2563†	72.1	86.1	26.3	79.3	30.0	30.9	69.7	68.8	49.8	87.8	60.1
50%	20k	70.2	87.3	27.1	80.6	33.5	31.9	70.0	68.6	50.8	87.4	60.8
20%	0	52.0	81.8	19.1	76.3	20.6	14.8	60.9	55.7	44.7	86.6	51.2
20%	4228†	60.4	84.3	21.0	77.8	29.3	19.1	60.1	66.3	46.3	85.2	55.0
20%	20k	56.8	83.8	20.0	78.3	30.1	17.0	58.6	65.5	46.5	83.5	54.0
100%	10k	79.8	87.6	31.1	80.9	34.5	32.4	70.3	72.2	53.3	92.2	63.4
100%	50k	79.1	88.1	31.2	82.3	38.3	39.0	72.8	71.6	54.5	90.6	64.8

Table 3.6: Per-class mAP@0.25 on the SUN RGB-D test set with FCAF3D (Rukhovich, Vorontsova, and Konushin, 2021) with different amounts of real and synthetic data. Values denoted by † are chosen to make the total number of scenes used for training equal to the total number of scenes present in the real training dataset.

real data %	# of synthetic scenes	bathtub	bed	bkshf	chair	desk	drser	nigtstd	sofa	table	toilet	mAP
100%	0	59.0	69.5	10.8	67.2	12.0	29.3	58.9	56.4	33.9	72.0	46.9
100%	20k	66.5	69.8	13.0	69.3	19.9	25.8	58.8	62.3	36.8	76.7	49.9
50%	0	45.9	65.6	5.9	64.4	8.1	19.8	54.1	49.9	28.5	68.0	41.0
50%	2563†	56.6	66.1	8.5	66.0	11.8	24.3	57.0	56.2	30.9	73.0	45.0
50%	20k	55.8	69.8	11.3	67.9	15.1	24.4	57.6	58.5	31.9	72.8	46.5
20%	0	21.7	58.0	3.3	60.0	4.6	6.4	48.3	38.8	22.6	61.5	32.5
20%	4228†	40.5	66.5	8.8	62.8	11.5	13.5	49.0	54.3	28.1	69.2	40.4
20%	20k	38.1	66.7	9.0	64.7	13.5	13.1	46.7	55.8	29.6	67.3	40.4
100%	10k	67.2	72.9	12.8	68.4	17.0	23.2	59.7	61.9	35.7	73.0	49.2
100%	50k	71.7	72.2	14.0	70.2	19.3	30.3	61.3	60.9	36.9	77.0	51.4

Table 3.7: Per-class mAP@0.50 on the SUN RGB-D test set with FCAF3D (Rukhovich, Vorontsova, and Konushin, 2021) with different amounts of real and synthetic data. Values denoted by † are chosen to make the total number of scenes used for training equal to the total number of scenes present in the real training dataset.

mAP@0.5 by 5.5.

A synthetic dataset containing only 2,563 scenes was also prepared, so that the total number of scenes equals the total number of scenes in the original real training set. As expected, the results of this experiment are intermediate between the results obtained using only real data and using 20,000 synthetic images for pretraining. When only 20% of the real data is used for generation and fine-tuning, the results are significantly worse: mAP@0.25 drops from 63.5 to 54.0 and mAP@0.5 drops from 49.9 to 40.4. Despite this significant drop in performance, the results are much better than when training with only 20% of the real data. In comparison, mAP@0.25 increases by 2.8 and mAP@0.5 by 7.9, which equates to a 19.6% improvement.

Similar to the experiments with 50% of the real data, 4,228 synthetic scenes were used for pretraining, bringing the total number of scenes to the number in the original real training set. However, in this experiment, the results are similar to those obtained using 20,000 synthetic images for pretraining. This suggests that using large amounts of synthetic data is not particularly useful when very small amounts of real data are used, and that smaller amounts of synthetic data may be sufficient to achieve similar results.

In addition, the network is pretrained with different amounts of synthetic data before fine-tuning on real data. Pretraining on 10,000 synthetic scenes results in slightly worse outcomes than pretraining on 20,000 synthetic scenes but still outperforms the baseline trained only on real data. Pretraining on 50,000 synthetic scenes results in mAP@0.25 of 64.8 and mAP@0.5 of 51.4, both of which are higher than the results obtained when the network is trained on real RGB-D data only.

3.4 Conclusion

This chapter provides insight into the process of generating synthetic data and the different factors involved. For this purpose, a modular method for generating single-view synthetic depth images of indoor scenes was developed. The modular architecture of the method allows data to be generated for a specific purpose and enables good analysis of the importance of the following five factors in generating synthetic data: camera noise, presence of background objects, positioning of objects, context of scenes, and object sizes. Through experiments, the importance of these factors is analysed. Although the object detector trained with only synthetic data does not perform as well as the one trained with real data, it's shown that pre-training with synthetic data and then fine-tuning with real data improves the performance of the network. The method presented in this chapter has been published in Džijan et al., 2023.

This work serves as a good starting point for further research into synthetic 3D data generation and what areas to focus on when designing methods for synthetic data generation. One future research option is to investigate the improvement of object detection performance by introducing more realistic backgrounds and clutter in synthetic scenes, e.g. small objects on tables, shelves, and other flat surfaces. Another option would be to investigate other factors in the generation of synthetic data, e.g., simulating different reflectance properties of object surfaces and adding texture to objects. Furthermore, since the research in 3D object detection is moving toward using coloured point clouds, it would be worth exploring generating RGB data together with the depth which would add even more complexity to the whole process.

4 Furniture opening classification

Once objects have been detected in a scene, a robot can have a clearer understanding of the scene and may interact with the detected object. Another important ability of a service robot designed for use in indoor environments is the ability to open doors and other openable objects. This ability enables the robot to perform many tasks such as reaching behind closed doors or opening drawers to access the contents inside them.

Traditionally, cabinets and drawers are opened by grabbing and pulling the handle that is attached to the front of this furniture. Therefore, there are various algorithms which have been developed for handle detection and grasping. However, in modern environments, it has become increasingly common to see cabinet and drawer doors without visible handles on their outer surface. These types of doors are usually opened by pressing on the outer surface near a spring mechanism located inside the door, as shown in Figure 4.1b. When this spring, shown in Figure 4.1a, is pressed, it stretches and pushes the door from the inside, allowing it to be opened. This mechanism is often called the push latch mechanism. On the other hand, there are also cabinet doors that are opened by grasping the edges of the front surface of the door, as shown in Figure 4.1c, which are referred to in this paper as hidden handle doors.

As the current methods deal with doors and drawers with visible handles suitable for grasping, it is important to develop methods which deal with these other types of opening types. This chapter focuses on the first step in this process, which is the classification of the opening type. One of the challenges we identified when attempting to classify the opening type of these doors is that when the regular handle is not present on the door surface, it can be difficult to determine the correct method for opening the door. In some cases, doors with hidden handles may resemble those with a push mechanism, making it difficult to distinguish between the two. However, when a human opens a door with a push mechanism, they typically press the door surface with their spread palm. On the other hand, when a human opens a door with a hidden handle, their fingers usually bend to grip the edge of the surface.

To tackle this issue, methods based on human demonstration were developed. To test the developed approach, a new dataset needed to created which would feature annotated images of such furniture. The dataset (Šimundić et al., 2023) consists of RGB-D images of various cabinets and drawers in their natural environments. It is described in

(a) A push mechanism

(b) An example of opening of the push mechanism door

(c) An example of opening of the door with hidden handle

Figure 4.1: Opening doors without handles

detail in Section 4.1.

The proposed methods use both the information available just from an image of an object and information available from an image of a human demonstrating the approach to opening of the object. It is described in detail in Section 4.2. Section 4.3 covers the experiments which were conducted in order to test the effectiveness of the proposed methods. The experiments show the importance of utilizing human demonstration for this task.

4.1 Dataset

For this task, a new dataset was created called HoDoor (Hands on Door) dataset (Šimundić et al., 2023). The dataset consists of RGB-D images of annotated doors, drawers, cabinet doors and closet doors of various opening types. The resolution of the images in the dataset is 640×480 pixels and the RGB and depth images are aligned. While the method presented in this chapter utilizes only the RGB data, the dataset also contains depth data making it suitable for evaluating methods which require depth data as well. Intrinsic camera parameters are included in the dataset. Some examples of various furniture found in this dataset are shown in Figure 4.2.

The dataset consists of three classes that differ in the way they are operated: push, pull, and handle. The push class describes doors without regular handles that need to be pressed on the front surface to open, while the pull class describes doors without regular handles that have some sort of indentation on one of the sides that provides space for fingers to grab and open the door. The pull class is also referred to as the hidden handle class. In contrast to the other two classes, the handle class represents ordinary doors with a regular handle on the front. The object instances are also divided into 4 categories of objects that can be opened: cabinet, closet, door and drawer. All of the objects were captured from one or more viewpoints.

The number of viewpoints is not the same for each object, as is the pose with respect to the camera. Poses of the camera between different viewpoints are not included in the dataset. For each viewpoint of the object, several images are captured. The first image represents the baseline image of the object where it is not occluded by a human hand. The other images were taken from the same viewpoint but with a hand grasping several different interaction spots of the object. Interaction spots refer to spots on an object where the object can be opened. These can be a regular handle on the door, edges of the front surface of the door, or a part of the outer surface around which the push mechanism is located inside the door. The method of capturing the images with hands consisted of a person naturally touching or holding the interaction spots at various locations on the object, trying to occlude the object as little as possible while still maintaining the natural opening approach.

Each image is annotated with a 2D bounding box of the object of interest. These objects are near the center of the images. The bounding boxes are referred to as Regions of Interest (RoIs). Figure 4.3 shows some examples of the annotations. The top row in the figure contains the baseline images with the corresponding bounding boxes, while the bottom row contains images taken from the same angle but with a hand placed on interaction spots.

Figure 4.4 shows several examples of the images in the dataset described. The figure is divided into three parts, each containing an object from the three classes described above. For each class, two viewpoints of the object are shown in separate rows. The first image in the row is the baseline image, while the other images contain hands placed on the interaction spots. Since this is a real-world dataset, the images were taken at multiple

Figure 4.2: Examples of various furniture found in the HoDoor dataset (Šimundić et al., 2023).

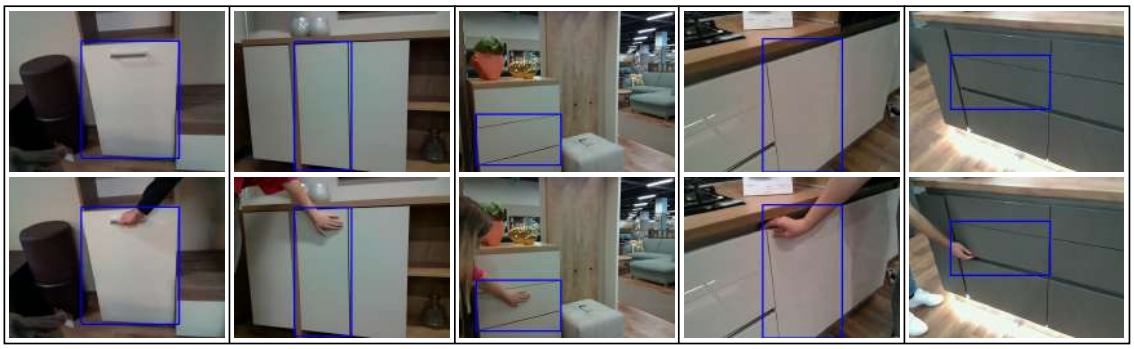

Figure 4.3: Sample images with annotations of the door of the object. Top row: images without human demonstration; Bottom row: corresponding images with human demonstration. The images in the bottom row were captured from the same angle as the top row.

	Push	Pull	Handle	All
Instances	43	104	77	224
Viewpoints	72	186	151	409
Images	489	1311	771	2571

Table 4.1: Distribution of object instances, viewpoints and images per class.

locations, and their scenes vary in complexity. This scene complexity varies depending on the number of objects in the image, clutter, and occlusion.

4.1.1 Statistics

The dataset consists of a total of 2571 images representing 409 viewpoints with 224 different object instances. A comparison of the number of instances, viewpoints, and images per class is shown in Table 4.1. The push class had the least amount of data collected because it is less common in households compared to the other two classes. The largest amount of data was collected for the pull class, as handleless kitchen objects that can be opened by pulling on one of the sides are widely used today.

The distribution of object instances among categories can be seen in Figure 4.5. Of the total 224 object instances, 109 instances are in the cabinet category, 101 in the drawer category, 13 in the closet category, while the door category, which includes ordinary doors, has 1 instance. The closet and door categories have a small number of instances because there are not many instances that belong to the push or pull classes.

In Table 4.2, a distribution of the number of viewpoints across the three classes is shown. The objects are mostly captured from 2 viewpoints, as this covers most of the object's environment, giving it enough context. If an object is placed in an area where it will be partially obscured or difficult to reach at most desired recording angles, that object is only captured from one angle. Objects were rarely captured from three angles, mainly because the third angle does not provide much new information about the object and the environment is very similar to the other viewpoints.

Figure 4.6 shows the distribution of the number of images with the baseline image and the images with hands in the viewpoints per class and overall. While there is one viewpoint with 20 images in the pull class, most viewpoints in all classes have 5 to 8 images, which are sufficient to describe how to open the respective doors with the pose of the hand. The handle class has a lower median number of images in the viewpoints

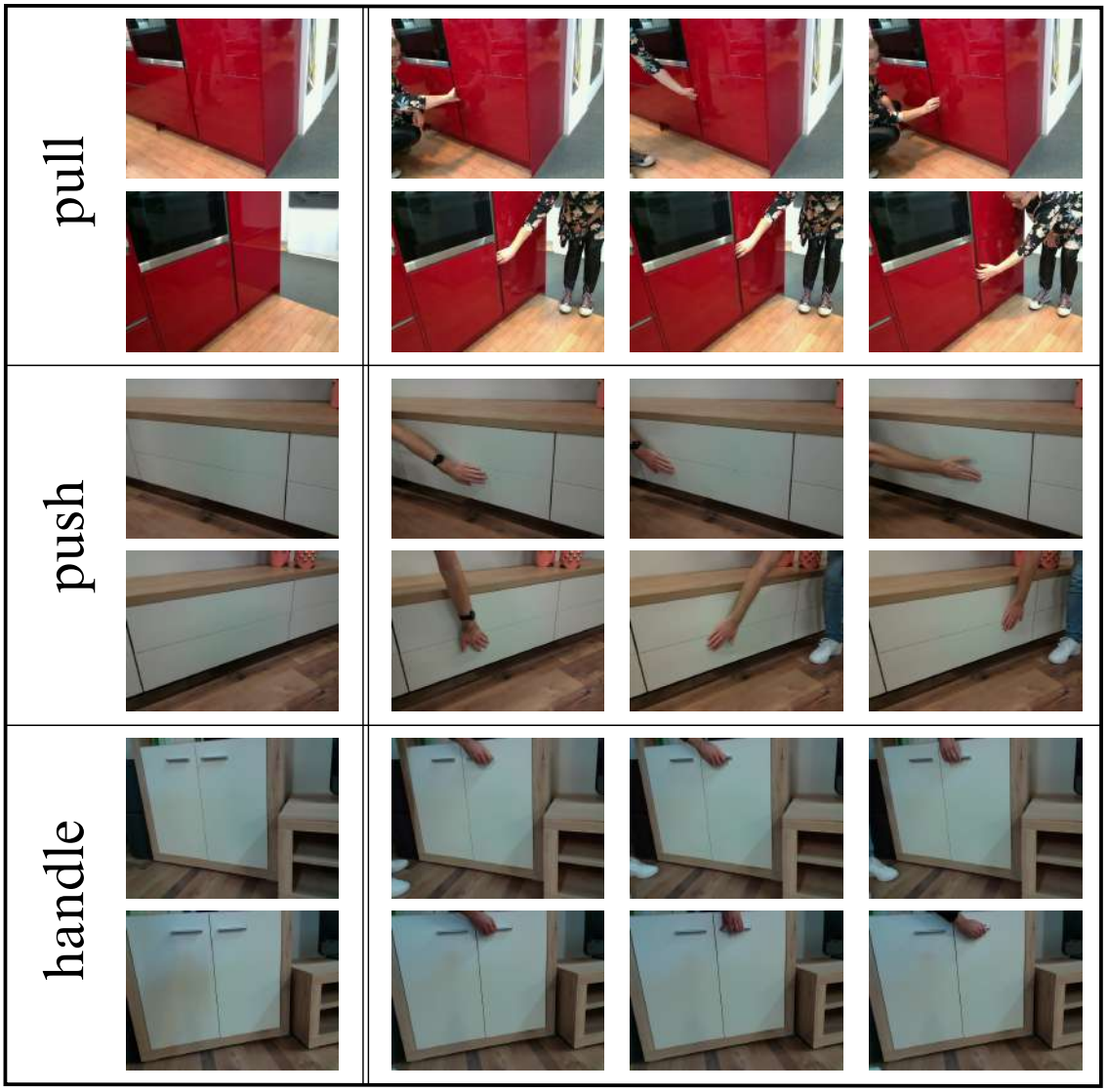

Figure 4.4: Sample images from the HoDoor dataset representing the three types of handling considered. Left: Baseline images of the same object captured from different angles. Right: images with a human demonstrating the type of opening.

Class / Viewpoints	**Push**	**Pull**	**Handle**
1	14	24	5
2	29	78	70
3	0	2	2

Table 4.2: Distribution of viewpoints across classes.

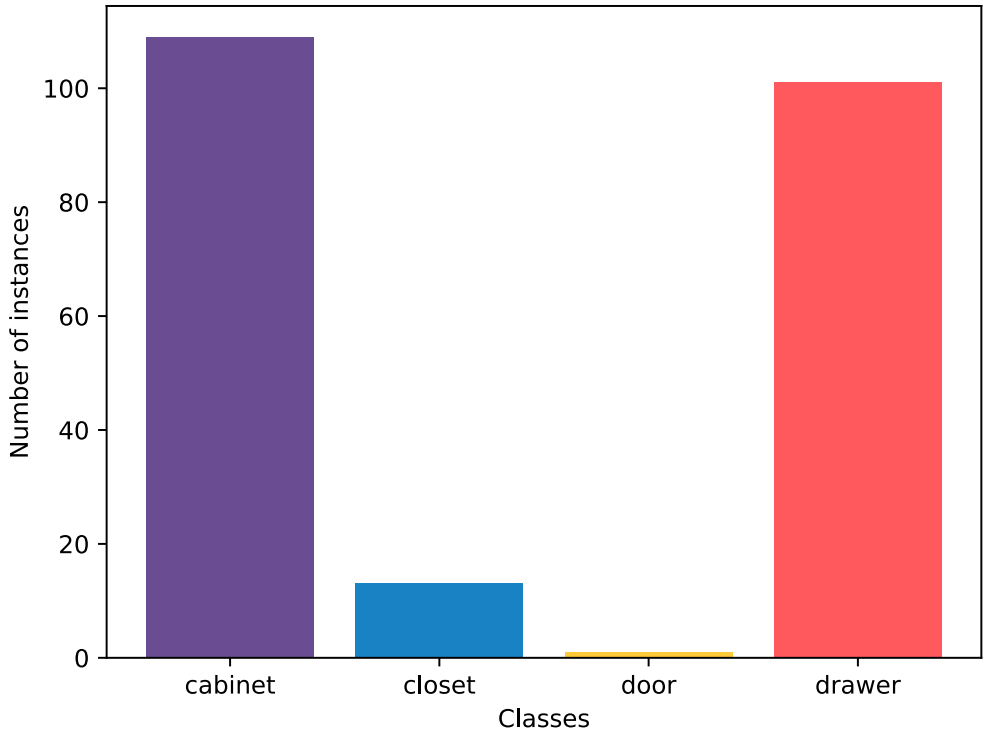

Figure 4.5: Distribution of object instances per category.

since the area of the handle is usually smaller than the area covered by interaction spots of other classes. This limits the number of possible hand poses on the handle.

4.1.2 Partitions

While there are 224 object instances in the dataset, there are a lot of sets of objects of interest which are actually parts of the same piece of furniture. For example, multiple doors and drawers might belong to the same cabinet. This makes it difficult to simply partition the dataset into train, validation, and test sets. If done randomly, similar objects could end up in different partitions, leading to a classifier that is not properly evaluated due to the overlap between the sets. To ensure meaningful evaluation, it is crucial to partition the dataset in a way that keeps related object parts within the same partition. Some objects are grouped as parts of the same piece of furniture even though they are not because they are part of the same kitchen or furniture set. Additionally, there are many similar objects, specifically, there are a lot of white kitchens and white cabinets in the dataset. For these reasons, the dataset is partitioned manually to ensure each partition is as diverse as possible, making sure that the test partition is representative of real-world data.

The dataset is divided into 45 different large objects which have various numbers of objects of interests, viewpoints, and images in total. The distribution of number of objects of interest per piece of furniture per class is shown in Figure 4.7. While some pieces of furniture have only a single object of interest, most have multiple, with the highest number being 20. After partitioning, the training set consists of 25 of these pieces

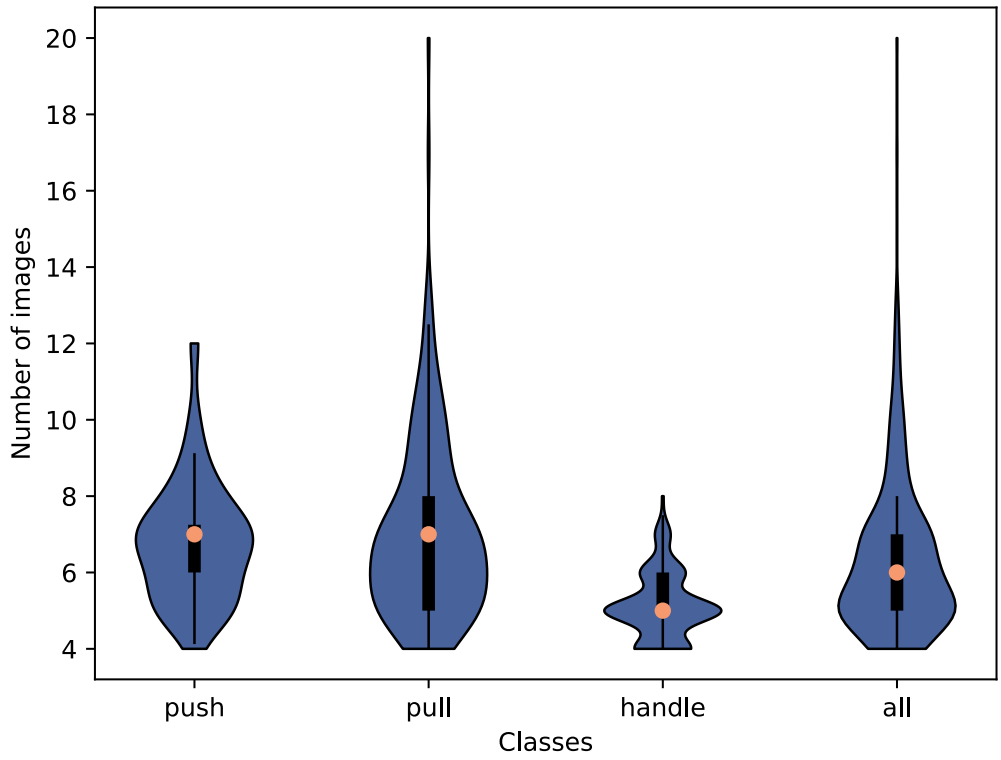

Figure 4.6: Distribution of images in the viewpoints per class and overall.

of furniture, the validation set consists of 9, and the test set consists of 11. The number of object instances, viewpoints, and images for each partition is shown in Table 4.3, which also displays the distribution of viewpoints and images for each class. Note that the number of images without human demonstration equals the number of viewpoints, while the shown number of images represents those with human demonstration.

4.2 Method

In this chapter, four methods of furniture opening classification are presented. Even though the dataset consists of RGB and depth data, only the RGB data is utilized for these methods. As previously mentioned, three types of handling are considered: regular handles (handle), hidden handles (pull), and push mechanisms (push). The goal is to classify regions of RGB images representing doors or drawers into one of these categories. As stated earlier, two sets of images are available: those with a human hand demonstrating the opening type and those without. Before an object can be classified, it first needs to be selected. This can be done by manually selecting a *Region of Interest* (RoI) or by utilizing an object detection network to detect the RoI which contains the object which needs to be classified.

Four methods of classification are proposed in this chapter. One method which uses only an image without human demonstration as input, one method which uses only an image with human demonstration as input, and two methods which use both images as input. The first two methods serve as baselines to check how well can classifiers perform with just one type of data. As the objects of the push and pull classes are very visually similar, it is expected that the first method might have issues differentiating those two classes. On the other hand, the second method might have issues differentiating objects

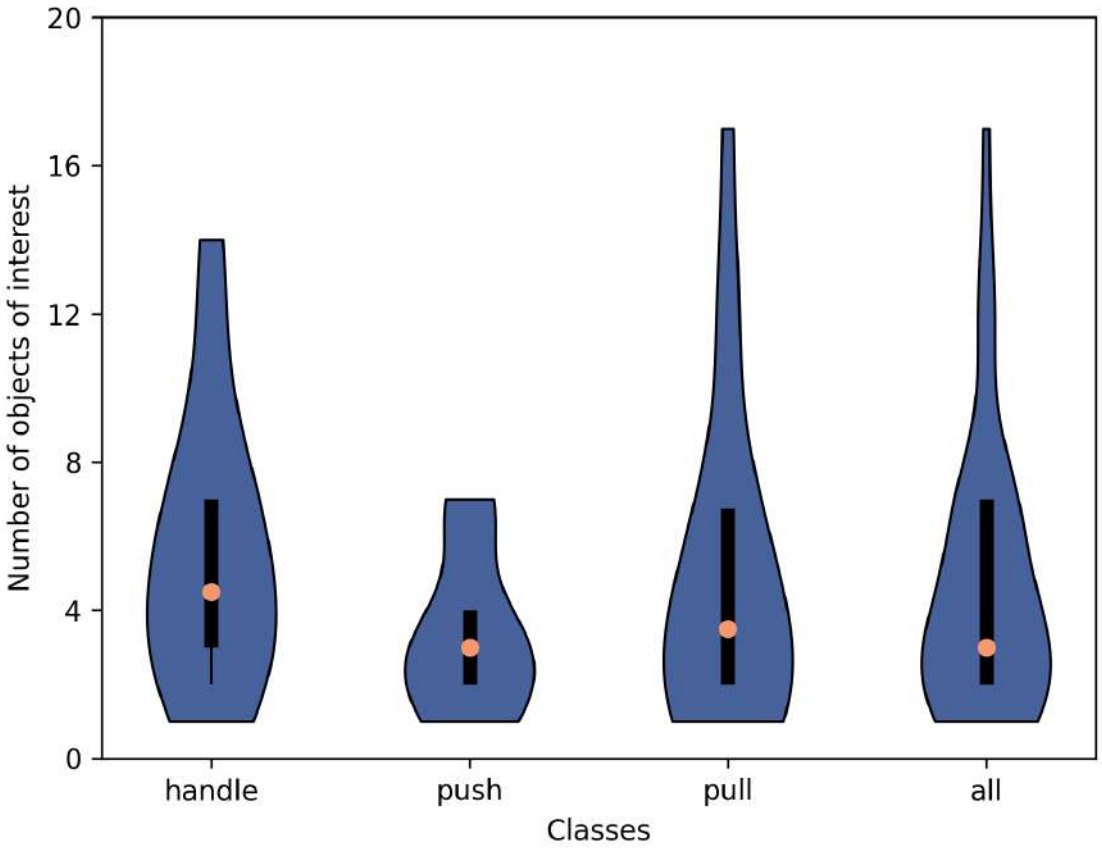

Figure 4.7: Distribution of objects of interest per piece of furniture per class.

		partition			
		train	validation	test	all
instances	push	30 (73.17%)	4 (9.76%)	9 (21.95%)	41
	pull	72 (69.23%)	11 (10.58%)	21 (20.19%)	104
	handle	56 (72.73%)	7 (9.09%)	14 (18.18%)	77
	all	158 (70.54%)	22 (9.82%)	44 (19.64%)	224
viewpoints	push	50 (69.44%)	7 (9.72%)	15 (20.83%)	72
	pull	128 (68.82%)	18 (9.68%)	40 (21.51%)	186
	handle	108 (71.52%)	15 (9.93%)	28 (18.54%)	151
	all	286 (69.93%)	40 (9.78%)	83 (20.29%)	409
images	push	272 (65.23%)	47 (11.27%)	98 (23.5%)	417
	pull	816 (72.53%)	81 (7.2%)	228 (20.27%)	1125
	handle	417 (70.68%)	61 (10.34%)	112 (18.98%)	590
	all	1535 (71.0%)	189 (8.74%)	438 (20.26%)	2162

Table 4.3: Distribution of instances, viewpoints, and images with human demonstration for all partitions. The percentages shown in parentheses represent are relative to the *all* column. For example, *50 (69.44%)* in the 7th row and 2nd column means that 69.44% of all viewpoints of the push class are in the train partition.

of the pull and handle classes since the human approach to opening of these types of objects can be similar. The latter two methods combine the information from both types of data. The input RoIs are optionally *squarified* before being used as inputs to the networks to reduce distortion, as explained in the following section.

4.2.1 *Squarification*

Since the RoIs can have very different width-to-height ratios, it can be useful to first *squarify* the RoIs to decrease the distortion of the input for the classification networks. Given a minimum size of a RoI dim_{min} and the input ROI's *width* and *height*, a new dimension w is determined by selecting the largest among *width*, *height* and dim_{min}, and increasing it by a given percentage $dim_{increase}$. The RoI is increased as to capture more context which can be useful for classifier networks. More context might also be needed for better hand detection. Given the center of the original RoI and w, a new RoI is determined using w both as its width and height. This RoI is then cropped to the sides of the image. Since the RoI is cropped to the image, that ROI may still not be a square but will be close to a square and the distortion as a result of resizing will be decreased. In addition to this, due to the increase of the RoI, this process can provide additional context for the classifier. A few examples of *squarification* are shown in Figure 4.8.

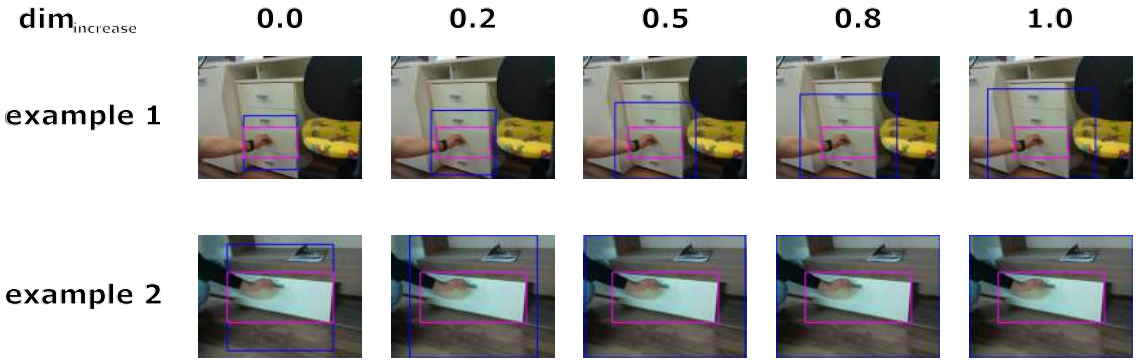

Figure 4.8: Examples of squarification with different values of $dim_{increase}$. The ground truth bounding boxes are shown in magenta, while the squarified bounding boxes are shown in blue.

4.2.2 Classification

The following four methods of classification are proposed. The first method, referred to as $method_{RGB}$, is a standard classifier based on CNN which takes an image without human demonstration as input and outputs one of the three classes. The second method, $method_{heatmaps}$, uses a hand pose detection network to detect the hand present on the image with human demonstration. This network outputs heatmaps, sometimes referred to as belief maps or confidence maps in the literature, for hand keypoints. The proposed method uses OpenPose (Cao et al., 2019) for hand pose detection. This network outputs 22 heatmaps. The first 21 heatmaps are used for the 21 keypoints of a hand (pictured in Figure 4.9), where the peak of each heatmap predicts the position of the corresponding keypoint. The 22nd heatmap is a belief map for the background. Belief maps for the background refer to the probability maps that indicate the likelihood of each pixel belonging to the background rather than to a hand. An example of a well predicted and an example of a poorly predicted hand pose and a few of the corresponding heatmaps are shown in Figure 4.10. These heatmaps are then fed to a CNN classifier to obtain the handling type. This network for hand pose detection is used instead of just using the

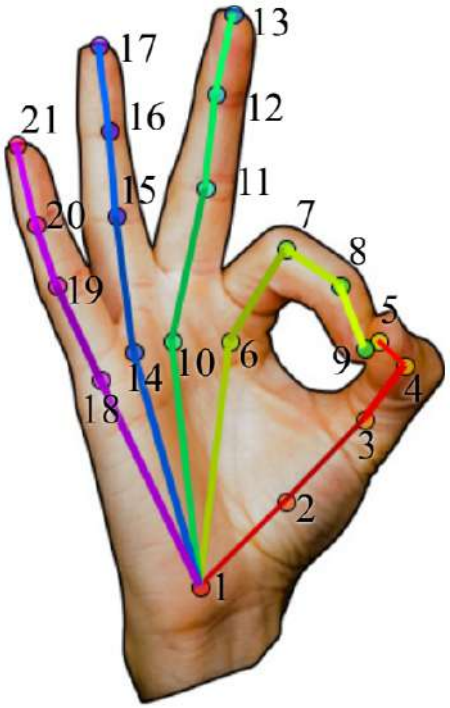

Figure 4.9: The 21 keypoints of a hand (Simon et al., 2017).

RGB data as input for the classifier because the available dataset is not very large and there are only a few different hands demonstrating the opening type. On the other hand, a network trained on a large dataset of hand poses should be more robust to images with different types of hands, environments, and conditions, thus making the proposed method more robust as well.

The last two methods combine the aforementioned methods. The third method, dubbed HoDoorNet (Figure 4.11), uses the feature extractors used for the first two methods, concatenates the features they output and classifies the set of images. As input this method, thus, requires both an image without human demonstration and the heatmaps for hand keypoints extracted from the image with human demonstration. The fourth method, referred to as $method_{hierarchical}$ (Figure 4.12), utilizes the fact that the handle can be easily detected on images without human demonstration so it can be expected that a classifier network would be able to differentiate between cabinets with handles and those without. This is confirmed by the conducted experiments. Furthermore, it is also very hard to differentiate between objects of pull and push classes based solely on the image of the object. However, the hand gestures used for opening these two types of objects are very different. Motivated by these assumptions, a hierarchical classifier is proposed. For this method, a binary classifier is first used to determine if an object has a handle or not. This classifier uses only the image without human demonstration as input. If the object does not have a handle, another binary classifier is then used to determine if the handleless object is of the push or pull class. This classifier, on the other hand, uses only the heatmaps for hand keypoints as input.

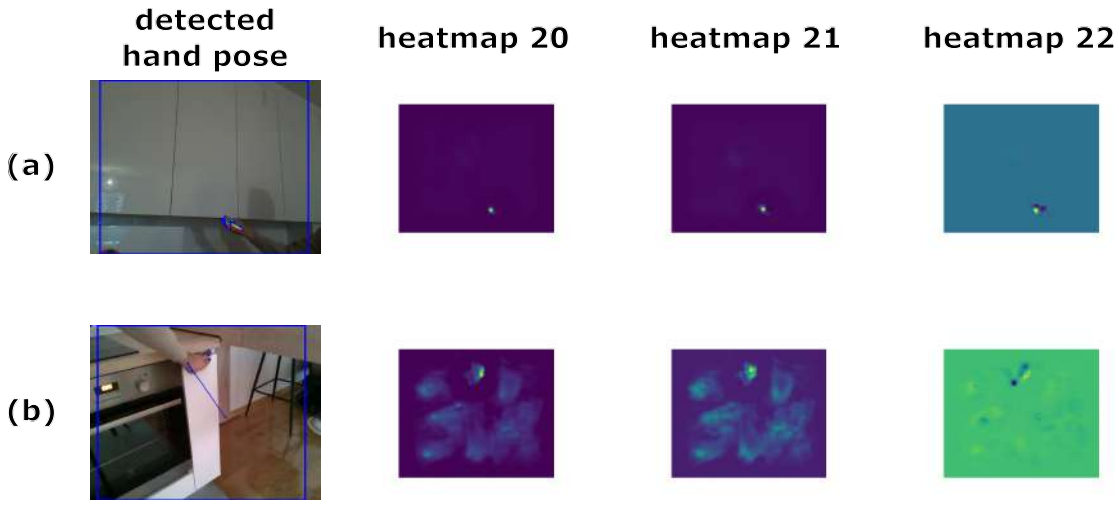

Figure 4.10: Two examples of results from hand pose detection using OpenPose. In (**a**) a well predicted hand pose is shown, while in (**b**) a poorly predicted hand pose is shown. When the predictions are good, the heatmaps 1-21 have well defined keypoints.

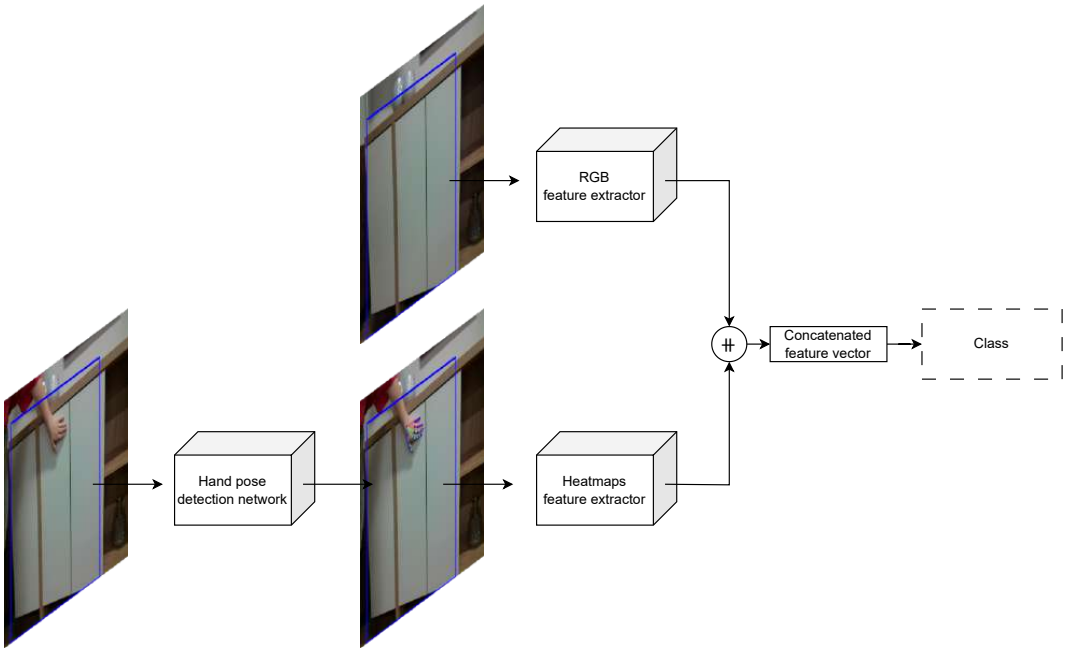

Figure 4.11: Scheme of the HoDoorNet method of classification. The top feature extractor accepts a regular RGB image without human demonstration as input, while the bottom feature extractor accepts the predicted heatmaps of hand keypoints as input. These features are then concatenated and ran through a fully connected layer to obtain the object class.

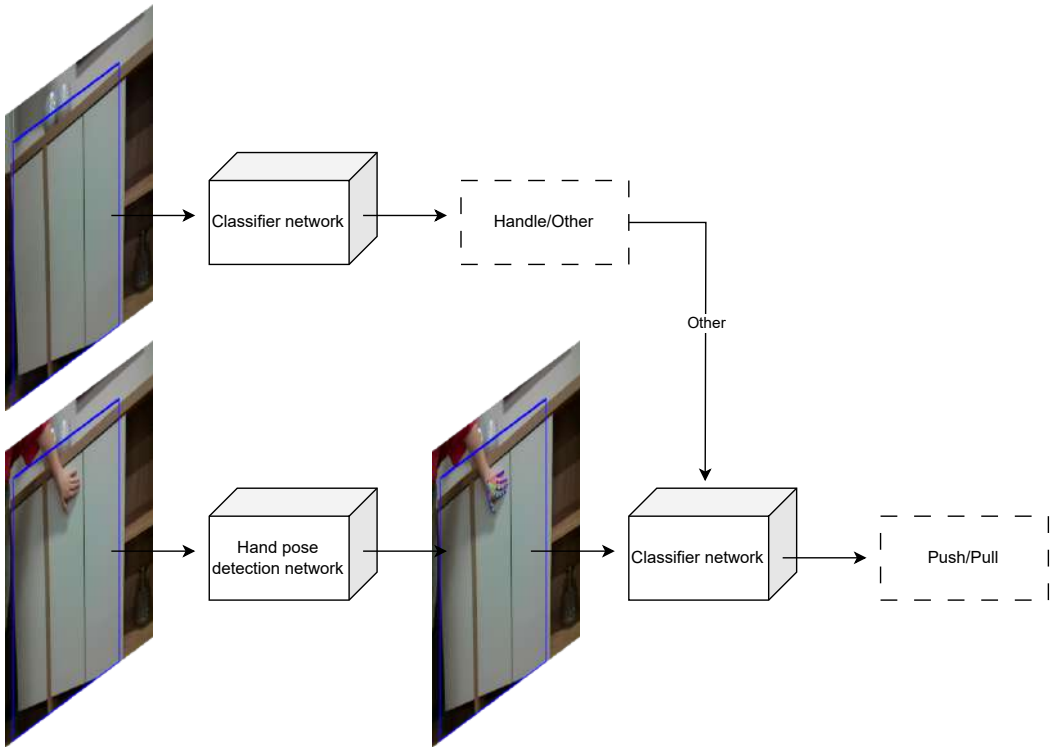

Figure 4.12: Scheme of the $method_{hierarchical}$. The top classifier extractor accepts an RGB image without human demonstration as input and outputs whether the RoI is of the handle class or not. If the region is determined not to be of the handle class, the bottom classifier uses the predicted heatmaps of hand keypoints as input and outputs whether the object is of push or pull class

4.3 Experiments

To test the effectiveness of the proposed methods, several experiments are conducted. For all the experiments, preprocessing described in Section 4.3.1 is used. Networks are trained, validated, and tested on partitions described in Section 4.1.2. Further experiments are conducted to test the consistency of the proposed methods which is described in Section 4.3.5. Finally, experiments with an object detector network are conducted. The results of these are shown in Section 4.3.6.

4.3.1 Preprocessing

As previously described, the dataset is arranged into scenes containing images of the same furniture. The dataset is split into training, validation, and test sets such that no images of the same furniture are put into different partitions, as described in Section 4.1.2.

Each region of interest, regardless if it is with human demonstration or without, is first optionally squarified as described in Section 4.2.1. In the case of images with human demonstration, these RoIs are then fed to OpenPose (Cao et al., 2019; Simon et al., 2017)[1].

[1]Instead of the code provided by the authors, the following pytorch implementation of the network is used: https://github.com/Hzzone/pytorch-openpose, accessed on 24 January 2023

The network outputs 22 heatmaps of size $92 \times N$, where N depends on the original aspect ratio of the input RoI. The heatmaps are then resized to the size 92×92. Exactly because of this step the ROIs are squarified, so as to prevent high levels of distortion in the heatmaps. Additionally, the heatmaps are augmented for training to reduce overfitting. The following transformations are used with the probability of each of 0.5:

- translation with maximum factor of 0.05,
- scaling with maximum factor of 0.15, and
- rotation with maximum angle of 25°.

The regions without human demonstration are preprocessed similarly. They are first rescaled to size 128×128 instead of 92×92. For training, the following transformations are used:

- translation with maximum factor of 0.05,
- scaling with maximum factor of 0.05,
- rotation with maximum angle of 15°,
- color shift with maximum value change of 15 for each color,
- brightness change with maximum factor of 0.2, and
- contrast change with maximum factor of 0.2.

The probability for each of these transformations is set to 0.5. Additionally, the input ROIs are normalized with mean and standard deviation of ImageNet (Deng et al., 2009) both for training and evaluation.

4.3.2 Experimental setup

For all four aforementioned methods, different combinations of squarification parameters and feature extractor networks are used. For squarification, dim_{min} is set to 50, and $dim_{increase}$ is from the set $0.0, 0.2, 0.5, 0.8, 1.0$. Additionally, using only the original RoI without squarification is also one of the setups [2]. This results in 6 total setups for RoI preprocessing. This value for dim_{min} is selected to ensure that the hand is visible and can be detected in the picture. The $dim_{increase}$ is there to give more context for hand pose detection. For feature extraction, 4 different ResNet (He et al., 2016) structures are used: ResNet18, ResNet34, ResNet50, and ResNet152.

Each network is trained for 100 epochs with batch size of 32. The networks are trained using the Adam optimizer with learning rate equal to 10^{-4} and L2 regularization with $\lambda = 10^{-4}$. They are trained on a single RTX 3080 GPU. The checkpoint with the highest F1 value on the validation set is chosen as the best one and used for evaluation on the test set.

For HoDoorNet, the feature extractors from the trained $method_{RGB}$ and $method_{heatmaps}$ are used. The last layer of each of the networks is removed and the final linear layer is added. The network is then trained for another 100 epochs as described above.

The binary networks which are part of the hierarchical classifier are trained separately. The first network uses the whole training dataset and classifies the regions into two categories, handle and no-handle. The second network uses only the subset of the dataset which contains regions of the push and pull categories.

[2] This is denoted as *none* in the following tables

4.3.3 Validation

All the networks are evaluated on the validation dataset to obtain the best models for further evaluation on the test dataset. The results of all 24 combinations of network architectures and RoI preprocessing setups for $method_{RGB}$ and $method_{heatmaps}$ on the validation set are shown in Tables 4.4 and 4.5, respectively. The tables show the F1 scores in percentage on the validation set. It can be seen that the $method_{RGB}$ works best when there is no RoI preprocessing, i.e. when the original region of interest is used as input, without squarification. On the other hand $method_{heatmaps}$ works better with higher values of $dim_{increase}$ and does not work very well when there is no RoI preprocessing. This is most likely due to the fact that the demonstrating hand is often not entirely inside the original RoI and more context is needed for the hand detection network to detect the hand well. It can be seen that the $method_{RGB}$ performs slightly better with the best model having F1 score of 84.98%, while the best $method_{heatmaps}$ model has 80.90% F1 score. $method_{RGB}$ results are also more consistent across the different combinations of network architecture and RoI preprocessing setups.

network \ $dim_{increase}$	**none**	**0.0**	**0.2**	**0.5**	**0.8**	**1.0**
resnet18	**83.91**	80.89	69.45	76.10	72.04	73.92
resnet34	**81.90**	77.37	63.36	63.52	69.07	72.46
resnet50	**84.98**	78.86	74.99	77.32	75.37	75.70
resnet152	80.77	79.95	75.65	75.70	76.60	69.51

Table 4.4: F1 score in percentage on the validation set with different combinations of network architecture and $dim_{increase}$ parameter using the $method_{RGB}$.

network \ $dim_{increase}$	**none**	**0.0**	**0.2**	**0.5**	**0.8**	**1.0**
resnet18	59.90	67.04	76.99	79.15	**80.00**	76.70
resnet34	59.01	68.79	77.67	**79.50**	79.11	**80.90**
resnet50	65.18	68.51	77.07	77.18	78.92	77.52
resnet152	57.25	71.74	76.97	76.28	77.93	77.41

Table 4.5: F1 score in percentage on the validation set with different combinations of network architecture and $dim_{increase}$ parameter using $method_{heatmaps}$.

The best 3 models of each method are shown in **bold**. The feature extractors of these models are chosen for HoDoorNet. All 9 combinations of these feature extractors are used for training HoDoorNet and the results on the validation set are shown in Table 4.6. The results are much better using this method, reaching 96.20% F1 score. This suggests that using the combination of the data without human demonstration and the data with human demonstration greatly improves the performance of a classifier.

RGB \ heatmaps	**resnet18 @ 0.8**	**resnet34 @ 0.5**	**resnet34 @ 1.0**
resnet18 @ none	92.65	89.46	92.65
resnet34 @ none	93.17	93.95	96.20
resnet50 @ none	96.10	91.31	94.26

Table 4.6: F1 score in percentage on the validation set with different combinations of network architectures and $dim_{increase}$ parameter for the feature extractors of HoDoorNet. The combinations are denoted as $network$ @ $dim_{increase}$. Different combinations for the feature extractor from images without human demonstration (RGB) are shown in rows while the feature extractors from images with human demonstration (heatmaps) are shown in columns.

Table 4.7 shows the results of the binary classifier differentiating between RoIs with handles and those without on the whole validation dataset. Most configurations have the perfect accuracy on the validation dataset, indicating that the network can easily recognize which RoIs contain handles and which don't. Table 4.8, on the other hand, shows the results of the binary classifier differentiating between the push and pull classes on a subset of the validation dataset. These networks are evaluated only on the RoIs of the push and pull classes. While these networks do not reach perfect scores, they still have really high accuracies, the best one having 96.88% accuracy.

network \ $dim_{increase}$	**none**	**0.0**	**0.2**	**0.5**	**0.8**	**1.0**
resnet18	**100.0**	**100.0**	100.0	**100.0**	97.88	93.12
resnet34	100.0	100.0	100.0	96.3	96.83	96.3
resnet50	97.88	100.0	100.0	100.0	100.0	100.0
resnet152	97.35	100.0	100.0	100.0	100.0	96.83

Table 4.7: Accuracy in percentage on the validation set with different combinations of network architecture and $dim_{increase}$ parameter using only the image without human demonstration. The images are categorizes into handle and no-handle categories.

network \ $dim_{increase}$	**none**	**0.0**	**0.2**	**0.5**	**0.8**	**1.0**
resnet18	80.47	82.03	93.75	93.75	92.97	**95.31**
resnet34	78.12	87.5	92.97	**96.09**	93.75	**96.88**
resnet50	78.12	85.16	93.75	93.75	91.41	93.75
resnet152	75.78	90.62	94.53	92.97	95.31	94.53

Table 4.8: Accuracy in percentage on the validation set with different combinations of network architecture and $dim_{increase}$ parameter using only the image with human demonstration. Only the images without handles are used and they are categorizes into push and pull categories.

The best 3 binary classifiers for both handle/no-handle and push/pull classification are chosen for the hierarchical classifier. Since there are many models which have 100% accuracy for handle/no-handle classification, the models with the fewest parameters are chosen, i.e. ResNet18. Since there are 4 ResNet18 models which have 100% accuracy, 3 are chosen arbitrarily. All 9 combinations of the hierarchical classifier are then evaluated

on the validation dataset and the results are shown in Table 4.9. Since all 3 models of the first classifier work perfectly on the validation dataset, the results only differ depending on the second classifier performance. The best of these classifiers reached F1 score of 97.76%, which is slightly better than the best HoDoorNet result on the validation dataset.

RGB \ heatmaps	**resnet18 @ 1.0**	**resnet34 @ 0.5**	**resnet34 @ 1.0**
resnet18 @ none	96.76	97.19	97.76
resnet18 @ 0.0	96.76	97.19	97.76
resnet18 @ 0.5	96.76	97.19	97.76

Table 4.9: F1 score in percentage on the validation set with different combinations of network architecture and $dim_{increase}$ parameter for the binary classifiers of the hierarchical model. The combinations are denoted as *network* @ $dim_{increase}$. Different combinations for the binary classifier on images without human demonstration (RGB) are shown in rows while the binary classifiers on images with human demonstration (heatmaps) are shown in columns.

4.3.4 Testing

The best three models of each $method_{RGB}$ and $method_{heatmaps}$ are chosen to be evaluated on the test dataset. Furthermore, all 9 aforementioned models of HoDoorNet and all 9 combinations of the hierarchical classifier are also evaluated on the test dataset. The results are shown in Table 4.10. This table shows the accuracy, average precision for all classes and per-class precision for each class, average recall for all classes and per-class recall for each class, and the F1 score.

Table 4.10: Results of various methods on the test dataset. The table shows the accuracy, average precision on the whole dataset followed by precision for each class, average recall on the whole dataset and for each class separately, and the F1 score. The configurations for *HoDoorNet* and $method_{hierarchical}$ are shown as follow <RGB feature extractor ResNet>-<heatmaps feature extractor ResNet> @ <RGB $dim_{increase}$> & <heatmaps $dim_{increase}$>. e.g. 18-34 @ none & 0.5 means that the feature extractor for the image without human demonstration is ResNet18 and there is no RoI preprocessing, while the feature extractor for the RoI with human demonstration is ResNet34 and RoI is squarified with $dim_{increase} = 0.5$.

network	**acc**	**prec**	push	pull	handle	**rec**	push	pull	handle	**F1**
$method_{RGB}$										
resnet18 @ none	60.24	48.61	12.5	60.00	73.33	50.91	6.67	67.50	78.57	49.74
resnet34 @ none	68.67	54.86	20.0	67.44	77.14	58.53	6.67	72.50	96.43	56.64
resnet50 @ none	63.86	51.19	12.5	65.22	75.86	53.41	6.67	75.00	78.57	52.28
$method_{heatmaps}$										
resnet18 @ 0.8	66.21	64.63	74.53	67.61	51.76	64.38	80.61	73.25	39.29	64.51
resnet34 @ 0.5	66.67	66.56	76.74	69.49	53.45	64.88	67.35	71.93	55.36	65.71
resnet34 @ 1.0	71.46	73.56	89.16	68.29	63.24	66.62	75.51	85.96	38.39	69.92

Continued on next page

Table 4.10: Results of various methods on the test dataset. The table shows the accuracy, average precision on the whole dataset followed by precision for each class, average recall on the whole dataset and for each class separately, and the F1 score. The configurations for *HoDoorNet* and $method_{hierarchical}$ are shown as follow <RGB feature extractor ResNet>-<heatmaps feature extractor ResNet> @ <RGB $dim_{increase}$> & <heatmaps $dim_{increase}$>. e.g. 18-34 @ none & 0.5 means that the feature extractor for the image without human demonstration is ResNet18 and there is no RoI preprocessing, while the feature extractor for the RoI with human demonstration is ResNet34 and RoI is squarified with $dim_{increase} = 0.5$. (Continued)

network	**acc**	**prec**	push	pull	handle	**rec**	push	pull	handle	**F1**
HoDoorNet										
18-18 @ none & 0.8	70.09	71.52	66.04	66.90	81.63	63.78	35.71	84.21	71.43	67.43
18-34 @ none & 0.5	73.52	74.98	70.37	71.64	82.93	68.43	58.16	86.40	60.71	71.55
18-34 @ none & 1.0	**84.93**	**84.97**	77.42	84.10	93.40	**83.34**	73.47	88.16	88.39	**84.15**
34-18 @ none & 0.8	72.83	75.10	86.76	75.70	62.82	72.92	60.20	71.05	87.50	73.99
34-34 @ none & 0.5	77.40	77.82	81.43	78.92	73.10	76.67	58.16	77.19	94.64	77.24
34-34 @ none & 1.0	70.32	71.01	79.52	71.43	62.10	69.49	67.35	72.37	68.75	70.24
50-18 @ none & 0.8	74.66	75.00	63.64	71.58	89.80	67.92	35.71	89.47	78.57	71.29
50-34 @ none & 0.5	69.86	73.43	77.55	66.44	76.29	63.31	38.78	85.09	66.07	67.99
50-34 @ none & 1.0	68.72	69.50	69.64	67.29	71.55	64.14	39.80	78.51	74.11	66.71
$method_{hierarchical}$										
18-18 @ none & 1.0	79.45	79.82	79.82	78.75	80.90	78.65	88.78	82.89	64.29	79.23
18-34 @ none & 0.5	76.94	77.67	76.34	75.78	80.90	73.94	72.45	85.09	64.29	75.76
18-34 @ none & 1.0	78.08	81.16	88.31	74.26	80.90	74.09	69.39	88.60	64.29	77.46
18-18 @ 0.0 & 1.0	76.03	77.48	84.47	88.20	59.77	81.30	88.78	62.28	92.86	79.35
18-34 @ 0.0 & 0.5	72.83	74.41	82.56	80.90	59.77	76.15	72.45	63.16	92.86	75.27
18-34 @ 0.0 & 1.0	74.20	77.68	93.15	80.10	59.77	76.45	69.39	67.11	92.86	77.06
18-18 @ 0.5 & 1.0	74.66	74.21	82.22	80.56	59.85	74.12	75.51	76.32	70.54	74.16
18-34 @ 0.5 & 0.5	71.92	70.98	75.31	77.78	59.85	69.84	62.24	76.75	70.54	70.41
18-34 @ 0.5 & 1.0	73.52	75.66	90.77	76.35	59.85	70.48	60.20	80.70	70.54	72.98

As was expected, the results of $method_{RGB}$ and $method_{heatmaps}$ are worse than those of the other two methods across different metrics. $method_{RGB}$ performs well on pull and handle classes but both recall and precision on the push class are really bad. This is due to the fact that the networks often confuse RoIs of push and pull class which look very similar without the help of human demonstrator. This can be seen in Figure 4.13 which shows the confusion matrix of the ResNet34 model. The network is good at predicting the handle class but often confuses the other two classes, rarely classifying anything as the push class. On the other hand, $method_{heatmaps}$ performs better on the push and pull classes, but not as well on the handle class. It still performs better on the handle class than the $method_{RGB}$ does on the push class. The confusion matrix of ResNet34 with $dim_{increase} = 1.0$ is shown in Figure 4.14. This method also overpredicts the pull class, but it also underpredicts the handle class, unlike the previous method. Exactly because

each method performs poorly on one of the classes and not on the same one, the other two methods, HoDoorNet and $method_{hierarchical}$, are needed.

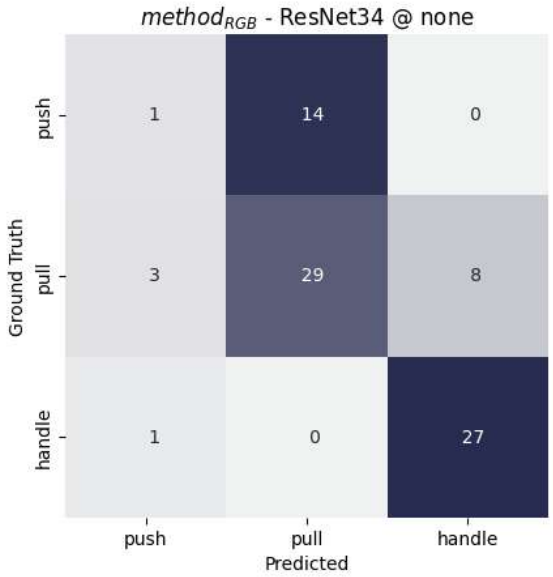

Figure 4.13: Confusion matrix on the test dataset of the $method_{RGB}$ ResNet34 model.

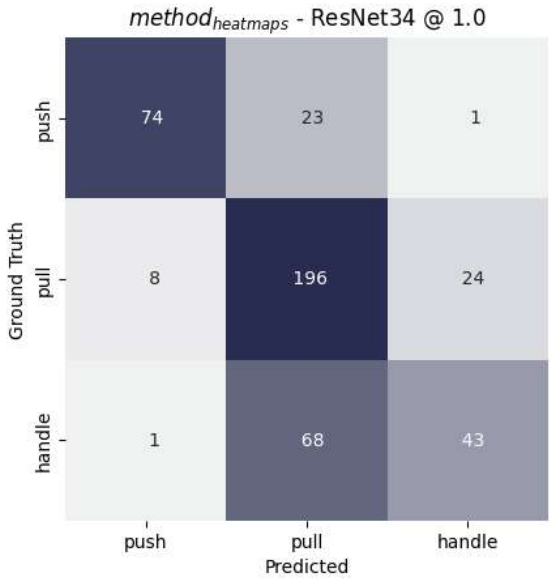

Figure 4.14: Confusion matrix on the test dataset of the $method_{heatmaps}$ ResNet34 model with $dim_{increase} = 1.0$.

With almost all configurations, HoDoorNet outperforms the previous two methods. The results are volatile and sometimes worse than those of the $method_{heatmaps}$. However, one of the configurations, where the feature extractor for the image without human demonstration is ResNet18 and there is no squarification, and the feature extractor for the image with human demonstration is ResNet34 with squarification with $dim_{increase} = 1.0$, obtains the best results of all methods on the test dataset with F1 score of 84.15%. The high volatility of the results suggests that there are some images in the

test dataset which are much different than what the network has learned on the training set. Furthermore, seeing how there is no such volatility on the validation set, it suggests that there is no such problem between the validation and training sets. Some overfitting is, of course, expected due to the small size of the dataset. The confusion matrix of the best model is shown in Figure 4.15. The model predicts the handle class very well and differentiates it from the other two classes, which is seen in the high precision and recall of that class. It is very good at predicting the other two classes as well, but it does sometimes confuse the two.

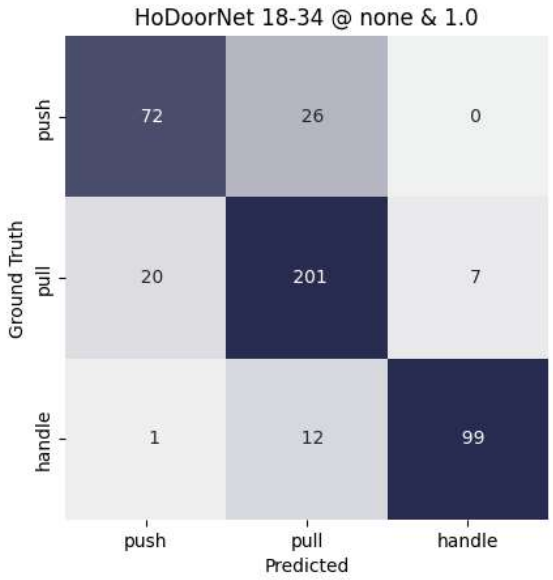

Figure 4.15: Confusion matrix on the test dataset of the HoDoorNet 18-34 @ none & 1.0 configuration.

While $method_{hierarchical}$ does perform more consistently than HoDoorNet when looking at the metrics for all three classes, there is some volatility when looking at the results on each class separately. For instance, precision for the handle class is sometimes around 60%, and sometimes around 80%. This is affected only by the first binary classifier, the one working on the images without human demonstration, and the results show that the ResNet18 without RoI preprocessing has the best precision on the handle class. However, this same classifier underpredicts the handle class causing the recall of that class to be low. The opposite is often true for the other handle/no-handle classifiers. It is harder to look at the results on the other two classes in isolation because those depend on both binary classifiers. The best performing configuration achieves F1 score of 79.35%, which is better than all the other methods except the best result of HoDoorNet. The confusion matrix of this model is shown in Figure 4.16. As can be seen from the confusion matrix and the precision of the handle class, this method overpredicts the aforementioned class. As is the case with all of these methods, there is little or no confusion between the handle and push classes. This is to be expected as these classes are very different both as just images of the objects and when looking at human's approach to opening these objects.

HoDoorNet	Expt. 1	Expt. 2	Expt. 3	Expt. 4	Avg. F1	F1 SD	no pt
18-18 @ none & 0.8	67.43	74.84	68.06	**78.30**	72.16	4.58	64.21
18-34 @ none & 0.5	**71.55**	67.04	67.53	71.24	69.34	2.07	54.83
18-34 @ none & 1.0	**84.15**	66.74	61.33	71.17	70.85	8.43	62.01
34-18 @ none & 0.8	**73.99**	62.37	62.34	69.56	67.06	4.96	73.17
34-34 @ none & 0.5	**77.24**	59.18	69.74	70.34	69.12	6.45	59.27
34-34 @ none & 1.0	70.24	69.78	66.96	**73.17**	70.04	2.20	73.33
50-18 @ none & 0.8	**71.29**	64.13	71.03	65.25	67.93	3.26	70.34
50-34 @ none & 0.5	67.99	**74.43**	55.20	67.96	66.40	6.98	59.97
50-34 @ none & 1.0	66.71	**74.01**	65.20	69.46	68.84	3.35	59.34

Table 4.11: F1 scores on the test set in percentage of multiple training experiments for the HoDoorNets. The third and second to last columns show the average F1 score and the standard deviation of F1 (*F1 SD*) in these experiments, respectively. The last column (*no pt*) shows the F1 scores for the experiment where the feature extractors were not pretrained.

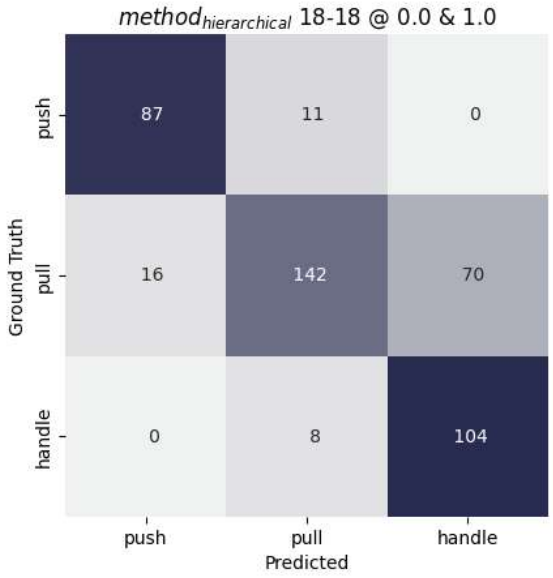

Figure 4.16: Confusion matrix on the test dataset of the $method_{hierarchical}$ 18-18 @ 0.0 & 1.0 configuration.

4.3.5 Testing variance

To check how consistent these methods are, further training experiments were done. The same HoDoorNet configurations as reported in subsection 4.3.4 were trained 4 more times, three times with separate pretraining of the feature extractors as previously described and once without this pretraining. The extractor for the RGB image without human demonstration is still pretrained on the ImageNet dataset. Additionally, the same $method_{hierarchical}$ configurations were trained 3 more times. The results of all of these are shown in Tables 4.11 and 4.12.

It is evident from these results that the $method_{hierarcical}$ is much more consistent than HoDoorNet, i.e., HoDoorNet's testing variance is higher. This is most likely due to the fact that $method_{hierarcical}$ splits the task of ternary classification into two simpler, binary classification tasks. Due to the size of the HoDoor dataset, a deeper network such as

$method_{hierarchical}$	Expt. 1	Expt. 2	Expt. 3	Expt. 4	Avg. F1	F1 SD
18-18 @ none & 1.0	**79.23**	75.19	76.59	71.96	75.74	2.62
18-34 @ none & 0.5	75.76	77.38	**79.24**	76.45	77.21	1.31
18-34 @ none & 1.0	**77.46**	74.11	75.19	73.38	75.04	1.54
18-18 @ 0.0 & 1.0	79.35	76.78	**79.81**	77.34	78.32	1.29
18-34 @ 0.0 & 0.5	75.27	79.10	**81.99**	80.81	79.29	2.54
18-34 @ 0.0 & 1.0	70.41	74.60	**79.48**	73.71	74.55	3.25
18-18 @ 0.5 & 1.0	74.16	72.59	**76.74**	69.82	73.33	2.51
18-34 @ 0.5 & 0.5	77.06	75.98	78.47	**79.07**	77.65	1.21
18-34 @ 0.5 & 1.0	72.98	71.87	**76.23**	71.86	73.24	1.79

Table 4.12: F1 scores on the test set in percentage of multiple training experiments for the $method_{hierarchical}$. The second to last column and the last column show the average F1 score and the standard deviation of F1 (*F1 SD*) in these experiments, respectively.

HoDoorNet cannot consistently perform well on test data which is different than the training data.

While the network performs well on training, the network is inconsistent on test data. The illustration shown in Figure 4.17 demonstrates how a model can fit to training data well in various different ways, but due to the small amount of training data, it can be somewhat random how well it will perform on test data. These results indicate overfitting and there are various techniques to fix this issue. As previously described, regularization and data augmentation were employed in these experiments to alleviate the issue. Another way to fix this issue would be to provide more training data. Results of the $method_{hierarchical}$ show that simplifying the task can alleviate the issue somewhat.

4.3.6 Classifying detected regions

An important usecase for the proposed methods would be classifying automatically detected objects. For this purpose, an object detection network, YOLOv7 (Wang, Bochkovskiy, and Liao, 2022), was used. This network takes an RGB image as input and outputs a set of object predictions. These predictions are in the form of bounding boxes, which can be used as RoIs for the classifiers. Since the HoDoor dataset is not large enough to train a robust object detector, another dataset needed to be used. Furthermore, annotations in the HoDoor dataset are not adequate for object detection task since there's only one annotated object per image.

The DoorDetect dataset (Arduengo, Torras, and Sentis, 2021) consists of annotated images taken from the Open Images Dataset containing doors and handles. The door class is split into three classes so there are four annotated classes in total: door, cabinet door, which also includes drawers, refrigerator door, and handle. The authors trained a YOLO v3 detector (Redmon and Farhadi, 2018) on this dataset and achieved mAP of 0.45, but it is unclear at what IoU this mAP was calculated at.

For this experiment, YOLO v7 was trained on the DoorDetect dataset with the hyperparameters the same as in (Wang, Bochkovskiy, and Liao, 2022), except batch size, which was 8. The network is trained on a single RTX 3080 GPU. The results on the DoorDetect validation set are shown in Table 4.13.

The detector is then used on the HoDoor dataset to detect the RoIs. Only the images without human demonstration are used for this experiment. Since the HoDoor dataset does not include annotations of all the cabinet doors and drawers in each scene, but only a single bounding box per scene, it is illogical to evaluate the detector based on mAP.

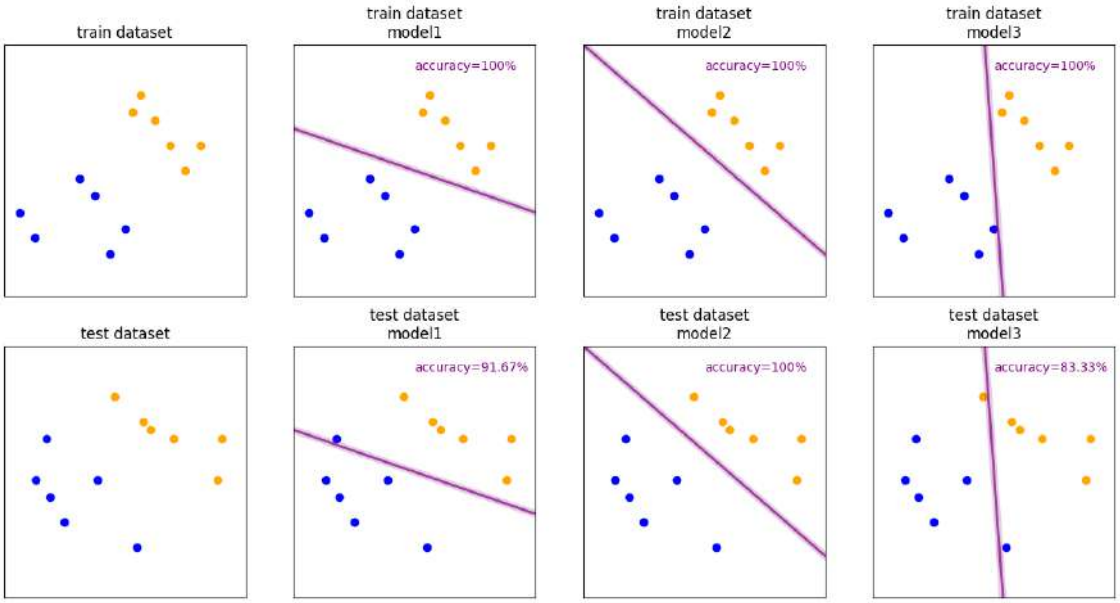

Figure 4.17: An illustration of different classifier models performing perfectly on the training set but sometimes not performing perfectly on the test set. The illustration represents a binary classification task with only two input dimensions.

Class	Labels	Precision	Recall	mAP@.5	mAP@.5:.95
door	73	0.755	0.548	0.624	0.379
handle	388	0.683	0.521	0.579	0.193
cabinet door	334	0.813	0.766	0.834	0.462
fridge door	106	0.840	0.604	0.683	0.472
all	901	0.772	0.609	0.680	0.376

Table 4.13: Results of object detection with YOLOv7 on the DoorDetect validation set.

Instead, only the recall of the detector is evaluated. Any bounding box classified as door, cabinet door or fridge with IoU with the ground truth annotation of 0.5 or higher is considered a true positive detection. It achieves recall of 93.64% on the whole HoDoor dataset and 89.16% on the test partition of the dataset, i.e., it correctly identifies 74 out of the 83 RoIs.

The classifiers presented in Section 4.3.4 are then used on the true positive detections. The results are shown in Table 4.14. It is apparent that the results are similar to those presented in Table 4.10 showing that the proposed methods perform well with detected RoIs as well. In some cases, the performance is slightly better on detected RoIs. This can be due to the fact that the classifiers are tested only on the positive detections. The best classifier achieved F1 score of 80.92% compared to the best classifier on GT RoIs which achieved F1 score of 84.15%.

Table 4.14: Results of the networks presented in Section 4.3.4 on the bounding boxes detected by YOLO v7 trained on the DoorDetect dataset. Notation is the same as in Table 4.10.

network	**acc**	**prec**	push	pull	handle	**rec**	push	pull	handle	**F1**
$method_{RGB}$										
resnet18 @ none	56.76	51.04	33.33	60.53	59.26	50.14	25.00	63.89	61.54	50.59
resnet34 @ none	70.27	64.80	50.00	71.05	73.33	61.54	25.00	75.00	84.62	63.12
resnet50 @ none	66.22	60.83	40.00	67.50	75.00	59.19	33.33	75.00	69.23	60.00
$method_{heatmaps}$										
resnet18 @ 0.8	65.73	62.99	74.23	67.70	47.06	65.47	92.31	73.91	30.19	64.21
resnet34 @ 0.5	62.92	61.90	70.27	68.25	47.17	61.13	66.67	69.57	47.17	61.51
resnet34 @ 1.0	71.36	73.51	90.14	68.32	62.07	67.50	82.05	86.47	33.96	70.37
HoDoorNet										
18-18 @ none & 0.8	75.19	74.24	78.48	79.41	64.81	74.60	79.49	78.26	66.04	74.42
18-34 @ none & 0.5	71.10	69.80	63.29	73.09	73.02	65.14	64.10	87.92	43.40	67.39
18-34 @ none & 1.0	76.98	75.55	65.18	90.26	71.20	81.57	93.59	67.15	83.96	78.44
34-18 @ none & 0.8	72.12	74.31	85.71	79.21	58.00	73.14	69.23	68.12	82.08	73.72
34-34 @ none & 0.5	73.40	75.03	85.94	77.20	61.94	73.60	70.51	71.98	78.30	74.31
34-34 @ none & 1.0	66.24	66.45	73.33	72.62	53.38	70.18	84.62	58.94	66.98	68.26
50-18 @ none & 0.8	80.56	81.72	86.36	79.37	79.41	78.33	73.08	85.51	76.42	79.99
50-34 @ none & 0.5	71.61	70.90	76.62	73.97	62.11	69.85	75.64	78.26	55.66	70.37
50-34 @ none & 1.0	72.63	72.37	75.95	74.16	66.99	72.30	76.92	74.88	65.09	72.33
$method_{hierarchical}$										
18-18 @ none & 1.0	74.94	73.91	71.15	78.06	72.53	77.02	94.87	73.91	62.26	75.43
18-34 @ none & 0.5	76.47	75.59	75.90	78.34	72.53	75.05	80.77	82.13	62.26	75.32
18-34 @ none & 1.0	77.49	79.70	91.04	75.54	72.53	75.16	78.21	85.02	62.26	77.37
18-18 @ 0.0 & 1.0	76.98	77.16	75.56	93.10	62.82	81.62	87.18	65.22	92.45	79.33
18-34 @ 0.0 & 0.5	77.49	78.26	82.86	89.09	62.82	79.28	74.36	71.01	92.45	78.76

Continued on next page

Table 4.14: Results of the networks presented in Section 4.3.4 on the bounding boxes detected by YOLO v7 trained on the DoorDetect dataset. Notation is the same as in Table 4.10. (Continued)

network	**acc**	**prec**	push	pull	handle	**rec**	push	pull	handle	**F1**
18-34 @ 0.0 & 1.0	79.03	81.60	93.55	88.44	62.82	80.24	74.36	73.91	92.45	**80.92**
18-18 @ 0.5 & 1.0	76.98	74.66	69.32	84.02	70.64	76.53	78.21	78.74	72.64	75.58
18-34 @ 0.5 & 0.5	78.01	75.47	71.62	84.13	70.64	75.04	67.95	84.54	72.64	75.25
18-34 @ 0.5 & 1.0	79.80	80.63	89.47	81.78	70.64	75.64	65.38	88.89	72.64	78.05

4.4 Conclusion

The problem with which this chapter deals is an important step in robotic manipulation of various types of articulated furniture. After localization of doors and drawers, the classification of their opening type is the next step toward manipulation. The methods presented in this chapter provide an insight into the feasibility of accomplishing this task using modern classifier networks. While the networks are sometimes inconsistent, the experiments show that even with the small amount of available data, the classifier network can perform well.

Furthermore, the experiments show that the addition of human demonstration in the classification process greatly improves the performance of classifiers. Without human demonstration, the classifiers have issues differentiating between push and pull classes which are usually difficult even for humans to differentiate on the first look. By using a combination of an image without human demonstration and an image with human demonstration, the classifiers are able differentiate well between all the three classes.

While this is a great step towards robotic manipulation of these kinds of furniture, there are still areas which can be improved. One of them being collection of additional data and, possibly, collection of data with different kinds of cameras and in different environments. One way of doing this would be annotating already available data. Another direction of further research could be fully automating the whole process, i.e., the localization of a RoI where a human demonstrates the opening type together with the proposed classification. Such a system could then be used in an environment where a robot could passively learn from humans without need for manual annotation by studying regular human interaction with furniture.

5 Conclusion

This thesis deals with two issues relating to use of service robots in household environments. The first issue is trying to explain the field of synthetic 3D data generation and how some factors of realism present in this process can affect the performance of an object detector trained on synthetic datasets. For this purpose, a modular method for synthetic data generation was developed. The method includes modules which have two modes of operation, one which produces more realistic scenes and one which produces less realistic scenes. The modularity of this method is essential in order to make it possible to study the effect some factors of realism can have on an object detector. The following five factors of realism in synthetic 3D data generation are considered: realistic camera noise, presence of background objects, sizes of objects, context of a scene, and positioning of objects.

It is very important that the baseline synthetic dataset is realistic for the ablation study described in this thesis. If the dataset is not realistic at all, it is not feasible to determine the importance of factors of realism. The experiments show that the baseline dataset created by this synthetic 3D data generation method can be used to improve performance of a 3D object detector (Rukhovich, Vorontsova, and Konushin, 2021). The detector pretrained on this dataset and finetuned on uncoloured real data outperforms the same detector trained on coloured real data only. This, together with the good results obtained when training only on the synthetic data, indicates that the generated synthetic dataset is at least somewhat realistic. This is further confirmed by visual inspection of the scenes the method generates.

The ablation study provides insight into the five aforementioned factors of realism. The experiments show that, depending on the network, the most important factor of realism can be the positioning of objects or the realistic sizes of objects. The CNN-based network, FCAF3D (Rukhovich, Vorontsova, and Konushin, 2021), is affected the most by realistic positioning, followed up by the presence of background objects. On the other hand, the PointNet-based network, VoteNet (Qi et al., 2019), is affected by the realistic object sizes substantially more than by any other factor of realism. This network is greatly affected by the presence of background objects as well.

These results indicate that realistic object size can be very important for synthetic scene generation, which is promising for future methods, as ensuring realistic object sizes is relatively easy to achieve. Furthermore, the presence of background objects is very important for both detectors. This indicates that including background objects is very important. The proposed method could be improved by including more background objects such as small objects on tables, more walls and so on. The results show that realistic camera noise is also very important, and further research into realistic camera noise could be beneficial to the field of synthetic 3D data generation.

The method could be further improved by including color in the generated point clouds. This would, however, require a 3D model dataset with textured and coloured models. Other than the proposed improvements to the baseline generation method, some further research options include studying the importance of other factors of realism, such as color, texture and reflectance properties of objects, and studying how these factors affect various different 3D object detectors.

The second issue tackled in this thesis is the classification of furniture opening type, specifically, the openable objects are split into the following categories: push, pull, and handle. Because the classes push and pull can hardly be distinguished by only looking at an image of an object, the proposed method also utilizes human demonstration for classification. Methods which utilize only images of the objects, only images of a human demonstrating how to open these objects, or a combination of these two are proposed in this thesis.

The experiments conducted using these methods show that using only images without demonstration leads to difficulty in distinguishing between push and pull classes as was hypothesized. However, using only images with human demonstration leads to confusion between pull and handle classes. Finally, the experiments show that using the combination of these two types of input leads to significantly better results on the test dataset. Furthermore, experiments on bounding boxes which were detected by an object detector (Wang, Bochkovskiy, and Liao, 2022) show that the methods perform well even when the regions of interest are not annotated perfectly. These results are very promising considering the small amount of available data for this task.

Further research in this field should include collecting more data. This includes collecting data in new environments, with different cameras and so on. Furthermore, the current method utilizes bounding boxes which were annotated by a human or which were detected and then chosen by humans. Excluding human intervention in this step would make the method fully automatic so research into this is also one of the possible directions of future research.

6 Curriculum Vitae

Matej Džijan was born in 1997 in Vinkovci, Croatia. He graduated from Ivan Goran Kovačić Elementary School and Matija Antun Reljković Gymnasium, both in Vinkovci. In 2018, he obtained a Bachelor's degree in Computer Engineering from the Faculty of Electrical Engineering, Computer Science, and Information Technology Osijek (FERIT) at the J. J. Strossmayer University of Osijek, Croatia. In 2020, he earned a Master's degree in Computer Engineering with a focus on Information and Data Science from the same faculty. That same year, he received the Dean's Award for his academic excellence. In 2020, he enrolled in the Postgraduate Doctoral Study in Electrical Engineering and Computer Science at FERIT. He is currently employed as a teaching and research assistant at FERIT, where he is involved in research on the application of artificial intelligence in robotics.

Matej Džijan
in Osijek, Croatia, August 16, 2024

Bibliography

Arduengo, Miguel, Carme Torras, and Luis Sentis (July 2021). "Robust and Adaptive Door Operation with a Mobile Robot". In: *Intelligent Service Robotics* 14.3, pp. 409–425. ISSN: 1861-2784. DOI: 10.1007/s11370-021-00366-7.

Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao (Apr. 2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. DOI: 10.48550/arXiv.2004.10934. arXiv: 2004.10934 [cs, eess].

Bousmalis, Konstantinos et al. (2017). "Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 95–104. DOI: 10.1109/CVPR.2017.18.

Caesar, Holger et al. (2020). "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11621–11631.

Cao, Zhe et al. (May 2019). *OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields*. DOI: 10.48550/arXiv.1812.08008. arXiv: 1812.08008 [cs].

Carion, Nicolas et al. (2020). "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer, pp. 213–229.

Chang, Angel X. et al. (2015). *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.

Cheng, Bowen et al. (2021). "Back-tracing representative points for voting-based 3d object detection in point clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8963–8972.

Community, Blender Online (2018). *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam. URL: http://www.blender.org.

Dai, Angela et al. (2017). "Scannet: Richly-annotated 3d reconstructions of indoor scenes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5828–5839.

Dalal, N. and B. Triggs (2005). "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

Demirdjian, David and Trevor Darrell (2001). "Motion estimation from disparity images". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 1. IEEE, pp. 213–218.

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Džijan, Matej et al. (2023). "Towards fully synthetic training of 3D indoor object detectors: Ablation study". In: *Expert systems with applications* 232, p. 120723.

Felzenszwalb, Pedro F et al. (2009). "Object detection with discriminatively trained part-based models". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9, pp. 1627–1645.

Geiger, Andreas, Philip Lenz, and Raquel Urtasun (2012). "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Geiger, Andreas et al. (2013). "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)*.

Girshick, Ross (2015). "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Girshick, Ross et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

Gschwandtner, Michael et al. (2011). "BlenSor: Blender sensor simulation toolbox". In: *International Symposium on Visual Computing*. Springer, pp. 199–208.

Gwak, JunYoung, Christopher Choy, and Silvio Savarese (2020). "Generative sparse detection networks for 3d single-shot object detection". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer, pp. 297–313.

Handa, Ankur et al. (2016). "Understanding RealWorld Indoor Scenes with Synthetic Data". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4077–4085. DOI: 10.1109/CVPR.2016.442.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hu, Yuan-Ting et al. (2019). "Sail-vos: Semantic amodal instance level video object segmentation-a synthetic dataset and baselines". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3105–3115.

Johnson-Roberson, Matthew et al. (2017). "Driving in the Matrix: Can virtual worlds replace human-generated annotations for real world tasks?" In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 746–753.

Karayev, S et al. (2011). "A category-level 3-D object dataset: putting the Kinect to work". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 1167–1174.

Khoshelham, Kourosh and Sander Oude Elberink (2012). "Accuracy and resolution of kinect depth data for indoor mapping applications". In: *sensors* 12.2, pp. 1437–1454.

Koonce, Brett and Brett Koonce (2021). "EfficientNet". In: *Convolutional neural networks with swift for Tensorflow: image recognition and dataset categorization*, pp. 109–123.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25.

Landau, Michael J., Benjamin Y. Choo, and Peter A. Beling (2016). "Simulating Kinect Infrared and Depth Images". In: *IEEE Transactions on Cybernetics* 46.12, pp. 3018–3031. DOI: 10.1109/TCYB.2015.2494877.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Leonardi, Rosario et al. (2022). "Egocentric Human-Object Interaction Detection Exploiting Synthetic Data". In: *International Conference on Image Analysis and Processing*. Springer, pp. 237–248.

Lin, Tsung-Yi et al. (2014). "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, pp. 740–755.

Liu, Shu et al. (2018). "Path aggregation network for instance segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768.

Liu, Wei et al. (2016). "SSD: Single Shot MultiBox Detector". In: vol. 9905, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325 [cs].

Liu, Ze et al. (2021). "Group-free 3d object detection via transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2949–2958.

McCormac, John et al. (2017). "SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?" In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2697–2706. DOI: 10.1109/ICCV.2017.292.

Misra, Ishan, Rohit Girdhar, and Armand Joulin (2021). "An end-to-end transformer model for 3d object detection". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2906–2917.

Planche, Benjamin et al. (2017). "DepthSynth: Real-Time Realistic Synthetic Data Generation from CAD Models for 2.5D Recognition". In: *2017 International Conference on 3D Vision (3DV)*, pp. 1–10. DOI: 10.1109/3DV.2017.00011.

Qi, Charles R et al. (2017a). "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660.

Qi, Charles R et al. (2018). "Frustum pointnets for 3d object detection from rgb-d data". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 918–927.

Qi, Charles R. et al. (2019). "Deep Hough Voting for 3D Object Detection in Point Clouds". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9276–9285. DOI: 10.1109/ICCV.2019.00937.

Qi, Charles Ruizhongtai et al. (2017b). "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

Redmon, Joseph and Ali Farhadi (Dec. 2016). *YOLO9000: Better, Faster, Stronger*. arXiv: 1612.08242 [cs].

— (Apr. 2018). *YOLOv3: An Incremental Improvement*. DOI: 10.48550/arXiv.1804.02767. arXiv: 1804.02767 [cs].

Redmon, Joseph et al. (June 2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, pp. 779–788. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.91.

Ren, Shaoqing et al. (Jan. 2016). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. DOI: 10.48550/arXiv.1506.01497. arXiv: 1506.01497 [cs].

Ros, German et al. (2016). "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243.

Rukhovich, Danila, Anna Vorontsova, and Anton Konushin (2021). "FCAF3D: Fully Convolutional Anchor-Free 3D Object Detection". In: *arXiv preprint arXiv:2112.00322*.

Saleh, Fatemeh Sadat et al. (2018). "Effective use of synthetic data for urban scene semantic segmentation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 84–100.

Savva, Manolis, Angel X. Chang, and Pat Hanrahan (2015). "Semantically-Enriched 3D Models for Common-sense Knowledge". In: *CVPR 2015 Workshop on Functionality, Physics, Intentionality and Causality*.

Shi, Shaoshuai et al. (2020). "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10529–10538.

Shi, Weijing and Raj Rajkumar (2020). "Point-gnn: Graph neural network for 3d object detection in a point cloud". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1711–1719.

Shrivastava, Ashish et al. (2017). "Learning from Simulated and Unsupervised Images through Adversarial Training". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2242–2251. DOI: 10.1109/CVPR.2017.241.

Silberman, Nathan et al. (2012). "Indoor segmentation and support inference from rgbd images". In: *European conference on computer vision*. Springer, pp. 746–760.

Simon, Tomas et al. (2017). "Hand Keypoint Detection in Single Images using Multiview Bootstrapping". In: *CVPR*.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Šimundić, Valentin et al. (2023). "Introduction to door opening type classification based on human demonstration". In: *Sensors* 23.6, p. 3093.

Song, Shuran, Samuel P. Lichtenberg, and Jianxiong Xiao (2015). "SUN RGB-D: A RGB-D scene understanding benchmark suite". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 567–576. DOI: 10.1109/CVPR.2015.7298655.

Stutz, David and Andreas Geiger (2020). "Learning 3d shape completion under weak supervision". In: *International Journal of Computer Vision* 128.5, pp. 1162–1181.

Sun, Pei et al. (2020). "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2446–2454.

Sweeney, Chris, Greg Izatt, and Russ Tedrake (2019). "A Supervised Approach to Predicting Noise in Depth Images". In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 796–802. DOI: 10.1109/ICRA.2019.8793820.

To, Thang et al. (2018). *NDDS: NVIDIA Deep Learning Dataset Synthesizer*. https://github.com/NVIDIA/Dataset_Synthesizer.

Tremblay, Jonathan, Thang To, and Stan Birchfield (2018). "Falling things: A synthetic dataset for 3d object detection and pose estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2038–2041.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Viola, Paul and Michael Jones (2001). "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. Ieee, pp. I–I.

Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao (July 2022). *YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors*. DOI: 10.48550/arXiv.2207.02696. arXiv: 2207.02696 [cs].

Wang, Chien-Yao, Hong-Yuan Mark Liao, and I-Hau Yeh (2022). "Designing network design strategies through gradient path analysis". In: *arXiv preprint arXiv:2211.04800*.

Wang, Chien-Yao et al. (2020). "CSPNet: A new backbone that can enhance learning capability of CNN". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391.

Xiang, Yi, Haoran Sun, and Wenting Tu (2023). "HDResNet: Hierarchical-Decomposition Residual Network for Hierarchical Time Series Forecasting". In: *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Xiao, Jianxiong, Andrew Owens, and Antonio Torralba (2013). "Sun3d: A database of big spaces reconstructed using sfm and object labels". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1625–1632.

Yan, Yan, Yuxing Mao, and Bo Li (2018). "Second: Sparsely embedded convolutional detection". In: *Sensors* 18.10, p. 3337.

Yang, Zetong et al. (2020). "3dssd: Point-based 3d single stage object detector". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11040–11048.

Zhang, Zaiwei et al. (2020). "H3dnet: 3d object detection using hybrid geometric primitives". In: *European Conference on Computer Vision*. Springer, pp. 311–329.

Zhou, Yin and Oncel Tuzel (2018). "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4490–4499.