Doctoral theses at NTNU, 2023:266

Jonatan Sjølund Dyrstad

Robot learning with visual processing in arbitrarily sized, high resolution volumes

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Jonatan Sjølund Dyrstad

Robot learning with visual processing in arbitrarily sized, high resolution volumes

Thesis for the Degree of Philosophiae Doctor

Trondheim, September 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

© Jonatan Sjølund Dyrstad

ISBN 978-82-326-7232-5 (printed ver.)
ISBN 978-82-326-7231-8 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

ITK-nr.: 2023-18-W

Doctoral theses at NTNU, 2023:266

Printed by NTNU Grafisk senter

Abstract

Flexible robots, capable of manipulating objects in unstructured environments under changing conditions, will lead to a paradigm shift in automation. Such robotic solutions can potentially transform entire industries currently subject to very little or no automation and they will eventually also have a profound impact on our daily lives.

Today's robotic solutions are not good at coping with large degrees of variability. These are highly specialized machines that typically operate in structured environments, which is designed to cater to the robot's strengths. The robots' inability to handle unstructured environments and cluttered scenery makes them unsuited for many real-world manipulation tasks. Many of these tasks are inherently cluttered and subject to large variations, such as tasks involving manipulation of raw materials or sorting of objects. Furthermore, automating these tasks requires developing highly specialized machinery that comes with costly and long development cycles. This is a limiting factor in today's application of robotics to automation.

To handle unstructured and cluttered domains robots need to sense and reason about their environment in order to select and execute an appropriate action, given the situation. This necessitates a visual processing system capable of extracting relevant information from the environment at high speeds.

The goal of this thesis has been to contribute towards the development of a visual processing system suitable for real-world robotic manipulation tasks. We approach this goal by first considering the task of open-loop grasping. To predict precise, collision-free grasps, the system needs to extract precise features in the face of noisy data. Further, robotic grasping is a highly relevant automation task across many industries and there is a need for a robust grasping system, which can handle noisy data and large variability in the appearance of objects.

The result of our work is a visual processing system that can be trained to process large point clouds by sequential focusing of attention. By learning to attend to the relevant parts of the volume, the proposed system can extract high-precision features from large volumes at high speeds.

In our work on grasping, we consider the task of bin-picking of fish, which is a difficult task, subject to clutter and noisy depth measurements. With the proposed visual processing system we achieve a 95 % grasp success rate on this task and the system is able to successfully correct its' own mistakes by trying again. Further, the system is trained solely on synthetically generated data sets and a generic pipeline for generation of such data sets for grasping has been developed. We envision that this approach might enable significantly shorter development cycles for new robotic applications and enable easy repurposing of robots for new tasks. In turn, we hope that this can open up for more automation in domains previously less suited for automation, such as producers dealing with smaller quanta or more varied materials or raw-materials or productions subject to seasonal variability.

The proposed system can process arbitrarily large volumes with a processing speed of 15 Hz. This indicates that the system can be used for closed-loop control, which can enable learning of more complex robot actions and sequences of actions. In this thesis we present the results of preliminary tests that were designed to test the system's capabilities in real-time applications. In these tests we considered two simple visual servoing tasks and trained the system with behavioural cloning. Through these two experiments, the system has proven capable of learning how to effectively summarize the contents of larger volumes through the attention mechanism and use this summary for subsequent decision making. When dealing with sequences of actions it is also able to infer the context based on the observations and shift its' focus of attention upon completion of a sub-task. However, these are only preliminary tests and the limits of the current system for tasks involving more complex relationships between objects and long-term memory are still unknown. Further, more research is needed in order to achieve robust robotic control-policies based on the extracted features, as the trained policies in this work suffer from the distribution shift-problem typical when training with behavioral cloning.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU), Trondheim. The work presented has been conducted at the Department of Engineering Cybernetics (ITK), NTNU and at SINTEF Ocean. The main supervisor has been Associate Professor Annette Stahl from the Department of Engineering Cybernetics. Senior Researcher John Reidar Mathiassen from SINTEF Ocean has been co-supervisor. The work was supported by the Research Council of Norway (grant no. 262900).

Acknowledgements

Many have contributed in various ways towards the completion of this work, both directly and indirectly, and I want to thank you all. In particular, thank you to my supervisors John Reidar Mathiassen and Annette Stahl for all discussions and for motivating me and giving me the push that I needed to finish this PhD. I've really enjoyed working with you on this exciting topic over these years.

Thank you to friends and colleagues at SINTEF Ocean, with a special thanks to Elling R. Øye, Aleksander Lillienskiold and Joakim Haugen, both for direct contributions to the work in this thesis and for the countless hours of highly diverging discussion.

To Ana's family, Guillermo, Margarita, Sebastian and Paulina, thank you for all the help, especially during these last weeks. This thesis would have been a lot shorter if it weren't for you.

To my parents Toril and Per and my brother Markus. Thank you for always supporting my projects, large or small, and for always genuinely believing in me even if there is no reason to.

Finally, to Theo. Thank you for being such a constant source of joy in my life and for distracting me from work. And to Ana. Be it in a sleep-deprived home-office

situation with a newborn baby, or in the final stages of a long PhD-period, you are always there for me. Without you, this thesis would never have been finished. Thank you for everything.

Jonatan Sjølund Dyrstad

Trondheim, June, 2023

Contents

Abbreviations

3D-CNN	3 Dimensional Convolutional Neural Network
BC	Behavioural Cloning
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
EA	Evolutionary Algorithm
FOV	Field of View
HDR	High Dynamic Range
IL	Imitation Learning
LSTM	Long Short-Term Memory
MDN	Mixture Density Network
MLP	Multi Layer Perceptron
ML	Machine Learning
MSE	Mean Squared Error
NLL	Negative log-likelihood
OOD	Out-Of-Distribution
PPO	Proximal Policy Optimization

RAM	Recurrent Attention Model
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
TSDF	Truncated Signed Distance Functions
VAE	Variational Autoencoder

Chapter 1

Introduction

This chapter aims to motivate the research presented in this thesis and to define the scope of the work. A brief overview of the contributions is given, followed by an outline of the remainder of the thesis.

1.1 Motivation

As the field of robotics continue to advance, the robots' increasing capabilities are making a significant impact across diverse industries. Worldwide, more than 3.5 million robots are tirelessly performing tasks like stacking items on pallets, welding, painting and assembling [1]. These robots are reliable, accurate and posses endurance vastly superior to their human counterparts. However, as of yet, robots are not cleaning our houses for us or helping out in the kitchen. In general, robots are rarely seen working next to humans, although robotic assistants could help many of us in our daily lives, assisting with heavy lifting and tedious tasks.

While many robots are much stronger than humans and better able to follow specified trajectories, they still fall short when tasked with mundane work humans find trivial. They struggle when faced with the unstructured and ever-changing real-world we humans handle so easily. As we humans act in these environments, we sense the world around us while continuously and effortlessly filtering relevant from irrelevant information. With our nuanced understanding of the intricate dynamics of the world and the intricate interactions at play, we are able to make quick judgements when faced with new situations. These are capabilities that are difficult to replicate in machines. Therefore, the robots of today are for the most part confined to domains where the environment can be tailored to better suit their strengths. Typical examples are the manufacturing and logistics industries.

If, however, robots were to acquire these human-like capabilities, the potential use-cases for robotics would explode. If this were the case, robots could work alongside humans, helping us with repetitive, hazardous, heavy or in general less desirable tasks, even *as they arise*, without the need for reprogramming. This is in stark contrast to the typical robotic solution found in the industry today. This is often a highly specialized machine, which is doing the same thing over and over again, only capable of handling ever so slight variations in its working conditions.

If robots are to make the transition from the controlled environments of e.g., car factories and warehouses to the unstructured environments our daily lives, they need to be able to act accordingly to different and constantly changing conditions. This entails sensing the environment, recognizing situations and, in light of the given task, selecting and executing the appropriate action. Acquiring this capability of acting in accordance with situations is a continuum, with different tasks requiring various amounts of high-level reasoning and long-term planning capabilities. Reducing the gap between robotic and human capabilities is a very large topic.

1.1.1 The scope of this work

Broadly, the work presented in this thesis aims at contributing to the realization of flexible robots capable of performing tasks in unstructured and dynamic real-world environments. We consider *visuomotor tasks*, which can be loosely defined as manipulation tasks requiring precision that can be performed by humans without involving complex reasoning or long-term planning. As steps towards this goal we define our research objectives.

Research objective 1: Visual processing for generic robotic applications

A good visual processing system for robotic applications should have the following attributes: 1) high-speed processing, 2) high-resolution input, 3) coverage of large work spaces. In practice, designing a visual processing system without trading off one of these attributes for the other two can be difficult. The result of processing the raw visual input should be a descriptive representation of the robot's workspace containing all information necessary to perform the task at hand.

The work in this thesis aims to contribute towards the design of such a visual processing system, well suited for generic robotic applications.

Research objective 2: Robust task-specific grasping

Pick-and-place operations remain a relevant task for automation, especially in domains subject to clutter and high variability. The most critical part of a pick-and-place operation is typically the picking part. In order to successfully pick an object, the robot needs to correctly parse the scene and effectively manipulate the object without damaging the object, neighbouring objects, the robot itself or the environment.

In this work we aim to contribute towards designing a system for generic picking applications. By generic, we do not mean a system capable of picking anything out of the box, but rather a system that can be deployed in a wide variety of applications and be *trained* to perform a specific task. This could for instance entail picking of certain objects in a specific way. The system should be robust to real-life noise and clutter and be compatible with domains subject to large variations in the appearance of objects. Further, the robot should be able to work in conditions tailored to human workers without need for modification of the environment. This would make for easier deployment and repurposing in line with research objective 3. As grasping is inherently linked to visual processing and parsing of scenes, the grasping domain also serves as a test-bed for the development of a visual processing system, the subject of research question 1.

Research objective 3: Repurposable robotic solutions

Development cycles for robotic solutions are costly. Typically, automated solutions are developed with a goal of automating one, highly specific task. In order to justify these costly development cycles, therefore, there has to be a need over time for a machine that is only capable of performing this single task. This effectively excludes a wide variety of industries subject to more varied tasks and producers dealing with smaller quanta or more varied materials or raw-materials or productions subject to seasonal variability. Flexible robots that, if necessary, can be easily repurposed for new tasks could significantly reduce the risk of investment in robotics and thereby enable more automation in these domains.

In this work we consider repurposing of robots between similar tasks, such as repurposing of a pick-and-place robot for a new pick-and-place task, handling a different raw-material. We contribute towards this research objective by considering two approaches to repurposing of robots: 1) Less costly development cycles with quick repurposing of robots by robotic experts in cooperation with domain experts. 2) In-situ repurposing of robots by domain experts without involvement of robotic experts.

1.2 Contributions at a glance

The long-term goal of the research presented in this work is to enable flexible robotic solutions, capable of performing tasks in unstructured and dynamic environments.

As a step towards this larger goal, we initially focus our research on the problem of robotic grasping. There is a need for robust grasping solutions in the industry and grasping can simultaneously serve as a useful test-bed for some of the components which needs to be in place for the larger, more capable system. Specifically, we used the grasping domain to explore feature extraction from visual input, which is tightly coupled with input representations and inference speeds.

To successfully grasp objects in cluttered environments, a robot needs a visual processing system capable of differentiating between graspable and non-graspable objects and parts of objects, which is robust to real-world, noisy data. As such, it needs to reason about the environment in relation to its own gripper, e.g., "where does the gripper fit", in order to avoid collisions. We view open-loop grasping of objects as a special case of reasoning about the space of possibility conditioned on a task and the geometry of the end-effector, which is fundamental to robotic manipulation in general.

Further, we show how the visual processing system initially used for grasping can be modified to enable more complex robotic control, i.e., closed-loop visual servoing.

The contributions of this work can be summarized as follows:

- **Grasp detection with sliding windows:** In papers A and C we demonstrate how 3D-Convolutional Neural Networks (CNNs) with small receptive fields can be used to predict precise, collision-free grasps for the challenging task of bin-picking of fish. This is a very cluttered domain subject to noisy data with large variations in the appearance objects and variations in the raw-materials. In paper B we demonstrate how 2D-CNNs can be used in a similar domain, bin picking of small, reflective steel parts.

- **Improved grasping by learning where to look:** In paper D we demonstrate how some of the issues related to small receptive fields and slow inference speeds in papers A and C can be mitigated by training a neural network to *attend* only to task relevant parts of a scene. The attending is done by sequentially deploying a small 3D-CNN at select positions in the volume, and the system produces precise grasps at high inference speeds. We also demonstrate that the attention mechanism functions as a filtering mechanism making the system robust to out-of-distribution (OOD) examples.

- **Extending to closed loop control:** In chapter 3.2 we demonstrate how the proposed grasping network of paper D can be used for closed loop control. We modify the network to predict velocities, rather than grasps, and train a model on two simple example tasks to motivate the use of the proposed visual processing system for closed-loop control.

We highlight the focus of our research in Fig. 1.1 on the following page. The figure is complementary to the bullet list above and shows the overall progress of the research through incremental improvements to the visual processing system with robotic grasping as test-bed.

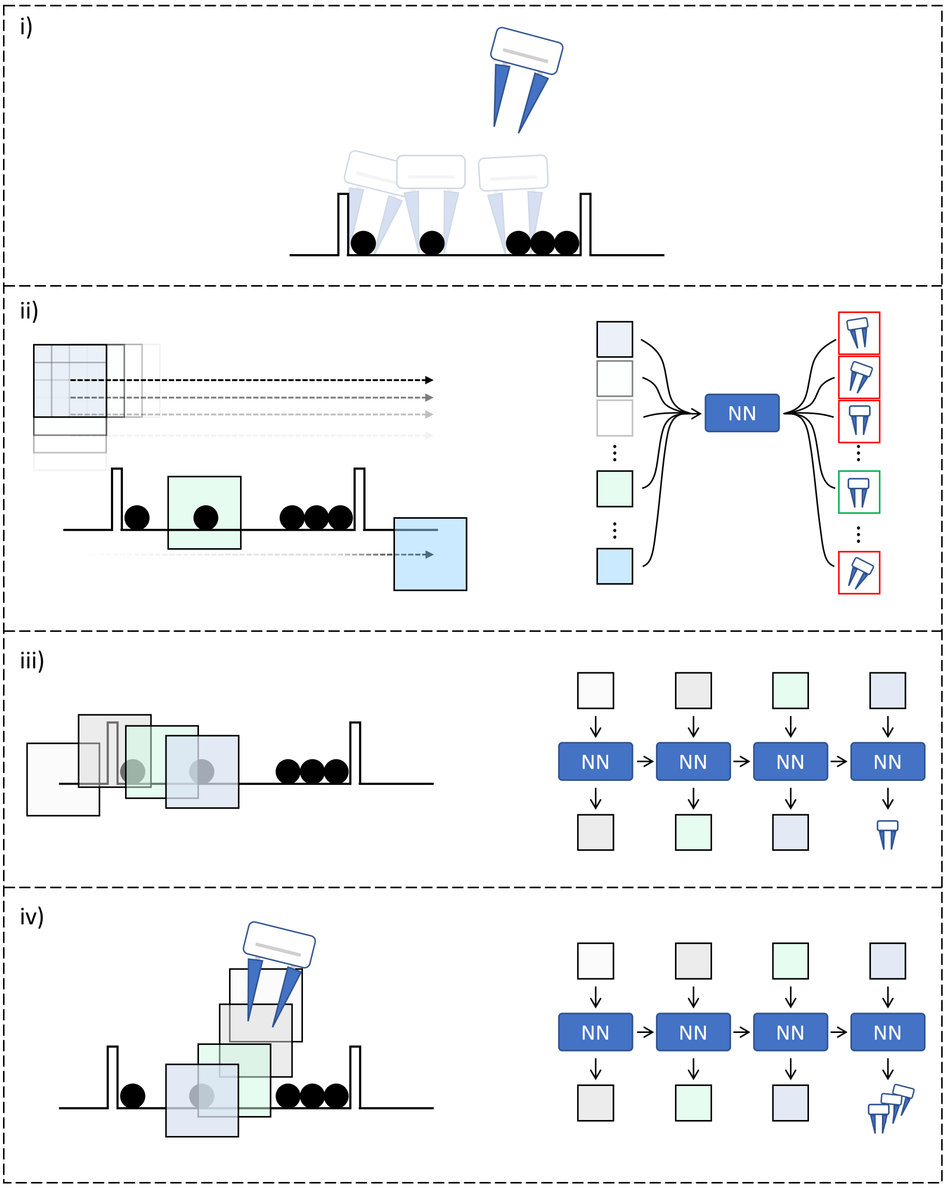

Figure 1.1: i) An example task, with a volume visualized in 2D as viewed from the side. The goal is to pick one of the graspable objects, with valid grasps visualized by transparent grippers. ii) Grasp prediction with sliding windows. A neural network processes all parts of the volume independently and looks for valid grasps at each location. The grasp, which the network is most certain of is attempted by the robot. iii) Grasp prediction with attention. The neural network decides where to look in the volume and predicts a single grasp based on what it has seen. iv) Closed-loop control with attention. The neural network decides where to look in the volume and predicts an end-effector velocity based on what it has seen, steering the robot towards the object in real-time.

1.3 Publications

Paper A

Jonatan S. Dyrstad and John Reidar Mathiassen. “Grasping virtual fish: A step towards robotic deep learning from demonstration in virtual reality”. In: 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO) (2017), pp. 1181–1187. [2]

Paper B

Jonatan S. Dyrstad et al. “Bin Picking of Reflective Steel Parts Using a Dual-Resolution Convolutional Neural Network Trained in a Simulated Environ- ment”. In: 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO) (2018), pp. 530–537. [3]

Paper C

Jonatan S. Dyrstad et al. “Teaching a Robot to Grasp Real Fish by Imitation Learning from a Human Supervisor in Virtual Reality”. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2018), pp. 7185–7192. [4]

Paper D

Jonatan S. Dyrstad, Elling R. Øye, Annette Stahl, John R. Mathiassen "Robotic grasping in arbitrarily sized volumes with recurrent attention models". Under review in: IEEE Transactions on Robotics (2023), [5]

1.4 Outline

The remainder of this thesis is structured as follows:

Chapter 2 outlines relevant background material from the literature and introduces concepts central to the work presented in this thesis. This includes topics from computer vision, robotics and learning for robotic control. Further, this chapter seeks to put the work of this thesis into context with the current state-of-the-art in the field.

Chapter 3 presents the contributions of this thesis. This chapter is split into two sections. In the first, Sec. 3.1, we introduce and expand upon our published research on grasping. In the second, Sec. 3.2, we discuss how to extend our approach to grasping for closed-loop robotic control. In this section, we present unpublished qualitative results of preliminary experiments.

Chapter 4 discusses the progress made towards the research questions, and possible future directions for the proposed methods in this thesis at a high level.

Chapter 5 concludes the work presented in this thesis.

Chapter 6 contains reprints of the papers on which this thesis is based.

Chapter 2

Background

This chapter seeks to outline relevant background material from the literature and to introduce concepts central to the work presented in this thesis.

2.1 Computer vision

Computer vision is an essential ingredient in today's robotic applications. By giving robots the ability to sense their surroundings, it empowers them to recognize different situations and respond appropriately to them. Computer vision is concerned with the task of acquiring visual information from the world and analyzing it to extract relevant information with regards to some specific application. Besides robotics, computer vision has numerous applications across many domains, ranging from augmented reality, e.g., putting dog ears on selfies, to medical imaging, e.g., assisting in diagnosis and analysis of X-rays and MRI-scans.

The year 2012, is by many regarded as the beginning of the *deep learning* (DL) revolution in computer vision. That year, the ImageNet Large Scale Visual Recognition Challenge [6] was won by a deep convolutional neural network (CNN) called AlexNet [7], which beat the competition by a large margin. This effectively showed that deep neural networks (DNNs), i.e., neural networks with many hidden layers, could learn end-to-end how to extract progressively higher level features from unstructured, high dimensional images when trained directly as a classifier. Prior to this revolution, traditional computer vision systems were often based on hand-crafted feature extractors, such as SIFT and HOG [8, 9], to reduce the dimensionality of the input space for a subsequent machine learning (ML) algorithm (e.g., SVM or kNN).

When trained on huge and varied data sets, the feature extractors learnt by CNNs have been shown to generalize well to new image analysis tasks. However, neural networks are notoriously data hungry, and when faced with a new problem without access to a large labeled training set, it can be difficult to train DNNs from scratch. In practice, therefore, it is common to do *transfer learning* either by *pre-training* the DNNs on a large available data set such as ImageNet [10] or by starting with a pre-trained model, available in many deep learning libraries [11, 12]. However, transfer learning works best when the data the model is pre-trained on is similar to the data the model is fine-tuned on. Therefore, when working with very different images from, say, ImageNet, or when working with other input formats, like 3D-data, transfer learning might not be possible. In such cases, unsupervised pre-training of the CNN feature extractors can be an option. There are several ways of going about this type of pre-training, and two broad categories are: 1) *Autoencoders* (AE), where a neural network is trained to compress and subsequently reconstruct the input from the compressed encoding [13]. 2) *Contrastive learning*, where a neural network is trained explicitly to encode "similar" inputs in similar parts of the encoded *latent space* (e.g. [14]).

2.2 Computer vision for robotics

Images can provide rich information about a robots surroundings, and in robotics therefore, computer vision is used to capture the *environment state*. As robots are expected to interact with their environment, computer vision for robotics is more tightly coupled with the physical world than pure computer vision systems. E.g., if a robot is supposed to manipulate an object, it is not sufficient to know where the object is in terms of pixels, but rather in terms of its position and orientation relative to the robot. Therefore, in addition to regular RGB-cameras, the use of 3D-sensors has become widespread. These 3D-sensors can capture geometric information by measuring the distance from the camera to points in the scene. Different technologies exists for 3D-data acquisition, some commonly used in robotics are: Structured light projection, time-of-flight (ToF), stereo vision (traditional feature matching based, and learning based) and light detection and ranging (LiDAR). Different sensors can have different properties, such as precision, max viewing distance, compatibility with reflective or textureless objects, compatibility with dynamic scenes and interference with other sensors and therefore, the choice of sensor will depend on the application.

2.2.1 Representation and processing of 3D data

Data acquired from different sensors can be represented in different formats. For color data, RGB images are typically used, and this format can be extended with 3D data by adding an extra channel containing the depth information, the result being an RGB-D image. One large advantage of these representations is that they can be processed using well proven 2D convolutional neural networks (2D-CNNs) developed for computer vision (e.g. [15, 16]). Additionally, both acquisition and processing of these formats can be done relatively fast, which is often a requirement in robotic applications. Further, there are several available data sets, which could potentially be used for pre-training [17]. However, for applications requiring high resolution inputs, the input space can become very high dimensional as the number of pixels increase. Further, if the work-space of the robot is large, it can be difficult to fit the entire scene within the frame of a single image, while maintaining the required resolution. Additionally, when working with images directly, the appearance of objects in the images will be inherently linked to the platform used for acquisition. For instance, the appearance of an object will be affected by the viewing angle of the camera relative to the object, the distance between the camera and the object, other objects in the line-of-sight of the camera and the object, and the intrinsic parameters of the camera. Therefore, trained computer vision models for robotics based on image data are largely platform-specific, and can not be transferred directly between different types of robots.

Point clouds offer a compact way of representing 3D data. A point cloud is simply a list of all the 3D measurements acquired from the scene

$$pcl = \begin{bmatrix} x_0 & y_0 & z_0 & r_0 & g_0 & b_0 \\ x_1 & y_1 & z_1 & r_1 & g_1 & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & r_N & g_N & b_N \end{bmatrix}, \tag{2.1}$$

where x_i, y_i, z_i describes the location of point i and r_i, g_i, b_i describes its color. As such, point clouds can be easily constructed based on data acquired from multiple sensors from different views, providing dense coverage of the work space of a robot in arbitrary resolution. By representing the acquired data in a fixed coordinate system, the state of the scene can be captured in a way that is less dependent on the location(s) and type(s) of sensor(s) used. However, because point clouds are large unordered sets, they cannot be processed using traditional CNNs. Point cloud processing with neural networks is an active area of research and has seen many breakthroughs in recent years [18, 19, 20, 21, 22], but the field has not yet converged to a "standard" architecture for feature extraction, such as CNNs have been for RGB-images.

CNNs can be used on 3D-data by discretizing space into *voxel grids* where each *voxel* can be viewed as a 3D-pixel [23, 24]. Typically, *occupancy grids* are constructed by assigning each voxel with a value of 1 if occupied and 0 otherwise. Alternatively, *truncated signed distance functions* (TSDF) can be constructed by assigning a real value to each voxel representing the distance to the closest occupied point in space. Like 2D-CNNs, 3D-CNNs are well proven and have been used successfully in many different applications such as 3D-object classification [23] and detection [24], analysis of medical data [25] and video analysis [26]. However, 3D-CNNs are computationally expensive with a memory consumption that grows cubically with resolution. This introduces a trade-off between the size and resolution of the input volume that can be processed by 3D-CNNs and in practice this has until recently made them unsuitable for real-time applications. However, in most real world applications, only a small part of the input space is occupied, meaning that most of the computation and memory is spent on processing of empty space. Recent works seeks to address this issue by exploiting the sparse nature of 3D data [27, 28, 29, 30]

2.2.2 The recurrent attention model

A drawback of CNNs is that the amount of compute needed to process an input scales linearly with the input's size. Further, the larger the input gets, the more data is typically needed in order to train the CNNs, to make sure they become sensitive to the things that matter in the input, and invariant to the rest.

In 2014, Mnih et al. proposed the Recurrent Attention Model (RAM) [31]. In their work, they propose processing of arbitrarily sized input images with a small CNN with a fixed input size. E.g., they process 60×60 pixel images with a CNN with an input receptive field of 12×12 pixels. They are able to to this by training the model to *focus its attention* on the important parts of the input and subsequently predict the correct label based on what it has seen. Specifically, the model processes the input sequentially, by repeatedly extracting 12×12 sized crops from the image and building up an internal representation of the input in the state of a recurrent neural network (RNN). The model is trained jointly to both learn an optimal processing strategy, i.e., where to look, and to predict the correct label for the input. This approach effectively decouples the computation needs from the size of the input. Further, they show that the model is able to ignore distractions and clutter, by focusing its attention on the important part of the input, outperforming CNNs tested on the same data sets. They emphasize the generality of their model by training it for multiple tasks, including a simple video game, and explicitly state that it can be applied to robotics. Additionally, a desirable property of the RAM is that it is easier to explain the behaviour of the model than typical CNNs, because one can visualize what parts of the input the model attends to, and thus what it bases its predictions on.

In 2016, Haque et al. extended the RAM to a significantly higher-dimensional domain, demonstrating that RAMs can be used to process point cloud videos for person identification, extending the attention mechanism to focusing in both 3D-locations in space and 4D-locations in space and time [32].

2.3 Learning for robotic control

Computer vision can be applied to robotics for both *open-loop* and *closed-loop* control. Robotic grasping is typically done in an open-loop fashion. In such cases, grasps are predicted from, e.g., RGB-D images and the robot performs the predicted grasps "blindly", i.e., without visual feedback. Deep learning based systems have been shown to be very effective in this domain, enabling high success rates when grasping novel objects in cluttered scenes [33].

Deep learning is also being applied to closed-loop robotic control from visual data. Unlike open-loop systems, these systems can potentially handle dynamic scenery and account for disturbances during execution by observing the robot while performing the action. In these cases, the neural networks take *observations* of the environment as input, e.g., RGB images, and are tasked with predicting appropriate *actions*. These actions can for instance be motor torques or target end-effector velocities or positions. In general, the goal of learning for robotic control is to learn a policy π_θ, i.e., a state-to-action-mapping, parameterized by the learnable parameters θ, typically, the weights of a neural network. In this work we follow the notation of [34], where $a_t \sim \pi_\theta(a_t|s_t)$, refers to the sampling of action, a_t, at time step t, from the stochastic policy π_θ, given the observed state s_t.

Two common approaches to learning of robotic policies are *imitation learning* (IL) and *reinforcement learning* (RL) [34]. IL is concerned with learning robotic behaviours by imitating an expert. In a typical setup, the expert demonstrates how to perform a task by teleoperating (i.e., remote controlling) the robot, while following his/hers expert policy π^*. The result is a data set of states (observations), and corresponding actions and the goal of IL is to learn a new policy π_θ which matches the expert's from the collected data set. One of the simplest approaches to IL is behavioural cloning (BC) [35]. In BC, the policy is trained with regular supervised learning by minimizing e.g., the negative log-likelihood (NLL) loss for discrete action spaces, or mean squared error (MSE) for continuous action spaces, between the expert's actions, $\pi^*(s_t)$, given state s_t and the learnt distribution over actions $\pi_\theta(s_t)$. In practice, however, BC often fails [36]. This is because the action the robot takes when in a state s_t at some point in time t, directly influences the state it will see next, s_{t+1}. Therefore, while running the learnt policy, π_θ, on the robot at test-time, it will encounter a set of observations conditioned on π_θ, rather then observations conditioned on the expert policy π^*. This leads to a *distribution shift* between the distribution over observations seen during training and the distribution over observations seen at test-time. During execution, the robot will initially find itself in a familiar part of state space, but as time progresses and the robot takes slightly different actions than the expert, given the same states, it might eventually find itself in parts of the state space not covered in the training data. When this happens, the policy will likely produce even less accurate actions, which leads to compounding errors that can grow quadratically over time [37]. This is a large problem with BC, and learning from offline data in general. In practice, in some applications the problem can be mitigated to some degree by gathering large amounts of data and by ensuring the training sets contains corrective actions, incentivizing the model to stay in the known parts of state space. This can be done by explicitly adding corrective actions for known critical states [38], by having humans provide corrective actions during demonstration by injecting noise into the experts policy [39] or by iteratively deploying the trained policy, having humans demonstrating corrective actions for encountered failure states [40].

Reinforcement learning considers the problem of learning of policies through trial and error [34]. In RL, an *agent* is trained without labelled training data, but rather with a *reward function*, providing rewards for desired behaviour and punishments (negative reward) for unwanted behaviour. In online RL, the training data is gathered by following the policy of the agent, and therefore, the distribution shift problem that arises in IL is not an issue in online RL.

2.4 Relation to state of the art

The work presented in this thesis revolves around two central topics: Learning for robotic grasping (3.1) and learning for closed-loop robotic control (3.2). This section aims to put the thesis into context with the current state-of-the-art wrt. these two topics[1].

2.4.1 The current state-of-the-art in robotic grasping

Early works on grasping were often based on object recognition and 3D pose estimation of known objects [41, 42, 43] followed by a grasp planning step or leveraged databases of CAD-models at test-time [44]. However, resent work typically favor deep learning based approaches.

Deep learning for robotic grasp prediction is often based on RGB-D images. Typically, a grasp is predicted in image space and is represented as an oriented rectangle [45]. The position of a grasp is represented by the position of a center pixel and optionally an offset from this pixel. The orientation is defined by the orientation of the rectangle and the image plane [46, 33, 47, 48, 49]. These approaches are well-proven and have demonstrated high accuracies and generalization to novel

[1]The following paragraphs are to some degree based the background section of paper D.

objects.

Other works predict grasps in 3D space. In these cases, the input is typically represented as occupancy grids, TSDFs, or point clouds. Despite slow inference speeds, 3D-CNNs can be used in a sliding window fashion on today's hardware, given a constrained volume [50]. However, multi-step methods are more common, which only process partial volumes with the 3D-CNNs e.g., after an initial segmentation step [51]. One of the earliest works employing 3D-CNNs for grasping was published in 2016 by Varley et al. [52] and they used a 3D-CNN for shape completion after a prior segmentation step for subsequent meshing and grasp planning with a simulator [53]. In [54], a set of *candidate grasps* is sampled uniformly from the point cloud, avoiding the need for voxelization of the entire work area by voxelizing only a subregion centered around each grasp candidate. Subsequently, the voxelized candidate grasps are projected along three axes before the grasp quality of the candidate is estimated by a 2D convolutional neural network. In [55], candidate grasps are sampled based on a set of heuristics, and the points within the closing area of the gripper for a given candidate grasp (i.e. the ones expected to come into contact with the gripper when closing) are transformed to local gripper coordinates and evaluated by a PointNet architecture [18], circumventing the need for voxelization, which inherently leads to resolution loss.

In [56], 6 degrees of freedom (DoF) grasps are predicted directly from a partial point cloud containing only the object which should be grasped. They train a variational autoencoder (VAE) with a PointNet++ [19] architecture so that they can sample 6 DoF grasps for a given object from latent space. They also employ an iterative refinement step to further improve the grasp quality.

A number of recent works have explored the use of attention in the context of robotic manipulation and grasping [57, 58, 59, 60, 61, 62]. In these works, different attention-based, systems are proposed that can learn to "zoom in" on the relevant parts of the scene with increasing resolution before predicting an action. In [62], James et al. propose a "coarse-to-fine" attention mechanism for 3D voxel grids. In their work, they first voxelize the entire work space for the robot in a coarse resolution voxel grid and task their model with selecting one of the input voxels as the focus of attention for the next step. On the subsequent step, a new, finer resolution voxel grid is created centered at the selected voxel and this process continues for a number of steps. On the final step the model predicts a 6D next-best pose for the end-effector, which is given as input to a low-level control agent.

2.4.2 Our work on grasping

While the focus of most related work is on training of grasping policies which generalize well to grasping of novel objects, the goal of our research has been to create a generic system for grasping of known objects. I.e., creating a system which can be deployed in a wide variety of factories and be trained for grasping of the specific types of objects which are expected to be seen.

Domain differences

Most related work considers rather controlled environments with objects lying singulated or in a pile on a table or in shallow bins with sloped walls, e.g., [44, 16, 63, 52, 51, 54, 55]. In such cases, collision free grasps can often be found by predicting top down-grasps and by e.g., picking objects along the edge of a pile. In contrast, our focus has been on making the system compatible with difficult real-life scenarios. This entails being robust to real-life clutter, occlusion, sensor noise and obstacles while handling both rigid and deformable objects subject to intra-class variations. As the main test-bed for our approach therefore, we chose the problem of picking of fish lying in fish crates, which is a challenging domain for a few reasons. Visually, a box of fish is densely cluttered. Being deformable, fish can be forced into a wide variety of poses and configurations with fish overlapping and occluding each other. Further, the semi-translucent plastic fish crates coupled with water-films, blood and the reflective surface of the fish themselves with both very dark and very bright textures leads to noisy depth data. Typically, depth images of fish in fish crates have both large patches of missing measurements and the acquired measurements can be very inaccurate. Additionally, grasping of fish is a very unforgiving domain wrt. grasp placement. Even when grasping singulated fish on a table, grasps have to be placed precisely, close to the fish's center of mass and with the fingers almost touching the table. Because of the slippery nature of the fish, the fish will slide up into the gripper upon closing if a grasp is placed correctly, but if misplaced, the fish will slide out of the gripper when lifted. This expected movement of the graspable object upon closing of the gripper is atypical wrt. related work on open-loop grasping. When lying in a bin, grasps have to be placed even more meticulously. Often, a grasp can only succeed if the fingers of the gripper are slid in-between other fish. If a fish is lying in a sharp corner, the grasp must be placed such that the gripper slightly touches the bottom and the side of the box at the same time.

Technical differences

In our papers A and C we propose the use of sliding window 3D-CNNs for grasping in the fish-picking domain described above. We propose end-to-end supervised training of mappings from voxel grids directly to 6 DoF grasps and to the best of our knowledge, our paper was among the first to propose this in the literature. Around the same time (2016) and towards the same end, Varley et al. [52] also employed a 3D-CNN, however, it was used for shape completion for subsequent meshing and grasp planning.

In our paper B, we consider a different domain subject to similar noisy depth data: Grasping of small reflective machined parts ($19mm$ to $30mm$). In this work, we propose an approach for generation of realistic looking simulated depth data for reflective objects and train a multi-resolution 2D-CNN for sliding window grasp detection in this domain.

In our work in paper D, we employ an attention mechanism, based on the RAM of Mnih et al. [31], with a network structure very similar to Haque et al. [32], in order to find viable grasps for large, high-resolution volumes in real-time. We test the system in the challenging fish-picking domain and achieve a grasp success rate of 95%, beating our previous approach (paper C) with 15 percentage points. Our approach with attention in voxel grids for grasping is similar to the system of James et al. [62] from 2022, and was developed independently from and approximately simultaneously as theirs.

2.4.3 Learning for robotic control

Most related works on grasping are not intended for, and do not extend naturally to, closed-loop control, which is a key motivating factor for the design choices made in our work. In our work in paper D we do not aim to find all valid grasps for a scene, but rather a single "action" in correspondence with the "state-to-action" mapping needed for closed-loop control. While the model of paper D was only tested in the fish-grasping application, the paper hypothesized about the possibility of altering the proposed system for real-time robotic control, and this is further expanded upon in Sec. 3.2 of this thesis. We therefore include some selected background works on learning for robotic control to highlight how our work ties in with the current state-of-the-art in this field.

In the visual servoing literature, the state on which the learnt policy π_θ bases its actions is often represented as RGB images. Recently, impressive results have been demonstrated on a variety of tasks [64, 65, 66, 67], such as zero-shot task generalization for complex sequences of actions [66]. These works addresses some of the difficult open research questions in imitation learning, such as the distributional shift (described in Sec. 2.3), dealing with long time-horizons and generalization to unseen tasks.

As these methods are based on imitation and/or reinforcement learning, they suffer from the *curse of dimensionality*. Essentially, in RL the problem is that as the state space grows, the number of data points needed for learning grows exponentially. In practice therefore, exploring for optimal control in high-dimensional state spaces can become infeasible. To combat this it is common to try to keep the dimensionality of the state-space for the RL-algorithms small. This can be done by, e.g., doing principal component analysis (PCA) on the state space [68], or more typically, by ensuring that the RL-algorithms are provided with small, descriptive states through transfer learning or unsupervised pre-training. Alternatively, the algorithms can be trained on the raw inputs while simultaneously being trained on auxiliary tasks such as open-loop predictions and/or input-reconstruction tasks (VAEs), which incentivizes the policy to encode the raw image features meaningfully. Further, most works reduce the state space by working with quite coarse input images, e.g. 80×64, 128×128, 150×150, 160×120 [64, 65, 66, 67]. Inherently, all of these methods for state-space reduction introduces a trade-off between the information the state contains, and the dimensionality of the state-space.

In this context, we propose the use of RAMs as a way of extracting features from large point clouds, which can provide small descriptive state spaces for learning of robotic policies. We argue that RAMs are well suited for this task as the attention mechanism can learn to efficiently mask out distractions while maintaining high resolution information about task-relevant parts of the volume, as shown by [31, 32]. Additionally, the RAM has the desired property of decoupling inference speeds from the size of the input, which is crucial when working with large point clouds subject to real-time constraints.

In Sec. 3.2, we demonstrate that the proposed system of paper D can be modified for real-time robotic control on simple robotic tasks. To the best of our knowledge, this thesis represents one of few examples in the literature of employing RAMs for robotic control, with real-time manipulation of objects directly from large point clouds containing millions of points. The work we have found most similar to ours is the work of James et al. [62]

Chapter 3

Contributions

This chapter presents the contributions of this thesis and it is split into two sections. In the first, Sec. 3.1, we introduce and expand upon our published research on grasping. In the second, Sec. 3.2, we discuss how to extend our approach to grasping for closed-loop robotic control. In this section, we present unpublished qualitative results of preliminary experiments.

3.1 Grasping

The aim of our work on grasping has been to find a combination of input representation and feature extraction pipeline that can be used for robotic applications in general. Towards this goal we try to solve a real world automation task, which is currently only done by humans in today's industry.

We chose picking of fish as the main test-case for our approach to grasping. Working with fish is challenging because it is difficult to get good depth measurements of fish using typical 3D-sensors. The specularity of the wet fish skin combined with the dark backs and light bellies of the fish leads to under- and over-exposure, which in turn leads to missing depth data in addition to the overall quite noisy measurements. Because fish are soft-body objects, the position and orientation of the fish body cannot be inferred by simply observing, e.g., the head of the fish. Therefore, precise feature extraction is the only way to know the full state of the fish. In sum, grasping of fish serves as a good baseline for our work and we consider success in this domain as a proof-of-concept of the proposed system.

In all our works on grasping we train neural networks for the task of *grasp prediction*. With neural networks being notoriously data hungry, and to avoid spending the bulk of our time gathering and labelling data, we rely on simulation and synthetic data generation to create the data sets needed for training. Because it is easier to simulate realistic looking depth images than color images, we limit ourselves to working solely with depth data in all our experiments. We do however believe that color data contains valuable information that the neural networks could learn to utilize in order to predict even more precise grasps. It is, for instance, easy to distinguish the head-end from the tail-end of a fish with color information, while it can be quite difficult to do so in a colorless point cloud subject to noise. This can in turn affect grasp performance, as knowledge about the position of the head and tail of the fish could provide information about the fish' center of mass. However, the convenience of being able to generate a lot of training data, especially in the early phase of the research period when the data needs of our application was unclear, led us to focus on depth data alone. Intuitively, this also makes the amount of data needed for training smaller, as the neural networks only have to be sensitive or invariant to different geometric features and noise, not, e.g., lighting conditions, different shadows, inter-/intra-class color variations and so on.

3.1.1 Grasping with sliding windows

This section summarizes our work on grasp detection with *sliding windows*. In this context, sliding windows refer to the act of "sliding" a detector, i.e., neural network, over an image or a volume, processing each location in the input independently. For instance, sliding a 2D-CNN over a 32×32 image will yield an output with $32 \times 32 = 1024$ detection-results[1].

Grasping with sliding windows was done in three of our papers. In papers A and C we train 3D-CNNs and in paper B we train a 2D-CNN for the task of grasp detection in a cluttered and noisy environment. The 3D-CNNs are trained and tested on the fish-picking task, while the 2D-CNN is tested on a similar task, bin-picking of reflective steel cylinders. In both cases the main challenge is low quality depth data, see Fig 3.1.

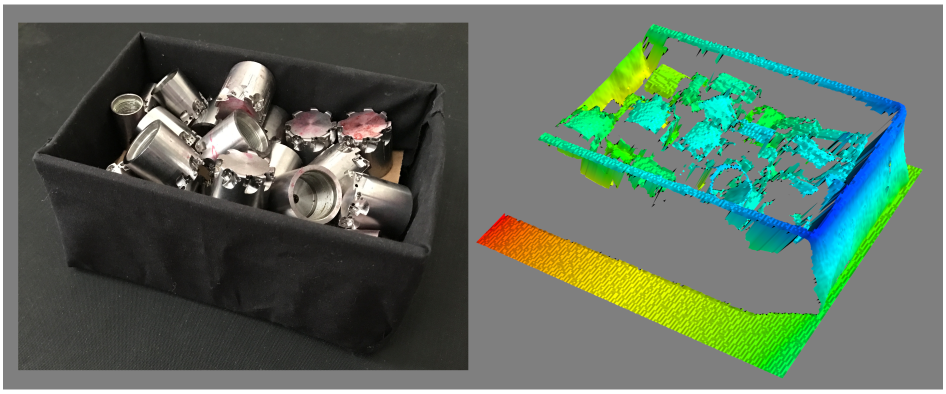

Figure 3.1: Left: The box of reflective cylinders used when testing in paper B. **Right:** An example depth image from the test set, visualized as a surface plot.

Data acquisition and pre-processing

The data for all real world experiments presented in this section was acquired with the use of depth sensors based on structured light projection. In paper C, a realsense SR300 was used, while a Zivid camera was used in paper B. In paper A, training and testing was done entirely in simulation.

By assuming a stationary scene, several images with different exposures can be combined (HDR), and temporal filtering can be done on the results of multiple images to get more accurate depth data. These techniques were used in paper B. However, because we would like our system to be extendable to dynamic scenes, we try to tackle the problem with noisy data mainly by the use of multiple view-

[1] Assuming a stride of 1 and that the input is padded.

points. Specular hightlights on reflective objects are dependent on the viewing angle, i.e., the position of the camera relative to the objects, and more depth data can therefore be acquired from the scene by taking multiple images from different locations at the same time. In paper C we take 4 images from 4 different positions by attaching the camera to the robotic arm and moving it in a known pattern, combining the depth data acquired from all positions into a single point cloud. This approach can be used in dynamic scenes by swapping the camera on the robot for a calibrated camera rig trigging several cameras simultaneously (see 3.1.2). A drawback of this approach is that the noise, in the form of wrong depth measurements, from all depth acquisitions are accumulated in the resulting point cloud. This noise often manifests itself as points in mid-air, or "thick" surfaces in the point cloud. However, with sufficient overlap between the different depth images, we find that this can be combated to some degree by voxelizing the volume and setting a suitable threshold for what counts as an occupied voxel.

In addition to combating missing data because of specular highlights and reflections, the multi-view solution helps mitigate issues regarding occlusion. Objects which are occluded from one viewing angle might be visible from another and in a bin-picking setting it can be difficult to, e.g., get good depth measurements of the insides of all 4 sides of the box from one viewing angle.

Generating training data in simulation

To generate the data needed for training we implemented a simulator based on the Unity Engine [69]. In the simulator the physics of the objects were simulated while the objects were poured into the bin, and depth images were rendered of the scenes when the objects had come to rest. Grasps were not labelled per scene, but rather, a set of candidate grasps were labelled beforehand, in relation to the 3D models of fish and cylinders. When the objects had come to rest and the images were rendered, the candidate grasps for each object were evaluated, and the ones which didn't collide with the environment were kept as valid grasps for the current scene. Examples of non-valid grasps were also needed to train the classifier/grasp-detection heads of the neural networks and these were simply sampled in the volumes in locations not containing a valid grasp.

Discussion and Results

The neural networks trained in papers A, B and C were all trained as convolutional sliding window grasp detectors. Meaning that a small receptive field was slid over an input image (in the 2D case) or voxelized volume (in the 3D cases) and at each location the neural network predicted the probability of the input patch containing a valid grasp. Additionally, the networks output an offset from the center pixel/voxel and the orientation of the grasps, and together this constitutes the full predicted grasp pose. The 2D-CNN of paper B predict the 5 DoF pose of a suction gripper, while the 3D-CNNs of papers A and C predict the 6 DoF pose of a jaw gripper. For more details on the specific architectures we refer to the original work.

Importantly, in the sliding window approach to grasping, each location in the input is processed independently from the others. The result is a list of grasps, one grasp per input patch, with corresponding estimated probabilities of that patch actually containing a valid grasp. In order to select the best grasp for a scene, one simply selects the grasp with the highest predicted probability in this list. This means that the systems cannot reason about the bigger picture and perform tasks like, e.g., "pick the biggest fish in the pile". However, working with small receptive fields independently can be advantageous in some cases, because the state space the neural network needs to consider is reduced. When sliding a 3D receptive field over a volume, only the geometry within some 3D cube is processed by the neural network. The smaller this cube is, the less variability is expected, which in turn should make it easier for a neural network to learn a reasonable mapping from input to grasp pose. Similarly, when sliding a small 2D receptive field over a depth image, the state space is reduced by reducing the size of the receptive field, but only in the image plane (x- and y-dimensions). The depth information in the z-dimension (the direction the camera is pointing) is not affected by changes in the size of the receptive field. Therefore, 2D-CNNs can be affected by irrelevant foreground and background objects which CNNs, in general, are known to be sensitive to [70].

We posit that all the information needed to predict a collision free grasp for an object is available in a confined region around the grasp point, where the size of this region is given by the size of the gripper used.

We find that both 2D-CNNs and 3D-CNNs can be used to predict grasps with relatively small receptive fields in our considered domains. However, both of the networks make mistakes when tested on the real robot. Out of the two approaches, we find the 3D-CNN of paper C's errors easier to explain. In general, the 3D-CNN is very good at avoiding collisions with the observable part of the environment. It also recognizes the orientation of the fish well, and predicts precise grasp orientations and off-sets, placing the grasps precisely on the visible part of the fish. However, for the fish picking case, the receptive field is not large enough to determine the center of mass of the fish accurately. The neural network ends up processing the different parts of the body of the fish independently and because a large part of the fish has an almost cylindrical shape, the neural network is unable to distinguish a part which is too close to the head- or tail-end from a part suitable for grasping. Therefore the fish is often grasped too close to the head or tail, which leads to the fish sliding out of the gripper.

Interacting with the environment by picking fish one by one inevitably introduces a *distribution shift* in the test time data vs. the training data in the sense that the configuration of fish in the box after a few picks will differ from the distribution of fish in the training set. This happens because the state of the scene at test time becomes conditioned on the policy of the grasping network, while the simulated data is simply generated with different numbers of fish in the box. However, because the 3D-CNN processes small receptive fields independently, it is agnostic to the global state of the scene, i.e., the environment state. And because the space of observable states seems to be sufficiently covered in the training data, the neural network works well, also in scenarios which are unlikely to have occurred in the training data (see Fig. 3.2)

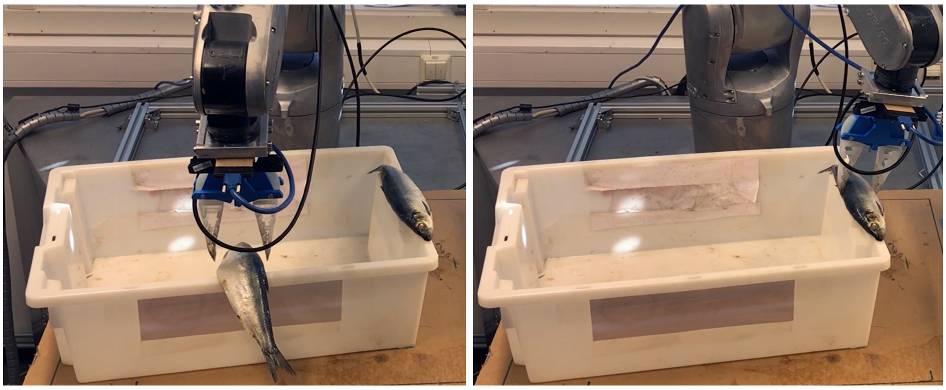

Figure 3.2: An unlikely case of an empty box with two fish lying on the sides of the box. The neural network, being unaware of the global state of the scene, picks these fish like any other fish within its small receptive field.

The main caveat of using 3D-CNNs for robotics is that voxelization introduces a trade-off between the size and the resolution of the input volume. This leads to slow inference speeds, which makes sliding window 3D-CNNs unsuited for closed loop robotic control. The network trained in paper C uses more than 2 seconds to predict grasps for a volume of $1.1m \times 0.6m \times 0.4m$ with a resolution of $5\frac{mm}{voxel}$, excluding the time needed for pre-processing.

The 2D-CNN is also quite good at predicting grasp poses relative to the steel cylinders. It seemingly is able to infer the pose of the parts and as such it is able to predict good grasps, even in the face of missing depth measurements at the grasp position, which is often the case, see Fig 3.3. However, in our experiments, the network is unable to consistently predict collision free grasps, even if the obstacles it collides with are within the receptive field of the network. The reasons for this are less clear, but it is likely a combination of several factors. There might be issues with the architecture of the neural network and it is also likely that the synthetic training data differs too much from the real data distribution indicated by a large drop in performance between the synthetic validation data and the real robot experiments. Additionally, where the receptive field was relatively small in relation to the graspable objects in the fish-picking case, the opposite is the true for the cylinder picking case. Meaning, that many cylinders fit within the receptive field of the 2D-CNN, e.g., more than 10, contrasted with the fish-picking case where the receptive field was too small to cover a single fish. This means that the network needs to be invariant to large amounts of background clutter. With this many objects visible to the network at the same time, inaccurate simulation of the physics and interaction between the different objects can also come into play and lead to differences in the simulated and real data distributions. With the simulator being simplified, it is therefore likely that there are parts of the state-space visible to the neural network at test time which are out of the training distribution.

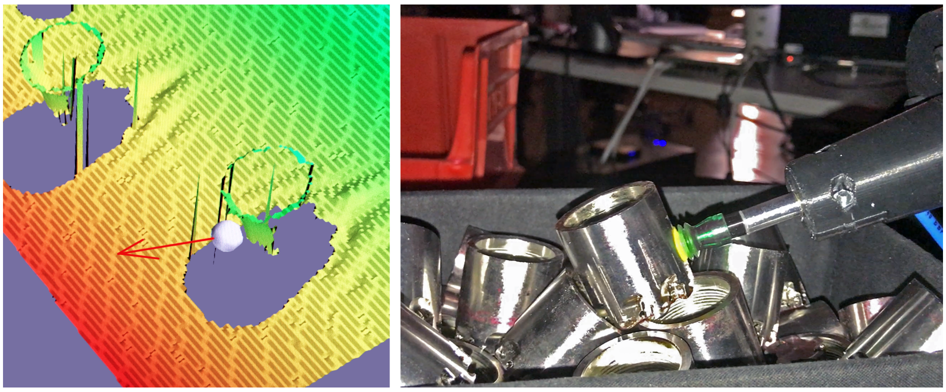

Figure 3.3: Left: The neural network is able to predict grasps (white ball represents contact point and the red arrow the approach vector) by inferring the pose of the parts. Right: The robot performing a grasp.

Conclusion

In this work, we experiment with the use of sliding window approaches for predicting 5 and 6 DoF grasps directly from 2D depth images and 3D volumes in cluttered and noisy domains. In particular, we find that 3D-CNNs are capable of learning good mappings from volumes to grasps, demonstrating good implicit obstacle avoidance and robustness to the type of noise typically seen in 3D-data. However, there is a trade-off between the size and resolution of the input volume which leads to slow inference speeds even for moderately sized volumes. As such, 3D-CNNs are not practical to apply in a sliding window fashion for large input volumes.

3.1.2 Grasping with attention

In section 3.1.1 we concluded that 3D-CNNs are able to learn good grasps which avoid collisions, directly from a voxel grid. Additionally, training on synthetic data sets seems to sufficiently cover the input space for networks with small receptive fields, allowing the system to work on real data and unseen states. However, employing 3D-CNNs in a sliding window fashion leads to slow inference speeds and a neural network which lacks global context awareness, making it unsuitable for more complex tasks, beyond detection. In paper D we address these issues and move towards a more flexible grasping architecture, which can be extended to general robotic control.

Volumes are often sparsely populated. Let's consider a typical automation task: Pick and place operations. In these settings, usually, the work space of the robot will contain a table, some objects lying either on the table, or in some sort of container, e.g., a box, and the task is to move these objects to another destination, perhaps assembling something from the picked objects. Depending on what part of the task the robot is currently performing, different parts of the volume might be of importance. If the robot is about to reach into a box to pick a small object, then the immediate area around the object's location is most important in order to avoid collision and pick the object successfully. When the object is picked, however, the part of the volume where the object used to be is suddenly irrelevant. The important part for the next step is the destination where the object should be placed.

In general, the entire workspace of a robot does not need to be processed with the same amount of scrutiny. Some parts might be irrelevant, while millimeter precision is needed in others. The challenge is that there is no way of knowing which parts of the volume that are of importance in advance. However, there might be some areas of the volume that are more likely to be of importance than others. For instance, consider the table in front of the robot in the pick-and-place case. It is more likely that an object-to-be-picked is lying somewhere on the table, than being under it or floating in mid air. Perhaps the object is usually located to the left of the robot, while the destination for the placement is usually located to the right. In such cases, *searching* over the surface of the table might be a good idea. Different optimal search strategies might exist for different cases, and this can be *learnt* from data.

In section 2.2.2 we introduced the Recurrent Attention Model as proposed by Mnih et al. in 2014 [31]. In their work they trained a RAM on the "60x60 Cluttered Translated MNIST" data set which consists of 28×28 pixel handwritten digits pasted on a larger 60×60 pixel image with added noise. The resulting data set consists of sparsely populated binary images subject to noise, which are analogous to *slices* in a voxel grid. They demonstrate that the RAM is able to successfully focus its attention on the relevant parts of the input, ignoring clutter and classifying the correct digit.

In paper D we propose using a RAM for grasping of fish in large volumes. The network processes large volumes by sequentially attending to small regions of space and building up an internal representation of the scene with a network structure very similar to [32]. Then, the network predicts a probability distribution over grasps, given the internalized representation. We achieve a higher success rate on the comparable fish-picking task of paper C and simultaneously achieve inference speeds compatible with closed loop control.

Further, as discussed in 3.1.1, the sliding window approaches to grasping lack global context awareness, and as such, do not extend naturally to more complex robotic tasks. In contrast, in paper D, we aim to condense the relevant information in the input volume into a global task-relevant feature vector and use this vector to predict a single grasp for the entire volume. This is in correspondence with the “state-to-action” mapping needed for closed-loop control.

The Recurrent Attention Model for Grasping

The neural network of paper D uses a 3D-CNN with similar structure to the one used in paper C as a feature extractor. However, instead of sliding the feature extractor over the entire volume, the 3D-CNN functions as a *virtual sensor* which the RAM can deploy at different locations in the volume to encode the content of the volume at that location. The RAM processes the volume sequentially by deploying the virtual sensor at different locations, building up an internal representation of the work space, conditioned on the task at hand, see Fig. 3.4.

Initially, the network receives the raw, unfiltered, point cloud, and deploys the virtual sensor at a random location l_0 in the volume. A $15cm \times 15cm \times 15cm$ sized crop centered at the location l_0 is extracted, voxelized, and encoded with the 3D-CNN into a 256-dimensional feature vector, z_0. The RAM receives this encoded feature vector z_0, together with the location of the volume it encodes, l_0, and outputs the next sensor deployment location, l_1. A new crop is extracted, centered at location l_1, and this process continues for a fixed number of time-steps, N, where $N = 10$ in paper D. During this sequential processing of the point cloud, the RAM can accumulate information about the seen locations in the volume in the state of a Long Short-Term Memory (LSTM) cell. Lastly, on the final time-step, the model predicts a grasp based on the accumulated state of the scene.

Because extraction of crops from a point cloud is a non-differentiable operation,

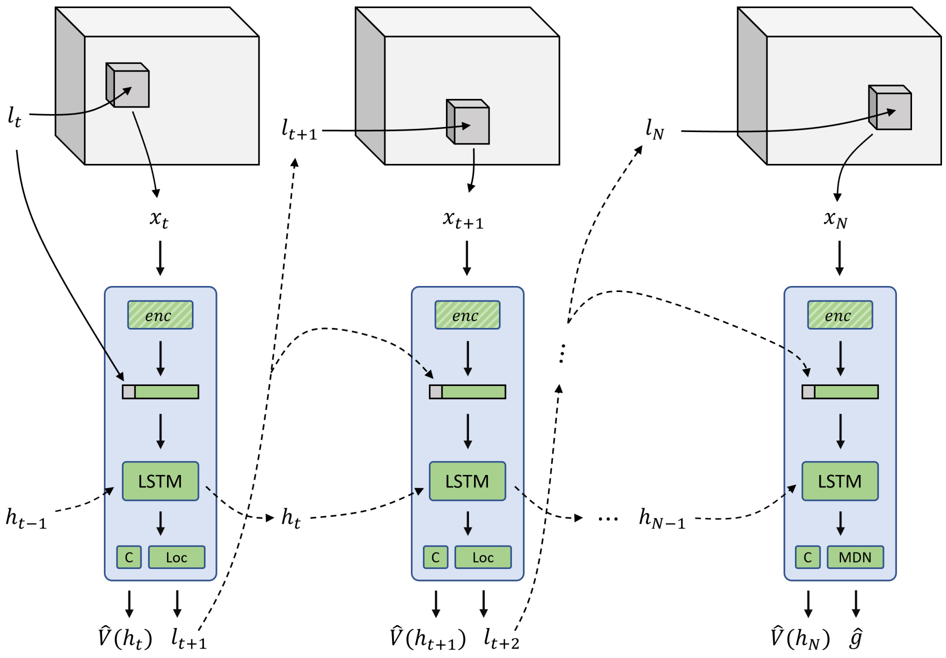

Figure 3.4: The RAM processes the input volume sequentially by deploying a virtual sensor at locations l_t in the volume, extracting crops x_t. At each time step, t, a new location, l_{t+1}, for the deployment of the virtual sensor at the next time step is predicted, based on the current accumulated state of the scene, h_t. On the final time step, N, the accumulated state, containing information about the content and locations of each crop, is used to predict a valid grasp for the volume, $\hat{g}$. The RAM also estimates the *value* of the accumulated state $\hat{V}(h_t)$, which can be interpreted as how likely it is that the network has seen a valid grasp.

the attention mechanism is trained with RL. The aim for the policy is to predict positions in the volume, in which to deploy the virtual sensor, that are likely to contain a graspable object. On each time-step, t, during the sequential processing of the point cloud, the *attention-head* of the RAM receives the accumulated state of the scene and predicts the parameters of a 3D-Gaussian. The predicted distribution is sampled to produce the next the location in which to deploy the virtual sensor, l_{t+1}. The attention mechanism has an actor-critic structure, with the *critic head* predicting the expected sum of discounted rewards, and is trained with Proximal Policy Optimization (PPO) [71]. To enable faster learning, a dense reward signal was designed based on the labeled grasps for the point clouds. On each time step, t, the agent was rewarded with a reward of $r_t = 1$ if at least one labeled grasp was within the receptive field of the deployed sensor at the location l_t, and 0 otherwise. This, essentially, incentivizes the model to search for valid grasps, and upon finding one, to keep the focus of attention on that part of the volume as it collects more reward for keeping the grasp within the receptive field.

In order to minimize the number of parameters to train with RL, the 3D-CNN encoder is trained separately as a variational autoencoder (VAE) and stays fixed during training of the RAM, similarly as in [32].

A separate *grasp prediction-head* receives the accumulated state of the scene and predicts a probability distribution over valid grasps on the final time-step. In the previous work of papers A, B and C, this was framed as a regression problem, where 9-dimensional vectors representing grasp poses were optimized with the mean-squared-error-loss (MSE) between the predictions and the targets. This is equivalent to fitting a Gaussian distribution, and it proved to be a reasonable approximation in the case with small 3D receptive fields. However, with the size of the receptive field in this work, and in general when the RAM makes predictions based on the accumulated state of the scene, this distribution is assumed to be multimodal. Meaning, that there might exist several valid grasps given the current state, while the average of these grasps might not be a valid grasp. Therefore, the grasp prediction head in paper D is designed as a Mixture Density Network (MDN) [72], which predicts the parameters of a mixture of Gaussians. In addition to predicting the 6 DoF pose of the grasps, the network also predicts the gripper opening for each grasp. For more details we refer to the original work in paper D.

Generating training data in simulation

As in the previous work, the neural networks in paper D are trained on synthetically generated data for picking of fish in a bin. The data generating procedures of papers A, B and C relied on 3D-models with predefined candidate grasps labelled relative to the objects. After physics simulation, these candidate grasps were pruned for each object to remove the candidate grasps that were invalid for the particular simulated run. While this approach ensures that all remaining grasps are valid, it does not ensure that all graspable objects in the current scene contains a grasp (as there might exist a valid grasp for an object not in the candidate grasps). While this is fine when training sliding window detectors, it is not ideal when training a RAM. Because the RAM can attend freely to all parts of a given point cloud, all possible grasps for that point cloud should be labeled to avoid noisy gradients.

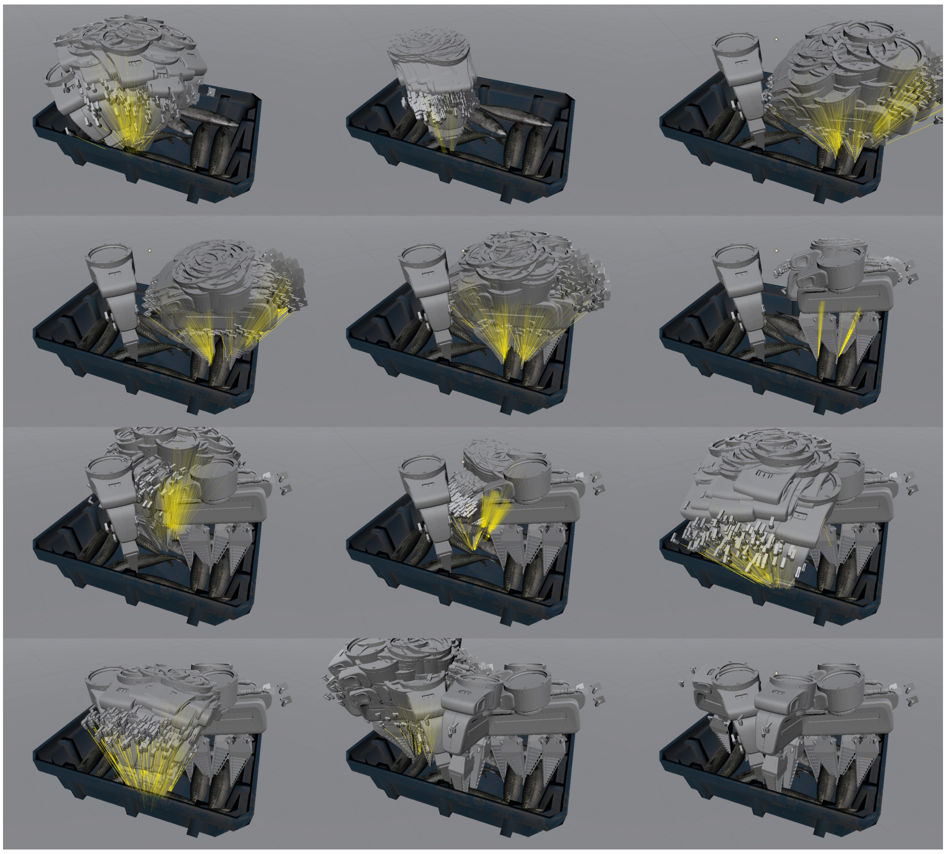

Figure 3.5: We employ an evolutionary algorithm to find the valid grasps for a scene. For each fish, the EA tries to *evolve* an optimal grasp by iteratively replacing the least fit grasps with modified versions of the most fit ones. The fingers of the gripper are approximated by four rays (in yellow) which checks for collisions and is used to evaluate a grasps fitness. Here the algorithm is run with rendered grippers for the entire population for visualization purposes. In the top row, read from left to right, you can see the algorithm finding a valid grasp for the first fish and moving on to searching for grasps for the next fish. This process continues, and the bottom right image shows the resulting valid grasps for this box of fish.

To find all valid grasps for a given scene we assume that each graspable fish has one optimal grasp and employ an Evolutionary Algorithm (EA) in order to find this optimal grasp, see Fig. 3.5. If the EA is unable to find the grasp, the fish is defined as not graspable. The *fitness* of a grasp was evaluated based on a set of task-specific heuristics which included the distance from the grasp to the object's centre of mass, the orientation of the grasp relative to the part of the object between the fingers, and the extent of collision between the gripper and the environment or other objects. As boxes of fish often are tightly packed, we find that allowing for different amounts of collisions between the fingers and the fish, and the fingers and the environment is crucial to the algorithm finding good grasps.

The sliding window 3D-CNN of paper C could be expected to work well on boxes of fish lying in configurations not seen in the training data, because small parts of the volume were processed independently. The same does not necessarily apply when processing unseen types of inputs with the RAM, which has global context awareness. The RAM might learn that fish are unlikely to be located in some areas of the volume by training on the synthetic distribution, while this might not be the case for the real distribution. Therefore, we attempt to make the synthetic boxes of fish as similar to real boxes of fish as possible. However, at test-time, the robots actions will influence future distributions because of the sequential picking of fish, and this cannot be simulated beforehand. The robot will pick the fish it considers to be most likely to contain a valid grasp first (as measured by the estimated value of the current state by the critic), and as the robot picks fish out of the box one by one, it will bias boxes with fewer fish in them towards containing "more difficult" grasps. To address this issue we make assumptions about which types of grasps that might be considered "more difficult" by the model, and bias boxes with few fish in them towards containing more of these types of grasps. We assume the most difficult grasps would be the ones, where very precise grasp placement is required in order to grasp the fish and avoid collisions. Therefore, in order to address the possible distribution shift between the synthetic and real data, we ensure that simulated boxes with few fish in them are biased towards containing fish close to the walls and corners of the box.

During training of the RAM we augment the point clouds with ±180 degree rotations around the world-z-axis, ±15 degrees on the other other two, left-right-flipping, and random translations of ±200 mm on the x- and y-axis and ±50 mm on the z-axis. Additionally, random point-noise was added to all points.

To create the data set for training of the 3D-CNN as a VAE, different crops were extracted from the synthetic point clouds. They were extracted from random locations in the volumes, but with a bias towards the parts of the volumes containing valid grasps. This was done on the intuition that it is most important for the 3D-CNN to encode the regions surrounding valid grasps precisely. Random rotations and random amounts of point noise were added before voxelizing the crops into $32 \times 32 \times 32$ voxel grids,

Discussion and Results

In paper D we improved the data acquisition pipeline from paper C and calibrated an inward facing camera rig with 6 Intel Realsense D415 cameras with overlapping field of views (FOV) covering the robots work area. To further improve processing speeds, no global pre-processing of the point cloud was done, but a suitable threshold for what counts as an occupied voxel, as discussed in 3.1.1, was used locally, when voxelizing the small crop extracted by the virtual sensor. Combined with the RAMs inference speed, which is decoupled from the size of the input volume, the result is a system which can process point clouds with more than 6 million points, at a speed of 15 Hz, including acquisition times. This indicates that the proposed system can be used for closed-loop control, which is the key motivation for this approach to grasping.

For the case of open-loop grasping of fish, fast inference speeds enables us to process the same point cloud several times independently and select the grasp which the RAM is most certain of, as estimated by the critic-head. Because multiple inferences on the same input can be done in parallel without need for re-acquisition of data from the camera rig with subsequent data transfer to the GPU, we can process each point cloud 150 times without affecting the speed of the experiment. Because of the stochastic nature of the policy and the random initial deployment position for the virtual sensor, this will yield 150 different grasp predictions.

The result of the experiments done in paper D is a grasp success rate of 95 % and a task success rate of 100 % (succeeding on second try for grasps that were unsuccessful on the first and emptying all boxes) on the fish-picking task. An example of a predicted grasp from our experiments is shown in Fig. 3.6.

The RAM is able to search the volume to find graspable fish and predict successful grasps based on the seen parts of the volume. In a domain subject to noisy inputs, where the gripper is supposed to be very close to obstacles, often slightly touching the walls of the box, it demonstrates impressive implicit collision avoidance (see Fig. 3.7). During the entire main experiment, and the initial debugging phase with several differently trained models, it never predicted grasps which would collide severely with the environment.

Notably, in the data set, which the model was trained on, the simulated boxes never contained more than 15 fish, while in the experiments of paper D, the boxes were filled with 30 fish. This means that about half of the predicted grasps in the experiments were predicted based on an environment state which was OOD wrt. the training data. Additionally, the real-world point clouds are subject to distractions never seen in simulation, such as the robot itself and the camera-rig.

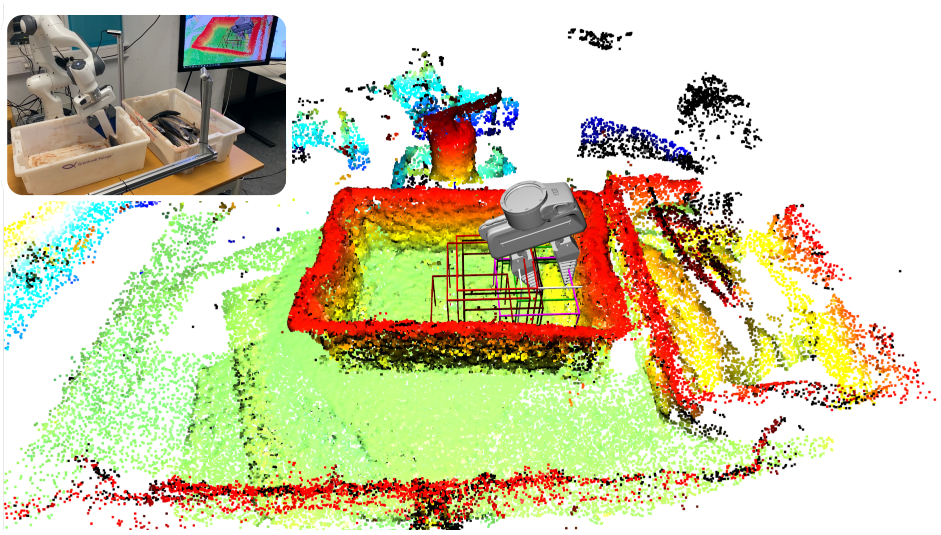

Figure 3.6: An example of a point cloud from the experiments conducted in paper D. The overlaid boxes shows the regions of the volume which the RAM attends to, and a 3D-model of the gripper shows the predicted grasp. Top left: The robot performing the task

The attention mechanism, however, seems to be effectively filtering out all of these distractions and by simply attending to the surface of the pile it is agnostic to the number of fish under the surface.

The grasp prediction head seems to have learnt a good multi-modal distribution over grasps. It does not appear to affect the precision of the predicted grasps when several fish are visible within the receptive field of the encoder[2].

In the experiments of paper D, the robot automatically stopped picking upon emptying the boxes because the critic-head predicted a low value for the state, i.e., the processed volumes were unlikely to contain valid grasps. Keeping in mind that each empty box was processed 150 times in order to actively search for grasps, this indicates some robustness and that the critic has learnt a good value function.

The attention head appears to have learnt to process the volume from left to right in its search for graspable fish. If it encounters what it believes to be a graspable part of a fish, it will keep its attention on it, predicting the remaining virtual sensor deployment locations around the same location. If, by chance, the initial random sensor deployment location happens to contain a graspable fish, the same happens,

[2]With the receptive field being of size $15cm \times 15cm \times 15cm$, usually no more than two graspable parts of fish are visible at the same time.

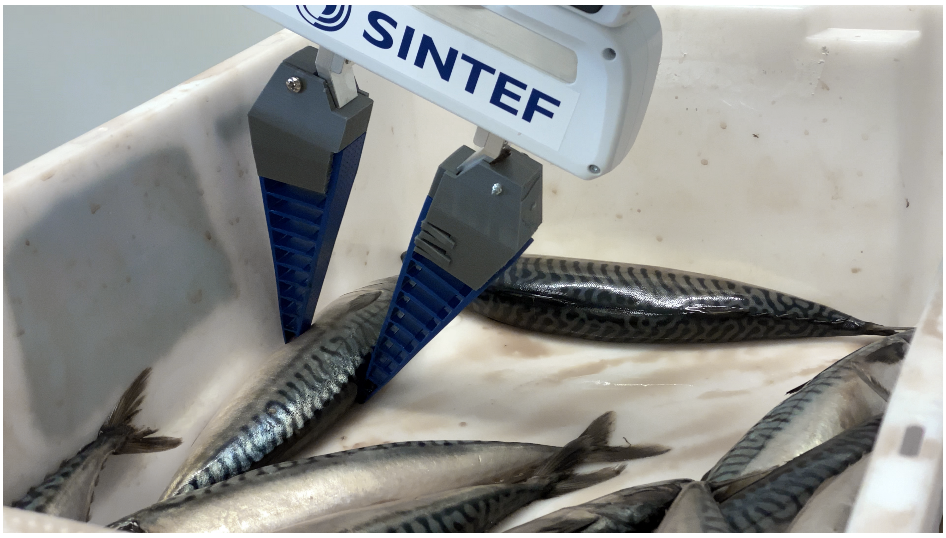

Figure 3.7: The robot performing a grasp predicted by the RAM. Often, as hypothesised in Sec. 3.1.2, after a few picks, the remaining fish in a box are lying along the walls. In these cases, the trained model consistently displays good obstacle avoidance, slightly touching the side and the bottom, as it has to, in order to grasp successfully.

and the RAM focuses its attention on the graspable part of that particular fish for all time steps. This is sub-optimal wrt. the actual goal of the system, which is to predict precise grasps, as there might exist an even better grasp somewhere else in the volume, or more information which could be useful for the already chosen grasp. However, it is expected behaviour with the designed dense reward signal rewarding the agent per time step for keeping the graspable part of a fish within the receptive field.

Most of the errors observed are as a result of slightly imprecise grasp placements either along the long axis of the fish or in the approach direction of the grasp. In these cases, respectively, the fish are grasped either too far from their center of mass, or they are not sufficiently inserted into the gripper, and the result in both cases is that the fish slips out during transport. This likely happens because the grasp prediction head lacks the information needed to infer the full pose of the fish. The policy head is rewarded for deploying the virtual sensor on the graspable part of a fish, not for deploying it in all the locations which could be useful to predict a precise grasp for that fish. For instance, it could be useful for the grasp prediction head to know the position of the head and the tail of a fish in order to find the center of mass precisely. However, as attending to these parts of the volume is not incentivized in any way for the attention policy, these things are not seen by the network and thus unknown also for the grasp prediction head. The solution to this is to modify the reward function. There are several ways of doing this, especially when training in simulation where the full state of the scene is known, and could be used to design a different dense reward signal. However, the most general solution would be to design a sparse reward signal directly coupled to the actual goal of the task, i.e., the quality of the predicted grasps, at the cost of longer training times.

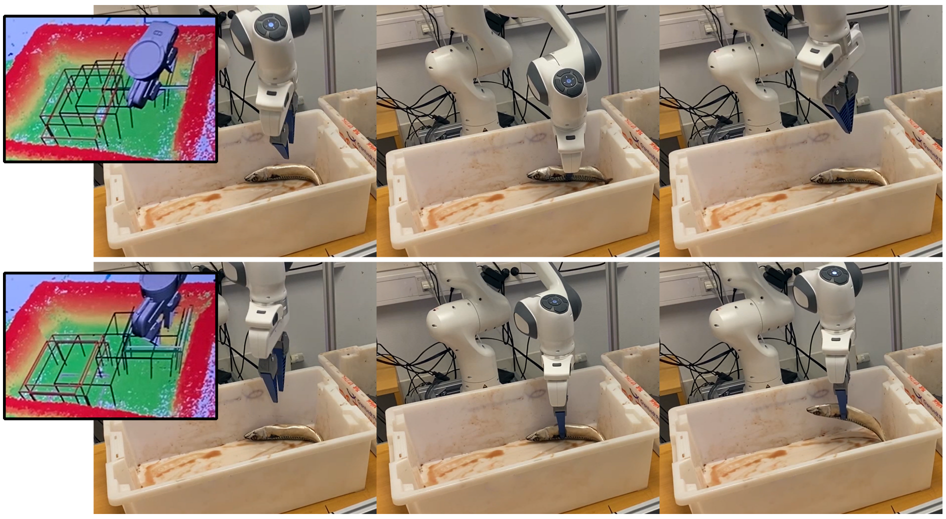

Figure 3.8: Top row: One of the failed picks from paper D. Bottom row: The next predicted grasp which successfully picks the fish. Inset image on both rows: The point cloud visualizing the sequential processing with a set of boxes and the predicted grasp with a 3D-model. As shown, in both cases the RAM attends to the inside of the box, moving from the bottom left corner, up towards the top right, where it finds the fish.

Even if the learnt attention policy isn't optimal wrt. the task of precise grasp placement, the end-result is very good on the challenging task of bin-picking of fish. Even when a grasp fails, the stochastic nature of the policy allows the robot to "try again" and in the experiments of paper D, it never makes the same mistake twice. A failure case is shown in Fig. 3.8. In this case, the last fish in the box is lying in a rare configuration in one of the corners of the box. Additionally, the fish is lying with its belly up, making it more difficult to pick the fish because this part is very soft (in general a grasp closing only on the belly of a fish will likely fail, while a grasp closing on the more firm back of a fish has a larger chance of success). On the first attempt, the robot attempts a grasp, but as the fish isn't inserted far enough into the gripper, the gripper ends up "pinching" the belly and the fish slides out when the robot retracts. On the second attempt, even though the state of the box hasn't changed much, the RAM is able to predict a grasp which successfully picks the fish.

Both the RAM and the separately trained 3D-CNN encoder were trained entirely on synthetically generated data. A total of 2000 point clouds were created with the total number of grasps in the data set counting 9887. Considering that all parameters were trained from scratch, this is not a very large training set, by computer vision standards. We hypothesise that the stochastic sampling of the volume combined with the internal state of the RAM makes the network somewhat robust to over-training, because the network never truly visits the same state twice. Additionally, the augmentations done to the point clouds during training removes unwanted biases in the training set, both forcing the RAM to learn where to look based on what it has already seen, and ensuring that the input space is sufficiently covered for the grasp prediction head to learn good mappings from voxels to grasps. Depending on the complexity of a given task, this opens up the possibility for training of RAMs on real data, removing the need for realistic simulated data and labels, which could be costly to obtain.

Conclusion

By learning to *attend* to the relevant parts of the volume we achieve a grasp success rate of 95 % on the fish picking task, similar to the one in paper C, an increase of 15 percentage points relative to the results of paper C. Additionally, the robot recovers from its errors and successfully empties all boxes in our experiments. With this, we demonstrate that the RAM can be used for robotic grasping with large point clouds in a cluttered domain.

Where the sliding window approaches described in section 3.1.1 were robust to OOD-states because they were unaware of the full state of the scene, the RAM displays robustness to unseen states while maintaining global context awareness, which makes the system suitable for more complex tasks. Simultaneously, we achieve processing speeds which could enable processing at 15 Hz for point clouds with more than 6 million points. This leads us to believe that the RAM is well suited as a feature extractor for robotic applications dealing with point clouds in general, as it is fast, can keep fine detail about some areas of the work space, while on the other hand, filtering out distractions and being robust to noisy depth measurements.

3.2 Closed-Loop Control

In section 3.1 we tackled the problem of grasp detection. In the presented works, grasps were predicted based on the state of the scene as captured by depth cameras, and the predicted grasps were performed by the robot in an open-loop fashion. With this approach to grasping there is a fundamental underlying assumption: That everything is lying perfectly still, i.e., that the scenes are static. While true initially for many pick and place tasks, it might not be true as soon as the gripper comes into contact with the objects it is going to grasp. Indeed, in paper D, one of the main reasons why grasps fail is movement in the box between the time of grasp prediction and closing of the gripper. When this happens, the state of the environment, and thus the conditions on which the predicted grasps were based, changes. The robot, however, unaware of this state change, continues towards the predicted grasp and attempts to pick the object based on outdated information about the objects whereabouts. This is a fundamental limitation of the open-loop approach to robotic control and if we want a system that is able to react to changes in the scene we need to *close the loop*.

Often times, robots need to be able to operate in dynamic and unpredictable environments, and in such cases, closed-loop control is essential. With closed-loop control, the robot continuously senses its environment, i.e., acquires new data from the depth cameras, which enables it to respond to changes in the scene and perform more complex tasks, such as, e.g., pushing and pulling of objects. This feedback from the environment during execution of a task gives the robot the means to compensate for errors and disturbances as they occur during execution.

In particular, in this section we re-purpose the RAM of Sec. 3.1.2 for the task of *visual servoing*. We modify the architecture of paper D and use the modified RAM to predict end-effector velocities, rather than grasps, directly from the point cloud data. We chose to work with velocities as a wide variety of robotic manipulation tasks can be described by sequences of velocities. However, the system has limitations, as it has no notion of force or torque it cannot perform tasks like e.g., apply x Newtons of pressure to an object. However, by working with velocities the training pipeline becomes independent of the dynamics of any specific robotic platform. This makes it easier to create training data in simulation, as in previous work. We assume that the robot controller is able to follow the given velocity commands with little error and take care when creating the data sets to simulate velocities and accelerations that are well within the limits for the robot. Further, working with velocities also makes it easy to create data sets without simulation, through demonstration by teleoperation with a handheld controller without the need for haptic feedback.

In Sec. 3.1.2 we concluded that the RAM has several desirable properties which makes it well suited as a feature extractor for generic robotic manipulation. These properties can be summarized as follows:

- It can process large point clouds at relatively high speeds.
- It has global context awareness and can learn a mapping from arbitrarily sized volumes to a single action.
- It can extract precise features which keeps fine detail about areas of interest.
- It can ignore irrelevant parts of the volume which can make it easier for a model to generalize to OOD-states.
- It is robust to sensor noise and missing values in depth data.

In the following sections, preliminary, unpublished work is presented to demonstrate how the RAM can be used for closed-loop control. In contrast to the the work in section 3.1, which focused on an unsolved real world automation case, the following work focuses on simpler "toy examples" to explore the capabilities and limitations of the proposed system.

In particular, the research in this section addresses the following:

- How to modify the proposed RAM architecture for closed loop control.
- How to reward the attention mechanism for processing of point clouds for robotic control.
- Is the RAM able to attend to multiple objects and infer velocities based on relative distances?
- Is the RAM able to select what it should focus on based the current state of the environment?
- Can the high correlation between subsequent point clouds be exploited to enable faster control loops?

In order to address the outlined research questions, we design two simple example tasks for the robot to execute. The following two sections are dedicated to specific subsets of the research questions, with each section aimed at solving one of the designed example tasks.

In Sec. 3.2.1 a task is designed such that the robot has to steer the end-effector to a specific position, with the gripper placed over a block as if to grasp the block. The section is aimed at addressing the first three questions listed above.

In Sec. 3.2.2 the task from the previous section is extended to a pick and place task, where the block has to be picked up and placed on top of a larger cube. This section primarily addresses the last two questions listed above, but also touches upon the other questions.

Caveat: In the following sections, qualitative results are presented based on preliminary tests and experiments. However, we find it helpful to include these experiments in the thesis as they highlight the motivations and future direction of the research.

3.2.1 Visual servoing in large point clouds with the recurrent attention model

In this section we design a simple RAM for the task of predicting end-effector velocities directly from point clouds. The RAM processes point clouds similarly as discussed in Sec 3.1.2, by predicting locations in the volume in which to deploy a virtual sensor. As before, it processes the point cloud by deploying the virtual sensor, i.e., attending, to a total of $N = 10$ locations, building up an internal representation of the scene in the state of an LSTM. On the final time step during the sequential processing, the model predicts a 3D end-effector velocity which is sent directly as a velocity command to the robot. In order to test the proposed system, we consider a simple task, where the robot has to position its gripper over a block, shown in Fig. 3.9.

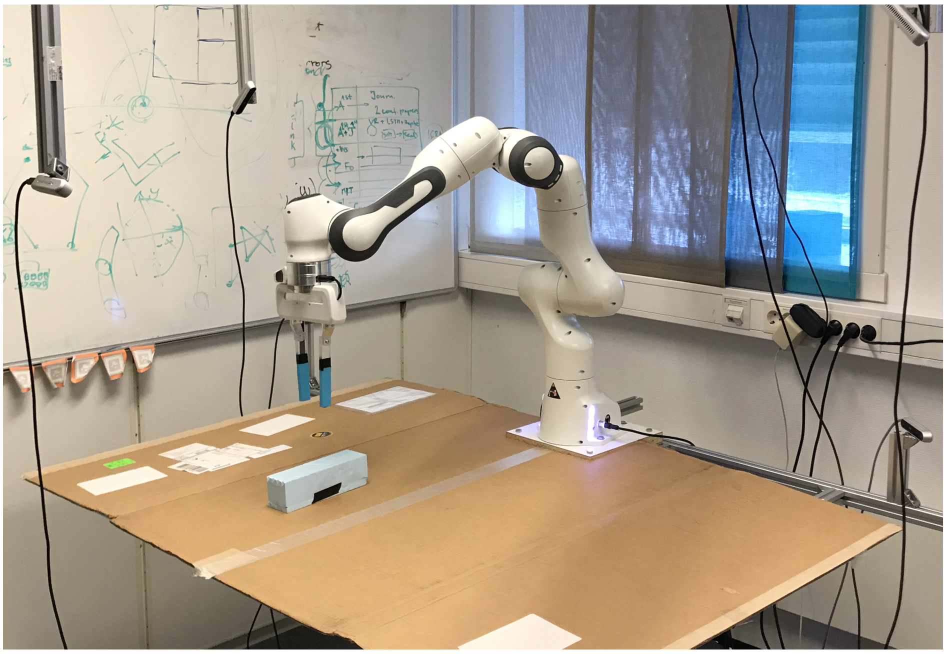

Figure 3.9: The test setup for the experiment in this section. The task for the robot is to position its gripper over the block, so that if the gripper is closed, the object would be grasped successfully.

When doing visual servoing from, essentially, a point cloud video, there is high correlation between consecutive point clouds, which could be exploited by the model. However, in the work presented in this section, we process each point cloud independently and by doing so, we can train the model with simple supervised learning, i.e., behavioural cloning (BC), from static, labelled, point clouds. We expect that it is easier to cover the space of possible inputs and outputs in the training data when learning a mapping from point clouds to actions rather than sets of point clouds to trajectories. In turn, this should make it easier to avoid over-fitting and significantly reduce the time needed for training. There are, however, good arguments for keeping information about the state of the scene between consecutive point clouds, and we address this in Sec. 3.2.2.

The main difference between the work of this section and the work presented in Sec. 3.1.2 is the processing speed requirements. In this work we aim for a control loop of 10 Hz, meaning that the total time spent on data acquisition, data transfer and processing has to take less than 0.1 seconds.

In the work of paper D, the processing speeds of the RAM could be exploited by processing each point cloud 150 times before attempting a grasp based on the most certain prediction. This was necessary because, given the limited receptive field of the virtual sensor and the size of the volume the model needed to search over, no attention policy could be learnt which would guarantee that the graspable object was found every time. The fast processing speeds, however, allowed us to rely on the stochasticity of the model and the initial random sensor deployment location to find all graspable objects. When doing visual servoing, the RAMs fast processing speeds are needed in order to meet the speed requirements for closed-loop control, and thus we cannot afford to process each point cloud several times. Therefore, the RAM needs to learn an attention policy which can reliably find the objects of interest, *every time* it processes the volume. In order to enable this, the model needs a larger receptive field, so it can cover the volume in as few as 10 processing steps. However, at the same time it needs the extracted crops to have high enough resolution to allow for precise robotic control.

Based on these criteria we modify the virtual sensor. Following [31] and [32] we modify the virtual sensor, so that when deployed at location l_t in the volume, it extracts, not one, but two crops, centered on l_t at different scales, x^{focus} and $x^{context}$. The crop with the smallest receptive field, x^{focus} has a size of $30cm \times 30cm \times 30cm$, while the one with the larges receptive field, $x^{context}$, has a size of $90cm \times 90cm \times 90cm$. Both of these extracted crops are voxelized into voxel grids of size $30 \times 30 \times 30$, yielding a resolution of $1cm/voxel$ for x^{focus} and a coarser resolution $3cm/voxel$ for $x^{context}$. Given the size of the work-space of the robot, the two extracted crops together should enable the RAM to extract the information necessary, both for finding the objects of interest and to precisely control the robot.

The goal for the RAM in this work is two-fold: 1) Find an optimal policy for deployment of the virtual sensor in the volume. 2) Predict precise end-effector velocities given the set of partial observations of the volume. In contrast to the work presented in Sec. 3.1.2, where the 3D-CNN encoder was trained separately, in this work, we train the entire model from scratch. Additionally, in this work, we do not hand-design a dense reward signal, but rather use a sparse reward signal directly coupled to the level of success on the velocity prediction task. Therefore, in order to simplify training and to avoid complex weighting of different losses, in this work we essentially train two models in parallel. One model, *the attention network*, is trained to find the optimal attention policy π^* for deployment of the virtual sensor, and the other, *the velocity network*, is trained to predict the end-effector velocities, see Fig. 3.10. As discussed, the attention network needs a large receptive field in order to effectively search the volume, and similarly, the velocity prediction network needs a high resolution input in order to predict precise velocities. Thus, the attention network, the model tasked with learning the optimal policy for extraction of crops $\pi(l_{t+1}|s^\pi)$, does so based on the state, $s^\pi = (x_t^{context}, l_t, h_{t-1}^\pi)$, where $x_t^{context}$ is the large receptive field extracted by the virtual sensor on location l_t at time step t, and h_{t-1}^π is the hidden state of the LSTM of the attention network from the previous time-step. Similarly, the velocity prediction network predicts the velocity of the end-effector, $\hat{y}_t = f_{vel}(x_t^{focus}, l_t, h_{t-1}^{f_{vel}})$, based on the small receptive field extracted by the virtual sensor, x_t^{focus} and the hidden state, $h_{t-1}^{f_{vel}}$, of a separate LSTM-cell. Both networks have Multi Layer Perceptron (MLP)-heads, where the attention network predicts the parameters of a Gaussian distribution and the velocity prediction network predicts a 3D velocity vector.

In summary, two separate networks, the attention and the velocity prediction networks, predict the next sensor deployment location l_{t+1} and the velocity for the end-effector $\hat{y}_t$. This separation of responsibilities allows for easier training with different gradient signals. We do stress, however, that for more advanced tasks, the attention network should have access to the high resolution features as well, as the coarse resolution of $x^{context}$ might limit the attention networks ability to know what it is focusing on.

The velocity prediction network predicts a 3D vector representing the velocity of the end-effector. In this preliminary work we keep the orientation of the gripper fixed. The network is trained with the MSE-loss between the true and predicted velocities, which is fine in this case because the target policy for the robot is known to be unimodal (see Sec. 3.2.1).

The attention network is trained with the standard REINFORCE algorithm [73], as in [31], and in contrast to paper D, without a critic. A sparse reward is given at the final time step $N = 10$, and is calculated as the clamped negative l1-loss between the predicted and labelled velocities, $r_{t=N} = min(c, -|y - \hat{y}|)$, where c is the clamping threshold. As such, the reward for the policy is directly coupled to how well the velocity prediction network is able to predict correctly, and a perfect

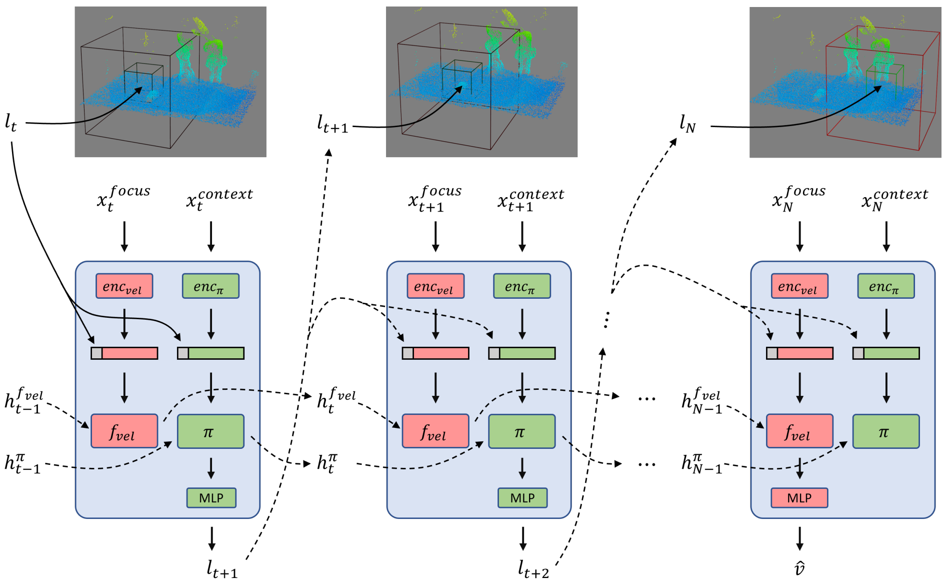

Figure 3.10: In this work we decouple the velocity prediction task from the task of learning to sequentially process the volume, meaning that no parameters are shared between the two tasks. We train two networks simultaneously, where the attention network (green) predicts the next position to deploy the virtual sensor, l_{t+1}, while the velocity network (red) predicts the end-effector velocity $\hat{v}$ on the last time-step N, based on the sequence of observations. The attention network takes the largest receptive field, $x_t^{context}$, as input and the velocity network takes the smallest, x_t^{focus}. Both networks keep their own accumulated states of the scene, h_t^{π} and h_t^{vel}, respectively.

velocity prediction would yield a reward of 0, and an erroneous prediction would yield a negative reward. The complete advantage for the policy for each time step is calculated as the discounted sum of future rewards.

Generating training data in simulation

In order to test the proposed system, we design a simple task, where the robot has to position its gripper over a block, as shown in Fig. 3.9. The model is trained on a synthetically generated data set of point clouds and corresponding end-effector velocities. The target velocities are given based on the relative position between the end-effector and the target block. In order to achieve the given task, the RAM needs to find and encode the sub-volume containing the block together with its location, and find and encode the sub-volume containing the gripper together with its location. Subsequently, it needs to infer the relative distances between the objects to predict an appropriate velocity for the end-effector. Additionally, the RAM needs to reason about obstacles in the robots path and adjust its predictions in order to avoid collisions. To test the RAMs capability of detecting and avoiding obstacles, we consider two types of simple obstacles: 1) A plank placed horizontally between the robots end-effector and the target block. 2) The robot base. In order to avoid collision with the plank, the RAM needs to adjust the predicted velocity in order to pass over it. Similarly, the robots own base might be an obstacle if the object and end-effector are located on opposite sides of the base, and the RAM needs to adjust the prediction to go around the base to avoid collision.

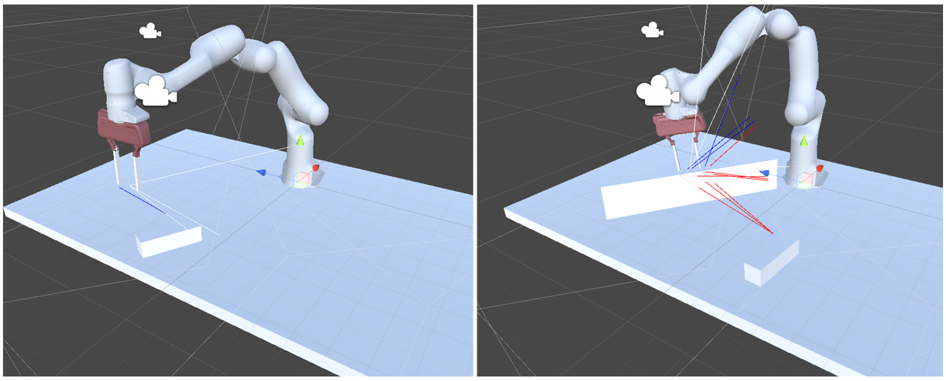
Figure 3.11: The data sets used for training were generated in simulation. The velocity for the end-effector is given by the relative position between the target block and the end effector. Left: The virtual robot regulating towards a way-point above the target block. Right: The velocity given by the controller would lead to a collision with the plank, and is adjusted so that the end-effecor passes over.

The experimental setup is designed in such a way that the environment state has the Markov property, meaning that the correct velocity for any point cloud is directly observable from that point cloud alone. Additionally, as there is only one target block in each scene, there is only one correct velocity for each point cloud. In simulation, the velocity for the end-effector, i.e., the labels for the training set, are simply set by a proportional (P-)controller, regulating towards a target pose based on the error in position $e_t = target\_pos_t - ef\_pos_t$. Initially, the target position for the regulator is set $10cm$ above the block as a way-point, and when the end-effector comes within a threshold distance of this way-point, the final target position is set. The final target position is set such that if the gripper is closed, the object would be secured in the gripper, i.e., equal to the way-point position, off-set by $-10cm$ in the world z direction. If the velocity, as given by the P-controller would lead to a collision with the plank, the velocity is simply off-set so that the gripper passes over (see Fig.3.11). Similarly, if the velocity given by the controller would lead to a collision with the robot base, the velocity is adjusted so that the gripper passes around the base, in front of the robot. During simulation, point-cloud-velocity-pairs were extracted at 2 Hz to create a data set of 20 000 training examples, and the velocities were clamped at a conservative speed.

Discussion and Results

To test the proposed system, the camera rig of paper D was used to acquire a point cloud with more than 6 million points. The point cloud was given without pre-processing to the RAM, which in turn predicted velocities which were sent directly to the robot as velocity commands. With this setup, we achieve a 10 Hz control loop.

The attention network π has learnt to process the volume in front of the robot from left to right. It starts on the left side of the table and moves step by step to the right, tracing down the center line of the table. When one of the objects of interest (either the gripper or the target block) comes into view of the large receptive field $x_t^{context}$, the attention deviates from the center line and is focused around the object of interest for a varying number of time steps. Subsequently, the attention returns to the center line of the table in search for other objects of interest. If not already observed near the first object, the same thing happens when the second object comes into view of the large receptive field. Interestingly, the attention never leaves the surface of the table, and is always placed at a height which ensures that the small receptive field, x_t^{focus}, both encapsulates the surface of the table and, if present, the gripper. An example is shown in Fig. 3.12 and more examples are given in Appendix A. The model seems to have learnt a robust policy, which successfully locates the objects of interest in our experiments.

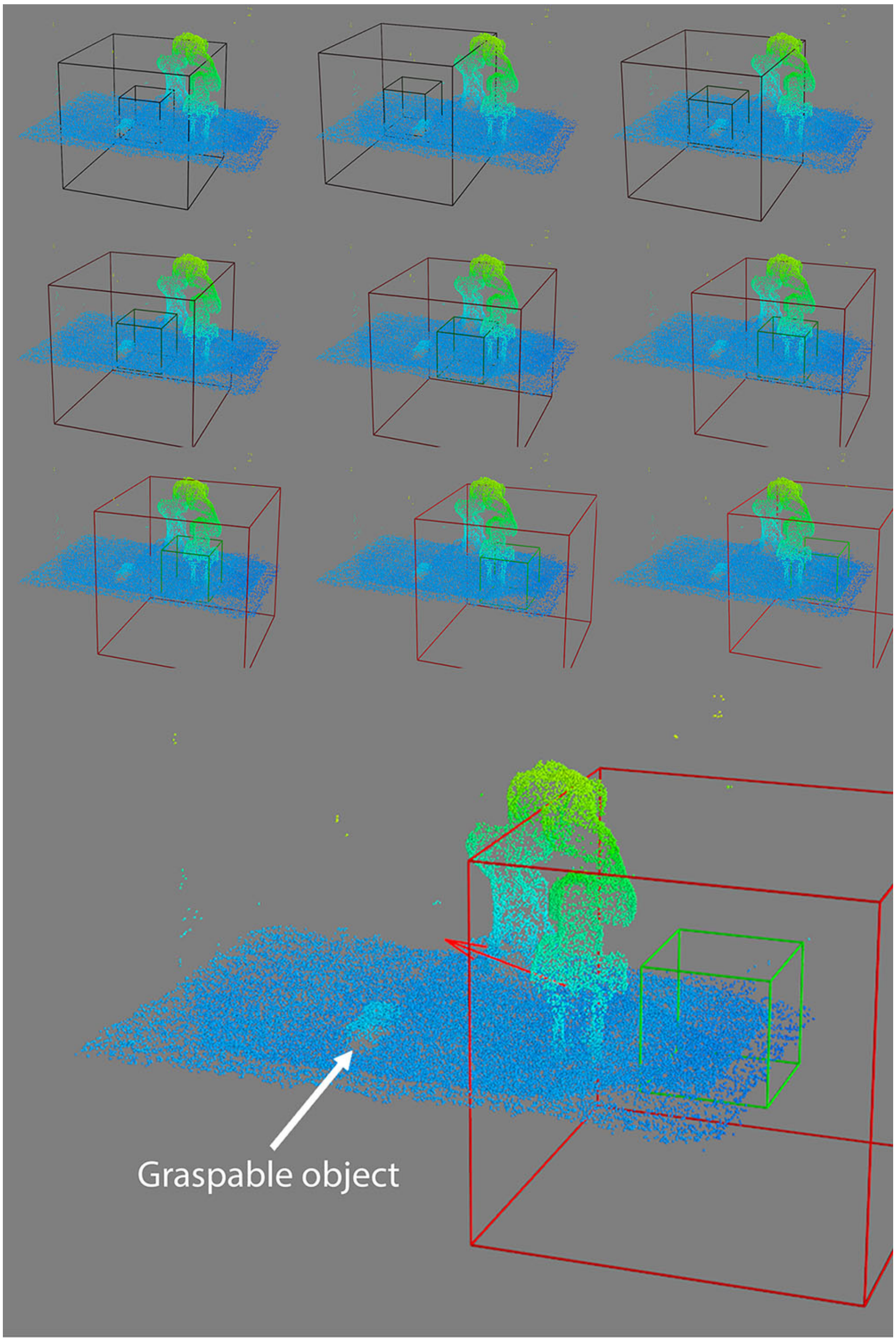

Figure 3.12: The RAM has learned to process the volume from left to right with the receptive field of the focus network placed on the table surface. The red arrow indicates the predicted velocity towards the graspable object. (The point clouds are cropped and subsampled for visualization.)

When the model is allowed to control the robot at test time, it is able to steer the robot directly towards the target block and place the target block between the fingers of the gripper, see Fig. 3.13. However, it sometimes spends more time on the final part of the trajectory than the robot does in simulation, i.e., when placing the gripper around the target block on the way to the final target position. Because the robot has no notion of its own speed and predicts velocities directly from a set of samples taken from a noisy point cloud, it is subject varying amounts of jerky movement. As the gripper opening only has a $0.5cm$ clearing between the fingers and the target block on both sides, very little jerky movement is tolerated in the final phase of the trajectory. The model therefore, keeps regulating the gripper back and forth sometimes, before deciding to go down. However, considering the noisy input, and that the target object is to some degree occluded by the gripper, the velocity prediction network has learnt to predict, to us, surprisingly precise velocities based on the extracted set of observations from the point cloud. By doing so consistently, it also confirms that the attention network has learnt a robust policy for processing of the volume, as the velocity network would not be able to predict precise velocities without observing the objects. Some examples of predicted velocities for different environment states are visualized in Fig. 3.14.

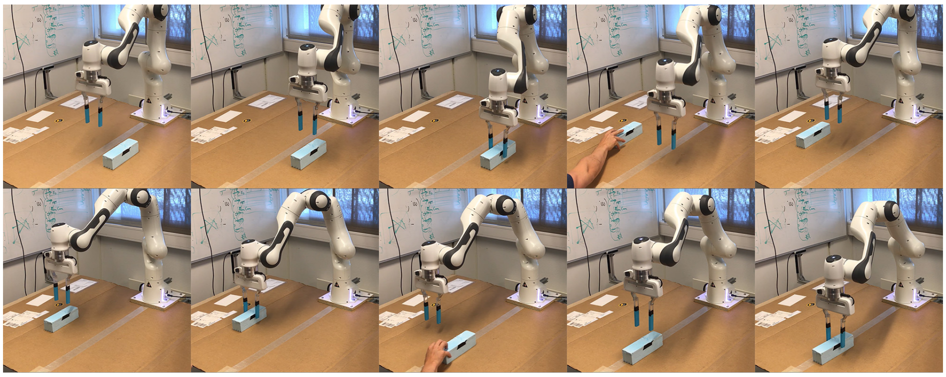

Figure 3.13: The robot executing the task at test-time. The robot is able to consistently place the gripper over the target block, when the block is moved around in the work space.

The model has also learnt to try to avoid collisions, although it does not always succeed (see Fig. 3.15). The model consistently adjusts the predicted velocities to account for the obstacles, but not always sufficiently, especially when navigating around the robot base or very close to the plank. This could likely be improved by having larger margins when avoiding collisions in simulation during data generation. In simulation, the end-effector always takes the shortest path around the obstacles, which leaves very little room for error for the robot. Additionally, the controller in simulation accounts for the obstacles from the very beginning of a

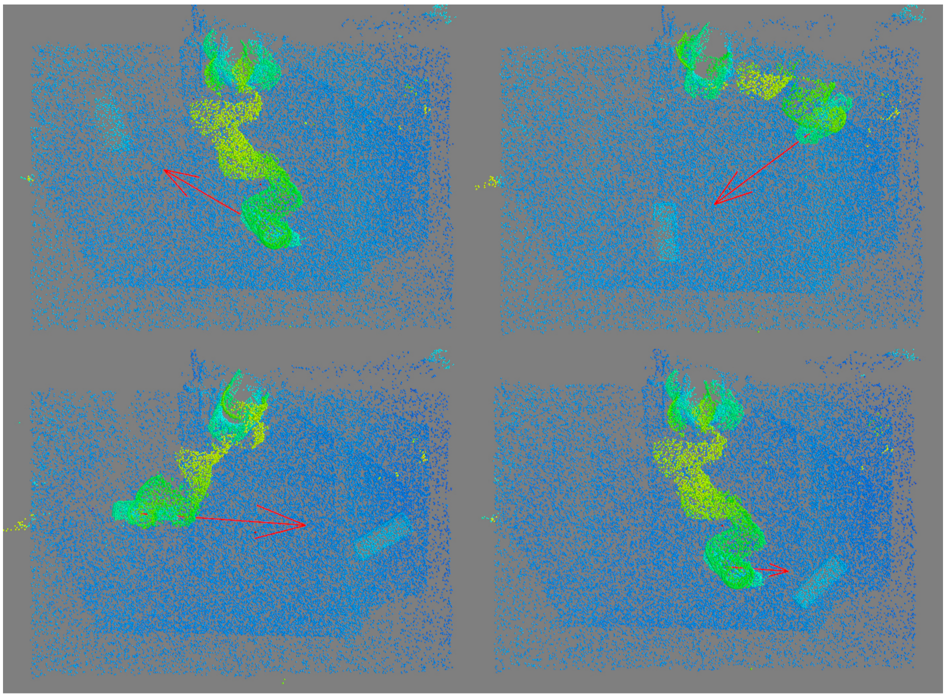

Figure 3.14: The red arrow indicates the predicted velocity towards the graspable object. (The point clouds are cropped and subsampled for visualization.)

trajectory (i.e., when still far away from the obstacle), slightly adjusting the velocity in order to closely pass by the obstacles. The result is that the robot very rarely gets close to the plank while still being close to the table, and similarly, the robot rarely gets close to its own base when the target block is located on the opposite side of the base initially. In fact, the only time this happens in the training data is when, by chance, the initial conditions of the scene happens to be such that this is the case. E.g., the end-effector is instantiated very close to the plank and close to the table, with the object on the other side of the plank. In the extreme case, when the gripper is almost touching the plank, the correct velocity to predict would be one pointing directly upward, or perhaps slightly in the opposite direction of the plank and the object. However, as the robot in simulation never gets this close to the obstacles, it never needs to take these types of corrective action and, thus, these types of examples do not exist in the training data. Therefore, during execution, as the errors in the velocity networks predictions compound over time, we get a distribution shift. There are several ways of addressing this issue, and the simplest solution would be to sample these difficult environment states more often in simulation. Alternatively noise can be added to the simulated trajectories similar to

DART [40], with the result in both cases being training examples in the data set for corrective actions. Another approach would be to do online RL either in the real world or in simulation in order to fine tune the velocity prediction network.

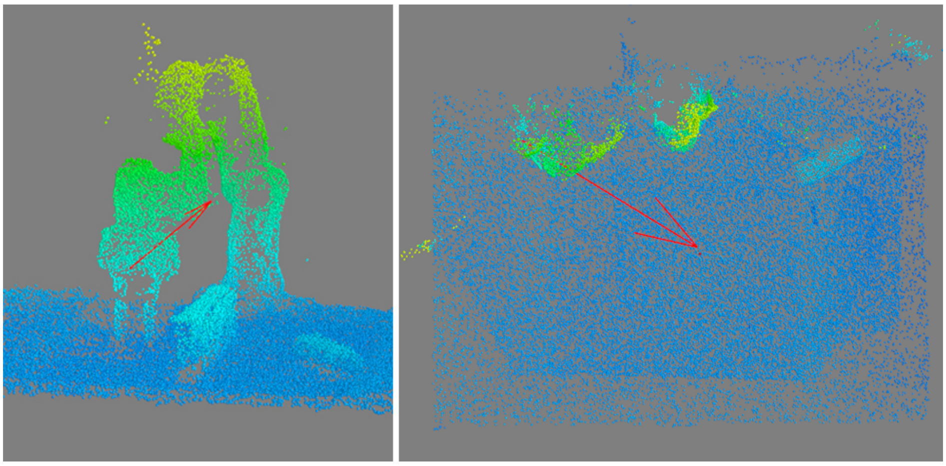

Figure 3.15: Avoiding obstacles. Left: The model tries to avoid colliding with the plank by predicting the velocity visualized by the red arrow. This velocity, however, might still lead to collision, or just barely miss, and as such, the model should ideally predict a larger z-component to be on the safe side. Right: The model has also learnt to adjust the velocity in order to avoid colliding with the robot base, but the end-effector, in general, ends up passing a bit too close too the base. (The point clouds are cropped and subsampled for visualization.)

For the simple example task tested in this experiment, the model shows quite good data efficiency. The data set the model was trained on was gathered by capturing point clouds at 2 Hz while the robot was executing the task in simulation. This only amounts to about 2,8 hours of real-time simulation which indicates that it might be feasible to gather the needed amounts of training data in the real world. It is impractical to create new simulations for each new scenario, and for more physics dependent tasks than the one considered in this work it can be very difficult to make the simulators accurate enough. The possibility of learning from demonstrations in the real world, e.g., by teleoperation, is therefore appealing. However, the data needs will vary with the complexity of the tasks, and simulation will likely remain a very useful tool during further development.

Conclusion

In conclusion, the attention network has learnt to process the input volume effectively in a way that enables the robot to complete the task. It consistently finds the gripper, target block and obstacles, and these are the only things affecting the labels in the training set. We achieve a 10 Hz control loop when processing each point cloud independently, and hypothesize that even faster control loops can be enabled by exploiting the high correlation between subsequent point clouds.

The velocity network has learnt a robust mapping from the observations given by the attention network to velocities when no obstacles are present. The model is capable of remembering positions, inferring relative distances and predicting the correct velocities. However, the robot is not always successful in avoiding collisions, because of the distributional shift introduced at test-time by following the robots policy. We do, however, not take this as a evidence against the use of RAMs for robotic control, as it is a known problem with offline training of agents in general.

3.2.2 Visual servoing using the recurrent attention model with memory through time

In this section, we further motivate the use of the RAM for closed loop control, by demonstrating that the model can learn to exploit object permanence, and thus enable even faster control loops than shown in Sec. 3.2.1. By keeping memory through time, the model can make assumption about the likely positions of objects of interest, based on their positions in the previous point clouds, thereby allowing for fewer processing steps per point cloud and overall faster processing.

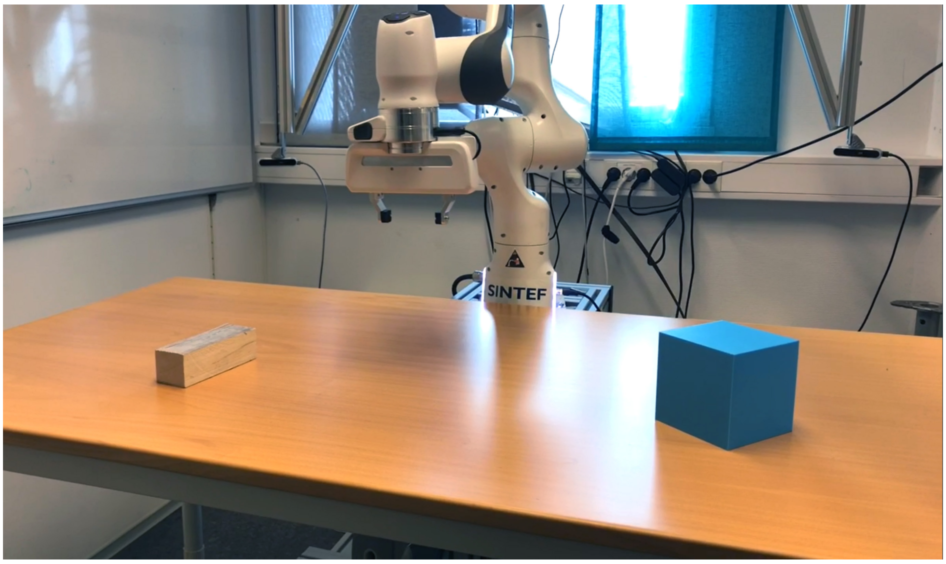

Figure 3.16: The test setup for the experiment conducted in this section. The goal for the robot is to pick up the small wooden block and place it on the larger blue cube.

Inspired by the data efficiency shown by the model trained in Sec. 3.2.1, we choose to train the model in this section using only real world data gathered by teleoperation of the real robot, i.e., learning from demonstration. Similarly as in the previous section, the goal is to predict velocities directly from the point clouds and we consider a simple pick and place task as a test bed for our approach. In this case, a block should be picked up and placed on a larger cube, with both objects being located at random positions on a table. The test setup for the experiment presented in this section is shown in Fig. 3.16. A small experiment was done, where 100 trajectories were demonstrated with the teleoperated robot and 75 of these were used to train a RAM and 25 were kept for validation. The main objective of the experiment was to train the attention mechanism to attend to the important parts of the volume, given the current sub-task, by keeping memory through time. The secondary objective was to do imitation learning on the demonstrated trajectories. We hypothesize that the filtering effect of the attention network might encourage the velocity prediction network to learn a good mapping from point clouds to velocities, even with very little data, as the network has nothing else to base its decisions on but the extracted, task-relevant parts of the volume.

We do a few changes to the architecture of Sec. 3.2.1 and a block diagram is shown in Fig. 3.17. The main difference however, is that we no longer reset the hidden states of the LSTMs between processing of different point clouds. This enables the model to remember the locations of objects from the previous point cloud when processing the next. Thus, it can exploit object permanence, i.e., the high correlation between consecutive point clouds, and process each point cloud in fewer steps. Specifically, in this experiment we reduce the number of processing steps per point cloud, from $N = 10$ in the previous section, to $N = 5$.

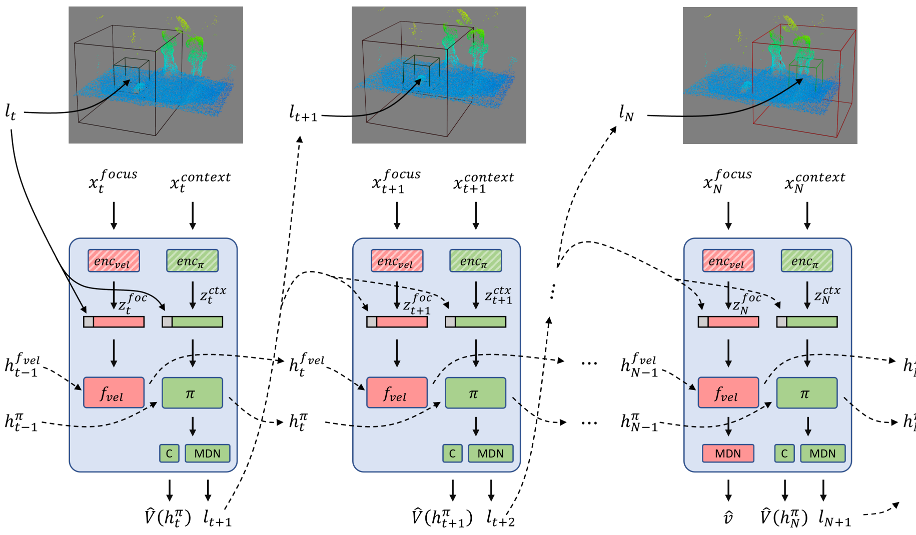

Figure 3.17: The architecture of the model trained in this work. The encoders are shaded to indicate that these are trained separately.

In this work we no longer assume that the target policy for the velocity prediction network is unimodal. Because we are training on real demonstrated trajectories, this can no longer be guaranteed and, in general, different operators may teleoperate the robot differently, and the same operator might also act differently each time he or she visits the same state. Therefore, we swap the simple MLP used to regress velocities in section 3.2.1 for an MDN-head similar to the one used to predict grasps in paper D. This allows the network to predict a multimodal probability distribution over possible velocities given very similar or equal states.

We also modify the attention network, which predicts the next virtual sensor deployment location, by swapping the Gaussian policy head for the more expressive MDN. To solve the given pick and place task, the attention network initially needs to attend to the gripper and the target block, and upon closing the gripper, it needs to shift its attention towards the drop point, indicated by a larger cube. As the model in this work needs to attend to several objects we hypothesize that it can benefit from a multimodal policy. Especially in "transition phases", where the model needs to keep its attention on what it is doing at the moment, e.g., grasping the block, while at the same having to explore the volume to figure out the next action, e.g., find the large cube.

In this work, we add a critic head to the attention network, similarly as in Sec. 3.1.2. The critic estimates the value of the current state at time step t, i.e., the expected sum of future rewards, $\hat{V}(h_t^\pi)$, given the hidden state of the attention networks LSTM, h_{t-1}^π, which crucially is not reset, so that it can contain information from previously processed point clouds.

The attention network is trained with PPO, similarly as in paper D, with the exception that we do not reset the LSTM-state. Like in the previous section, the velocity prediction network is trained supervised, through behavioural cloning and we keep the orientation of the gripper fixed.

In the experiment described in this section, the model is allowed to deploy the virtual sensor on $N = 5$ different locations per point cloud. On every N^{th} processing step, a velocity is predicted, and a new point cloud is acquired from the camera rig. Subsequently, on every $N + 1^{\text{th}}$ processing step, the model proceeds to process the new point cloud, and this processing cycle goes on indefinitely. Initially, we let the attention network "get its bearings" by processing the first 8 point clouds without involving the velocity prediction network. This allows the attention network to deploy the virtual sensor at 40 different locations in order to search for the important parts of the volume. This can be done in less than 0.5 seconds when not voxelizing or doing inference on the small receptive field with the velocity network. We find that this effectively eliminates undesired erratic behaviour of the robot in the very beginning of a new trajectory.

The encoders in this work are 3D-CNNs similar to the ones used in Sec. 3.2.1, and are trained separately as VAEs, like the encoder in paper D. In this work, the large crop extracted by the virtual sensor, $x_t^{context}$, is encoded as a 256-dimensional feature vector, $z_t^{context} = enc_\pi(x_t^{context})$. Similarly, the smaller, high resolution, crop is encoded by $z_t^{focus} = enc_{vel}(x_t^{focus})$. The extracted crops for the context and focus encoders have resolutions of $12mm/voxel$ and $6mm/voxel$, yielding receptive fields of $38.4cm$ and $19.2cm$ respectively. These receptive fields are significantly smaller than the ones used in Sec. 3.2.1. In the previous section, because each point cloud was processed independently, we had to make sure that the network could cover the entire work space of the robot with attention in only $N = 10$ processing steps. This was accomplished by setting the size of the receptive field for the attention network relatively large. In this experiment, however, we can allow for smaller receptive fields as the model can keep memory through time and, as such, does not need to process the entire point cloud every time. The large receptive field in this work is therefore more comparable to the small receptive field in Sec. 3.2.1.

Small receptive fields for the 3D-CNNs lead to higher resolution inputs or faster inference speeds, both of which are beneficial properties for robotic control. Another reason to keep the receptive fields small is to reduce the risk of over-training on the limited amounts of training data. With only 75 demonstrated trajectories used for training, we have only visited a very small sub-set of the total number of possible environment states. As a receptive field is increased, a larger and larger portion of this state is visible to the encoder, and we hypothesize that this could lead to over-training on the seen environment states. By keeping the receptive fields small, we ensure that the full environment state, e.g., containing information about distances between objects and the configuration of the robot arm, does not affect the encoding of small regions of the volume. The encoders job, should rather be to encode small regions precisely, while the complete model with access to many encoded sub-volumes and their locations should be able to approximate the environment state. The model as a whole could still over-train, however, but the stochastic sampling of the volume and the accumulated hidden state of the LSTMs ensures that the model never truly sees the exact same state twice, which might help mitigate the issue.

Both encoders, enc_π and enc_{vel}, are trained as VAEs on programmatically extracted crops from the point clouds acquired through teleoperation. In order to make sure that the encoders are able to encode "the most important parts" of the volumes well, we take biased samples from the point clouds when building the data sets used to trained them. We specifically consider the parts of volumes close to *gripper actions*, i.e., locations where the gripper was opened or closed. On the assumption that the location where a gripper action occurred is of interest in the time leading up to that gripper action, e.g. that the graspable object is lying still, we bias our sampler towards sampling around that location in the time leading up to that gripper action. In our case, the result is a data set biased towards containing extracted crops of both the graspable object and the drop position with the gripper approaching at varying distances. The sampler is also biased towards sampling around the current location of the gripper in addition to sampling random locations spanning the entire work space for the robot. Before voxelizing and creating the data sets for the VAEs we do heavy augmentations with ± 180 degree rotations around the world-z-axis, ± 20 degrees around the x- and y-axis, random flipping along the x- and y- axis[3], varying amounts of point noise, point sub-sampling and point duplication with additional noise added to the duplicates.

Data acquisition

We collect training and validation data by demonstrating the task by teleoperating the robot with a virtual reality controller. During teleoperation, point clouds were gathered at 30 Hz from a camera rig similar to the one used in Sec. 3.2.1. These point clouds were stored with the corresponding target end-effector velocities, angular velocities and gripper actions, and together they constituted the demonstrated trajectories.

The task was demonstrated starting from different initial conditions, one of which is shown in Fig. 3.16. We also demonstrate some *recovery trajectories* starting from initial conditions based on likely failure states, i.e., states the robot is likely to end up in which would not be covered by simply demonstrating the task. An example of this would be that we start a demonstration with the gripper touching the table next to the wooden block.

A total of 100 trajectories were demonstrated, and 75 of these were used for training, and the remaining 25 were used for validation. With the test setup being approximately bilaterally symmetric, we double size of the training set with left-right flip augmentation. We also apply small random shifts in the point clouds of up to $\pm 10cm$ on all axes during training. A side effect of this is that the robot base and table are offset in the volume in ways which will not be seen by the network at test time. We do, however, hypothesize that these shifts will keep the networks, and particularly the attention network, from over-training on the exact locations of the objects. As the target velocities are dependent on the relative distance between the objects, rather than the absolute positions of the objects, the random shifts should not affect the training labels.

The data acquired when demonstrating the trajectories were gathered at 30 Hz, while the target for the control loop is to run at 15 Hz. Therefore, during training, we stride through the time dimension with a stride of 2 and simulate varying amounts of jerky movement by randomly skipping forwards or backwards by up to 15 time-steps, i.e., ± 0.5 seconds.

[3]Although the gripper is not perfectly symmetrical, we treat it as such.

Reward

The attention network in this experiment was trained with a dense reward signal in order to decouple the performance of the attention network from the performance of the velocity prediction network. This allowed for investigation of the main objective of the experiment separately, namely the performance of the attention mechanism with memory through time.

We assume that a sequence of actions consists of a set of independent sub-tasks. I.e., that the picking part of a pick and place task can be executed without focusing on the placing part. With this assumption we design a reward function which rewards the model for observing the robotic gripper and the *next target position*. In this experiment, the next target position would be the wooden block while executing the part of a trajectory leading up to a grasp, and upon closing of the gripper, the next target position would be the drop point for the wooden block, the blue cube. We assume that all objects, except for the gripper, are stationary and this allows us to find these target positions, l_t^{target}, based on the demonstrated trajectories. We define gripper actions, as earlier, as the points in a trajectory where the gripper is opened or closed. Specifically we set

$$l_t^{target} = l_i^{gripper\ action}, \quad t_{i-1} < t \leq t_i, \tag{3.1}$$

where $l_i^{gripper\ action}$ is the location where gripper action i occurred (e.g., the location where the gripper was opened or closed), and t_i is the time at which gripper action i occurred.

In this case, we encourage the model to observe *only* its own gripper and the part of the volume where next gripper action occurs, and ignore everything else. By doing so, we ensure that the accumulated state of the scene in the velocity prediction network contains only the information relevant to predicting the velocity. At any point in time t, there are two interest points in the volume, the current location of the gripper $l_t^{gripper}$ and the location of the next gripper action, l_t^{target}. We design the reward function so that the optimal policy would be to alternate between observing these two interest points:

$$r_t = r_{seen}(l_t, l_t^{gripper}) * (1 - G) + r_{seen}(l_t, l_t^{target}) * G \tag{3.2}$$

In Eq. 3.2, G determines if the agent gets a reward for observing the gripper or the current target position by having $G = 1$ if the gripper was seen last and $G = 0$ otherwise. r_{seen} determines the amount of reward the agent gets for attending in the vicinity of the correct interest point and is given by

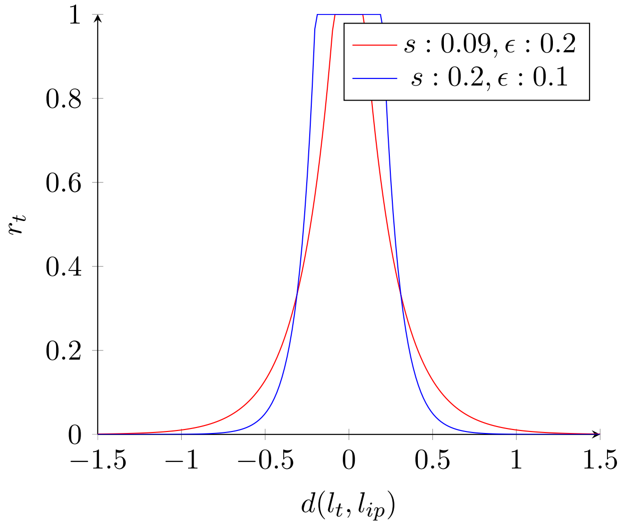

Figure 3.18: The reward r_t plotted as a function of the distance between the location of attention l_t and the location of an interest point l_{ip}. Here, s controls the threshold for when maximum reward is given, and ϵ controls how the reward decays with distance.

$$r_{seen}(l_t, l_{ip}) = min(1, \exp - \frac{d(l_t, l_{ip}) - s}{\epsilon}), \tag{3.3}$$

where d is the euclidean distance between the location of the deployed virtual sensor, l_t and the interest point, l_{ip}, s is the threshold for how close to the actual interest point the attention needs to be placed in order to get the maximum reward and ϵ controls how the reward decays with distance. The result is a reward of 1 if the location of attention is close enough to the location of the interest point to regard the interest point as "seen" by the network and then it decays towards zero with distance. A plot of how the reward changes with distance is shown in Fig. 3.18.

Discussion and Results

By reducing the number of processing steps, i.e., sensor deployment locations, per point cloud we achieve a control loop running at 15 Hz. The attention network has learned to attend to the gripper and target locations, and an example from the validation set is shown in Fig. 3.19. The network is able to find the objects of interest quickly, and also tracks them well through time. It has learnt to focus solely on the gripper and the next target position, i.e., one sub-task at the time, as incentivized by the reward function. When the gripper closes in on the graspable object, the model "goes looking for" the drop point, as seen in Fig. 3.19, and upon closing of the grasp it switches its attention from the location of grasping, towards the drop point.

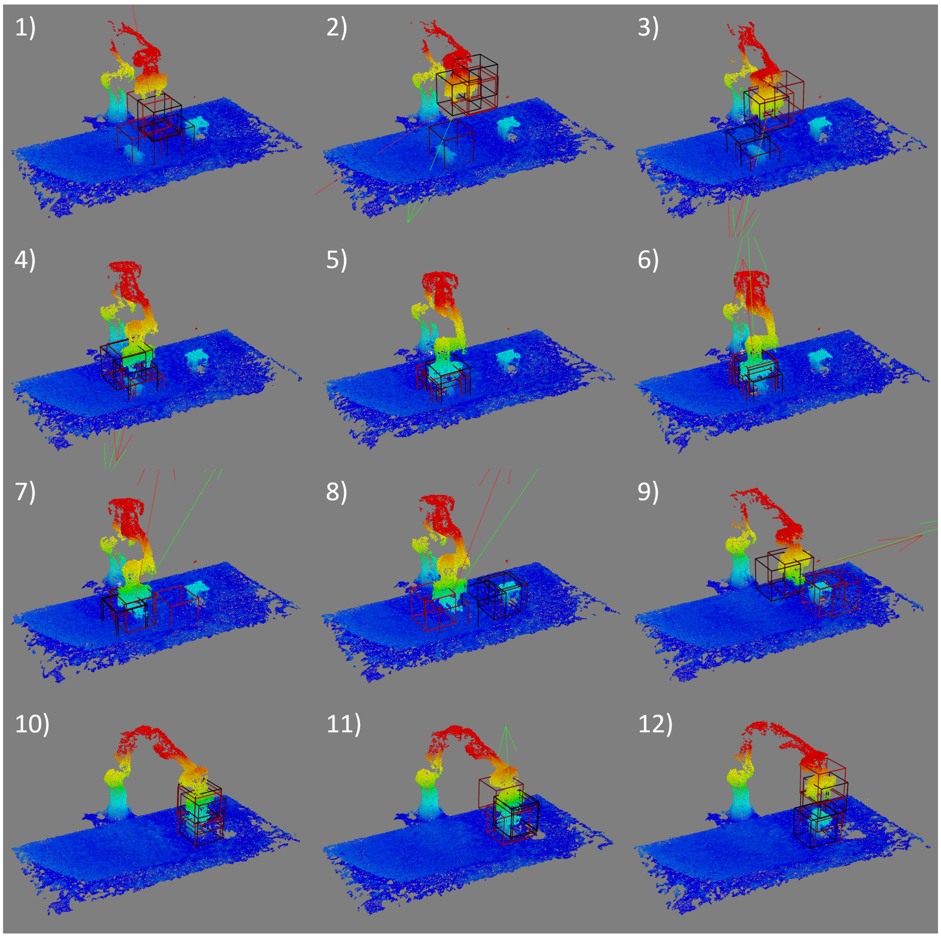

Figure 3.19: An example of the network processing one of the trajectories from the validation set. The small receptive field of the virtual sensor is visualized with small cubes. Notice how, upon completing the grasp of the wooden block in subplot 7, the attention wanders off to the right, to find the drop position where the block should be placed. (The point clouds are cropped and subsampled for visualization.)

Although the velocity network is presented only with the information it needs by the attention network (e.i., the gripper, and the next target position), it fails to predict correct velocities consistently. Ironically, the main pressure point seems to be to regulate precisely over the graspable wooden block, the part which worked very well in the work presented in Sec. 3.2.1. It does however accomplish the entire task from time to time, and a successful execution is shown in Fig. 3.20 and a failed attempt is shown in Fig. 3.21.

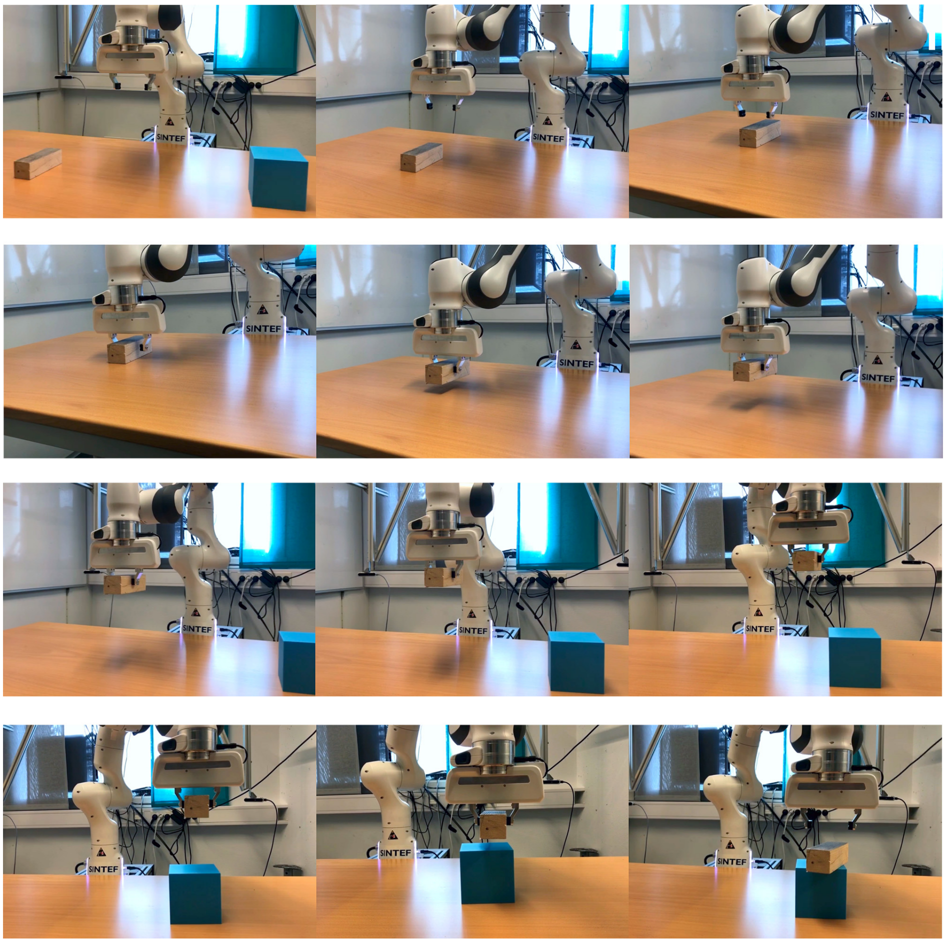

Figure 3.20: An example of a successful execution of the task. The robot picks up the wooden block and drops it on the target cube.

There are several potential reasons why the velocity network has not learned a reliable mapping from observations to velocities, even performing worse than the network of Sec. 3.2.1. One factor, likely affecting the results to a large degree, is the size of, and variation in, the training data set. It is likely that the network has seen too few examples of different environment states and that the ones it has seen are too similar for the network to learn a good mapping. For instance, with 75 demonstrated trajectories, there are only 75 different initial conditions.

This only very sparsely covers the space of possibilities for different locations and rotations of the three objects, and can make it hard for the network to generalize. In the previous experiment, the state space was likely much better covered because the initial conditions were more uniformly sampled in simulation and because the trajectories were only sampled at 2 Hz.

If the environment state-space is not sufficiently covered in the training data, one needs to make sure that the robot stays close to the familiar states visited during demonstration. If this isn't enforced, errors will compound over time leading the robot into OOD states as the robot executes the task, similarly as discussed in 3.2.1. This issue becomes more severe in this work because the network considers both the current observation and the running history of the previous observations as encoded in the hidden state of the LSTM. As the robot executes the task, it needn't take long before the model encounters an unseen combination of where the end effector currently is and where it has been. Like in the previous experiment, the issue arises from training on a data set which lacks examples of corrective actions. A possible solution to this is to extend the data set with these types of examples either by demonstrating how to recover from typical failure cases directly or with methods like DART [40] and DAGGER [39] or alternatively with on-policy fine tuning with RL.

The network does sometimes take corrective actions, typically when it encounters one of the expected failure states, for which specific recovery trajectories were demonstrated as described in Sec. 3.2.2. Sometimes, this leads to a successful recovery, but other times the robot fails to achieve the task despite the corrective action bringing the end-effector into a presumably familiar environment state. An example of this is shown in Fig. 3.21, where the robot goes down too quickly and ends up next to, rather than over, the wooden block. In this case, the model correctly guides the end-effector back up over the wooden block, only to go down again too early and collide with the object. The robot keeps repeating this motion, alternating between going up and down only to collide with the block over and over while pushing the block across the table. In this case, it seems plausible that the initial corrective action brought the robot back into a well explored part of the environment state space. When it still fails, this indicates that the velocity network weights the history of previous observations too much when taking the next action, as only the combination of the observed environment state and the accumulated history together should constitute an unfamiliar observation. This is further indicated by the, in general, very smooth robot motion observed, resulting from very similar predictions made at subsequent time steps. This is in contrast to the very jerky motion observed in Sec. 3.2.1, where the robot relied solely on the current observations. A pragmatic approach to solving this issue for the problem in this experiment would be to reset the state of the velocity network per point cloud, like in the previous experiment. In this particular case, the velocity network does not need memory through time to accomplish the task as all the information it needs is provided by the attention network's correct sensor deployment locations. This solution, however, would limit the model to only working with static scenery, as the model would lose the possibility of inferring the velocities of objects by tracking them through time. Again, an effective solution addressing the issue of ODD-states directly could be to use DAGGER [39], as this would effectively append the data set with corrective actions for the combinations of environment states and histories, thereby teaching the network to correctly weight the current environment states vs. the history in such cases.

A related concern is that the network could be able to fit the training data simply by observing the current end-effector velocity and use this as the estimate for the next velocity. The model could achieve this simply by observing the gripper, and it would likely lead to a very good fit as the velocity changes very little from frame to frame. However, this doesn't seem to be what the model has learnt, as the robot often successfully completes the task, and only fails when regulating very close to the target positions. We speculate that it is easy for the model to learn a mapping to velocities from the xyz locations of the end-effector and target positions, and these positions are more or less given to the model by the attention network through the precise placement of the virtual sensor. It is possibly more difficult for the model to predict the current velocity for the end-effector because the end-effector doesn't move much from frame to frame when new point clouds are acquired at 15 Hz. Additionally, the precise position of the end-effector at a given time is not known to the model, as the volume is only stochastically sampled in the vicinity of the gripper. Coupled with the lossy compression introduced by the encoder, the only way the model could estimate the velocity then is by relying on longer term memory to do the calculations. However, in the future, the current end-effector state (velocity, angular velocity and gripper opening) will likely be given to the model as an input, as this is useful information for the model. The velocity head will then have to be changed, e.g., to predicting delta velocities.

The encoders were also trained on the point clouds collected during the 75 demonstrated training trajectories. Although more diverse data sets were created for the unsupervised learning by means of heavy augmentation, the input space for the encoders is likely not sufficiently covered in the training data. Further, training the encoders as VAEs might not necessarily emphasize precise encoding of the parts of the input that are most important for the given task. This, in turn, could make it more difficult for the velocity network to generalize from few examples, as it needs to learn how to decode the entangled latent space of the encoders. A well structured latent space for the encoders could therefore also improve the model, one from which the end-effector and target poses are easily inferable. As both the gripper and target poses are known in this case, one could, for instance, train the encoders with auxiliary objectives trying to predict these poses directly.

The above-mentioned issues with the velocity network are largely known issues with imitation learning and behavioural cloning in general, mentioned in Sec. 2.3. While it has not been the focus of this work to address these fundamental problems with behavioral cloning, we hypothesized that the filtering effect of the attention network could allow the model to learn the correct mapping, even from very little data. To some extent, it seems that this has happened. The model seems to base the predicted velocity on the relative positions of the end-effector and the target position. However, it is not able to do so very precisely, which leads to a distribution shift and the robot ending ut in unrecoverable states.

Conclusion

In conclusion, by letting the attention network have memory through time we achieve a control loop running at 15 Hz with arbitrarily sized, high resolution volumes as input. In this preliminary experiment, the attention network has successfully learnt to find the objects of interest, and to keep track of them through time. The network has learnt, as incentivized by a designed reward function, to focus its attention on one sub-task at the time, and switch its focus of attention to the next sub-task upon completing the first.

The velocity network has not learnt a robust mapping from observations to velocities. The reasons for failure seems to be poor coverage of the environment state-space during demonstration coupled with the distributional shift problem, common when doing behavioural cloning. However, the robot successfully completes the whole task from time to time, sometimes several times in a row, despite being trained only on very little and highly correlated data. We hypothesize that more, and crucially, more diverse training data would improve the performance of the model when trained using behavioral cloning. Alternatively, online RL could be used for fine-tuning, which would eliminate the distributional shift problem entirely.

In summary, the model has proven capable of extracting precise information (in Sec. 3.1.2) and achieves fast inference speeds when processing large, high resolution volumes. Therefore, we conclude that RAMs are a viable approach to feature extraction for robotic learning in applications requiring high fidelity inputs. However, more work is needed on how to successfully train a robot policy from human demonstrations.

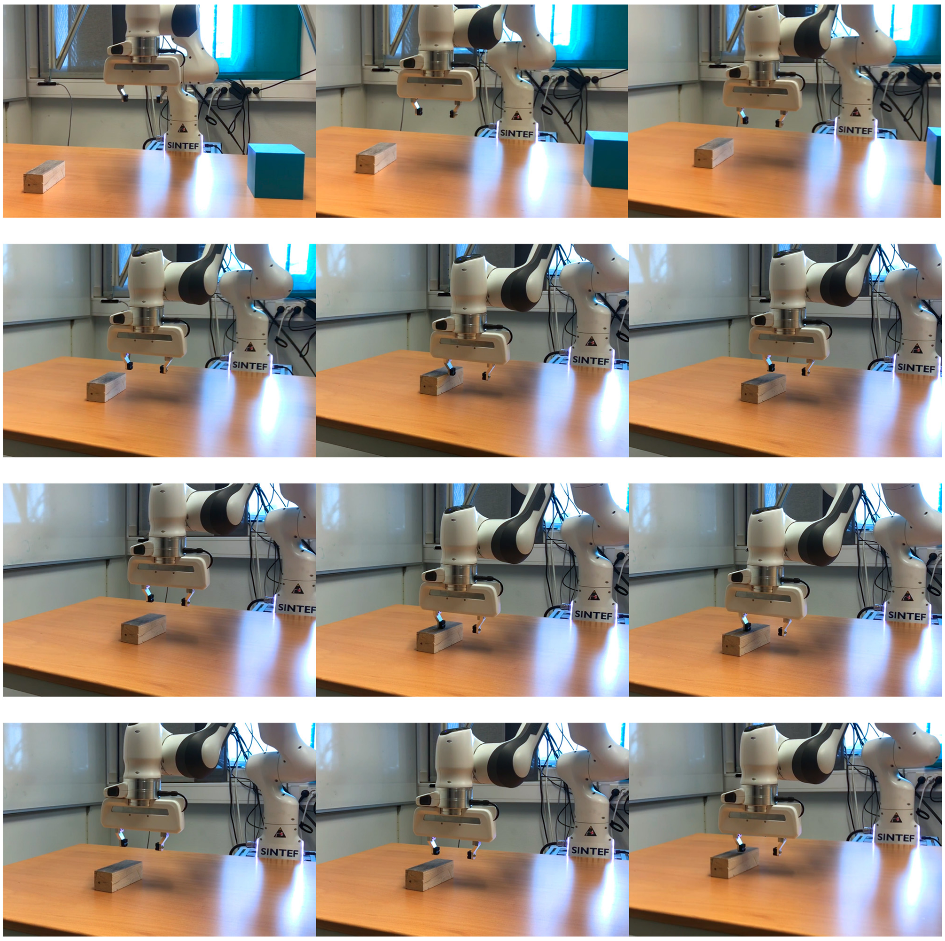

Figure 3.21: An example of a failed attempt. In this case, the robot loses height too quickly and ends up in a state next to the wooden block and never recovers. Even though it correctly goes back up, it misses again on both the second and third attempt at grasping, and ends up crashing into and pushing the block around.

Chapter 4

Discussion

The work presented in this thesis has aimed at contributing to the realization of robots capable of performing tasks in unstructured and dynamic real-world environments.

This chapter discusses the contributions of the thesis relative to the research objectives (defined in Sec. 1.1.1) and outlines possible future directions for the research.

4.1 Visual processing for generic robotic applications

The result of our work towards a generic visual processing system for robotics is an approach based on processing of point clouds with attention. A few variants of the proposed visual processing system have been applied to grasping of fish and to two simple visual servoing tasks. In this section we downplay the role of these differences and focus on the overall approach to visual processing and discuss whether or not it is a viable approach suitable for generic robotic manipulation tasks.

A multitude of different sensors can be combined to produce high-resolution point clouds with coverage of arbitrarily sized volumes. As such, the point cloud representation on its own has properties that are desirable for a visual processing system. The challenge lies in effective processing of this representation. The processing needs to be fast and able to extract precise features from the raw point cloud that are relevant to the task at hand.

The proposed system aims to enable this type of processing by making two core assumptions: 1) The entire volume is not of equal importance when executing a manipulation task or sub-task. 2) When observing objects at a sufficiently high frequency, their positions in the volume remain nearly unchanged. In most relevant automation settings, we judge these assumptions as reasonable. When manipulating an object, the immediate region surrounding the object is of particular interest, as well as the region surrounding the gripper. However, a significant part of the volume might still be of importance to the task, in the sense that it needs to be searched over in order to find the object which should be manipulated. However, in this case, the model can learn to exploit prior knowledge of the domain to narrow down the search space, by e.g., only considering parts of the volume containing horizontal surfaces. Further, by viewing such manipulation tasks as consisting of sub-tasks, e.g., first search for the object, then manipulate the object, the model can leverage assumption number two upon finding the object and limit the search space to the immediate region where the object was found for future time-steps. For this assumption to hold true when objects in the scene are moving, it is a requirement that the acquisition and processing speeds are sufficient relative to the movement of the objects. As such, the assumptions on which the proposed system is based do not exclude many relevant automation tasks.

The proposed system has proven capable of extracting precise enough features to enable grasping of fish with high success rates. In order to do this it needs to reason about its own gripper in relation to the environment and the task at hand. The fish-picking task is difficult because successful grasping often entails sliding the gripper in between fish lying tightly packed together or placing the gripper so that it slightly touches the wall and bottom of the box. This calls for precise predictions if collisions and damaging of the raw material is to be avoided. Further, it is a very cluttered domain, subject to large variations in the appearance of the objects due to noisy depth data and the slippery and deformable nature of the fish. Success in this domain, therefore indicates that the proposed system is capable of extracting precise features under quite challenging conditions. However, the fish-picking domain is still quite narrow. Although there is some variation in the raw-material, all the fish are of the same species, and roughly the same size. Therefore, more work needs to be done to see how well the system tackles larger variations, e.g., the number of different types of objects of different sizes it can handle simultaneously.

The fish-picking task is a task subject to most of the types of noise typically seen when working with depth data. However, the reflective fish skin, semi-translucent box and water coated surfaces makes the amount of noise more severe in this domain than many others dealing with surfaces that scatter light more diffusely. As such, the system has demonstrated some robustness to the types of noise expected to be seen when working with point clouds in general.

By successfully picking fish, the proposed system demonstrated a capability of extracting precise *local* features from the point cloud. For instance, when attending to a location containing a fish, the model is able to predict a precise, collision-free grasp for that fish. However, such a collision-free grasp can be derived from the current location of attention alone, without involving memory. A general purpose visual processing system needs to be able to extract such precise local features from several locations in the volume and accumulate the information in memory for subsequent decision making based on the whole context. In Sec. 3.2 we explored these capabilities in a laboratory setting through two experiments. In the first experiment, the system proved capable of summarizing information by learning to precisely use visual servoing to position the gripper over a target block. This task could only be achieved by finding and encoding each object and their locations in memory, and subsequently predicting an appropriate velocity based on their relative distances. The model was also able to infer whether or not there were any obstacles in the predicted path, and adjust its behaviour. However it did not learn a robotic policy which avoided collisions consistently, likely mainly because of the distribution shift problem. In the second experiment, the model demonstrated that it can infer context by observing the scene and shift its focus of attention appropriately for execution of the current sub-task. In the tested pick-and-place setting, it initially, during the picking stage, attended to the gripper and the graspable object. Upon picking of the object it learned to shift its focus of attention towards the location where the picked object should be placed. In both of theses experiments, as well as in the fish-picking case, the attention network effectively found the relevant parts of the volume, and filtered out distractions. These are very encouraging results. However, ultimately, whether or not the current capabilities of the system are sufficient for it to be applied to a given automation task will depend on the complexity of that task. More research is needed to find the limits of the current system for tasks involving more complex relationships between objects and long-term memory. Additionally, a complete robotic system needs to both extract the relevant information from the scene, and act appropriately based on this information. While the main focus of this thesis has been on the former, there is no clear distinction between these two steps when training end-to-end for robotic control. Therefore, more work needs to be done in order to learn good robotic policies based on the extracted features. This is a large topic which is only briefly touched upon in this work and our initial attempts have led to either jerky motion or poor generalization.

Closed-loop control, enables robotic solutions to recover from errors and respond to disturbances. It also opens for the possibility of working in dynamic scenery, subject to movements. To be compatible with dynamic scenery, a robot's visual processing system should be able to precisely capture the velocities and accelerations of objects. This capability was not explicitly tested in our work, however in Sec. 3.2.2 we argued that it could be difficult for the model to infer velocities precisely. To do this, the current model needs to precisely encode the positions of objects over time, and because of the high processing speeds, rely on rather long-term memory to infer their velocities and accelerations. This can be difficult for the model to learn in practice. For dynamic scenery therefore, it could be better to swap the 3D-CNN encoder for a 4D-CNN encoder. By doing this, the model can attend to regions of space, and encode the content of that region for the last few time-steps. As such, it would be able to encode not only the object at that location, but also that object's speed and perhaps its' acceleration with a sufficiently large time-window. This ability, however, would likely come at a significant cost in terms of processing speed.

With regards to future research directions, there are many possibilities. The proposed system is composed by modules and evaluating each part's contribution and iterating on the architecture could lead to better results. Some examples are: Testing different sensors for point cloud acquisition, testing point-based encoders vs. CNNs vs. sparse CNNs, and swapping the LSTM for a transformer [74]. Another interesting topic for future research is pre-training of the model for generic robotic manipulation tasks. This can likely lead to faster training and possibly also better performance on several tasks, assuming that features extracted for, e.g., collision avoidance, finding graspable parts of objects and optimal processing of the volume, are to some degree transferable between the tasks.

4.2 Robust task-specific grasping solution

In our research we have worked towards designing a system that can be applied for different pick-and-place applications across several industries. Different tasks might entail different types of handling, even of the same object. Further, following research objective 3, no large modifications to the working environment should be needed in order to deploy or re-deploy the robot. Therefore, the robot also needs to be able to operate in a wide variety of environments, perhaps using different types of grippers for different task. Making a robotic solution that is capable of all this out-of-the-box is difficult. Our approach has therefore been to create a generic system for grasping which can be made task-specific through training on task-specific data sets. As task-specific data sets can be costly to obtain, we have also developed a pipeline for synthetic generation of such data sets.

The resulting system for grasping was presented in Sec. 3.1.2, and it was tested in the fish-picking domain. As discussed in the previous section (Sec. 4.1) this is a difficult domain where precise, collision-free grasps must be predicted from point clouds subject to large amounts of clutter and noise.

The results on the fish-picking task are promising. The model achieves a 95 %

grasp success rate and corrects its own mistakes when a grasp fails. Further, it demonstrates robustness to out-of-distribution-states by successfully emptying boxes with twice as many fish at test-time compared to those seen during training. It also exhibits high inference speeds, with the overall speed of operation limited by the robot's ability to move the fish without them sliding out of the gripper.

The fish-picking task was chosen because it is a difficult picking task, and no simplifications were made in our experiment, by, e.g., patting the fish dry or swapping the fish crate for one with less sharp corners, made by a different material. Handling of fish in piles is not something that is done by robots in today's industry. It is a particularly difficult task because of the atypical amounts of noise, and because the gripper has to touch other objects and obstacles in order to successfully grasp the fish. As no parts of the designed system were tailored to this specific task, we view the good performance displayed in this domain as an indication that the system might be used successfully in other domains as well. As the fish-picking task was designed to be difficult, many tasks, even within the fishing industry are to some degree "easier" than the tested case. For instance, in reality, the fish piles at processing factories are often much larger, and the containers rarely have as sharp corners as the ones found in the box in the experiments. As such, the collision avoidance capabilities of the proposed system can actually be less relevant in some industry cases. The ability to process large, cluttered work-spaces, however, is often highly relevant. Additionally, the fresh fish or frozen fish[1] typically seen in the industry are much firmer than the fish used in the experiments. Working with firmer fish will likely improve the grasp success rates as some of the errors in the experiments were due to the softness of the belly of the fish. Further, once lifted, a firmer fish is more likely to stay in the gripper which will also eliminate some of the errors seen. Thus, although the complete system is only tested on one test-case, we believe that the developed system has many more application areas because of the properties of the chosen test-case.

However, two important properties of the proposed system are not yet sufficiently tested: The systems ability to handle large variations in the sizes of the graspable objects and its ability to utilize color information. In many settings, these two properties are very important, e.g., in sorting applications. However, as the feature extractors are based on CNNs, we hypothesize that these things can be handled by the model. CNNs are very well proven as capable of extracting features for classification purposes from color data and object sizes are also derivable from voxel grids. However, the model's ability to discern fine details will be limited by the resolution of the discretizised volume. With more variation in the input, more

[1]Some processing steps in the industry only work if the fish holds a temperature of about -2°, -3°C, precisely because of the texture change in the fish muscle.

data will also be needed for training, and it should be noted that it is more difficult to simulate realistic looking color images than depth images.

With regards to future research directions for the grasping system, many of the ones listed in the previous section regarding the overall visual processing system are relevant. Further, as discussed in Sec. 3.1.2, the designed reward signal could be improved upon by coupling it more tightly with the success on the actual grasping task. Additionally, because the system already displays a quite high grasp success rate, a small, labelled, real-world validation set covering some typical examples and edge cases could be very useful in squeezing out the remaining percentage points.

4.3 Repurposable robotic solutions

Through the experiments on fish-picking it has been demonstrated that the proposed grasping system can be trained solely on synthetic data and perform well in the real world. Whether this would be possible or not was not known when the work on this dissertation began. Such training can possibly enable significantly cheaper development cycles for robotic solutions. In order to repurpose the robot trained for picking of fish for some other task, say, picking of bananas, one simply has to generate a new data set containing examples of how to pick bananas, and retrain the model. This is done by 3D-scanning some representative bananas, tweaking the physics parameters in simulation and adjusting the parameters of the evolutionary algorithm generating the labelled grasps. This is potentially done in only a few days. With some unlabelled examples of real point clouds of bananas for validation, one can quickly make an educated guess to whether or not the approach is viable without hampering a potentially active production line[2]. Factories and productions subject to varying tasks can potentially keep many differently trained models for the same physical robot, switching models based on what processing is done, e.g., on a daily basis. While this approach is limited in terms of applicability, we hope that it, in the short-term, might enable increased automation in domains where costly development cycles cannot be justified.

Relying on simulation has some drawbacks. There might be applications where simulating realistic physics or realistic looking sensory data is difficult. Additionally, if the robot needs to work in a dynamic environment subject to changing operating conditions within the work-space, this might need to be explicitly programmed in the simulator and randomized over during data set generation, which can be tedious. Further, it can be difficult in some cases to define a fitness function

[2]Do to restrictions during the pandemic, this is actually the way that the fish-picking system was trained, with the development being done almost entirely without access to the robot.

for the evolutionary algorithm that ensures good grasps in all scenarios for large domains. However, in many cases, setting up an environment with some parameters to randomize over and tweaking the fitness function is a substantially faster way of gathering large data sets, than through acquisition in real life and subsequent manual annotation. And as the simulator is applied in different domains it will mature, and contain more and more relevant functionality for following automation tasks. Further, labelling of grasps with an evolutionary algorithm, as opposed to, e.g., models based on grasp quality, provides a flexible labelling framework as the fitness function can take advantage of the entire known state of the simulator. By designing a fitness function relative to the geometry of a 3D-model and adding simple conditional statements, data sets can be created where different objects are grasped in different ways and some objects are grasped before others. This can be very useful in many automation tasks, as objects often have to be grasped in a certain way fitting of a subsequent task e.g. in assembly operations. However, the current simulator has limitations which makes it difficult to use for more complex manipulation tasks. Specifically, the physics simulation is not accurate enough, e.g., to model the intricate interactions between the soft gripper and the objects and use this as a basis for evaluating grasps. However, others, like NVIDIA [75], are developing general purpose simulators for robotics, and it is quite safe to assume that the number of domains where synthetic data generation can be applicable will continue to grow in the future.

In the long-term, a more elegant solution to repurposing of robots would be through in-situ learning from demonstration. This would remove the need for continued development of simulators and it would enable users to repurpose robots without involving robotic experts. In Sec. 3.2.1, the proposed visual processing system demonstrated the capability of learning a simple visual-servoing task after less than 3 hours of "demonstrations" in simulation (these could just as easily have been provided through teleoperation). However, it was not able to robustly execute the second, more complex task, in Sec. 3.2.2, after training on real demonstrated trajectories and we hypothesized that this was largely caused by an under-explored state-space. Despite the performance in the second experiment, we find these results encouraging. By working to some degree with direct behavioural cloning from small data sets, we hypothesize that the visual processing system can be compatible with imitation learning. We believe that it can work substantially better with larger data sets and more sophisticated training. Learning from demonstration and offline reinforcement learning are active areas of research where rapid advances are made. We believe that the advances made in those fields can be applied for learning of better robotic policies with a RAM based visual processing system.

During the research period we have developed the necessary infrastructure for demonstration and gathering of data and future research will focus more on the above mentioned issues with learning from demonstration. An interesting avenue for future research which might enable subsequent learning from demonstration is pre-training of the model on a multitude of tasks in simulation. Intuitively, if a good attention mechanism coupled with a robust mapping from observations to actions is learnt in simulation, then the learning from demonstration part could largely be used only to infer the goal of the task. This might require some architectural changes to the model, and this will be the focus of future work.

Chapter 5

Conclusions

The larger goal of this work has been to contribute towards the realization of flexible robotic solutions capable of operating in unstructured and cluttered environments. To this end we defined the following research objectives:

Research objective 1: Visual processing for generic robotic applications

Research objective 2: Robust task specific grasping

Research objective 3: Repurposable robotic solutions

To address these research objectives, we considered real-world automation cases in domains subject to noise and clutter. The main test-bed has been the challenging task of bin-picking of fish. This is a task subject to noisy depth measurements and large variation in the appearance of objects where precise grasps have to be predicted in order to avoid collisions and to successfully pick the objects. We consider success in this domain as a proof-of-concept of the proposed system.

Through our work on grasping, we find that sliding window 3D-CNNs can be used to detect 6 DoF grasps directly from 3D occupancy grids. We find that they are capable of learning collision-free grasps, while being robust to the types of noise typically seen in 3D-data. Further, by processing small 3D regions independently with restricted receptive fields, we find that we can sufficiently cover the input space for the CNNs with synthetically generated data sets for the model to transfer to the real world.

We leverage the strengths of 3D-CNNs and generalize the visual processing system by employing a RAM [31], that processes large point clouds by sequentially deploying a 3D-CNN at select locations in the volume while accumulating the information from each location in the state of an LSTM.

When applied to the fish-picking task, the system achieves a grasp success rate of 95 % after training solely on synthetic data and it is able to successfully correct its' own mistakes by trying again. Further, it demonstrates robustness to out-of-distribution-states by successfully emptying boxes with twice as many fish at test-time compared to those seen during training. The attention mechanism seems to be effectively filtering out distractions, while the 3D-CNN encoder enables prediction of precise, collision-free grasps in the face of noise and clutter.

In line with research objective 3, none of the design decisions made during development of the system were based of the particular domain used for testing, i.e. bin-picking of fish. Therefore, the system is easily repurposable for similar tasks by retraining the model with a different data set. A generic pipeline for generating such data sets for robotic grasping has been developed, based on 3D-scanning of objects with a subsequent evolutionary algorithm for labeling them with grasps.

The proposed visual processing system is designed so that it is capable of reducing arbitrarily sized volumes to single outputs with a processing speed of 15 Hz. As such, the system can be used to learn a state-to-action mapping for use in closed-loop control settings, enabling more complex robotic actions and sequences of actions. Through preliminary experiments on simple visual servoing tasks, the system has proven capable of inferring relative distances between objects and reasoning about simple obstacles. When trained to use visual servoing to position the gripper over a target block, the model is able to do this, albeit with somewhat jerky motion. It achieves this by continuously predicting end-effector velocities directly from point clouds with more than 6 million points. We view this as evidence for the feasibility of the method.

In the short-term, we hope that the proposed approach to grasping might enable increased automation in domains that were previously less suitable for it. Further, we believe that the RAMs unique capability of extracting precise features from large work spaces in real time makes it ideal for robotic solutions in general. In the long-term, we think that it can be used for closed-loop robotic control from point clouds by providing small descriptive state-spaces for robotic policies to operate on.

In conclusion, we find that 3D-CNNs can be used to find collision-free grasps in the presence of noisy depth measurements. By combining a 3D-CNN with a recurrent attention model, we find that precise grasps can be found in a difficult case subject to clutter and noisy input data, by learning to attend to the relevant parts of the volume. The resulting system can extract precise features from arbitrarily large volumes with high processing speeds and preliminary experiments indicate that the system can be suitable for closed-loop control.

Chapter 6

Publications

This chapter contains reprints of the publications included in this thesis.

A Grasping Virtual Fish: A Step Towards Robotic Deep Learning from Demonstration in Virtual Reality

Published by J. S. Dyrstad, J. R. Mathiassen. "Grasping virtual fish: A step towards robotic deep learning from demonstration in virtual reality". In: 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO) (2017), pp. 1181–1187

Bibliography entry [2].

Grasping Virtual Fish: A Step Towards Robotic Deep Learning from Demonstration in Virtual Reality

Jonatan S. Dyrstad[1] and John Reidar Mathiassen[2]

Abstract— **We present an approach to robotic deep learning from demonstration in virtual reality, which combines a deep 3D convolutional neural network, for grasp detection from 3D point clouds, with domain randomization to generate a large training data set. The use of virtual reality (VR) enables robot learning from demonstration in a virtual environment. In this environment, a human user can easily and intuitively demonstrate examples of how to grasp an object, such as a fish. From a few dozen of these demonstrations, we use domain randomization to generate a large synthetic training data set consisting of 76 000 example grasps of fish. After training the network using this data set, the network is able to guide a gripper to grasp virtual fish with good success rates. Our domain randomization approach is a step towards an efficient way to perform robotic deep learning from demonstration in virtual reality.**

I. INTRODUCTION

In robotics, robust grasping and manipulation of objects is still a challenging task to automate, in particular for biological and deformable objects such as fish. Inspired by the ability of humans to perform such tasks, we investigate how to efficiently transfer the knowledge of a human to the robot, via a combination of 1) a virtual reality (VR) interface for demonstrating the task, 2) domain randomization over components of the task to generate a large data set, and 3) deep learning on this large data set. Our hypothesis is that through our approach, VR can serve as a efficient medium for demonstrating complex tasks to robots. A first step towards testing this hypothesis is presented in this paper. We describe an approach for deep learning from demonstration in VR, where a human can easily teach a robot how to grasp fish. Grasping of fish from a box is an example of a challenging grasping task involving a cluttered scene of multiple highly deformable objects. In today's fishing industry, many simple and repetitive tasks are still performed by human workers due to difficulties in automating handling of the fish. For tasks with a large throughput, there exist processing machinery that moves fish and handles them automatically. Compared to a robotic solution, these systems are neither compact nor adaptive. The fish picking task is therefore ideal for demonstrating the capabilities of our approach applied to relevant industrial problems.

An essential aspect of learning from demonstration is the system in which the human teacher demonstrates the task. This system should be intuitive and easy to use and additionally, the teacher should not have to demonstrate the task many times, which is often necessary when training deep learning models. Therefore, we have developed a system where the teacher's actions are not used directly to train the robot, but rather used to generate large amounts of synthetic training data through domain randomization over relevant components of the grasping task. This enables us to train a deep neural network (DNN) for grasp detection, from scratch, using only a few dozen manually-demonstrated example grasps. In this paper we use our approach to robotic deep learning from demonstration to create a grasp detector with success rates that are sufficient to enable improvements in future research directions. These directions include using reinforcement learning algorithms to rapidly improve the performance of the system in VR and later in a real world scenario.

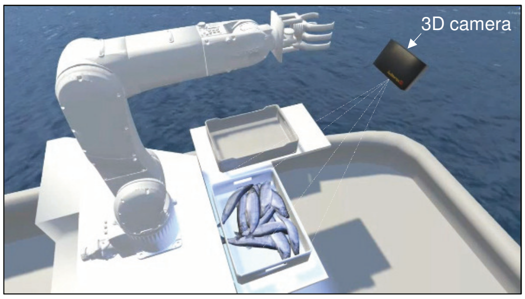

Fig. 1. The virtual robot is tasked with grasping virtual fish from a box and placing them in another box. The sensory input to the robot consists of a virtual 3D camera that generates depth images of the fish in the box.

In this paper, a grasp is defined by the 6 degrees of freedom (DOF) needed to define a robotic gripper in 3D-space. This is a simplification and successful grasps in a real world situation might require good path planning and a well regulated and agile gripper, capable of holding objects with different weights, textures and sizes.

Both training and testing of our system has been done on synthetic data, since this enables efficient development and testing of our approach. Future work will focus on the transferability of the features learned on the synthetic data to data from the real world.

Robot grasping is a field with a lot of research being done. There are several methodologies applicable to robot grasping based on visual input, such as learning from demonstration

[1]Jonatan S. Dyrstad is with the Processing department, SINTEF Ocean AS, Brattrkaia 17c, 7010 Trondheim, Norway `jonatan.dyrstad@sintef.no`

[2]John Reidar Mathiassen is with the Processing department, SINTEF Ocean AS, Brattrkaia 17c, 7010 Trondheim, Norway `john.reidar.mathiassen@sintef.no`

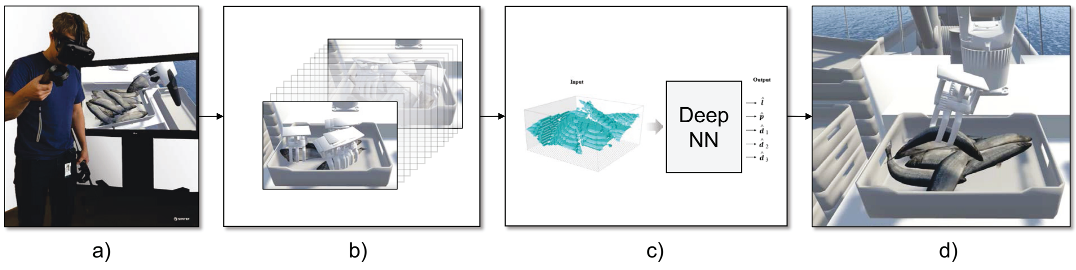

Fig. 2. The presented approach to deep learning from demonstration in VR, where a) the user provides a few example grasps by grasping fish from the box, b) domain randomization is used to generate a large data set based on the few example grasps, c) the deep neural network is trained on the large data set, and d) the trained deep neural network controls a gripper to grasp virtual fish.

(LfD) and reinforcement learning, based on real or virtual data. Learning from demonstration is a generic approach [1], [2], [3] in which a human or virtual teacher demonstrates a task, e.g. a grasping action specified by a pose and gripper configuration, with a corresponding state space consisting of visual information. Based on a set of demonstrations, a learning algorithm, such as support vector machines (SVM) and regularized regression [4] or artificial neural networks [5], [6], [7], [8], [9], [10], learns the mapping between the visual state and the grasping action. In reinforcement learning (RL) there is no teacher to demonstrate how to perform the task, instead there is a reinforcement signal provided to the learning algorithm. LfD has the advantage of efficiently discovering state-action mappings that work reasonably, with the disadvantage that this may require a large data set of demonstration examples. Contrary to this, RL, e.g. using artificial neural networks [11], has the disadvantages of slow learning speed and difficulty of accurately defining a suitable objective function for more complex tasks. The advantages of RL are that they do not require a human teacher, and they are capable of exploring and learning how to perform potentially beyond the capabilities of a teacher.

Synthetic data generation is a well-known and successful approach to training deep learning systems [13], [14], prior to applying them to real-world data. In particular, an approach called domain randomization has been shown to enable robust training on simulated data that directly works in the real-world without additional training [15]. Domain randomization involves randomizing over the relevant components of a task. A novelty of our approach for domain randomization is that it begins with an intuitive virtual reality interface within which a human teacher can easily provide *a low or moderate number* of demonstration examples. These examples are then used in a domain randomization process over the camera viewpoints, the physical interactions of the fish and the box, and the previously demonstrated intent of the teacher, to generate the large data sets required for deep learning.

To the best of our knowledge, we present the first work where a deep learning algorithm has solved robotic grasping using a 3DCNN, which has previously been successfully applied to 3D shape recognition [4], [12]. Our main motivation in working with voxel grid representations of point clouds, and 3DCNNs to analyze them, is to develop the foundation for deep learning in robotics applications that are camera- and viewpoint-agnostic. Hence the 3DCNN can work in a single voxel image including 3D information obtained from multiple depth images from multiple cameras and/or viewpoints. Multiple viewpoints and cameras can provide a more complete coverage of a scene. A single-view depth image will have occluded regions, and moving the depth camera to one or more other locations will provide a better view of the occluded regions. Fusing these two views into a single point cloud will provide a more complete view. The advantage of the 3DCNN approach is thus that can be invariant or agnostic to the number of views or the number of cameras that generate the 3D data, and it can work on the complete view, as long as domain randomization is done over the types of views that are possible.

Our main contributions are; 1) a novel domain randomization approach and its application to learning from demonstration in virtual reality, 2) using a deep 3D convolutional neural network (3DCNN) to detect potential grasps and estimate their pose. We develop and test our contributions in a virtual reality environment in this paper, as preparation for future research focusing on development and test in the real world.

II. METHOD

We use a deep 3DCNN to estimate grasps from a point cloud. We propose the use of VR to generate large amounts of synthetic training data in order to be able to train a deep learning model with many parameters. An illustration of our approach is shown in Fig. 2.

A. User interface for demonstrating the task

A virtual environment was created where a user, using a virtual reality head mounted display (HMD) and tracked hand controllers, can enter and demonstrate the task for the robot as shown in Fig. 2a. The user has a controller in his hand, which in VR appears to him as a gripper, similar to the end effector on the robot. The environment was created with the Unity game engine and the VR-equipment used the HTC-Vive head-mounted display and hand-held motion controllers.

Fish are instantiated in mid-air and dropped using simulated physics that model the deformation and friction characteristics of the pelagic fish species herring (*Clupea harengus*). This ensures that the fish land in natural poses in a fish box placed in front of the robot. The user's task is to grasp the fish and move the fish from this box to another box, using the gripper in his hand (see Fig. 2a). In this way, the user gets the impression that he is *showing* the robot how to perform the task, in an easy and intuitive way. Since the user is told to grasp the fish in a way that enables him to pick it up and place it in a second box, he will naturally use a grasp that is suited for that task. If e.g. the task had been a different one, such as placing the fish in a narrow hole, the user would probably grasp the fish differently. Hence, this is an effective way of getting the user to demonstrate grasps that are suitable for a given task. For each fish grasped by the user, the grasp is logged with regards to its position and orientation relative to the fish. An example of these logged grasps can be seen in Fig. 4. This is the demonstration part of our learning from demonstration (LfD) approach. In traditional LfD, a large number of demonstration examples are required. The number of required examples scales with the number of parameters in the model that is being taught. Models with very many parameters, such as large neural networks, may require on the order of tens of thousands of demonstrations in order to adequately learn the task without overfitting to the training data. For a user this is clearly too much work, and an alternative approach is needed to generate a sufficiently large and realistic training data set.

B. Generating Large Amounts of Synthetic Data by Domain Randomization

Based on the logged grasps, we propose to use domain randomization that includes information on the user's grasp intent, as a method for generating a large training data set from a few demonstrations, with no further human supervision. As in the previous section, the fish are instantiated and dropped into the fish box, as shown in Fig. 3a. By randomizing over the number of fish, and the position and orientation of each fish before dropping them into the box, this provides domain randomization over the possible ways in which fish can realistically be positioned relative to each other in a box.

Instead of the user demonstrating the grasp for each randomly generated box of fish, we instantiate all of the previously logged grasps onto each of the fish in the box, as shown in Fig. 3b. However, not all of the grasps are valid for all of the fish, given their current pose and position in the box (i.e. closeness to the walls etc.). Therefore, for every fish, all of the logged grasps are automatically checked using collision heuristics to see if they collide with the environment or with the other objects in the scene in any way. The ones that do not are kept and the rest are discarded, resulting in a set of plausible grasps as shown in Fig. 3c. This three-step approach provides domain randomization over the possible ways a user would *probably* grasp the fish, based on what know from the previously logged grasps.

An orthographically projected depth image is rendered of the entire fish box and the list of valid grip vectors are recorded along with the depth image (an example is shown in Fig. 7). The field of view and resolution of the virtual 3D camera is such that each pixel in the orthographically projected depth image can be read as an xyz-coordinate in millimeters given in camera coordinates (with an offset of $\frac{imagewidth/height}{2}$ in the xy-direction). Current 3D cameras, such as the Microsoft Kinect and Intel RealSense, work by projecting a light pattern from a projector that is offset from the actual camera. Because some of the scene is visible to the camera but occluded to the projector, the result is *depth shadows*, areas in the depth image with unknown depth values. This effect is simulated with the virtual 3D camera as well. The depth data we generate in simulation can therefore be thought of as coming from a perfectly calibrated real 3D camera. To provide robust learning, we randomize the position and orientation of the virtual 3D camera as well, thus creating variations in the amount of missing data in the depth images due to the occlusion of the projector illumination. This is our final component of domain randomization.

C. Neural Networks

The depth images are projected into a voxel grid, and we use a 3DCNN to estimate grasps from a volume in the voxel grid, and split the problem up into three sub-problems

- Detecting possible grasp locations
- Finding the precise grip point
- Finding the orientation of the gripper

The architecture of the network is shown in Fig. 5. For a volume of size $50 \times 50 \times 25$, the output is a vector

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{l} & \hat{\mathbf{p}} & \hat{\mathbf{d_1}} & \hat{\mathbf{d_2}} & \hat{\mathbf{d_3}} \end{bmatrix}, \tag{1}$$

where $\hat{l} \in [0,1]$ estimates the probability of the input volume containing a valid grasp, $\hat{\mathbf{p}}$ estimates the precise position of a grasp within the input volume and $\hat{\mathbf{d_1}}$, $\hat{\mathbf{d_2}}$ and $\hat{\mathbf{d_3}}$ estimate the orientation of the grasp.

The network is *fully convolutional* and has sliding dense layers, meaning that the dimensions of the output from the network is dependent on the dimensions of the input volume. For larger inputs, the result is a grasp detector, rather than a classifier, capable of detecting multiple grasps within the input volume.

The estimations for the three sub problems are output from the same network and trained jointly because of the high dependence between the different objectives. The total cost for training example i is given by

$$J^{(i)} = \alpha J_C^{(i)} + l^{(i)}(\beta J_O^{(i)} + \gamma J_P^{(i)}), \tag{2}$$

where $J_C^{(i)}$ is the classification cost, $J_O^{(i)}$ the orientation cost and $J_P^{(i)}$ the position cost, $l^{(i)}$ is the true label for training example i and the parameters α, β and γ are simple weighing factors used to prioritize the relative importance of the

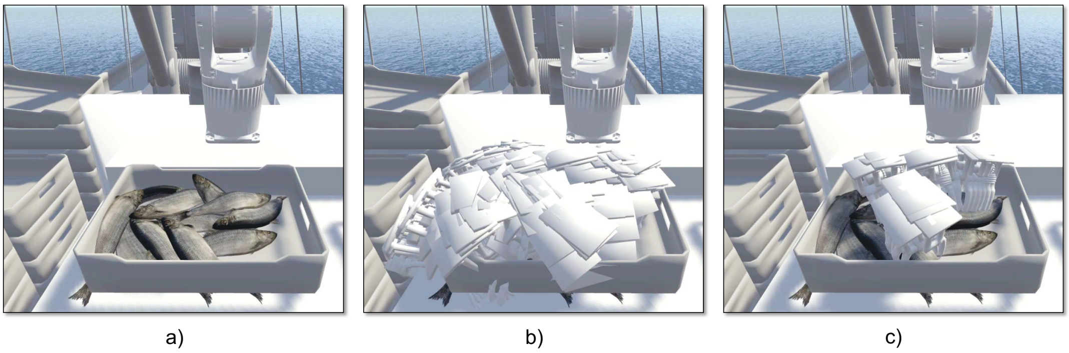

Fig. 3. The domain randomization approach for generating a large data set, by a) dropping a random number of fish in the box, using realistic fish physics, b) placing each of the logged grasps onto each fish, c) pruning the grasps based on collision heuristics.

different sub-problems. Note that for false examples, no updates are done to the position and orientation estimators. For classification of valid grasps, the binary cross entropy function

$$J_C^{(i)} = -l^{(i)}\log(\hat{l}^{(i)}) + (1-l^{(i)})\log(1-\hat{l}^{(i)}) \tag{3}$$

is used, and for regression on the precise grasp point $\mathbf{p}^{(i)}$ within the given volume we use the squared error cost function

$$J_P = \frac{1}{2}||\hat{\mathbf{p}}^{(i)} - \mathbf{p}^{(i)})||^2. \tag{4}$$

The orientation of the gripper is defined unambiguously with two three dimensional vectors, each describing a direction in 3D-space (see Fig. 6). However, for an object like a fish, which is almost a mirror image of itself along one axis, there are always two correct answers to any situation (i.e. an "overhand" and an "underhand" grip). To avoid any contradicting labels in the data set we assume a symmetrical gripper and can therefore simply reverse the sign of the vectors on one half of the data manifold. This leads to a discontinuity which we solve by having two estimates of one vector, each with the discontinuity at different places in the data manifold. In this way, one estimate becomes increasingly reliable as the other goes towards its uncertain region. Empirical observations of the training examples in the data set shows that one of the target vectors defining the grip orientation ($\mathbf{v_2}$ in Fig. 6) draws out a diffuse disk in 3D-space (as opposed to a sphere). This lets us get away with two estimates of the vector to avoid discontinuities ($\hat{\mathbf{d_1}}$ and $\hat{\mathbf{d_2}}$ in (1)). The variance in the other orientation vector ($\mathbf{v_1}$ in Fig. 6) is small (because most of the grasps are pointing up in camera coordinates) and therefore we only use one estimate for this vector ($\hat{\mathbf{d_3}}$ in (1)). The total orientation cost for the $N = 3$ orientation vectors is given by

$$J_O^{(i)} = \sum_{k=1}^{N} 1 - \frac{dot(\hat{\mathbf{d_k}^{(i)}}, \mathbf{d_k}^{(i)})}{\sqrt{dot(\hat{\mathbf{d_k}^{(i)}}, \hat{\mathbf{d_k}^{(i)}})dot(\mathbf{d_k}^{(i)}, \mathbf{d_k}^{(i)})}}, \tag{5}$$

where, $\mathbf{d_k}^{(i)}$ and $\hat{\mathbf{d_k}^{(i)}}$ for $k \in 1,2,3$ respectively denote the true and estimated orientation vectors for training example i.

D. Preparing Data For Training

During training, the input to the network is a volume of size $50 \times 50 \times 25$ and the classification label $l^{(i)}$ is either 1 or 0 (i.e. the volume does or does not contain a valid grasp). Training examples with true labels are simply created by cropping volumes of the synthetically created depth images centered around one of the valid grasp points for that image. The crop is offset randomly from the middle by some amount in order to create training vectors for the grip point estimator as well. Generation of false training examples is more problematic. The virtual environment outputs a depth image and some, but not all conceivable grip vectors for the given image. Thus, there is no way of knowing which parts of the image that are guaranteed not to contain a valid grasp. In our experiments we generate false training examples (i.e. areas with a low probability of containing a grasp), simply by cropping random parts of the image and labelling them as false examples. Because the volumes that contain valid grasp are vastly outnumbered by the volumes that do not,

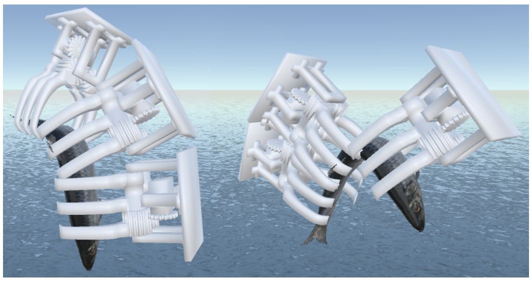

Fig. 4. The logged grasps after demonstration of three grasps in virtual reality. As the fish bends and twists, the grips follow, making them valid for the fish regardless of pose.

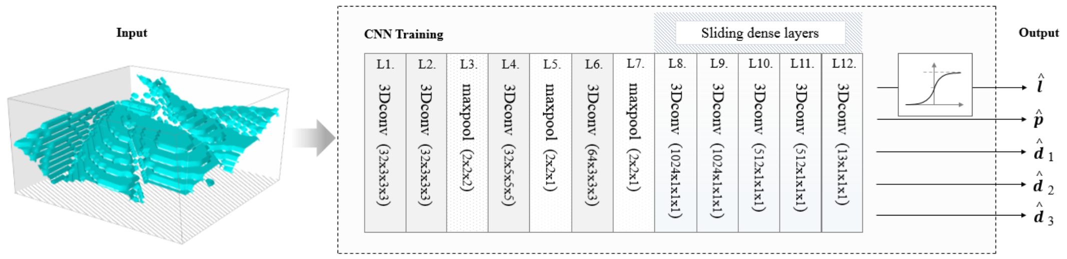

Fig. 5. The architecture of the 3DCNN consists of stacked convolutional and max pooling layers. The dense layers are swapped for $1 \times 1 \times 1$ convolutions to enable inputs of varying sizes. The activation functions for the feature extraction layers are rectified linear units, and in the top layer, $\hat{l}$ has a sigmoid activation function and the rest have linear activations.

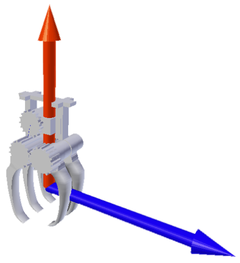

Fig. 6. The orientation of the gripper is defined by two vectors $\mathbf{v_1}$ (red) and $\mathbf{v_2}$ (blue).

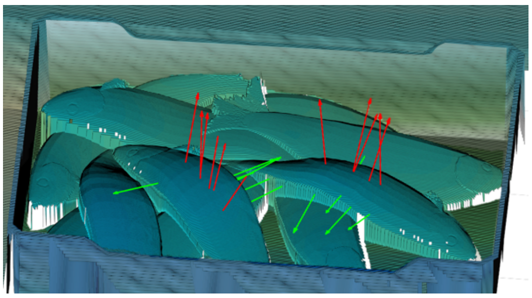

Fig. 7. An example of a synthetically generated depth image with the labeled grip vectors overlaid.

the result is a false-data set with mostly true, but also some false, negatives.

III. RESULTS

In our experiments, 43 grasps were shown in VR. With these grasps, 5,000 images were rendered of fish boxes containing between one and fifteen fish. Randomness was introduced in the rotation and position of the depth camera when the images were rendered. From these depth images, 76,000 training examples were cropped where $2/3$ of the data set did not contain a grasp. The input to the network during testing was a volume of size $197 \times 197 \times 50$ and the the depth images were decimated so that the resolution was 3.6 mm per voxel.

Testing was done in real time with the trained model attempting to pick fish in the virtual environment as shown in Fig. 9. Because no paths are output from the network, the gripper simply approached the estimated grasp point from the direction given by $\mathbf{v_1}$ in Fig. 6. If the gripper could not reach the grip point because it hit the environment, or if it failed to capture and hold the fish when the gripper was closed, the grasp was considered unsuccessful. If several grasps were detected in the image, the one with the largest detection certainty, $\hat{l}$, was chosen.

The neural network was created in Python with the Theano[1] and Lasagne[2] libraries and training was done on an NVIDIA Titan X GPU.

A. Generated Training Examples

The logic for generation of large amounts of labelled training data with only a few shown grasps works well. Many examples were inspected visually and all of the inspected grasps in the training set look plausible. An example of a rendered image with labelled grasps is shown in Fig. 7. In this case, and in general, the labeled grips are good, but the virtual environment only outputs a subset of all possible grips in the scene.

The generated depth images with added noise and simulated depth shadows still looks like stylized and ideal depth data. Visual comparison (not included in this paper) to real data from the Microsoft Kinect and Intel RealSense depth cameras, reveals that specular effects are what separates the real from the synthetic data the most. The large difference in albedo on different parts of the fish leads to areas in the real depth images with undefined values which are not present in the synthetic ones.

B. Testing the Trained Neural Network

The neural network was trained with stochastic batch gradient decent with momentum, and cross-validated on a

[1]http://deeplearning.net/software/theano/
[2]https://lasagne.readthedocs.io/en/latest/

testing set with respect to minimizing the objective function. This function is however not an exact representation of grasping success, and our primary test involves testing the trained network in virtual grasping scenarios that were not seen during training of the neural network.

The trained network was used to attempt 100 grasps with 1, 5, 10 and 15 fish in the box and the results are shown in Tab. I. These results show that the performance is similar regardless of the number of fish in the box. Hence the neural network handles scenarios with many fish, in overlapping and challenging conditions, almost equally well as with only a single fish in the box.

Observations of the grasping attempts during live testing reveals that most unsuccessful grasps are a result of

- The gripper hitting the environment. For the most part with the top of the gripper hitting the walls and sometimes with the tip of the fingers hitting the bottom of the box.
- The fish sliding out of the gripper because the fish was picked to close to the head or tail.
- The chosen grip was a valid grip, but not the best candidate in the scene. Meaning, the local grip area was good, but other fish should have been picked first. E.g. the gripper sometimes push other fish around on its way to the grip point changing the state of the box before arrival.

For almost all fish boxes there are a lot of false positives. Even with no fish in the box, several valid grasps are often detected. However, when fish is present, the most certain grasps are often good. A typical result for a fish box with the 10 most certain grasps overlaid is shown in Fig. 8.

The detector, grasp point and grasp orientation estimates are good, and surprisingly good grasps can be found for almost vertical fish, frozen in mid air, even if this has never been seen in the data set used for training the neural network.

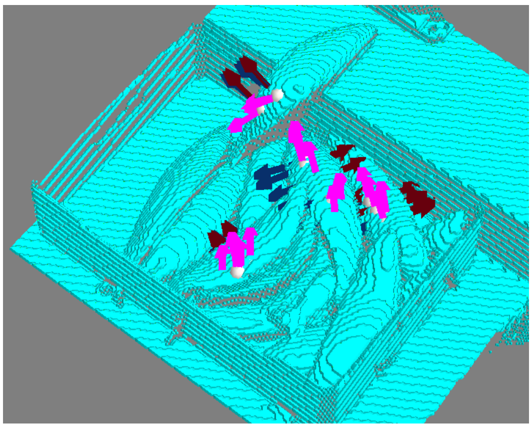
Fig. 8. The decimated volume input to the network, with the 10 most certain grasps overlaid. The pink arrow corresponds to $\mathbf{v_1}$ in Fig. 6, and the blue and red ones are the two estimates of $\mathbf{v_2}$ in Fig. 6.

TABLE I
GRASP SUCCESSES IN VR

No. of fish in the box	1	5	10	15
Success rate	70 %	74 %	61 %	71 %

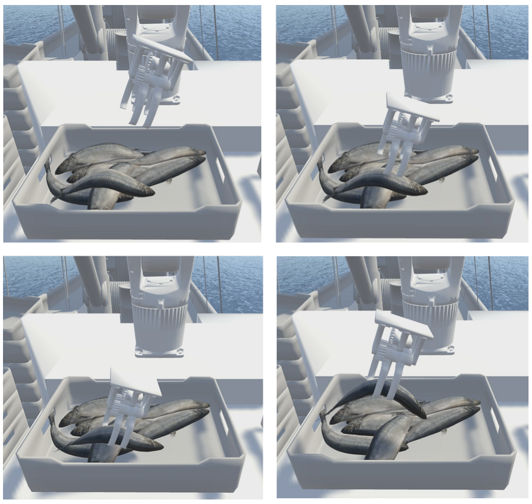
Fig. 9. The trained 3DCNN grasping a fish successfully in the virtual environment.

IV. DISCUSSION AND FUTURE WORK

The results of our work, suggest that our approach to domain randomization in virtual reality works intuitively and well. Based on only a few demonstration examples, a sufficiently large and diverse data set was generated that was capable of training a large 3D convolutional neural network. This network was tested on previously unseen scenarios, with success rates on the order of 70 %. This is good, but not sufficient for a working system. However, the results are good enough for us to proceed with implementation of this system on a real-world robot, and for refining the neural network using reinforcement learning in virtual reality as well as in the real world. One may argue that demonstrations in virtual reality do not prove the validity of our approach. Contrarily, we may argue that a methodology such as deep learning already has proven itself capable in transfer learning to the real world, as long as domain randomization is realistic [15]. As such, the domain randomization approach and neural network presented in this paper lay the groundwork for future implementation in a real-world scenario.

Our hypothesis from the onset is that through our approach, VR can serve as a efficient medium for demonstrating complex tasks to robots. A first step towards testing this hypothesis was presented in this paper, and the results are promising enough to maintain our hypothesis. However, more work is needed to develop this further. Future work will focus on transfer learning to apply what is learned in

VR to a real-world setting of picking fish from a box using an industrial robot. In addition to what we have presented in this paper, this may require better modelling of the depth images of fish. Fish are specularly reflective objects, and therefore affect the quality of depth images acquired with real 3D cameras such as the Microsoft Kinect or Intel RealSense. Accurate modeling of these reflective properties will enable more accurate training in virtual reality.

Beyond our application to grasping fish, we will also expand the domain randomization methodology and neural network architecture presented to the more generic problem of grasp detection for multiple types of objects and also to learning from demonstration of more complex task sequences.

ACKNOWLEDGMENTS

This project was funded by the Research Council of Norway, through its basic funding scheme, and in part from Research Council of Norway Grant No. 262900.

REFERENCES

[1] R. Pelossof, A. Miller, P. Allen and T. Jebara, "An SVM learning approach to robotic grasping," In Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA), 2004, Vol. 4, pp. 3512-3518.

[2] B. D. Argall, S. Chernova, M. Veloso and B. Browning, "A survey of robot learning from demonstration," Robotics and autonomous systems, 57(5), 469-483. May 2009.

[3] S. Choi, K. Lee and S. Oh, "Robust learning from demonstration using leveraged Gaussian processes and sparse-constrained optimization," In 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 470-475.

[4] S. Song and J. Xiao, "Sliding shapes for 3d object detection in depth images,". In European Conference on Computer Vision, 2014, pp. 634-651. Springer International Publishing.

[5] N. Rezzoug and P. Gorce, "Robotic grasping: A generic neural network architecture", 2006, INTECH Open Access Publisher.

[6] A. Saxena, J. Driemeyer and A. Y. Ng, "Robotic grasping of novel objects using vision". The International Journal of Robotics Research, 2008, 27(2), 157-173.

[7] Y. Jiang, S. Moseson and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," In 2011 IEE International Conference on Robotics and Automation (ICRA), 2011, pp. 3304-3311.

[8] P. C. Huang, J. Lehman, A. K. Mok, R. Miikkulainen and L. Sentis, "Grasping novel objects with a dexterous robotic hand through neuroevolution," In 2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA), 2014, pp. 1-8.

[9] I. Lenz, H. Lee and A. Saxena, "Deep learning for detecting robotic grasps," The International Journal of Robotics Research, 2015, 34(4-5), 705-724.

[10] J. Dyrstad, "Deep learning in virtual reality for grasp detection", Master's thesis, University of Stavanger, 2016.

[11] S. Levine, P. Pastor, A. Krizhevsky and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection", arXiv preprint arXiv:1603.02199. (2016)

[12] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1912-1920.

[13] J. Shotton et al. "Real-Time Human Pose Recognition in Parts from a Single Depth Image". In CVPR. IEEE, (2011)

[14] E. Wood et al. "Rendering of Eyes for Eye-Shape Registration and Gaze Estimation". In CoRR abs/1505.05916 (2015)

[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". arXiv preprint arXiv:1703.06907. (2017)

B Bin Picking of Reflective Steel Parts using a Dual-Resolution Convolutional Neural Network Trained in a Simulated Environment

Published by Jonatan S. Dyrstad et al. 'Bin Picking of Reflective Steel Parts Using a Dual- Resolution Convolutional Neural Network Trained in a Simulated Environment'. In: 2018 IEEE International Conference on Robotics and Biomimet- ics (ROBIO) (2018), pp. 530–537.

Bibliography entry [3].

Bin Picking of Reflective Steel Parts using a Dual-Resolution Convolutional Neural Network Trained in a Simulated Environment

Jonatan S. Dyrstad[1,2], Marianne Bakken[3], Esten I. Grøtli[3], Helene Schulerud[3] and John Reidar Mathiassen[1,*]

***Abstract*— We consider the case of robotic bin picking of reflective steel parts, using a structured light 3D camera as a depth imaging device. In this paper, we present a new method for bin picking, based on a dual-resolution convolutional neural network trained entirely in a simulated environment. The dual-resolution network consists of a high resolution focus network to compute the grasp and a low resolution context network to avoid local collisions.The reflectivity of the steel parts result in depth images that have a lot of missing data. To take this into account, training of the neural net is done by domain randomization on a large set of synthetic depth images that simulate the missing data problems of the real depth images. We demonstrate both in simulation and in a real-world test that our method can perform bin picking of reflective steel parts.**

I. INTRODUCTION

Bin picking is the problem of grasping objects randomly placed in a bin. This is a problem that often occurs in industrial settings where objects come out of a production line packaged in bulk, without isolating individual objects, and where the objects are transported to a second production line that subsequently must isolate and process these objects individually. Due to the importance and relevance of the problem, bin picking has been well studied [19], [24]–[26] in the literature. Challenges in bin picking arise when seeking to develop a bin picking algorithm that can be automatically customized for specific objects, and when these objects are very reflective. We present a method for bin picking that addresses these two challenges.

The input to the grasp detection network is a depth image and the output is a set of possible 3D grasps (e.g. 5-DOF or 6-DOF gripper poses). The use of a dual-resolution network enables both high accuracy in a focus region of interest for placing the grasp and estimating the grasp pose, as well as enabling a low-resolution context awareness that e.g. ensures that the grasps do not collide with other objects in cluttered scenes. Fig. 1 shows the robot and the Zivid[1] 3D camera used in our experiment and the steel parts in our bin picking case.

We first evaluate our approach on simulated test data and then demonstrate it in an exemplary real-world scenario involving bin picking of steel parts using a robot with 5-DOF placement of a vacuum suction gripper. Training of the neural network is done entirely on synthetic depth images generated by domain randomization in a simulated environment. This

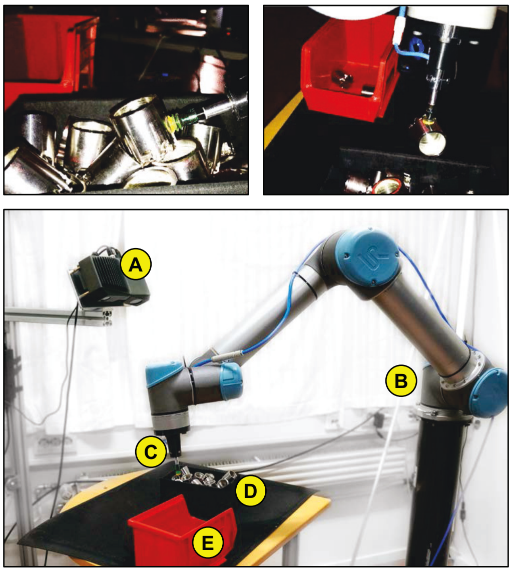

Fig. 1. Grasping steel parts with a suction gripper (top two images). An overview of the bin picking setup, including a Zivid 3D camera (A), a UR5 robot (B), a pneumatic suction gripper (C), a bin of reflective steel parts (D) and a bin (E) for placing the steel parts after picking.

approach is used to generate simulated data for training of the neural network [23] that will work well in the real world.

Our main contributions are:

- A dual-resolution convolutional neural network for end-to-end 5-DOF grasp estimation from depth images, which uses a high resolution focus network to compute the grasp and a low resolution context network to avoid local collisions.
- A simulation environment using domain randomization to automatically generate large data sets for training the neural network, given known reflectivity and geometric properties of objects in the bin-picking scenario.
- Demonstrating that the dual-resolution neural network can be trained entirely in a simulated environment on specific objects, and be deployed in a robot that performs bin picking of these objects in the real world.

Although our experiments are done using a suction gripper on smooth-surfaced metal objects, the methodology of our contributions should be applicable also for other types of

*Corresponding author, `John.Reidar.Mathiassen@sintef.no`
[1]SINTEF Ocean AS, Trondheim, Norway
[2]NTNU, Department of Engineering Cybernetics, Trondheim, Norway
[3]SINTEF Digital, Oslo, Norway

[1]http://www.zividlabs.com/

objects and grippers - as long as these can be simulated. The rest of the paper is organized as follows: We discuss related work in Section II. We present our grasping method in Section III. We then describe our experimental setup and results in Section IV. The conclusion and suggestions for future work are in Section V.

II. RELATED WORK

Detecting robot grasps from 3D or depth images is an active research field, both in terms of using geometry-based methods [8]–[12], [14] and deep learning [1]–[5], [7], [13], [20], [21]. Geometry-based methods attempt to match 3D CAD models to point clouds to compute the object pose [16]. Some research suggests that primates and humans have separate neural pathways for object recognition and grasping [17], and the object detection and pose estimation has often been treated as an isolated problem separated from grasp selection in the bin-picking literature. Geometry-based methods have been well explored for pose estimation in bin-picking, such as Abbeloos et al [24], that uses the popular point pair feature approach, first presented by Drost et al. [27]. Buchhilz et al. [26] suggests a two-stage approach where the full object pose is estimated after grasping based on inertial features. On the other hand, Ellekilde et al. [25] focuses on the grasp selection alone and proposes a learning framework to improve on this part. A different approach is to detect a valid grasp directly from 2D or 3D images without explicit pose estimation. Domae et al. [19] estimates the graspability of an object based on depth maps without the assumption of a 3D model, which makes it applicable to all objects. Saxena et al. [6] developed a grasp detection algorithm based on extracted hand-coded features from stereoscopic cameras, and machine learning (logistic regression). Other hand-coded feature-based approaches using machine learning have also been developed [11].

Instead of hand-coding features, one may use deep learning to extract the relevant features for grasping [3]–[5], [7]. These works use deep learning on depth images and output grasping rectangles with center points parameterized by (x, y) and θ in the plane of the depth image, and use the depth image values within the rectangle to compute the distance to that point. For a parallel-plate gripper approaching perpendicular to the viewing plane of the depth sensor, this approach works well. In general, this may not necessarily work, and a full 3D grasp may be required. This has previously been solved by using deep learning [1]. Here the 6-DOF grasps are generated randomly within a volume of interest and a convolutional neural network is used to evaluate the grasps by inputting multiple projections of a 3D point cloud volume centered at the grasp. Levine et al. [22] uses a convolutional neural network to learn hand-eye coordination from a large dataset of grasp attempts with real robots and a large variation of domestic objects in semi-cluttered bins. In contrast to Pinto et al. [5] they use the trained network to servo the gripper in real-time, which makes it more robust to mistakes and moving objects. Other approaches [2], [21] also use deep learning to evaluate the quality of a grasp. This differs from our approach, in which we use deep learning to compute the 3D grasp itself. In terms of input-output domain of the neural network, the work most related to ours is Huang et al. [13], where the output is the robot hand position, rotation axis and angle of rotation. Our approach differs, in that we use a dual-resolution convolutional neural network. Dual-resolution networks have successfully been used to recognize hand gestures from depth images [15]. Relative to this, our deep learning approach is novel in that we integrate two resolutions into fully connected layers before computing the output, whereas [15] integrates the final output of each resolution.

The use of a dual-resolution network enables both high accuracy in a focus region of interest, when placing the grasp and estimating the grasp pose, as well as enabling a low-resolution context awareness that e.g. ensures that the grasps do not collide with other objects in cluttered scenes. This approach is in principle similar to the fast filter-based bin-picking algorithm in [19], which uses binary and linear contact and collision filters that consider the geometry of the gripper, to filter the depth images and thereby locate 4-DOF grasps. The principle differences between our work and [19], is that we use a neural network to provide end-to-end training of a 5-DOF (extendable to 6-DOF) grasp detection network that automatically designs the appropriate general and nonlinear filters relevant to grasp estimation and collision, in a way that considers both the geometry of the gripper as well as the geometry and reflectivity of the objects.

One challenge with deep learning for robot grasping is the lack of large datasets of labeled training data, especially for depth images. Pinto et al. [5] collected grasp attempts on a real robot to train their network. Schwarz et al. [28] take another approach, and uses pretrained models from object classification to output bounding boxes and object boundaries for further grasp detection. Our solution to this problem is to generate our own labeled depth dataset in a simulated environment. The benefit of this approach is that it enables us to do end-to-end learning of the grasp itself, and utilize the depth data more directly such that we can handle challenging surfaces.

Generation of realistic looking synthetic images is an active research topic and [29] has proposed a method based on an adversarial network to improve the realism of simulated images using unlabeled real data.

Compared to the literature (e.g. [21]) our network does not require singulating or segmentation of the objects before estimating the grasp. Segmentation is difficult when the objects are highly reflective, since one is not guaranteed to have contiguous depth measurements within an object due to missing data in the depth images. To solve this problem, our neural network is trained entirely by domain randomization in a simulated environment that explicitly simulates the reflectivity of the objects in cluttered scenes by rendering missing data in the depth images similarly to how missing data occurs in real 3D images of reflective objects.

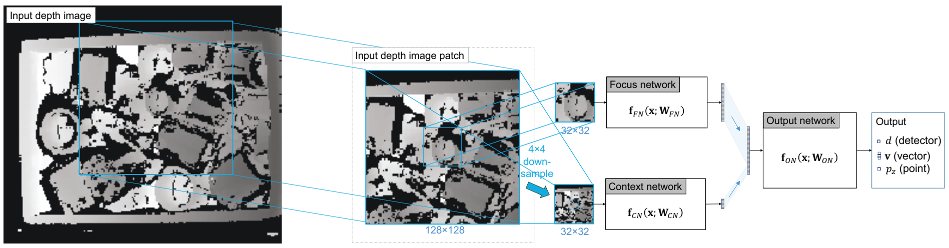

Fig. 2. Overview of the dual-resolution neural network, including scaling of input image patches.

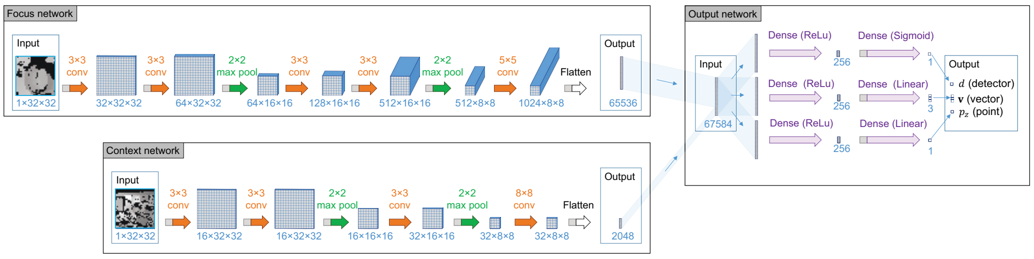

Fig. 3. Architecture of the focus, context and output networks.

III. GRASP POSE DETECTION

We propose the use of a dual-resolution convolutional neural network for estimation of grasps from depth images. Combining a high-resolution with a low-resolution image as input, provides the network with enough information to accurately place grasps on small objects, and enough of an overview of the scene to avoid collisions between the gripper and the local environment.

A. *Convolutional Neural Networks*

The neural network architecture is illustrated in Fig. 2. An input depth image $\mathbf{I}$ is used to compute a batch of image patches $\mathbf{I}_p$ of size 128×128. The neural network processes each of these image patches independently and produces five outputs for each image patch. The first output d, is the grasp detector confidence and estimates the probability of a valid grasp point in the center of the image patch. At test time, the image patch with the corresponding highest grasp detector confidence is selected, and from this image patch the neural network computes the grasp that the virtual or real robot will perform. Three outputs describe the 3D grasp vector $\mathbf{v}$, which is an approach vector for the grasp point in camera coordinates. The last output p_z, is the point estimator which estimates the distance to the grasp point along the z-axis of the 3D camera, at the x- and y- coordinates of the center of the depth image patch.

From the depth image patch $\mathbf{I}_p$ of size 128×128 pixels, the central 32×32 pixels are considered the focus of the grasp network, and are input to the convolutional focus network $\mathbf{f}_{FN}(\mathbf{x};\mathbf{W}_{FN})$. A second convolutional network, called the context network $\mathbf{f}_{CN}(\mathbf{x};\mathbf{W}_{CN})$, takes as input a 4×4 down-sampled version of $\mathbf{I}_p$. The architecture of the focus and context networks are shown in Fig. 3. Each have convolutional layers and max-pooling layers and use the rectified linear unit (ReLu) activation function after each convolutional layer. The vector outputs from these two networks are concatenated and input to the output network $\mathbf{f}_{ON}(\mathbf{x};\mathbf{W}_{ON})$. The output network has three sub-networks each having two dense layers, with each sub-network computing one of the three outputs d, $\mathbf{v}$, and p_z. The first dense layer in each sub-network uses a (ReLu) activation function. The second dense layer uses a sigmoid activation before the output d, and ReLu before the outputs $\mathbf{v}$ and p_z.

B. *Training*

The neural network was trained end-to-end, supervised on 400 000 synthetically created training examples. Training was done using the Adam optimizer [18]. One training example consists of a 128×128 image patch input $\mathbf{I}_p$ with its corresponding ground-truth output $\mathbf{y} = \begin{bmatrix} d & \mathbf{v} & p_z \end{bmatrix}$. For false examples, i.e. image patches not containing a grasp, the target vector is set to $\mathbf{y} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \end{bmatrix}$. Given the ground-truth output $\mathbf{y}$, and the output $\hat{\mathbf{y}}$ that the network predicts,

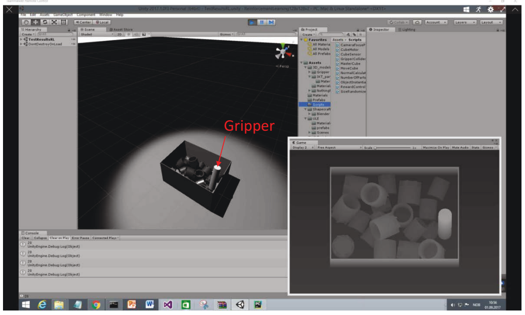

Fig. 4. The simulated environment, used for generating synthetic data set for training the neural network and for evaluating its performance on synthetic depth images.

the cost function is a weighted sum of the detector cost J_d, point regression cost J_{p_z} and the vector regression cost $J_{\mathbf{v}}$, expressed as

$$J_{total}(\mathbf{y}, \hat{\mathbf{y}}) = \alpha J_d(\mathbf{y}, \hat{\mathbf{y}}) + \beta J_{p_z}(\mathbf{y}, \hat{\mathbf{y}}) + \gamma J_{\mathbf{v}}(\mathbf{y}, \hat{\mathbf{y}}). \quad (1)$$

For the grasp detector cost, the cross entropy function is used, giving

$$J_d(\mathbf{y}, \hat{\mathbf{y}}) = -d \ln \hat{d} + (1-d) \ln (1-\hat{d}). \quad (2)$$

For both the point and vector regression costs, the squared error cost function is used. Note that for input training images not containing a valid grasp, we do not have real targets for the point and vector estimators. Therefore, we mask out the point and vector costs for false examples by multiplying with the classification label, giving us the cost functions

$$J_{p_z}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} d (p_z - \hat{p_z})^2 \quad (3)$$

and

$$J_{\mathbf{v}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} d \|\mathbf{v} - \hat{\mathbf{v}}\|_2^2. \quad (4)$$

C. Simulated Environment for Generating Training Data

There is a need for large amounts of data when training deep neural networks and hand labelling of a large data set for the bin picking task would be very time consuming. To avoid this tedious work, we used a simulated environment, shown in Fig. 4. Using the Unity3D game engine and its built-in physics, we created an environment for easy data generation.

The environment is generic and can be used to create data sets for 6 DOF grasping tasks from depth images for any type of rigid object. To generate a data set, a geometric model of the graspable objects needs to be provided. Additional information about weight, reflection and friction coefficients is also needed. Lastly, some valid grasps for each object need to be set. In our experiments we defined 21 preset grasps for each of the three types of graspable metal cylinders. The output from the simulated environment is a depth image and a list of valid grasps in camera coordinates. The data is generated as follows:

1) A random number of parts in the range 1 to 30 are instantiated in mid-air with random orientations.
2) The parts are dropped and allowed to fall to a rest in random positions in the box or on the table.
3) For each preset grasp on each instance of a part, perform a check for collision between the gripper and all other parts and the box. A grasp is a valid grasp if there is no collision.
4) For each instance of a part with at least one valid grasp, only a single valid grasp is selected. The grasp is selected in a manner that favors grasps that are close to the world z-axis and in the direction towards the simulated 3D camera.
5) The simulated 3D camera is randomly rotated and translated before an intensity image $\mathbf{I}_{cam}$ and a depth image $\mathbf{I}_{depth}$ is rendered and saved to disk along with a single valid grasp for each part in the scene.

The neural network requires a data set of 128×128 image patches where each patch either has a grasp in the center, or it has no grasp. The true examples were simply created by cropping patches from the generated depth image, centered around each grasp. An equal amount of false examples were created by cropping random patches from the same image. This way of creating false examples may lead to some false negatives. We assume that the number of possible grasps in the image are outnumbered with a large enough margin by the number of not possible grasps for this to not impact training negatively.

D. Synthetic and Real 3D Camera Images

Depth information is in principle invariant to lighting and texture, which makes it far easier to generate realistic depth images than RGB-images. However, with a real 3D camera, noise and missing depth data occurs even in controlled lighting conditions.

The 3D camera used in the experiments was the Zivid RGB-D camera based on projection of structured light. The camera has high resolution (0.1 mm), high speed (10Hz) and High-dynamic-range (HDR) imaging, which combat the over/under exposure problem, to some degree. However, the captured 3D data still suffers from noise and missing depth data, especially in cluttered scenes with many reflective surfaces, which leads to under exposure in some areas and over exposure in others, as shown to the right in Fig. 5. Additional noise comes as a result of the light from the projector reflecting off multiple objects before reaching the camera.

In order to generate realistic looking depth images, we simulate the missing data in depth images resulting from the reflectivity of the steel parts. We assume that all other objects consist of ideal diffuse reflective surfaces and that the structured light projector is the only light source in the scene. Because the experiments done with the real 3D camera and robot are done in controlled surroundings with controlled lighting conditions, the synthetically created depth images

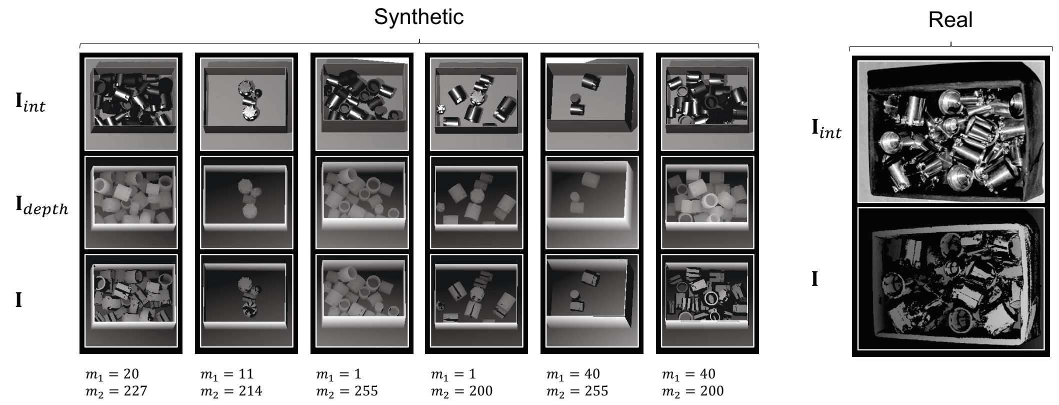

Fig. 5. Synthetic images generated with domain randomization, compared to real image of steel parts in a box. The intensity images $\mathbf{I}_{int}$ are used to mask the depth images $\mathbf{I}_{depth}$ resulting in masked depth images $\mathbf{I}$. The masking is done using randomized masking parameters m_1 and m_2 for each image.

should sufficiently approximate real depth images. Domain randomization over the dynamic range of the depth images is applied to generate training and test data for the neural network. This is done by combining an intensity image patch $\mathbf{I}_{p,int}$ with a depth image patch $\mathbf{I}_{p,depth}$ into an final image patch $\mathbf{I}_p$ defined by

$$\mathbf{I}_p = \mathbf{I}_{p,depth} \cdot \mathbb{I}(\mathbf{I}_{p,int} \geq m_1 \wedge \mathbf{I}_{p,int} \leq m_2) + \epsilon \qquad (5)$$

where $\mathbb{I}(\cdot)$ is the indicator function and ϵ is an image of uniformly-distributed random noise satisfying $-1 \leq \epsilon \leq 1$. The dynamic range of the 3D camera is randomly adjusted by setting m_1 and m_2 to uniformly-distributed random numbers satisfying $m_{1,min} \leq m_1 \leq m_{1,max}$ and $m_{2,min} \leq m_2 \leq m_{2,max}$. Additionally, we assume $m_1 \ll m_2$, so that m_1 corresponds to the minimum level of intensity required to provide a valid depth measurement, and m_2 corresponds to the maximum level, i.e. camera saturation. Examples of synthetic images and an example of a real image can be seen in Fig. 5. The synthetic images show the variations due to domain randomization over the following variables of the simulation:

- Number of steel parts
- Size of steel parts
- Position and orientation of steel parts
- Reflectivity of steel parts
- Position and orientation of the 3D camera
- Dynamic range of the 3D camera

In total, this domain randomization is expected to span the range of scenarios sufficiently that a neural net trained on synthetic images will work well on real images.

IV. EXPERIMENTS AND RESULTS

A. Procedure for Grasp Evaluation on Synthetic Data

In order to evaluate the performance of the neural network on a large data set, we tested it on data from a simulation not used during training, using a simplified simulation of a suction gripper. The experiment was conducted as follows:

1) A random number of parts in the range 1-30 were dropped in the box
2) The number of parts present in the scene was noted before the neural network was used to attempt a grasp.
3) If the grasp was successful:
    a) The grasp was logged as successful together with the number of parts present before the grasp was attempted.
    b) The picked part was removed from the scene, allowing the remaining parts to move according to their physics.
    c) If the picked part was the last part in the scene: Go to point 1. Else: Go to point 2.
4) If the grasp was unsuccessful:
    a) The grasp was logged as unsuccessful together with the number of parts present before the grasp was attempted.
    b) The cause of the failure was logged as either: Gripper collision (the virtual gripper collided with the environment or other parts) or poor grasp (outside a set tolerance).
5) Go to point 1.

The experiment in simulation was continued until 12000 parts were picked successfully. The physics of the vacuum gripper was not simulated, instead the criteria for a valid grasp were defined by the position of the gripper on the part and the gripper angle relative to the surface normal on the contact point.

B. Results of Grasp Evaluation on Synthetic Data

The grasp performance of the neural net tested in the simulated environment shows that we achieve an overall success rate of 83 % for the bin picking system, when using

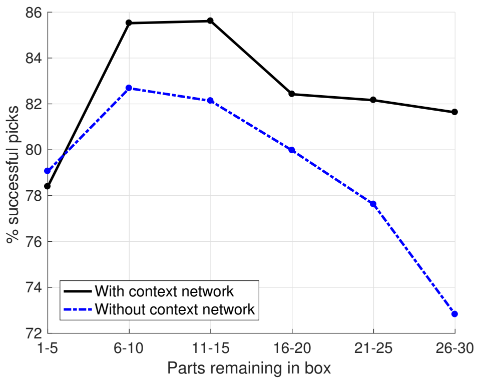

Fig. 6. Successful picks with and without context network, as a function of the number of remaining parts in the box.

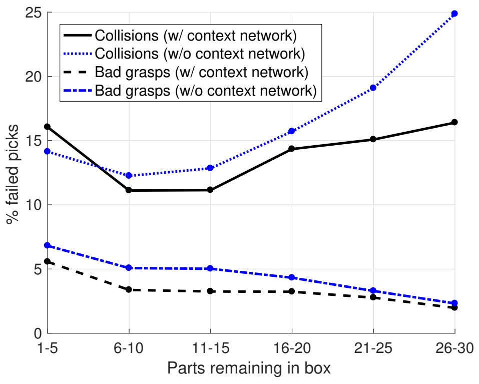

Fig. 7. Failed picks, due to collision and bad grasps, as a function of the number of remaining parts in the box.

the dual-resolution network. The success rate varies as a function of the number of parts remaining in the box (Fig. 6). Most of the errors made were due to collisions between the gripper and the environment. If these errors are ignored, i.e. we evaluate the performance of the system solely on the basis of correctly placed grasps on the steel parts, the success rate is 95 %. In a robot bin picking system this issue would be resolved by implementing a global collision check. It seems to be a trend that the success rate drops when there are few objects left in the box. The likely cause of this is that the network always chooses the most certain grasp in the scene first, leaving the most uncertain grips for last, e.g. difficult grasps like a part in the corner of the box that is only partially viewed by the camera.

C. Evaluation of the Context Network

To evaluate the effect of the context network we tested the proposed dual-resolution network, which includes the context network, and a single-resolution network without a context network. The single-resolution network has the same overall architecture as the dual-resolution network, with the difference being that the context network is removed entirely. A separate training was done on the single-resolution network. The single-resolution network without the context network has more picking failures as the number of parts in the box increase (Fig. 7), compared to the dual-resolution network with the context network. The number of collision failures are reduced by up to 34%. When there are few parts in the box there are few local collisions and the context network has little effect. As the number of parts in the box increases, there are potentially more local collision between the gripper and the parts. Overall, the use of a context network in a dual-resolution network can improve grasping success by reducing the number of collisions between the gripper and the parts.

D. Robot Bin Picking Data and Setup

For the bin picking experiments with a real robot we used reflective steel parts from an industrial automation application. We used a set of 40 cylindrical parts with diameters varying between 19 mm and 30.7 mm, and lengths varying between 25.7 mm and 31.7 mm. For these experiments, a UR5 6-DOF robotic manipulator from Universal Robots was used. A vacuum gripper with a single suction cup was attached to the end effector. The suction cup had a diameter of 10 mm and could be compressed 4-5 mm in the tool point direction. The design of the gripper puts some physical limitations on the set of grasps that can be executed without collision. Even though the dual-resolution neural network incorporates local collision checking, global collision checking for the whole robot arm was not implemented. This could be solved by running a global collision check (for instance OpenRAVE) on the suggested grasp before execution.

E. Robot Bin Picking Evaluation Procedure

A box was filled with steel parts of different sizes in a random manner. An image was captured of the box and used as input to the system, and the grasp with highest score was executed with the robot. If a grasp was unsuccessful, either because of collision or an erroneous grasp estimation, the part was removed manually. Otherwise, the robot removed the object by grasping and picking it up and placing it in a second box. A new image was acquired before attempting to remove a new part. This continued until the box was empty. The robot test was conducted on 6 different boxes, each initialized with 30 parts randomly placed.

F. Comparing Grasp Placement Performance on Simulated and Real Depth Images

The grasp placement performance of the dual-resolution network was evaluated on simulated and real depth images. In this evaluation, only the grasp placement is evaluated, and other robot system related errors are not considered. There is good correspondence between the results on simulated and real depth images. The grasp placement performance on real depth images has an overall lower success rate, see Fig. 8. The mean grasp placement success rate for the simulated

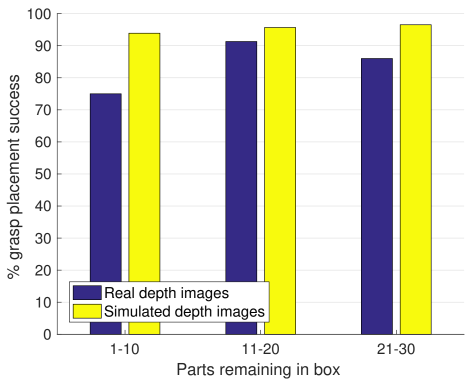

Fig. 8. Grasp placement performance on simulated and real depth images, as a function of the number of remaining parts in the box.

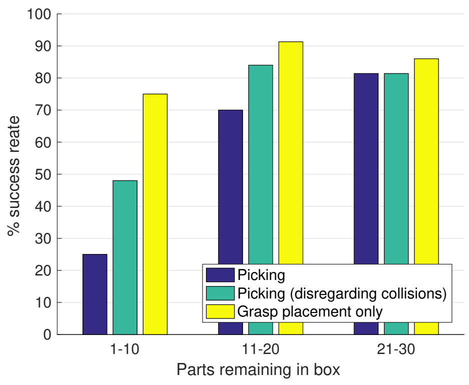

Fig. 9. Real-life bin picking performance, showing the picking success rate with and without disregarding local collisions, as well as grasp placement success.

depth images is 95% and 85% for the real depth images. In both cases there is a decrease in performance towards the bottom of the box, when there are few parts left. However, the performance drop on the real depth images is somewhat higher. This is most likely caused by more missing data in the real depth images, than in the simulated depth images, see Fig. 5.

G. Robot Bin Picking Performance

In the performance evaluation of the robot bin picking system we measure success in several ways, in order to understand the successes and failures of the system. This performance evaluation is summarized in Fig. 9. Picking success is measured by the success rate of the complete robot system performing a grasp. We also measured the picking success that could be achieved by disregarding grasps that could be removed by a global collision check. Finally, we compare this to grasp placement only. Compared to the simulation results, the robot has a very low success rate when picking the last parts in the box. This is mostly due to collisions between the real gripper and environment, and these errors stem from some differences between the physical and the virtual gripper. In addition to collisions, in the robot test, the failed grasps were due to small errors in grasp position or angle, resulting in failed grasps due to lack of suction. This error happened more frequently at the bottom of the box. The last objects are often the ones which are most difficult to grasp (e.g. in a corner), Levine et al. [22] reported a similar trend. Additionally, objects in the bottom of the box have less depth data due to occlusion. Also, parts oriented with the opening facing up occurred more frequently in the real-world test, because of a different weight distribution than in the simulations. Some differences between the real-world and the simulation combined lead to a more challenging test scenario in the real world than in the simulation. For situations when the real-world and the simulated environment are more similar, for instance when there are 21-30 parts in the box, the performance in the real-world and in simulation are equally good (81% and 82% picking success rates respectively, see Fig. 9 and Fig. 6). This result suggests that improvements in the simulation, to more accurately represent the real 3D camera, robot and gripper, will result in higher picking success in the real world.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a method for robotic bin picking of reflective steel parts. We proposed a dual-resolution convolutional neural network for 5-DOF grasping, trained entirely by domain randomization in a simulated environment. The system was tested on simulated data as well as in the real-world using a robot with a 5-DOF placement vacuum gripper. Using a context network improves the grasp performance by minimizing collisions between the gripper and the local environment. We demonstrate that training of the neural net by domain randomization on a large set of synthetic depth images is an effective and useful approach when the simulated environment closely resembles the real world. Future work will focus on more accurately simulating the 3D camera to improve the model of missing data in the depth images, and implementing a more accurate model of the gripper. We will also explore how we can improve the simulated data through Adversarial Networks.

ACKNOWLEDGMENT

This project was funded by the Research Council of Norway through the SFI Manufacturing Centre for Research-based Innovation, and in part from Research Council of Norway Grant No. 262900.

REFERENCES

[1] Gualtieri, M., ten Pas, A., Saenko, K., and Platt, R. (2016,October). High precision grasp pose detection in dense clutter. In Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on (pp. 598-605). IEEE.

[2] Kappler, D., Bohg, J., and Schaal, S. (2015, May). Leveraging big data for grasp planning. In Robotics and Automation (ICRA), 2015 IEEE International Conference on (pp. 4304-4311). IEEE.

[3] Lenz, I., Lee, H., and Saxena, A. (2015). Deep learning for detecting robotic grasps. The International Journal of Robotics Research, 34(4-5), 705-724.

[4] Redmon, J., and Angelova, A. (2015, May). Real-time grasp detection using convolutional neural networks. In Robotics and Automation (ICRA), 2015 IEEE International Conference on (pp. 1316-1322). IEEE.

[5] Pinto, L., and Gupta, A. (2016, May). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In Robotics and Automation (ICRA), 2016 IEEE International Conference on (pp. 3406-3413). IEEE.

[6] Saxena, A., Driemeyer, J., and Ng, A. Y. (2008). Robotic grasping of novel objects using vision. The International Journal of Robotics Research, 27(2), 157-173.

[7] Jiang, Y., Moseson, S., and Saxena, A. (2011, May). Efficient grasping from rgbd images: Learning using a new rectangle representation. In Robotics and Automation (ICRA), 2011 IEEE International Conference on (pp. 3304-3311). IEEE.

[8] Bicchi, A., and Kumar, V. (2000). Robotic grasping and contact: A review. In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on (Vol. 1, pp. 348-353). IEEE.

[9] Miller, A. T., Knoop, S., Christensen, H. I., and Allen, P. K. (2003, September). Automatic grasp planning using shape primitives. In Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on (Vol. 2, pp. 1824-1829). IEEE.

[10] Miller, A. T., and Allen, P. K. (2004). Graspit! a versatile simulator for robotic grasping. IEEE Robotics and Automation Magazine, 11(4), 110-122.

[11] Pelossof, R., Miller, A., Allen, P., and Jebara, T. (2004, April). An SVM learning approach to robotic grasping. In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on (Vol. 4, pp. 3512-3518). IEEE.

[12] Len, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., Morales, A., ... and Dillmann, R. (2010, November). Opengrasp: a toolkit for robot grasping simulation. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (pp. 109-120). Springer Berlin Heidelberg.

[13] Huang, P. C., Lehman, J., Mok, A. K., Miikkulainen, R., and Sentis, L. (2014, December). Grasping novel objects with a dexterous robotic hand through neuroevolution. In Computational Intelligence in Control and Automation (CICA), 2014 IEEE Symposium on (pp. 1-8). IEEE.

[14] Pas, A. T., and Platt, R. (2015). Using Geometry to Detect Grasps in 3D Point Clouds. arXiv preprint arXiv:1501.03100.

[15] Molchanov, P., Gupta, S., Kim, K., and Kautz, J. (2015). Hand gesture recognition with 3D convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 1-7).

[16] Zeng, A., Yu, K. T., Song, S., Suo, D., Walker Jr, E., Rodriguez, A., and Xiao, J. (2016). Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge. arXiv preprint arXiv:1609.09475.

[17] Goodale, M. A., and Milner, A. D. (1991). A neurological dissociation between perceiving objects and grasping them. Nature, 349(6305), 154.

[18] Kingma, D., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[19] Domae, Y., Okuda, H., Taguchi, Y., Sumi, K., and Hirai, T. (2014, May). Fast graspability evaluation on single depth maps for bin picking with general grippers. In Robotics and Automation (ICRA), 2014 IEEE International Conference on (pp. 1997-2004). IEEE.

[20] Mahler, J., Pokorny, F. T., Hou, B., Roderick, M., Laskey, M., Aubry, M., ... and Goldberg, K. (2016, May). Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In Robotics and Automation (ICRA), 2016 IEEE International Conference on (pp. 1957-1964). IEEE.

[21] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., and Goldberg, K. (2017). Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. arXiv preprint arXiv:1703.09312.

[22] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2016). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. The International Journal of Robotics Research, 0278364917710318.

[23] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. arXiv preprint arXiv:1703.06907.

[24] Abbeloos, W., and Goedem, T. (2016, June). Point Pair Feature based Object Detection for Random Bin Picking. In Computer and Robot Vision (CRV), 2016 13th Conference on (pp. 432-439). IEEE.

[25] Ellekilde, L. P., Jorgensen, J. A., Kraft, D., Kruger, N., Piater, J., and Petersen, H. G. (2012, October). Applying a learning framework for improving success rates in industrial bin picking. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on (pp. 1637-1643). IEEE.

[26] Buchholz, D., Kubus, D., Weidauer, I., Scholz, A., and Wahl, F. M. (2014, May). Combining visual and inertial features for efficient grasping and bin-picking. In Robotics and Automation (ICRA), 2014 IEEE International Conference on (pp. 875-882). IEEE.

[27] Drost, B., Ulrich, M., Navab, N., and Ilic, S. (2010, June). Model globally, match locally: Efficient and robust 3D object recognition. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on (pp. 998-1005).

[28] Schwarz, M., and Behnke, S. Data-efficient Deep Learning for RGB-D Object Perception in Cluttered Bin Picking. In Warehouse Picking Automation Workshop (WPAW), IEEE International Conference on Robotics and Automation (ICRA), 2017.

[29] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R. (2017, July). Learning from simulated and unsupervised images through adversarial training. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Vol. 3, No. 4, p. 6).

C Teaching a Robot to Grasp Real Fish by Imitation Learning from a Human Supervisor in Virtual Reality

Published by Jonatan S. Dyrstad et al. 'Teaching a Robot to Grasp Real Fish by Imitation Learning from a Human Supervisor in Virtual Reality'. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2018), pp. 7185–7192.

Bibliography entry [4].

Teaching a Robot to Grasp Real Fish by Imitation Learning from a Human Supervisor in Virtual Reality

Jonatan S. Dyrstad[1,2], Elling Ruud Øye[1], Annette Stahl[2] and John Reidar Mathiassen[1,*]

***Abstract*— We teach a real robot to grasp real fish, by training a virtual robot exclusively in virtual reality. Our approach implements robot imitation learning from a human supervisor in virtual reality. A deep 3D convolutional neural network computes grasps from a 3D occupancy grid obtained from depth imaging at multiple viewpoints. In virtual reality, a human supervisor can easily and intuitively demonstrate examples of how to grasp an object, such as a fish. From a few dozen of these demonstrations, we use domain randomization to generate a large synthetic training data set consisting of 100 000 example grasps of fish. Using this data set for training purposes, the network is able to guide a real robot and gripper to grasp real fish with good success rates. The newly proposed domain randomization approach constitutes the first step in how to efficiently perform robot imitation learning from a human supervisor in virtual reality in a way that transfers well to the real world.**

I. Introduction

In robotics, robust grasping and manipulation of objects is still a challenging task to automate, in particular for biological, non-rigid and deformable objects such as fish. Inspired by the ability of humans to perform such tasks, we investigate how to efficiently transfer the knowledge of a human to the robot, via a combination of 1) a virtual reality (VR) interface for demonstrating the task, 2) domain randomization over components of the task to generate a large synthetic data set, and 3) deep learning on this large data set. Our hypothesis is that through our approach, VR can serve as an efficient medium for demonstrating complex tasks to robots. A first step towards testing this hypothesis was presented in earlier work [16], where both training and testing was done entirely in VR. In this paper we take another step, testing this hypothesis, by demonstrating that a robot trained entirely in virtual reality can grasp real fish. We describe an approach where a human supervisor can easily teach a robot how to grasp fish in a VR environment. Grasping and picking fish from a box is an example of a challenging grasping task involving a cluttered scene of multiple highly deformable objects. In today's fishing industry, many simple and repetitive tasks are still performed by human workers due to difficulties in automating handling of the fish. This task is therefore ideal for demonstrating the capabilities of our approach applied to relevant industrial problems.

An essential aspect of imitation learning, from a human supervisor, is the system in which the human demonstrates the task. This system should be intuitive and easy to use and additionally, the supervisor should not have to demonstrate the task many times, which is often necessary when training deep learning models. Therefore, we have developed a system where the supervisor's actions are not used directly to train the robot, but rather used to generate large amounts of synthetic training data through domain randomization over relevant components of the grasping task. This enables us to train a deep neural network (DNN) for grasp detection, from scratch, using only a few dozen manually-demonstrated example grasps.

Training of our system is done exclusively on synthetic data, since this enables efficient development and testing of our approach. Testing of the system is done on real data and with a real robot and gripper. This succeeds in our case, since domain randomization, over the possible poses and dimensions of the virtual fish and the parameters of the virtual 3D camera, ensures that a 3D CNN can perform well on real data encountered during tests with real fish.

A. Related work

Robot grasping is an active ongoing research field. There are several methodologies applicable to robot grasping based on visual input, such as imitation learning and reinforcement learning, based on real or synthetic data. Imitation learning is a generic approach [1], [2], [3] in which a human or algorithmic supervisor demonstrates a task, e.g. a grasping action specified by a pose and gripper configuration, with a corresponding state space consisting of visual information. Based on a set of demonstrations, a learning algorithm, such as support vector machines (SVM), regularized regression [4] or artificial neural networks [5], [6], [7], [8], [9], [10], learns to find or evaluate the mapping between the visual state and the grasping action. In reinforcement learning (RL) there is no supervisor to demonstrate how to perform the task, instead there is a reinforcement signal provided to the learning algorithm. Imitation learning has the advantage of efficiently discovering state-action mappings that work reasonably, with the disadvantage that this may require a large data set of demonstration examples. Contrary to this, RL, e.g. using artificial neural networks [11], has the disadvantages of slow learning speed and difficulty of accurately defining a suitable objective function for more complex tasks. The advantages of RL algorithms are that they do not require a human supervisor, and RL algorithms can potentially learn to perform beyond the capabilities of a supervisor.

Synthetic data generation is a well-known and successful approach to training deep learning systems [13], [14], prior to applying them to real-world data. In particular, an approach

*Corresponding author, John.Reidar.Mathiassen@sintef.no
[1]SINTEF Ocean AS, Trondheim, Norway
[2]NTNU, Department of Engineering Cybernetics, Trondheim, Norway

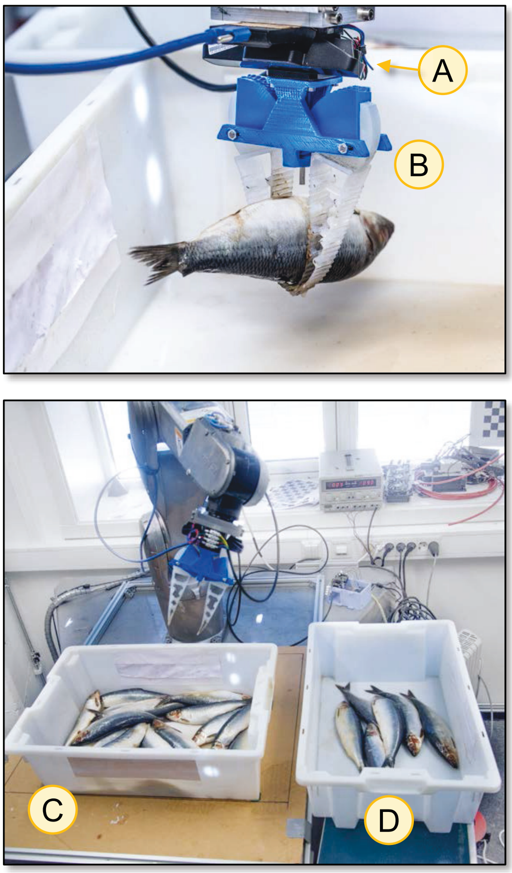

Fig. 1. A robot is tasked with picking fish from a box. The robot has a RealSense 3D camera (A) and gripper (B) mounted on its end effector. The task is to pick fish from a box (C) and place them in a second box (D).

called domain randomization has been shown to enable robust training on synthetic data that directly works on real-world data without additional training [15]. Domain randomization involves randomizing over the relevant components of a task to generate synthetic data that has enough variation to generalize to real data. A novelty of our approach for domain randomization is that it begins with an intuitive virtual reality interface where a human supervisor can easily provide *a low or moderate number* of demonstration examples. These examples are then used in a domain randomization process over the camera viewpoints, projector-camera occlusion, the physical interactions of the fish and the box, and the previously demonstrated grasps of the human supervisor. This enables us to generate the large data sets required for deep learning, while including the intent of the human supervisor. In principle, this is similar to the algorithmic supervisor approach presented in [17]. A few differences between our work and [17] are: 1) in our work the grasps are placed by a human supervisor in VR, instead of an algorithmic supervisor; 2) in our work an end-to-end 3D CNN directly computes 6-DOF grasps from the input 3D occupancy grid, whereas [17] uses a GQ-CNN to evaluate multiple randomly-sampled and pre-aligned parallel-jaw grasps on depth images and select the grasp with the highest quality.

Our previous work [16] applied 3D convolutional neural networks (3D CNN) to robot grasping. Previously, 3D CNNs have been successfully applied to e.g. 3D shape recognition [4], [12]. Our main motivation in working with volumetric occupancy grid representations of point clouds, and 3D CNNs to analyze them, is to develop the foundation for deep learning in robotics applications that are camera- and viewpoint-agnostic. Hence the 3D CNN can work in an occupancy grid that is constructed by integrating 3D information obtained from multiple depth images, multiple cameras and/or multiple viewpoints. Multiple viewpoints and cameras can provide a more complete coverage of a scene. A single-view depth image will have occluded regions, and moving the depth camera to one or more other locations will provide a better view of the occluded regions. Fusing these two views into a single occupancy grid will provide a more complete view. The advantage of the 3D CNN approach is thus that can be invariant or agnostic to the number of views or the number of cameras that generate the 3D data, and it can work on the complete view, as long as domain randomization is done over the types of views that are possible. Compared to our previous work [16], we have added domain randomization over the projector-camera offset, as well as coded an entirely new implementation of the 3D CNN in TensorFlow [18].

Our main contribution is to show that our approach to robot imitation learning from a human supervisor in VR transfers well to the real world, with a low or moderate number of demonstrations by the human supervisor. This validates our previous work [16], which was done entirely in VR.

II. System and task description

Our experiments are carried out on a 6-DOF Denso VS 087 robot arm mounted on a steel platform. An overview of the system can be seen in Fig. 1. The task is for a robot to pick up small fish of the species Atlantic herring (*Clupea harengus*) out of a box and place them in a second box, so that each fish can be weighed and processed individually. In this paper we focus on the first part of this task - grasping and picking the fish out of the box and placing them in the second box. A custom two-finger wormdrive gripper was designed, with compliant 3D-printed finger tips. The gripper consists of a single stepper motor attached to a 3D-printed hand. A wormscrew is mounted on the stepper motor axle and this drives two wormwheels - one for each finger. With the exception of the stepper motor, the entire gripper is 3D printed. Fig. 1 shows a closeup of the gripper and how the fingers are compliant enough to conform to the shape of small fish. The gripper is controlled by an Arduino, and has an adjustable gripper opening.

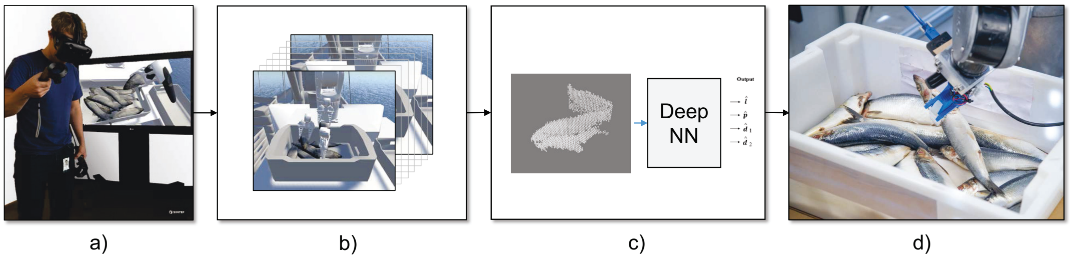

Fig. 2. The presented approach to robot imitation learning in VR, where a) the human supervisor provides a few example grasps by grasping fish from the box, b) domain randomization is used to generate a large synthetic data set based on the few example grasps, c) the deep neural network is trained on the large data set, and d) the grasp output from the trained deep neural network is used to control a gripper to grasp real fish.

An Intel RealSense SR300 depth camera is placed on the robot end-effector. The robot moves to three different poses and the camera acquires three depth images that are projected into an occupancy grid. The occupancy grid is processed by a 3D CNN implemented in TensorFlow. The output of the 3D CNN is a grasp certainty for each location in a downsampled 3D grid, and corresponding 6-DOF grasps defined by grasp placement position and orientation vectors for each location in that grid. A grasp is selected at the 3D location with the highest grasp certainty. The robot is commanded to perform this grasp, by placing the gripper, closing it and picking up the fish.

III. ROBOT LEARNING

We use a deep 3D CNN to estimate grasps from an occupancy grid. We propose the use of VR to generate large amounts of synthetic training data in order to be able to train a deep learning model with many parameters. An illustration of our approach is shown in Fig. 2.

A. VR interface for demonstrating the task

A VR interface was created where a user - in this case a human supervisor - can enter a virtual environment and demonstrate the task for the robot as shown in Fig. 2a. The user has a controller in his hand, which in VR appears to him as a gripper, similar to the gripper mounted on the robot end effector. The environment was created with the Unity game engine and the VR-equipment used the HTC-Vive head-mounted display and hand-held motion controllers.

Fish are instantiated in mid-air and dropped using simulated physics that model the deformation and friction characteristics of the pelagic fish species herring (*Clupea harengus*). The fish physics were modelled using joints and colliders in the Unity game engine. Each fish consists of seven 3-DOF spring-damper revolute joints connecting eight rigid collider segments that approximate the fish shape. The coefficients of the spring-damper joints were hand-tuned until they visually matched the pose and dynamics of real fish recorded in various static and dynamic deformation scenarios. For rendering a realistic fish mesh, a neutral-pose fish mesh was rigged and weight-painted using the colliders. The purpose of weight painting is to smoothly deform the continuous neutral-pose fish mesh using a discrete set of colliders as its ”skeleton”. Friction coefficients of each collider were hand-tuned to visually match the sliding motion of real wet fish. This relatively simple physics modelling was complex enough to provide visually realistic images, while simple enough to render tens of fish at interactive rates (90 Hz).

Simulated fish physics ensures that the fish land in natural poses in a fish box placed in front of the robot. The task of the human supervisor is to grasp the fish and move the fish from this box to another box, using the gripper in his hand (see Fig. 2a). In this way, the user gets the impression that he is *showing* the robot how to perform the task, in an easy and intuitive way. Since the user is told to grasp the fish in a way that enables him to pick it up and place it in a second box, he will naturally use a grasp that is suited for that task. If e.g. the task had been a different one, such as placing the fish in a narrow hole, the user would probably grasp the fish differently. Hence, this is an effective way of getting the user to demonstrate grasps that are suitable for a given task. For each fish grasped by the user, the grasp is logged with regards to its position and orientation relative to the fish. An example of these logged grasps can be seen in Fig. 5. This is the demonstration part of our imitation learning approach. In traditional imitation learning, a large number of demonstration examples are required. The number of required examples scales with the number of parameters in the model that is being taught. Models with very many parameters, such as large neural networks, may require on the order of tens of thousands of demonstrations in order to adequately learn the task without overfitting to the training data. For a user this is clearly too much work, and an alternative approach is needed to generate a sufficiently large and realistic training data set.

B. Generating large amounts of synthetic data by domain randomization

Based on the logged grasps, we propose to use domain randomization that includes information on the human supervisor’s grasp intent, as a method for generating a large training data set from a few demonstrations, with no further human supervision. As in the previous section, the fish are

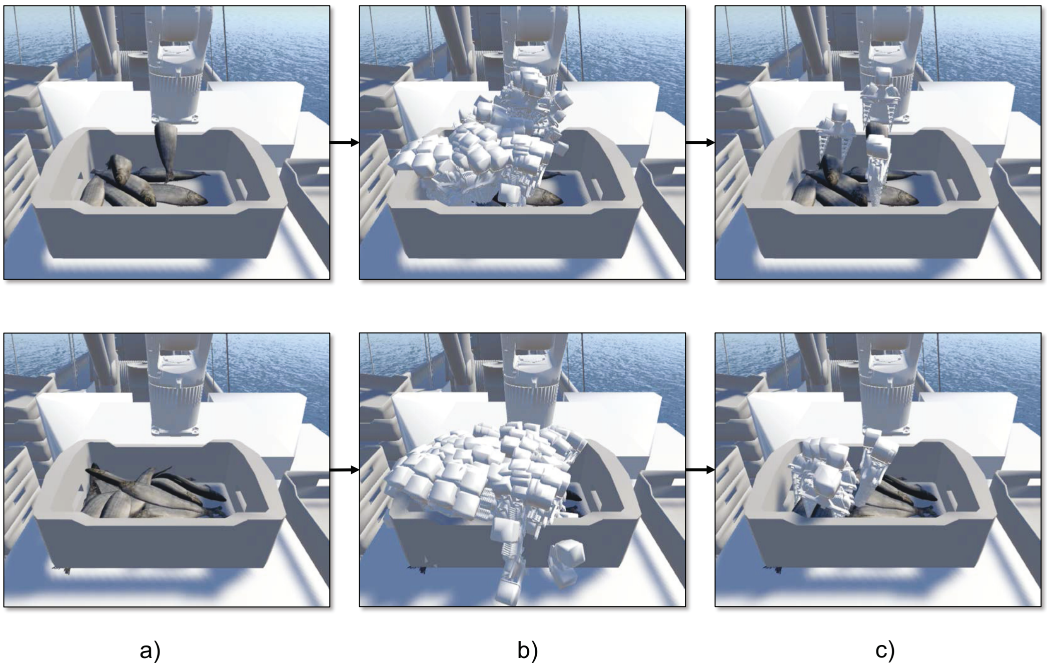

Fig. 3. Two examples (top row and botom row) illustrating the domain randomization approach for generating a large data set, by a) dropping a random number of fish in the box, using realistic fish physics, b) placing each of the logged grasps onto each fish, c) pruning the grasps based on collision heuristics.

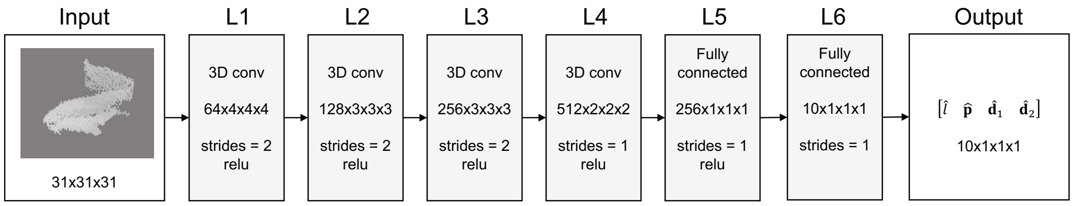

Fig. 4. The architecture of the 3D CNN consists of stacked convolutional layers, with striding (instead of max pooling) used to reduce the output resolution. The dense layers are swapped for $1 \times 1 \times 1$ convolutions to enable inputs of varying sizes. The activation functions for the feature extraction layers are rectified linear units, and in the last layer, $\hat{l}$ has a sigmoid activation function and the rest have linear activations.

instantiated and dropped into the fish box, as shown in Fig. 3a. By randomizing over the number of fish, and the position and orientation of each fish before dropping them into the box, this provides domain randomization over the possible ways in which fish can realistically be positioned relative to each other in a box.

Instead of the human supervisor demonstrating the grasp for each randomly generated box of fish, we instantiate all of the previously logged grasps onto each of the fish in the box, as shown in Fig. 3b. However, not all of the grasps are valid for all of the fish, given their current pose and position in the box (i.e. closeness to the walls etc.). Therefore, for every fish, all of the logged grasps are automatically checked using collision heuristics to see if they collide with the environment or with the other objects in the scene in any way. The ones that do not are kept and the rest are discarded, resulting in a set of plausible grasps as shown in Fig. 3c. This three-step approach provides domain randomization over the possible ways a human supervisor would *probably* grasp the fish, based on what know from the previously logged grasps.

An orthographically projected depth image is rendered of the entire fish box and the list of valid grip vectors are recorded along with the depth image. The field of view and resolution of the virtual 3D camera is such that each pixel in the orthographically projected depth image can be read as an xyz-coordinate in millimeters given in camera coordinates

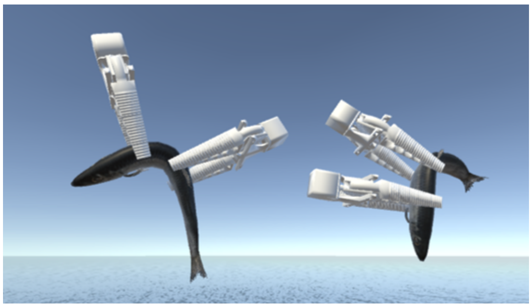

Fig. 5. The logged grasps after demonstration of two grasps in virtual reality. As the fish bends and twists, the grasps follow, making them valid for the fish regardless of pose.

(with an offset of $\frac{imagewidth/height}{2}$ in the xy-direction). Some 3D cameras, such as the Intel RealSense SR300, work by projecting a light pattern from a projector that is offset from the actual camera. Because some of the scene is visible to the camera but occluded to the projector, the result is *depth shadows*, areas in the depth image with unknown depth values. This effect is simulated with the virtual 3D camera as well. The depth data we generate in simulation can therefore be thought of as coming from a perfectly calibrated real 3D camera. To provide robust learning, we randomize the position and orientation of the virtual 3D camera and the offset between the projector and the camera, thus creating variations in the amount of missing data in the depth images due to the occlusion of the projector illumination. This is our final component of domain randomization.

C. Neural network

The depth images are projected into a 3D occupancy grid, and we use a 3D CNN to estimate grasps from a receptive field volume in the occupancy grid, and split the problem up into three sub-problems

- Detecting probable grasp locations
- Estimating the precise grip point
- Estimating the orientation of the gripper

The architecture of the network is shown in Fig. 4. For a volume of size $31 \times 31 \times 31$, the output is a vector

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{l} & \hat{\mathbf{p}} & \hat{\mathbf{d}}_1 & \hat{\mathbf{d}}_2 \end{bmatrix}, \tag{1}$$

where $\hat{l} \in [0, 1]$ is a label that estimates the certainty that the input volume contains a valid grasp, $\hat{\mathbf{p}}$ estimates the position of a grasp within the input volume, $\hat{\mathbf{d}}_1$ and $\hat{\mathbf{d}}_2$ estimate the orientation of the grasp.

The network is *fully convolutional* and has sliding dense layers, meaning that the dimensions of the output from the network is dependent on the dimensions of the input volume. For larger inputs, the result is a grasp detector, capable of detecting multiple grasps within the input volume.

The predictions for the three sub-problems are output from the same network and trained jointly because of the high dependence between the different objectives. The total cost for training example i is given by

$$J^{(i)} = J_C^{(i)} + l^{(i)}(0.1 \cdot J_O^{(i)} + 0.1 \cdot J_P^{(i)}), \tag{2}$$

where $J_C^{(i)}$ is the classification cost, $J_O^{(i)}$ the orientation estimation cost, $J_P^{(i)}$ the position estimation cost, $l^{(i)}$ is the true label for training example i. Note that for false examples, no updates are done to the position and orientation estimators. For classification of valid grasps, the binary cross entropy function

$$J_C^{(i)} = -l^{(i)} \log(\hat{l}^{(i)}) + (1 - l^{(i)}) \log(1 - \hat{l}^{(i)}) \tag{3}$$

is used, and for regression on the precise grasp point $\mathbf{p}^{(i)}$ within the given volume we use the squared error cost function

$$J_P^{(i)} = \frac{1}{2} ||\hat{\mathbf{p}}^{(i)} - \mathbf{p}^{(i)}||^2. \tag{4}$$

The orientation of the gripper is defined unambiguously with two three dimensional vectors, each describing a direction in 3D-space (see Fig. 6). The total orientation cost for the $N = 2$ orientation vectors is given by

$$J_O^{(i)} = \sum_{k=1}^{N} \frac{1}{2} ||\hat{\mathbf{d}}_k^{(i)} - \mathbf{d}_k^{(i)}||^2, \tag{5}$$

where $\mathbf{d_k}^{(i)}$ and $\hat{\mathbf{d}}_k^{(i)}$ for $k \in 1, 2$ respectively denote the true and estimated orientation vectors for training example i.

D. Preparing data for training

During training, the input to the network is a receptive field volume of size $31 \times 31 \times 31$ and the ground truth grasp certainty $l^{(i)}$ is either 1 or 0 (i.e. the volume does or does not contain a valid grasp). Training examples with true labels are simply created by cropping volumes of the synthetically created depth images centered around one of the valid grasp points for that image. The crop is offset randomly from the middle by some amount in order to create training vectors for the grip point estimator as well. In our experiments we generate false training examples (i.e. areas with a low probability of containing a grasp), simply by cropping random volumes of the occupancy grid and

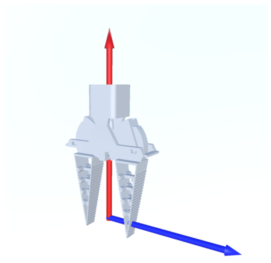

Fig. 6. The orientation of the gripper is defined by two vectors $\mathbf{d_1}$ (red) and $\mathbf{d_2}$ (blue). The position $\mathbf{p}$ is at the intersection of these two vectors.

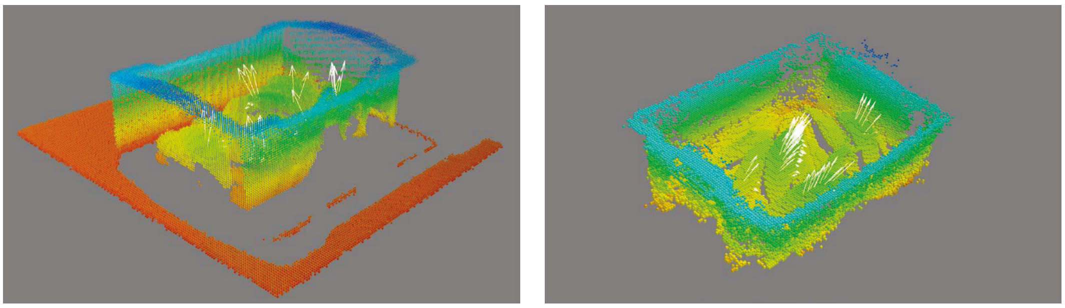

Fig. 7. Examples of grasp vectors placed by the 3D CNN in the occupancy grid of synthetic (left) and real (right) data.

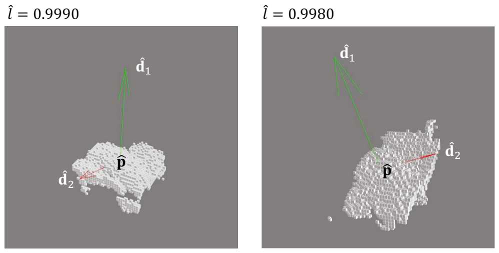

Fig. 8. Examples of predicted grasps on synthetic data, showing the contents of the receptive field and the predicted grasp vectors.

labelling them as false examples. Because the volumes that contain valid grasp are vastly outnumbered by the volumes that do not, the result is a false-data set with mostly true, but also some false, negatives.

IV. Experiments and Results

A. Generating synthetic data

Domain randomization was used to prepare the synthetic data set. In our experiments, 35 grasps were shown in VR. With these grasps, 3334 different random scenes were generated, each with a random number of fish between 1 and 25. For each of these scenes, 3 depth images were generated by randomly selecting a viewpoint and projector-camera offset. The result of this is more than 10000 synthetic depth images of fish boxes containing between 1 and 25 fish. From these depth images, 100000 examples were cropped where 50% of the data set did not contain a grasp.

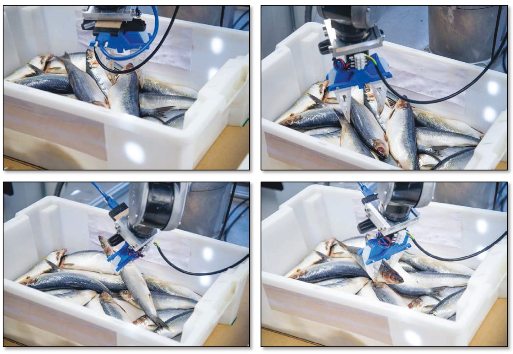

Fig. 9. Examples showing grasping of real fish.

B. Training the 3D CNN

The data was split into a training set of 100000 examples and a validation set of 4000 examples. The 3D CNN was trained with the Adam [19] optimizer on the training set. Early stopping was used to stop training before the cost function started increasing on the validation set. After a few hours the training was stopped, and the 3D CNN was tested on some synthetic data and visualized to inspect the outputs of the 3D CNN, before proceeding to real-world tests. The 3D CNN takes as input an occupancy grid and outputs grasp certainties and grasps for the entire volume of the occupancy grid. Examples of a full synthetic occupancy grid with predicted grasps is shown in Fig. 7 (left), and examples of two receptive fields with predictions can be seen in Fig. 8.

C. Experimental protocol

The 3D CNN was evaluated in an experiment with real fish. The setup included the robot with a box in front of it, where fish are placed. The fish box also contains some water. The robot is tasked with picking fish, one at a time, from that box and placing them into a second box. This setup is shown in Fig. 1. The goal of the experiment was to measure the grasping success rate and failure rate, in picking up the fish from the first box and placing it into the second box. We also wanted to determine the types of failures. The experimental protocol was as follows:

1) Place a box of 25 fish in front of the robot.
2) Scan the box and compute the occupancy grid.
3) Detect grasps for the entire occupancy grid and select the most certain grasp.
4) Attempt the grasp.

5) If successful grasp, goto point 2.
6) If unsuccessful, log the failure type and randomize box:
   a) Remove the box from in front of the robot.
   b) Pour contents of the box into a second box.
   c) Pour contents of the second box back into the first box.
   d) Place the box back in front of the robot.
   e) Goto point 2.

The grasping was continued until the box was empty.

D. Failure types

A grasp was judged as a success if a fish was successfully moved from one box to another. If the robot failed to do so the grasp was judged as a failure. The two main reasons for failure were bad grasps and collisions. A failure was logged in the bad grasps category if:

1) The robot failed to pick up the fish.
2) The fish was dropped during transfer to the second box.

Collisions were logged in three separate subcategories:

1) Gripper collisions within the 3D CNN's receptive field.
2) Gripper collisions on approach to an otherwise valid grasp.
3) Robot and camera collisions.

The category "NN failures" in Table I excludes the failures from the second and third collision failure types. These failures should not be credited to the 3D CNN because the conditions for success are unobservable for the neural network. Additionally failed grasps where the highest predicted grasp certainty was less than 0.50 were not included in 'NN failures'.

E. Real-world test results

Example grasps can be seen in Fig. 9, showing how the gripper approaches the grasp point, places the grasp and begins picking up the fish. Fig. 7 (right) shows an occupancy grid computed during the real-world tests.

The experimental protocol was performed on a total of seven boxes, and the successes and failures were categorized. The results are summarized quantitatively in Table I. Referring to that table, we see successes and failures of the grasping task, as well as the success and failures attributed to the neural network (NN). There are two boxes that deviate significantly from the others. Box 2 contained fish that were not very fresh, resulting in an oily film on the fish and a very greasy box after repeated randomization of it. This affected the depth imaging and resulted in more slippery and soft fish that were difficult to grasp. The average grasp certainty $\hat{l}$ is significantly lower for box two, suggesting the quality of the imaging was affected by the oily film. During the experiment on box 3, the gripper failed after 19 grasp attempts and a replacement part had to be 3D printed, and the experiments continued the next day.

The overall success rate was 74 %, and this increased to 80 % when excluding failures that could not be attributed to the neural network. On average the successful grasps had a higher predicted grasp certainty than the unsuccessful ones, 0.98 and 0.92 respectively. This suggests that a threshold on grasp certainty values could yield better results by minimizing failed grasp attempts.

TABLE I
GRASPING RESULTS ON REAL FISH

Box	Success (%)	Success	Failure	NN success (%)	NN failure	Avg. $\hat{l}$
1	71 %	25	10	78 %	7	1.00
2	57 %	25	19	71 %	10	0.84
3	74 %	14	5	74 %	5	1.00
4	86 %	25	4	86 %	4	1.00
5	81 %	25	6	83 %	5	0.99
6	83 %	25	5	96 %	1	0.99
7	76 %	25	8	76 %	8	1.00
All	74 %	164	57	80 %	40	0.97

V. Discussion and future work

The results of our work, suggest that our approach to domain randomization in virtual reality is an efficient method of transferring knowledge to a robot. Based on only a few demonstration examples, a sufficiently large and diverse synthetic data set was generated that was capable of training a large 3D convolutional neural network. The real-world experiments showed an overall grasping success rate of 74 %, which increases to 80 % when considering only the failures that could be attributed to the neural network. The task of grasping slippery fish, with a narrow elastomer gripper, is very unforgiving with respect to grasp placement errors. If the grasp is placed with a significant offset, in any direction, from fish's center of gravity, the probability of the fish sliding or dropping out of the gripper increases. Therefore the task requires precise grasp placement. To achieve this at greater than 80 % success rate, our conclusion is that the receptive field size will have to be increased. Increasing the receptive field size will enable the neural network to observe more of the fish and its pose and position relative to the box and other fish. These factors can reduce collisions and improve grasp placement accuracy. This results in a larger 3D CNN. For example, increasing the receptive field from $31 \times 31 \times 31$ to $63 \times 63 \times 63$ will result in an eightfold increase in the number of parameters in the neural network, and a similar increase in training time and the required amount of training data. Considering the limited observations that can be made within a small receptive field (see Fig. 8) it is understandable that an larger receptive field may improve the ability of the neural network to estimate accurate grasp placements. For this reason, our future work will be focused on increasing the receptive field size. We will also adjust the dimensions of the gripper used in the domain randomization, so that it better matches the dimensions of the actual end-effector and robot gripper. We expect that this will reduce the number of errors attributed to collisions with the box. Other avenues of future work, include learning sequences of actions, such as patterned packing (i.e. packing fish in a box according to a predefined number of layers and orientation of the fish in each layer), without re-grasping the fish after picking it out of the first box.

Our hypothesis from the onset is that through our approach, VR can serve as a efficient medium for demonstrating complex tasks to robots. A step towards testing this hypothesis was presented in this paper, and the results are promising enough to maintain our hypothesis. One may question whether the presented approach is unnecessarily complex for picking fish, and that we should have compared it to a baseline, such as antipodal grasp sampling with automated grasp placement. Investigating this was outside the scope of this paper, since our focus was on testing our hypothesis as a step towards more challenging applications of our approach.

Beyond our application to grasping and handling fish, we will also expand the domain randomization methodology and neural network architecture presented to the more generic problem of grasping and handling multiple types of objects.

ACKNOWLEDGMENT

This project was funded by Research Council of Norway Grant No. 262900.

REFERENCES

[1] R. Pelossof, A. Miller, P. Allen and T. Jebara, ”An SVM learning approach to robotic grasping,” In Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA), 2004, Vol. 4, pp. 3512-3518.

[2] B. D. Argall, S. Chernova, M. Veloso and B. Browning, ”A survey of robot learning from demonstration,” Robotics and autonomous systems, 57(5), 469-483. May 2009.

[3] S. Choi, K. Lee and S. Oh, ”Robust learning from demonstration using leveraged Gaussian processes and sparse-constrained optimization,” In 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 470-475.

[4] S. Song and J. Xiao, ”Sliding shapes for 3d object detection in depth images,”. In European Conference on Computer Vision, 2014, pp. 634-651. Springer International Publishing.

[5] N. Rezzoug and P. Gorce, ”Robotic grasping: A generic neural network architecture”, 2006, INTECH Open Access Publisher.

[6] A. Saxena, J. Driemeyer and A. Y. Ng, ”Robotic grasping of novel objects using vision”. The International Journal of Robotics Research, 2008, 27(2), 157-173.

[7] Y. Jiang, S. Moseson and A. Saxena, ”Efficient grasping from rgbd images: Learning using a new rectangle representation,” In 2011 IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 3304-3311.

[8] P. C. Huang, J. Lehman, A. K. Mok, R. Miikkulainen and L. Sentis, ”Grasping novel objects with a dexterous robotic hand through neuroevolution,” In 2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA), 2014, pp. 1-8.

[9] I. Lenz, H. Lee and A. Saxena, ”Deep learning for detecting robotic grasps,” The International Journal of Robotics Research, 2015, 34(4-5), 705-724.

[10] J. Dyrstad, ”Training convolutional neural networks in virtual reality for grasp detection from 3D images”, Master’s thesis, University of Stavanger, 2016. URL: http://hdl.handle.net/11250/2413962

[11] S. Levine, P. Pastor, A. Krizhevsky and D. Quillen, ”Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”, arXiv preprint arXiv:1603.02199. (2016)

[12] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, ”3d shapenets: A deep representation for volumetric shapes,” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1912-1920.

[13] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A. and Blake, A., 2011, June. Real-time human pose recognition in parts from single depth images. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on (pp. 1297-1304). IEEE.

[14] E. Wood et al. ”Rendering of Eyes for Eye-Shape Registration and Gaze Estimation”. In CoRR abs/1505.05916 (2015)

[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba and P. Abbeel, ”Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. arXiv preprint arXiv:1703.06907. (2017)

[16] J. Dyrstad and J. R. Mathiassen, ”Grasping Virtual Fish: A Step Towards Robotic Deep Learning from Demonstration in Virtual Reality,” In 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2017, In press.

[17] J. Mahler and K. Goldberg, ”Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences,” 1st Conference on Robot Learning (CoRL). Mt. View, CA. Nov 2017. Proceedings of Machine Learning Research volume 78.

[18] A. Martn, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin et al. ”TensorFlow: A System for Large-Scale Machine Learning.” In OSDI, vol. 16, pp. 265-283. 2016.

[19] Kingma, D., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

D Robotic grasping in arbitrarily sized volumes with recurrent attention models

By Jonatan S. Dyrstad, Elling R. Øye, Annette Stahl, John R. Mathiassen ‘Robotic grasping in arbitrarily sized volumes with recurrent attention models’. Under review in: IEEE Transactions on Robotics (2023)

Bibliography entry [5].

Appendix A

Learnt Attention Policy for Visual Servoing Experiment 1

Examples of the attention policy learnt in 3.2.1. The learnt policy processes the volume from left to right, tracing the center line of the table. It deviates from this center line only to focus on the objects of interest. Two examples are shown in Figs. A.1 and A.2

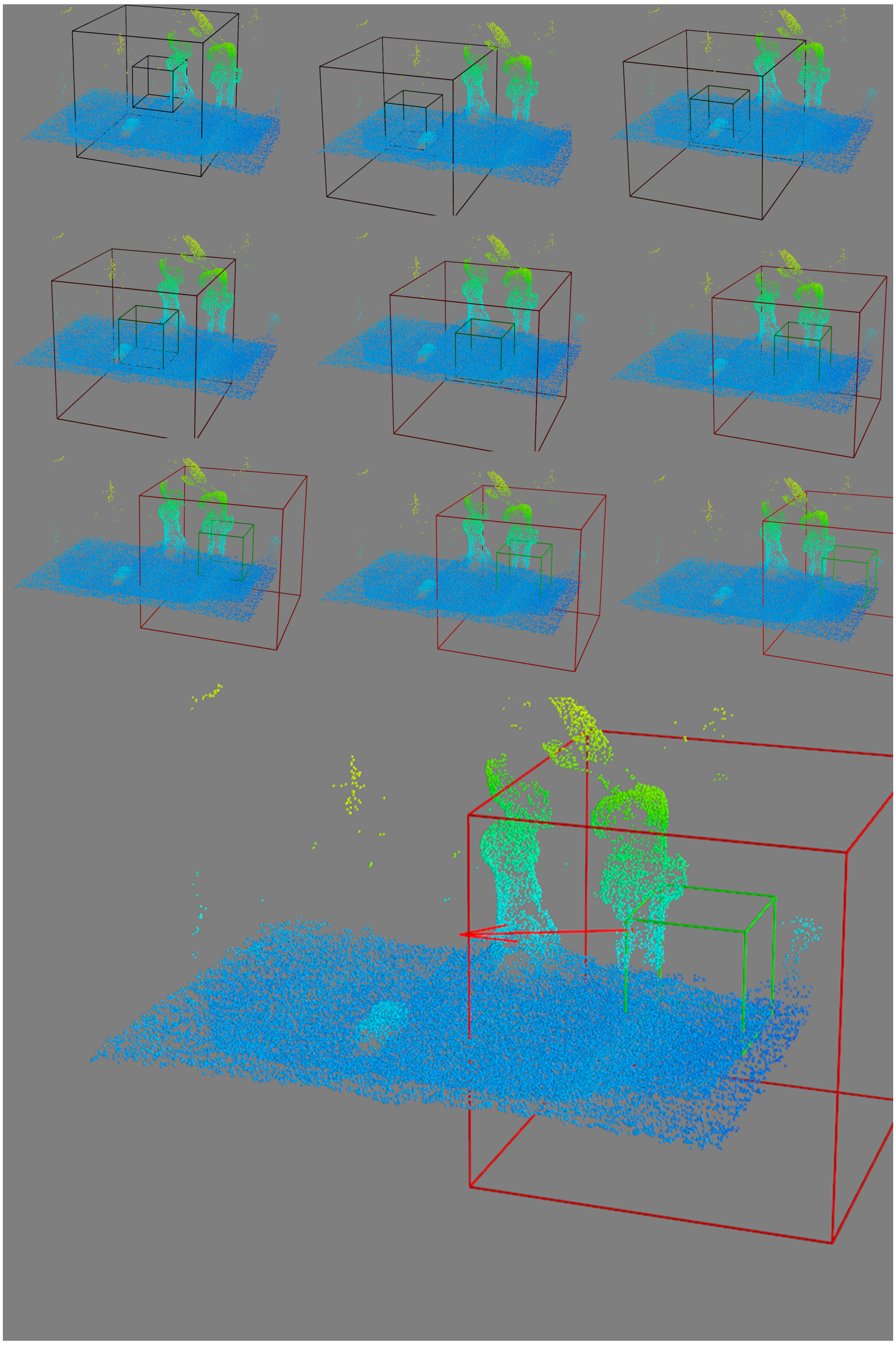

Figure A.1: The red arrow indicates the predicted velocity towards the graspable object.

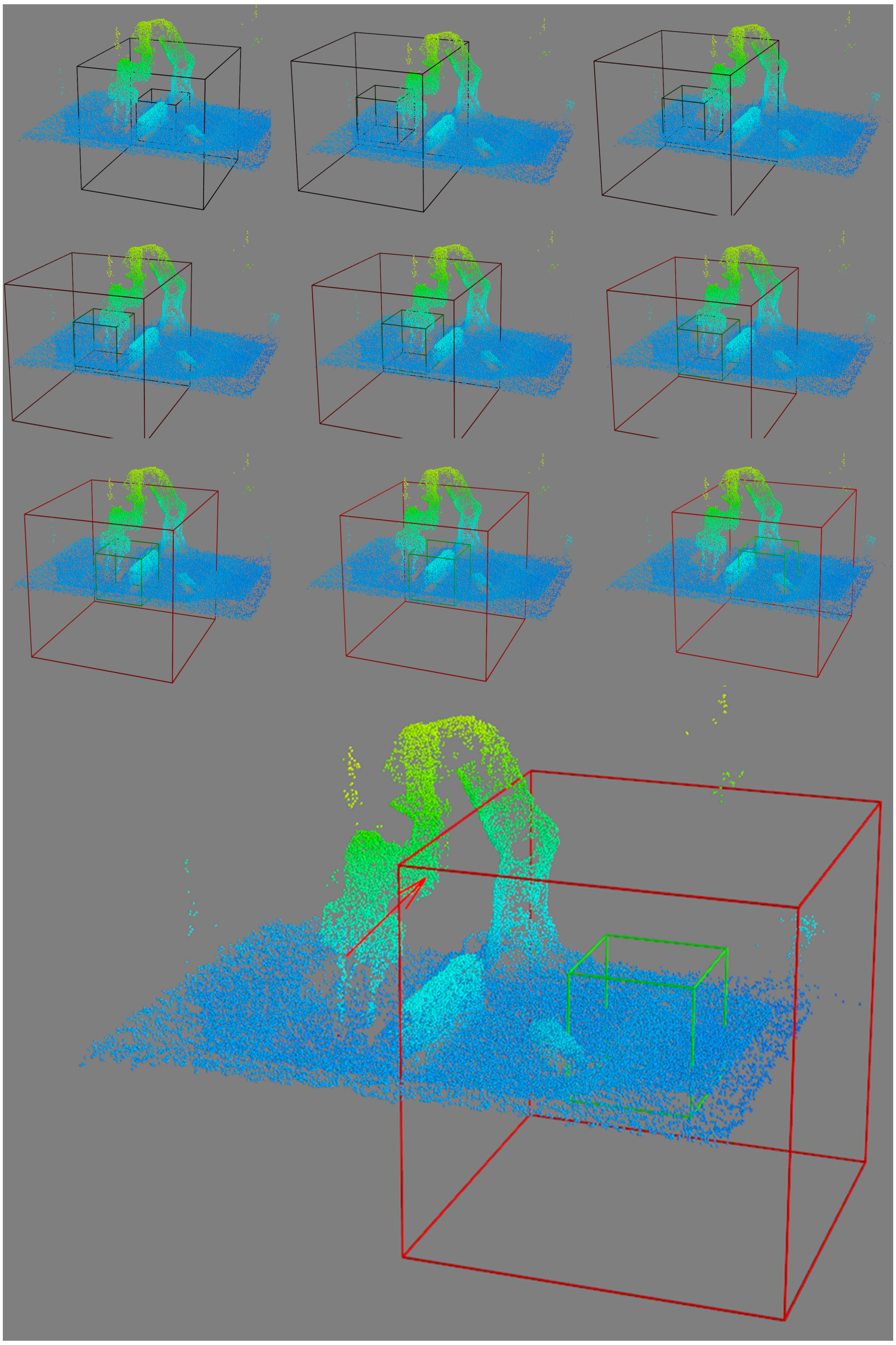

Figure A.2: The red arrow indicates the predicted velocity towards the graspable object.

ISBN 978-82-326-7232-5 (printed ver.)
ISBN 978-82-326-7231-8 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)