Karlsruhe Institute of Technology

High Performance Humanoid Technologies

Dynamical Deformation Network:

A deep generative model for 3d voxel object deformation using Kalman variational auto-encoder

Bachelor's Thesis
of

Jonas Frey

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
High Perfomance Humanoid Technologies Lab (H^2T)

Referees: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Prof. Dr.-Ing Tamim Asfour
Assoc. Prof. Ph.D Takamitsu Matsubara

Duration: June 1th, 2018 – November 30th, 2018

Erklärung:

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 20. September 2018

Jonas Frey

Abstract:

This bachelor's thesis incorporates the recent advances achieved in deep learning, to model the dynamics of deformable objects. The capability of a novel statistical deep learning approach to model deformation in a data driven manner is examined. We propose the Dynamical Deformation Network (DDN) which is able to learn key features of deformable objects in time series. This framework is based on the Kalman variational auto-encoder network and extends it to handle sequential 3D voxel data. The DDN is a deep generative model, which ia able to learn latent variables in an unsupervised fashion that describe the deformation characteristics of 3D objects and can capture the appearing deformation dynamics in a hidden space over time. It allows to interpolate missing data in time sequences and predict the future shape of deformable objects. We train this model end-to-end and validate it in simulation for a variety of simple modelling tasks. Experimental results are shown for two different slow recovering elastic foam objects, for which deformation datasets were generated by using a Universal Robot 5 robotic arm and an Intel Realsense camera. These experiments verify the DDN‘s capability to model elastic deformation of noise and partially observed 3D voxel objects. This work takes the first step to data driven deep learning modelling in application areas that classical parametric deformation modelling approaches can not fully cover.

Contents

1. Introduction **1**

2. Fundamentals **3**
2.1. Neural Networks . . . 3
2.2. 3D Data Representation . . . 4
2.2.1. Point Cloud Data . . . 4
2.2.2. Voxel Data . . . 5
2.2.3. Polygon Mesh Data . . . 5
2.2.4. Other Methods . . . 5
2.3. Deformation . . . 6

3. Related Work **8**
3.1. Deformation Modelling . . . 8
3.2. 3D Deep Learning . . . 9

4. Dynamical Deformation Network **10**
4.1. Introduction to Variational Auto-Encoders . . . 10
4.2. Introduction to Kalman Variational Auto-Encoder . . . 12
4.3. Overview and Implementation of Proposed Network . . . 14

5. Evaluation **16**
5.1. Simulation . . . 16
5.1.1. 3D Bouncing Ball . . . 16
5.1.2. Box Deformation . . . 23
5.1.3. Conclusion . . . 26
5.2. Robot Experiments . . . 26
5.2.1. Robot Setup . . . 26
5.2.2. Experimental Foam Deformation . . . 28
5.2.3. Experiment Toy-Ball Deformation . . . 34

6. Conclusion and Outlook **38**

A. Appendix **45**
A.1. Deep Learning Frameworks . . . 45
A.2. Robot Operating System . . . 46
A.3. Decoder and Encoder Tensorflow Implementation . . . 47
A.3.1. Encoder . . . 47
A.3.2. Decoder . . . 49

1. Introduction

The goal of this thesis is to develop and evaluate a system that reduces the dimensionality of a voxel representation of a soft object, learns the deformation dynamics in this reduced space over time, and allows to make predictions about the deformation in the next time-steps by means of a full-resolution voxel representation.
One of the most basic skills all humans have is the ability to forecast changes in our surrounding environment with knowledge about the present and past state. We reason between action and reaction and start to learn at an early age to predict the trajectory of a ball and how to utilize this knowledge to catch or dodge the ball. Rational reasoning is one of the main reasons why humans are superior to all other forms of life on earth in terms of cognitive abilities. Forecasting the future helps us choose the right action in short and long term to reach a desired result. We extended our human capabilities to forecast the future by creating all kinds of complex models, utilizing maths and modern computers, to make precise predictions of the real world. Modelling is a key challenge in all fields of engineering ranging from simulating car crashes or calculating payloads of bridges to describing the structure of complex proteins in order to gain a clearer understanding of how the human body works. One important subfield

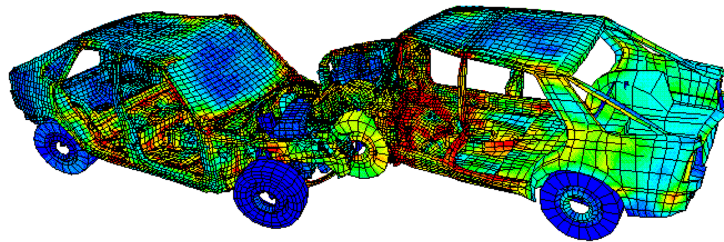


Fig. 1.1.: Finite element method car crash deformation simulation. Data Source [44]

of modelling regards deformation. Deformation in material science refers to changes in shape or size of objects [63], and is of significant importance in daily life. Deformation can be observed when folding clothes, handling food, or even shaking the hand of another person. Nowadays the majority of structural analysis including deformation is performed by Finite Element Methods (FEM), which were invented 70 years ago. With this methods complex analytic problems which include boundary value problems for partial differential equations can be simplified to a system of algebraic equations. By describing deformation with classical mechanical differential equations, constrained by additional boundary conditions, deformation can be converted into a mathematical problem. The basic idea behind applying FEM is to divide this large problem into simpler parts, called finite elements, which are easier to solve. Solving these algebraic equations approximates a solution for the analytic large-scale problem. Astonishingly precise results can be achieved by FEM, nearly perfectly describing the real world, when exact material properties, environmental constrains and enough time or computational power is available. For most applications where real time calculation is not required, computational power is not the limiting factor of this method.

The bigger limitation of FEM is knowing all exact material properties and other constraints of the scenario and environment. This is not a problem when calculating a car crash, where all the parts and material, with exact properties can be determined in advance, but in a lot of scenarios these properties

cannot even be measured or are simply not available. The development in structural analysis is mainly driven by mechanical engineers and material scientists. This leads to the fact that most problem scenarios where material properties cannot be acquired or are not available got neglected, because these scenarios are out of scope of these specific research fields.

With modern robotics expanding from pick and place jobs in carefully designed and limited industrial environments, to interacting in for-humans-made environments, new challenges arise. One of these challenges is how to manipulate and interact with deformable objects, for which it is hard or even not feasible to fully parameterize them. The goal of this new challenge is to model deformable objects, so the correct next motion or action, that the robot should perform, to reach a desired target can be calculated while taking the behavior of the deformable object into account. This abstract formulation can be simplified by looking at some actual scenarios, in which humans must deal with deformable objects.
One category of these objects are food related items. Tasks like preparing meals and helping old or disabled people to eat might be performed by robots soon. Another even more critical scenario where deformation is necessary to be modeled is for operating a robotic surgical system, where the behavior of the human body must be known to perform precise movements. For a successful operation it is necessary to model different tissues, the heart or other organs like the lungs which are constantly deforming and can‘t be easily parameterized by FEM.
The classic approach to deal with these problem scenarios is to at first simplify the objects so they can be captured by FEM or similar models, which underly the same basic constrains of parameterization. If this simplification is not precise enough to capture the object, the next step is to extend the applied modelling method in order to capture a more complex behavior of the deformable object, which can be extremely challenging or not even possible.

In this work we want to investigate the capability of a novel deep learning approach to model deformation. We propose the Dynamical Deformation Network, for which detailed object parameterization is not required. Instead it is capable to model deformation in a fully data-driven manner. This network intuitively learns the dynamics that underly the apparent deformation. Our proposed method is not meant to be competing with or be an alternative to other methods in the application area of classical deformation modelling. Instead our network shows that applying deep learning combined with the benefits of a statistic model is capable of modelling deformation without the parameterization boundary conditions, and therefore marks the first step to opening a new application field for deep learning.

2. Fundamentals

This section is for readers who are new to the topic of deep learning and deformations. It provides a short introduction to neural networks, 3D spatial data representations and deformation.

2.1. Neural Networks

"Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules." [62]
The concept of interconnected artificial neurons building a computational network, can be dated back to 1943. Neurophysiologist Warren McCulloch and logician Walter Pitts showed that a network consisting of a discrete number of simple units is able to compute complex functions [39]. This concept is inspired by the human brain, in which nerve cells in the brain (neurons) transmit information via electro-chemical signals. One neuron can be connected to up to 10.000 different neurons over synaptic connections.

This principle was adopted to computer science by Frank Rosenblatt and Charles Wightman with the building of the first neuron computer named Mark I Perceptron [34]. The idea of developing a network based on an architecture inspired by the brain structure slowly became less interesting over the next 50 years. One reason for this is the high computational cost that comes with implementing a neural network, and inevitably lead to limited application areas. Another cause was the rapid development and success in other classical electrical engineering disciplines at the time, which attracted most researchers and development budget.

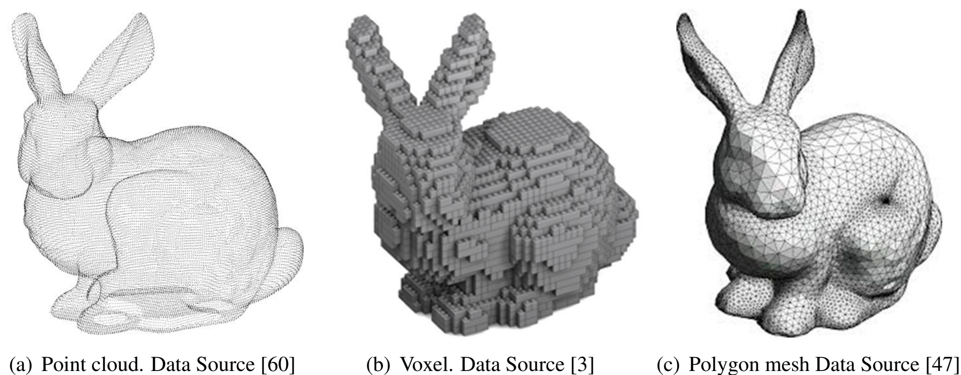
The latest development in parallel computing and cheap consumer GPUs allowed applying neural networks to a variety of different tasks and became the leading force in machine learning. The applied neural networks strongly differed from the brain structure and could achieve outstanding performance on different tasks. In this context the term "deep learning" that will be explained in the next paragraph was introduced. The goal of deep learning is to train a network, which defines the relation ship between a set of input and output values. The network is defined by multiple parameters (weights) that can be modified, in order to achieve the best performance on modelling the intended input/output relation. The cost function is a measurement of error e.g. in classification or reconstruction tasks, and the network weights are tuned to minimize this cost function. Deep learning can be performed supervised, semi-supervised and unsupervised. After the time-consuming training phase is finished correctly, the parameterized network can abstractly solve complex tasks, with very low computational power required.

A neural network can generally be split into different layers. The simplest neural network consists of an input and output layer, connected by a hidden layer. Each neuron in the hidden and output layer performs an operation on the input data, provided by the previous neurons, and passes the result to the next neurons. The performed operation can be a weighted summation of the input neurons (including a bias) or a nonlinear function, e.g. tanh, ReLu or sigmoid. Nonlinear functions are in the context of deep learning often called activation function and allow the network to respond nonlinear to the given input data which is necessary to solve non trivial problems. In this structure of the neural network the origin of the term "deep learning" can be found. The depth of a network is defined by the number of interconnected layers. With the increasing number of layers the applied network gained in depth and the term deep learning for training these deep neural networks was created.

Deep learning networks were successfully applied in computer vision, speech recognition, machine translation, social network filtering, playing board games, video games and medical diagnoses [62]. With the fast-growing number of application fields of deep learning different types of network structures were proposed. The most common network structures are feed-forward neural networks, recurrent neural networks and convolutional neural networks. For more information about deep learning in general and network structures we refer the reader to the appendix A.1 and the book "Deep Learning" by Goodfellow, Bengio, and Courville [19].

2.2. 3D Data Representation

In the past ten years smart phones developed from a luxury item to an essential tool in daily life. In January 2018 the estimated number of smartphone users was around 2.5 billion [40]. Most of these smartphones are equipped with a camera and so the sheer number of low-cost cameras accelerated development in the fields of image processing, recognition and segmentation. The same trend can be seen in 3D scanning technology. Recently devices able to capture 3D data became available, ranging from smartphones with built-in stereo cameras, game consoles (most revolutionary the Microsoft Kinect), Light Detection And Ranging (LIDAR) systems in modern cars and much more.



(a) Point cloud. Data Source [60] (b) Voxel. Data Source [3] (c) Polygon mesh Data Source [47]

Fig. 2.1.: Illustration of different 3D data representations

To understand the acquired 3D data it is necessary to gain a brief overview of the most common available 3D data representations.

2.2.1. Point Cloud Data

A point cloud consists of a list containing points. Each point is defined by its geometrical position, most common the cartesian coordinates in correspondence to a reference coordinate system. A individual point can contain additional information e.g. color data, surface normal or measurement accuracy. Most 3D scanners' unprocessed output data is a point cloud. The key goal of 3D data processing is extracting higher level features from raw data represented in a high dimensional space e.g. given by the parameters of multiple points. The unorganized point cloud data structure includes a high amount of redundancy, because e.g. one flat table surface is described by hundreds of points. The classical approach for extracting higher level features out of this data is to perform filtering, noise reduction and down sampling on the raw point cloud. Point clouds are also hard to manipulate and handle, so this data format is commonly converted to one of the following different 3D data representation to perform additional processing.
One example how higher level visual features are extracted form real world data is given by Kaiser et al. [28]. The authors introduced a perceptual pipeline that provides a robust and reliable perceptual mechanism for affordance-based action execution starting from raw point cloud data provided by LIDAR

systems and RGB-D cameras. It is important to keep in mind, that every one of the briefly introduced data representations comes with unique strengths and drawbacks when applied to different tasks.

2.2.2. Voxel Data

A normal image can be described by a two-dimensional grid of pixels. Each pixel stores one value (greyscale) or multiple values for different channels (RGB). This principle can be intuitively extended to a three-dimensional uniform cartesian grid, where we divide the observed space into smaller segments (voxels), with each voxel representing a discrete volume located by its cartesian grid coordinates. By sorting the voxel into a three-dimensional grid we keep the spatial relationship between the voxels and achieve an organized 3D data format. A voxel can contain information such as color, density or other data. The voxel representation comes with two major limitations. First the reconstruction loss for curved shapes and objects is very high. Secondly the computational complexity rises cubically (N^3) with the voxel resolution. Despite these disadvantages voxels are commonly used in medical imaging methods and computer games. The structural similarity of voxel and image data allows the usage of similar algorithm and applying previously established image processing methods can be done easily. In the following we will refer to voxels as volumetric units that are either filled (on) or empty (of).

2.2.3. Polygon Mesh Data

Another approach to building 3D models is to use polygons to describe surfaces. A polygon mesh is defined by a set of vertices, edges and faces. The research on polygon meshes is a large subfield of computer graphics with a variety of applications. The advantage of polygon meshes is the low reconstruction loss for curved objects and the excellent hardware acceleration, especially available for triangle meshes. This fast rendering makes polygon meshes the method of choice for real-time computer graphics and is used in computer aided design or video games, just to name a few out of many areas of application. For more technical details about polygon meshes we refer to "A Mesh Data Structure for Rendering and Subdivision" [58], where a brief introduction about polygon meshes is provided in a clear and graphically sophisticated way.

2.2.4. Other Methods

The field of computer graphics is an active research topic including different approaches and methods that have been applied to spatial data. It is difficult to give a complete overview about this big topic. But in the context of deep learning and data acquisition these 3 data representations cover a high percentage of the recent publications. Still worth mentioning are methods based on Partial Differential Equations (PDEs) to describe surfaces [59], Level-set methods (LSM) [41] used for numerical analysis of surfaces and shapes, surfels [43], Green functions and NURBS[1] [35].

[1] Non-Uniform Rational B-Spline

2.3. Deformation

Deformation in material science describes the change in shape or size of objects caused by an externally applied force or a shift in temperature [64]. We only need to consider deformation due to an externally applied force for this thesis and therefore we will neglect thermally caused deformations in this summary. The applied force can be classified by its point of contact and direction in tensile (pulling), compressive (pushing), shear, bending or torsion(twisting) (Fig. 2.2). Material behaviors can be divided to in elastic and plastic. Elastic deformation recovers to its original shape after the force responsible for the deformation of the object is released. This deformation behavior is referred to as reversible. Plastic deformation on the other hand appears when the force applied to the object is the reason for constant deformation (irreversible), even when the external force is released. When the deformation of a material is neglectable under a certain force, then the object can be described as rigid and there is no need to model or to take deformation into account.

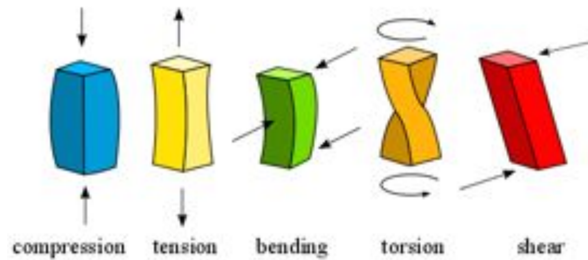


Fig. 2.2.: Schematic deformation caused by different forces. Data Source [52]

Materials can show elastic and plastic behaviors at different ranges of load. There are many factors, that decide which type and form of deformations occurs to an object. Therefore different material behaviors can be observed for rubbers, plastics or metals. A classic deformation example is a tension test, in which a certain pulling force is applied to a probe (often a metal rod) in order to measure the deformation of the object and thus calculate unknown material properties. A schematic example for this can be seen in the graphic 2.3.

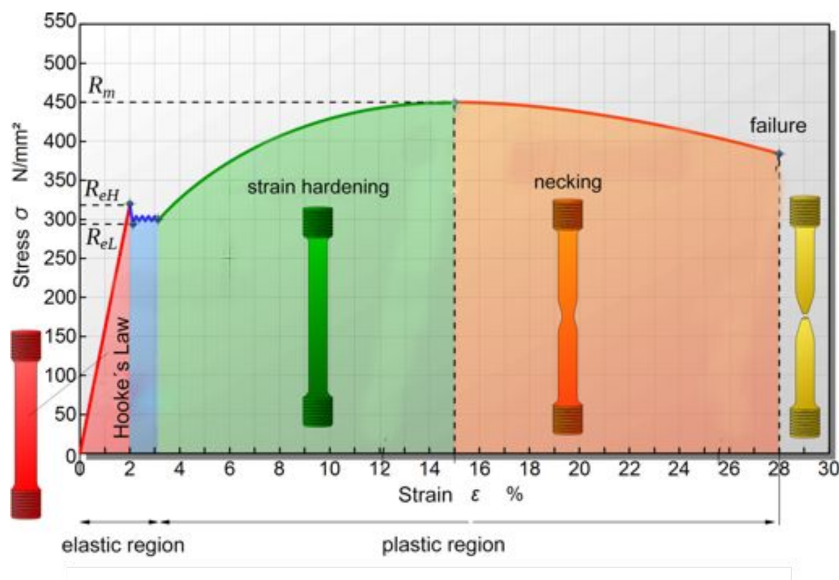


Fig. 2.3.: Tension test schematic. Stress/strain graph. Based on [10]

To describe deformation, defining stress and strain is necessary. Stress is the force per area applied to an object, while strain is the relative deformation in relation to the equilibrium state of the object. In figure 2.3 the schematic results of a tension test for a ductile metal can be seen. At first in the elastic

phase the object deforms nearly proportionally to the applied stress. This can be described by Hook's law [2, Chapter 2.2.1 Page 16], and the object is able to recover to its original shape if the force is released (this region is marked in red and partially blue in the graph). After exceeding a certain force, non-reversible deformation (plastic deformation) occurs until the mechanical stress is too high and the probe breaks apart. Different behaviors can be observed for other materials. There are different laws and material constants that help to describe the change in shape or size of objects in continuums mechanics. Since our research is clearly focused on statistical deep learning and investigating a novel approach for deformation modelling more knowledge about classical material science in the engineering context is not necessary. Herakovich [24] provides more information about elastic materials in his book "A Concise Introduction to Elastic Solids".

3. Related Work

Despite all the efforts spent in modelling deformation over the past decades, all proposed approaches underly certain limitations or restrictions. As a result, deformation modelling remains an active research topic with open issues and unsolved problems. This chapter introduces the motivations behind the proposal of our new method by giving an overview of the recent related work in modelling deformation. Since we want to incorporate the recent advances made in the field of 3D deep learning, we introduce different concepts that have been successfully applied to processing 3D objects.

3.1. Deformation Modelling

Several different methods have been presented in the literature to capture the dynamics of deformation of non-rigid objects. One popular category of models to achieve this are mass-spring-damper models. Here the 3D object is split into smaller parts that are interconnected by springs and dampers. The parameters of the springs and dampers are adjusted to best fit the real-world deformation behavior. In [73] and [37] parameterization of mass-spring models was performed by applying external forces to an unknown deformable object and thus generating the necessary reference deformation data. The parameterized model then allowed performing further motion planning for robotics grasping and manipulation application, while taking the deformation of the 3D object into account. A similar approach was taken by Burion et al. [8] to identify different stiffness properties by particle filters. The downside of the mass-spring models is that the stiffness and spring parameters do not describe actual material properties and are not intuitively understandable.

In contrast, Finite Element Methods describe physical properties of objects. They utilize elastic theory between small finite parts of an object and only few parameters are necessary to describe the actual behavior of homogeneous objects. With solving numerical differential equations between these finite elements, the computational costs are higher and real-time or closed loop control methods are not easily implementable [5, Chapter 5, 12]. Frank et al. [15] were able to learn elastic parameters by comparing simulation FEM results to real robot data, and adjusting the simulation parameters to minimize the error between simulation and the real acquired data. A finite element model was fitted to pre-segmented point cloud data recorded by an RGB-D sensor to model elasticity in real-time by the authors of [42]. A similar approach was applied by Frank et al. [16]. In their research they focused on utilizing the deformation modelling method to navigate a wheeled robot and manipulate objects in real deformable environments. Another application and research field of deformation is modelling human tissues for surgical assistant systems. Haouchine et al. [22] applied a linear tetrahedral co-rotational FEM model to heterogeneous tissue deformation of the liver surfaces. In medical technology a variety of other methods to model deformation have been investigated in order to perform successful operations.

Non-uniform rational B-splines (NURBS) can also be applied to model geometry representation and Lau et al. [35] were able to implement a collision detection framework which takes deformable object into account by rendering them as NURBS. Meshless shape matching (MSM) is widely used in computer graphics and Giiler et al. [17] verified its capability to estimate the deformability of elastic materials by this method. A very new approach incorporating deep learning in a novel way is presented by Tawbe and Cretu [57]. They used Recurrent Neural Networks (RNN) to map force-torque data acquired while probing an elastic object to the actual spatial deformation and could predict based on a neural gas fitting approach the shape of the object under a certain applied force.

3.2. 3D Deep Learning

Deep learning has made tremendous progress and achieved outstanding results in the past decade. AlexNet [32], developed by the SuperVision group at the University of Toronto, was the first convolutional neural network (CNN) that set new benchmark records in the ImageNet large Scale Visual Recognition Challenge in 2012 [49]. In this challenge a dataset containing 10.000.0000 images labeled into over 10.000+ categories was provided. The task is to reliably classify the given image into the corresponding category. In 2015 ResNet [23], developed by Microsoft, even outperformed human image classification capabilities on this dataset.
Researchers at Princeton University brought the same challenge to 3D models with the ModelNet dataset. They gathered 127.915 voxel CAD models categorized into over 662 classes with annotated orientation. Because of this dataset 3D deep learning has focused on voxel representation processing. Zhirong Wu's [67], 3D ShapeNet, a convolutional deep belief network, maps the ModelNet dataset to binary propability distributions and is capable of shape completion. The authors of [7] have applied a VAE network to learn a discriminative representation for the ModelNet 3D objects. Extending a 3D VAE network with an CNN to handle voxel and image data allowed extraction of additional information into a generative vector representation of 3D objects in [18] and [38]. A similar idea has been applied to detecting landing zones for autonomous helicopters [38]. "Generative Adversarial Networks"(GANs) proposed in [20] have been successfully applied to 2D image data and Wu et al. [66] extended this idea to 3D voxel data. They showed the capability of GANs to generate 3D objects from a probabilistic space.

Yumer and Kara [71] utilized an approach that is based on object deformation data to learn deformation handles for mesh data representation, which was later extended by Yumer et al. [72]. Tan et al. [56] applied a mesh based autoencoder with the goal of robust intuitive extraction and interpretation of deformation components applicable to large scale deformations. In [70] the challenge to find deformation flows in volatilized objects was examined. The authors were able to tune the appearance of 3D voxel objects and find novel shape representation of the initial object. These approaches clearly show that it is feasible to find a meaningful low dimensional representation of complex voxel objects and varieties in shape. For an extensive and general state of the art overview of shape analysis and processing methods we refer to the report by Xu et al. [69].

4. Dynamical Deformation Network

Through the recent development and advances made in deep learning, extending the application areas of 3D deep learning to model deformation seems feasible. To achieve this we will at first introduce the Variational Auto-Encoder (VAE) and Kalman Variational Auto-Encoder (KVAE) frameworks. These two frameworks build the foundation for the proposed "Dynamical Deformation Network" (DDN) that is able to capture deformation over time in voxel data. We decided to represent 3D object as voxels since the recent work made by the 3D deep learning community focused on this data structure, and the similarity to image data allows us to apply methods developed for image processing. In the following we will clarify the benefits and limitations of the VAE and KVAE framework. Utilizing the knowledge about both networks and the constraints of deformation will lead us to the implementation of the DDN. We will explain how this novel network benefits from the recent advantages achieved in deep learning and provide an overview over the concerns, that came up, while implementing the network. The DDN should not be seen as an alternative to the existing modeling methods previously introduced. It is supposed to take the first step in applying deep learning to the field of deformation modelling and with additional future work, deep learning might be able to complement the classical existing deformation modelling methods.

4.1. Introduction to Variational Auto-Encoders

"VAEs are appealing because they are built on top of standard function approximators (neural networks), and can be trained with stochastic gradient descent. VAEs have already shown promise in generating many kinds of complicated data, including handwritten digits [30, 51], [30, 48, 33], house numbers [31, 21], CIFAR images [21], physical models of scenes [33], segmentation [55], and predicting the future from static images [61]." [11]
Images and voxel data include a high amount of redundancy and unimportant information. If we take a $16x16x16$ voxel scene, with every voxel being either on or off, there are $2^{16x16x16}$ possible combinations of this scene appearing, but only a few of interest. Humans are great at understanding the semantics in this complex representation. This stays a simple task even if the shape or appearance of objects differ. We are able to extract semantically meaningful data out of these data structures and simplify the high dimensional voxel or image space intuitively. The variational auto-encoder tackles the same task of dimensionality reduction and is able to extract low dimensional features out of the original data in an unsupervised manner. It belongs to the categories of "Generative Models", which try to fit a probability representation to a given dataset. To introduce this framework, we follow the derivation presented in [11, 30, 36].

We make the assumption, that an observed datapoint $\mathbf{x} \in X$, is drawn from a distribution $P(\mathbf{x})$. The lower dimensional features of $\mathbf{x}$ can be described by latent variables $\mathbf{a} \in A$. The latent variables are unobserved and able to describe the important information in the original data. This latent representation is referred to as the bottleneck of the autoencoder.
For the VAE, we describe the latent space by a vector of random gaussian variables $\mathbf{a} \sim \mathcal{N}(\mu,\Sigma)$[1]. We can sample from the latent space by its probability density function $P(\mathbf{a})$ over A .
A noisy sample of the latent space is mapped by the decoder network $f_\theta(\mathbf{x}|\mathbf{a})$, implemented as a neural network, parameterized by ϕ, from the latent space A to the space Θ, which is similar to X.

$$f_\theta : \quad A \, x \, \Theta \rightarrow X \tag{4.1}$$

[1] $\mathcal{N}(\mu,\Sigma)$ ia a Gaussian Distribution with mean = μ and standard deviation = Σ

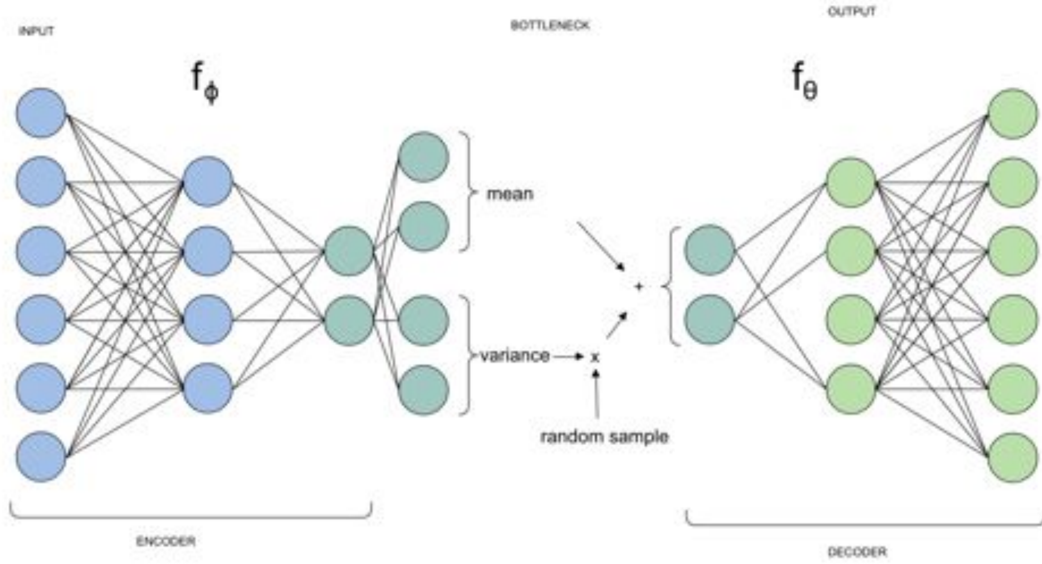


Fig. 4.1.: Variational Auto-Encoder schematic structure. f_θ and f_ϕ are implemented as neural networks. The last layer of the encoder network represents the mean and variance of the latent space. A random sample of the latent space is provided as the input to the decoder network. Data Source [36, Page 18, Fig. 3.2]

The encoder is a deterministic non-linear function $f_\phi(\mathbf{a}|\mathbf{x})$, parameterized by ϕ, and implemented as a neural network. Its function is to map the input X to the latent space A. The joint probability of the model is given by $P(\mathbf{x},\mathbf{a}) = P(\mathbf{x}|\mathbf{a})P(\mathbf{a})$. According to Bayes law the posterior is given by the following equation:

$$P(\mathbf{a}|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{a})P(\mathbf{a})}{P(\mathbf{x})} \tag{4.2}$$

$P(\mathbf{a})$ is the prior belief about the distribution. $P(\mathbf{x}|\mathbf{a})$ the likelihood and $P(\mathbf{a}|\mathbf{x})$ is the posterior. With the law of total probability we can obtain $P(\mathbf{x})$ by the following equation:

$$P(\mathbf{x}) = \int P(\mathbf{x}|\mathbf{a};\phi)P(\mathbf{a})\, d\mathbf{a} \tag{4.3}$$

To train the model we maximize the log-likelihood of $P(\mathbf{x})$. The logarithm is a monotone function and so by maximizing the log-likelihood instead of only $P(\mathbf{x})$ the same results are obtained.

$$\log P(\mathbf{x}) = \int \log\ P(\mathbf{x}|\mathbf{a};\phi)P(\mathbf{a})\, d\mathbf{a} \tag{4.4}$$

Integrating over the marginalized joint probability distribution $P(\mathbf{x},\mathbf{a})$ is computationally not feasible, since we have to integrate over the latent space A. This problem can be solved by applying the method of variational inference proposed in [30] and modelling the true distribution $P(\mathbf{a}|\mathbf{x})$ by a simpler Gaussian distribution $Q(\mathbf{a}|\mathbf{x})$. Minimizing the difference between both distributions can be achieved by using the Kullback-Leibler (KL)-divergence as a metric of similarity between the distributions. In the following we summarize the derivation given in [30].

The KL-divergence is defined by:

$$KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a}|\mathbf{x})\ ) = \mathbb{E}[\ \log Q(\mathbf{a}|\mathbf{x})\ -\ \log P(\mathbf{a}|\mathbf{x})\ ] \quad [2] \tag{4.5}$$

By applying Bayes' rule 4.2 to 4.5, the KL-divergence can be rewritten as:

$$KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a}|\mathbf{x})\ ) = \mathbb{E}[\ \log Q(\mathbf{a}|\mathbf{x})\ -\ \log P(\mathbf{x}|\mathbf{a})\ -\ P(\mathbf{a})\ ]\ +\ \log P(\mathbf{x}) \tag{4.6}$$

$\mathbb{E}[\ \log Q(\mathbf{a}|\mathbf{x}) - \log P(\mathbf{x}|\mathbf{a})\ ]$ on the right hand side of equation 4.1 is another KL-divergence. This leads us to the following equation:

$$\log P(\mathbf{x})\ -\ KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a}|\mathbf{x})\ ) = \mathbb{E}[\ \log P(\mathbf{x}|\mathbf{a})]\ -\ KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a})\ ) \tag{4.7}$$

$Q(\mathbf{a}|\mathbf{x})$ is the encoder network and $P(\mathbf{x}|\mathbf{a})$ is the decoder network of the VAE. The original goal was to maximize $P(\mathbf{x})$ in relation to the encoder and decoder parameter. Maximizing the equation on the right side correspondences to finding the lower bound of $P(\mathbf{x})$, because $KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a}|\mathbf{x})\ )$ is always greater than zero. Maximizing over $\mathbb{E}[\log P(\mathbf{x}|\mathbf{a})]$ is a common maximum likelihood estimation. For this the loss function can be evaluated e.g. by log-loss, regression-loss or cross-entropy and is referred to as the reconstruction loss. We also have to maximize over $KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a})\ )$. The reason why we previously chose to set the latent space to be Gaussian distributed can be obtained in this step. Calculating the KL-divergence between two multivariate Gaussian results in another multivariate Gaussian. It is worth mentioning that setting the latent space to other probability distributions is possible, but not common.

$$KL(\ Q(\mathbf{a}|\mathbf{x})\ \|\ P(\mathbf{a})\ ) = KL(\ \mathcal{N}(\mu(\mathbf{x}),\Sigma(\mathbf{x})\ \|\ \mathcal{N}(0,1)\ ) = \frac{1}{2}\sum(exp(\Sigma(\mathbf{x})) + \mu^2(\mathbf{x}) - 1 - \Sigma(\mathbf{x})) \tag{4.8}$$

The derivation for this can be found in [38]. The KL-divergence is referred to as the regularization term of the VAE. The main advantage of this method is that it allows us to apply back propagation. Therefore calculating the gradient of the network with respect to the loss function is possible. This allows us to apply stochastic gradient descent to the network and find the optimal network parameters.

The VAE has been successfully applied to a variety of different applications. The ability of the VAE, implemented with the correct decoder and encoder structure, to find a low dimensional meaningful representation of the training data has been shown in multiple previous works. Of course, this network comes with limitations and restrictions. Since this is a completely unsupervised learning method we are not able to choose which features we want to extract out of the data, and often the latent space does not encrypt data in a way that is semantically meaningful to humans. Another issue related to implementation of the decoder and encoder function as highly expressive neural networks, is the potential of over-fitting the given training dataset.
In summary the VAE allows learning latent variables of a voxel scene. Since we want to investigate deformation over time and focus on the appearing dynamics, a more complex model is necessary to learn the dynamics and map the appearance and shape of the object over time. For this we will introduce the Kalman Variational Auto-Encoder network in the next section.

4.2. Introduction to Kalman Variational Auto-Encoder

To understand the concept of the Kalman Variational Auto-Encoder framework (KVAE), it is important to understand Linear Gaussian State Space Models (LGSSM), Variational Auto-Encoders (VAE) and Kalman filtering. Since Kalman filtering is a more complex topic which cannot be briefly and at the same time mathematically adequately summarized, we would refer someone new to the topic to the tutorial provided by Bishop, Welch, et al. [6]. The KVAE tries to incorporate the benefits of a statistical framework and deep learning by reducing high dimensional input space with a VAE to a latent space,

[2] $\mathbb{E}$ denotes expected value

which then can be modeled over time by the LGSSM. In figure 4.2 we can see four different vectors over

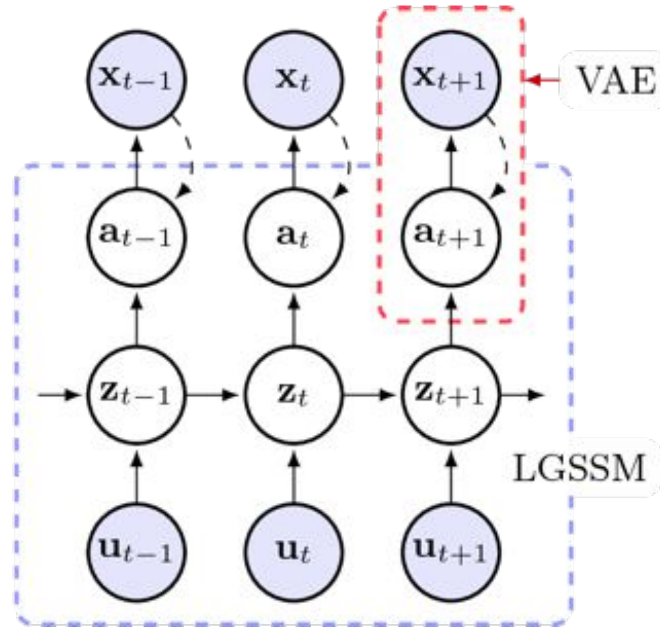


Fig. 4.2.: A KVAE is formed by stacking a LGSSM (dashed blue), and a VAE (dashed red). Shaded nodes denote observed variables. Solid arrows represent the generative model (with parameters θ) while dashed arrows represent the VAE inference network (with parameters ϕ). Data Source [14, Page 2, Fig. 1]

time. $\mathbf{x}_{1:T}$[3] is a sequence of high dimensional observed input data, normally captured by a sensor. In our case, this will be the voxel data observed by a depth camera. The VAE relates the latent variable $\mathbf{a}_t$, also referred to as the pseudo-observation, to the observed input $\mathbf{x}_t$. The decoder network $p_\theta(\mathbf{x}_t|\mathbf{a}_t)$ and encoder network $q_\phi(\mathbf{x}_t|\mathbf{a}_t)$ are constant over time. The emission matrix $\mathbf{C}_t$ establishes the connection between the pseudo-observation $\mathbf{a}_t$ and the hidden state $\mathbf{z}_t$. The state transition matrix, given by $\mathbf{A}_t$, connects the hidden variables $\mathbf{z}_t$ over time, and therefore is the key element to describe dynamics in the latent space. $\mathbf{u}_t$ can be observed and interpreted as an action or control input to the model. When we later apply this model to deformation, the action input might be the force or the location of the force applied to an object. Referring to $\mathbf{u}_t$ as the control signal is justified, because $\mathbf{u}_t$ is able to directly influence $\mathbf{z}_t$ by the control matrices $\mathbf{B}_t$. The LGSSM is parameterized at time t by γ_t = [$\mathbf{A}_t$, $\mathbf{B}_t$, $\mathbf{C}_t$]. By adding the covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ which represent the noise of the process and measurement, we obtain the two following important equations for the LGSSM:

$$p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t;\mathbf{A}_t\mathbf{z}_{t-1}+\mathbf{B}_t\mathbf{u}_t,\mathbf{Q}), \quad p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t) = \mathcal{N}(\mathbf{a}_t;\mathbf{C}_t\mathbf{z}_t,\mathbf{R}) \tag{4.9}$$

The joint probability of this model is given by:

$$p_\gamma(\mathbf{a},\mathbf{z}|\mathbf{u}) = p_\gamma(\mathbf{a}|\mathbf{z})p_\gamma(\mathbf{z}|\mathbf{u}) = \prod_{t=1}^{T} p_{\gamma_t}(\mathbf{a}_t|\mathbf{z}_t)p(\mathbf{z}_1) * \prod_{t=2}^{T} p_{\gamma_t}(\mathbf{z}_t|\mathbf{z}_{t-1},\mathbf{u}_t) \tag{4.10}$$

By mapping $\mathbf{x}_t$ to $\mathbf{a}_t$, $\mathbf{C}_t$ does not scale with the dimension of $\mathbf{x}_t$. This becomes especially useful when performing inference. To do this, only inversion of a low dimensional matrix $\mathbf{C}_t$ instead of a high dimensional matrix, scaling with the dimension of $\mathbf{x_t}$, is required. With this structure we can now perform Kalman filtering and smoothing algorithms to the latent space. The LGSSM allows us to reconstruct imputed data of unobserved time-frames. Also, prediction of future time-steps can be performed. For

[3]time dependet variables are subscripted with a time-index

more information about how to actually perform the inference for KVAE and how backprorogation is effectively implemented we refer to the original paper [14].

A key feature of this model is handling nonlinear dynamics in the latent space. Since learning the correct parameters for γ_t for every time-step, becomes computationally infeasible for longer sequences. The proposed idea to solve this is to only learn a set of K time independent different matrices instead of t time dependent ones. A dynamic parameter network is able to find weights $\alpha_t^{(k)}$ for each matrix $\mathbf{A}^{(K)}$ for every time step. We can describe the time dependent matrix $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{C}_t$ by:

$$\mathbf{A}_t = \sum_{k=1}^{K} \alpha_t^{(k)} (\mathbf{a}_{0:t-1}) \mathbf{A}^{(k)}, \quad \mathbf{B}_t = \sum_{k=1}^{K} \alpha_t^{(k)} (\mathbf{a}_{0:t-1}) \mathbf{B}^{(k)}, \quad \mathbf{C}_t = \sum_{k=1}^{K} \alpha_t^{(k)} (\mathbf{a}_{0:t-1}) \mathbf{C}^{(k)} \tag{4.11}$$

The network, calculating α_t, is implemented as recurrent neural network RNN, consisting of long Long Short-Term Memory (LSTM) cells. For more details about LSTM cells the reader is referred to Hochreiter and Schmidhuber [25].

In summary, the KVAE framework allows us to learn complex dynamics in a latent pseudo observed space and computationally efficiently perform back propagation in the network, even for long time-series. The network is not only able to handle linear dynamics, it has the potential to perform well under non-linear boundary conditions, by its dynamical network. It includes the advantages of the VAE framework to efficiently reduce the dimensionality of the input space for fast performance and also common advantages of LGSSM including reconstruction of missing frames by applying inference, or performing prediction of the future state of $\mathbf{z}$.
Fraccaro et al. [14] provide an implementation of the KVAE in Googles deep learning framework Tensorflow and made it publicly accessible under `https://github.com/simonkamronn/kvae` [54]. This implementation is able to process sequential image data and was tested for mini computer games with hidden physical dynamics (pong game and others).

4.3. Overview and Implementation of Proposed Network

The implementation of the DDN is based on Fraccaro et al. [14] implementation of the KVAE. We extend this framework to handle 3D sequential voxel data, in order to model deformation processes.
The proposed VAE has the function to find meaningful latent variables of 3D voxel objects. The DDN is able to capture deformation dynamics appearing in the voxel input space by learning hidden variables that include dynamical deformation information. The action input allows the network to utilize additional information about the deformation. This model, with the ability to predict or infer missing time-frames is capable of reconstructing the behavior of real non-rigid objects.

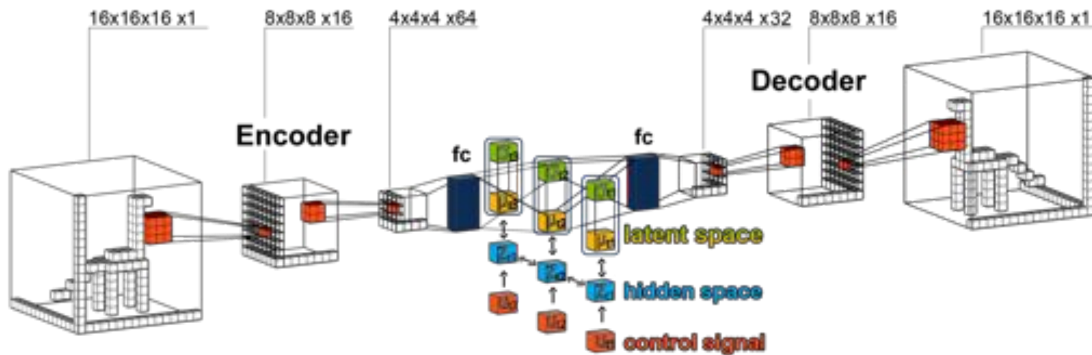


Fig. 4.3.: Dynamical Deformation Network Structure with simplified convolutional VAE

For the VAE we propose a volumetric CNN structure. Girdhar et al. [18], showed that CNN VAE structures are able to handle shape deformation in voxel data. We parameterize 3D convolution layers by its number of output channels, filter size and stride. While performing convolution we apply zero padding to keep fixed dimensions. For more details about convolution and its implementation we refer to the tutorial by Dumoulin and Visin [12]. In our architecture every 3D convolution layer is followed by a batch normalization layer.
We use leaky relu nonlinearity, proposed in [68] and applied to voxel data by Yumer and Mitra [70], with a negative slope of 0.2. For down-sampling we apply max-and average pooling on the same input layer. This is inspired by the voxception architecture introduced in [7], with the idea of providing the network with a higher expressiveness and allow multiple pathways of dataflow.
For both pooling operations we use a kernel size and stride of $[2,2,2]$. This results in the doubled amount of channels and reduces the voxel dimensionality by the factor of two. The $16x16x16$ architecture includes only two 3D convolutional layers. Since the DDN operates on this restricted resolution and should be able to reconstruct small shape varieties, we set the filter size of the first convolutional layer to [3,3,3] and for the second layer to [2,2,2]. The output of the last convolutional layer is provided to a fully connected layer with 256 neurons with sigmoid activation function. This layer is then mapped by a other fully connected layer to the latent space a_{mu} with no additional activation function and also to a_{var} by a fully connected layer with a sigmoid non-linearity.

The decoder implements the same 3D spatial convolutional structure as the encoder. We first added two fully connected layers then sequentially applied 3D upsampling and 3D upconvolution to the data. For the second upconvolution we doubled the amount of filters compared to the encoder, since we are not performing max and average "uppooling", which doubles the amount of filters. All layers are connected by leaky rule non-linearities and batch normalization. After the last convolutional layer two fully connected layers with 4096 neurons and leaky-relu activation function are added. One fully connected layer with sigmoid activation function followed by an additional Bernoulli function projects back to the 16x16x16 voxel space.

The reconstruction loss function for the VAE to compute voxel data is normally provided by the BCE. The authors of [7] discovered that weighting false-negative[4] and false-positive[5] reconstruction errors improved the performance of the VAE. Normally in the voxel space only a very small amount of the possible voxels are filled. So a static empty voxel space is already a good and easyly producible solution for the VAE. They introduced the following modified BCE with the hyper-parameter λ to avoid this:

$$L = -\hat{x}\,\lambda\,\log(\,x\,) - (1-\lambda)\,(1-x)\log(\,1-\hat{x}\,) \tag{4.12}$$

Here x is the original encoder input and $\hat{x}$ reconstructed decoder output. The implementation of the decoder and encoder structure in Tensorflow can be found in the appendix A.3.

[4]False-negative voxel: Wrongly classified voxel in the reconstruction sequence. In the original sequence this voxel is filled
[5]False-positive voxel: Wrongly classified voxel in the reconstruction sequence. In the original sequence this voxel is empty

5. Evaluation

In this chapter a detailed analysis of the DDN, split into two sections is given. The simulation section helps the reader to gain a clearer understanding of the problems that the DDN structurer faces and what the effect of certain parameters are. We also introduce different metrics for performance analyzing. In the following robot experiment section we apply the acquired knowledge to real world data and evaluate the network in two different experiments. This evaluation should test the basic capability of the network to model elastic deformation. Since we propose a complete novel strategy for this challenge, compared to the application area of FEMs and others, only simple experiments are validated.

5.1. Simulation

The simulation section of this thesis is not providing quantitative evaluation results of the proposed network. Instead this section is meant to help explaining and understanding the behaviour of the DDN, its limitations and possibility. There are two major reasons for this uncommon presentation of the simulation section. The first one is the lack of comparable fully data driven methods with the consequence that there are no standardized datasets available. The second reason for this is the iterative implementation of the DDN, in which we focused on different network tasks and behaviors for different simulations.
The first goal of the simulation phase was to extend the KVAE to voxel data. Fraccaro et al. [14] verified the capabilities of the KVAE via the “bouncing ball experiment”. In this experiment a video sequence, consisting out of a two dimensional planar ball movement with random starting position and direction, is given to the KVAE as an input. The results showed that the KVAE is capable of learning smooth dynamics in time-series and the ability of the network to handle non-linear dynamics.
In the first simulation we extend this experiment to the "3D bouncing ball" experiment with the challenge of learning non-linear dynamics that occur when the ball bounces off a wall in three dimensions. We modified and added a variety of different behaviours to this simulation game, to incrementally investigate the capabilities of this model . In preparation for real world deformation data modelling, in the second simulation we created a cube and deformed its surface on different faces over time. In the simulations we either used $8x8x8$ or $16x16x16$ voxel data. This reduced the training time and we were able to iterate and validate hyper-parameters faster. The structure of the $8x8x8$ and $16x16x16$ network only differs by the number of convolutions performed. The VAE structure changed in the development of the simulation phase, but the proposed model is capable of reproducing similar results.

5.1.1. 3D Bouncing Ball

We first introduce and visualize the generated dataset. Afterwards we discuss the influence of individual parameters to the training results, and introduce further evaluation methods for the network.

Dataset

The moving 3D ball represented by $3x3x3$ voxels can be described by a position vector $\mathbf{x}_t = [x, y, z]$ and velocity vector $\mathbf{v}_t = [v_x, v_y, v_z]$
The position $\mathbf{x}$ is initialized randomly at the beginning of every sequence. To capture the same dynamics that can be found in the 2D bouncing ball game we set $v_z = 0$ and $v_x, v_y \in [+1, 0, -1]$. To learn dynamics, v_x and v_y cannot be zero in the same sequence. Otherwise this would result in a constantly still-standing ball.

While the ball has no contact with the wall it follows a classic linear movement equation:

$$\mathbf{x}_{t+\Delta} = \mathbf{x}_t + \mathbf{v}_t * \Delta \tag{5.1}$$

When it hits the wall the velocity vector $\mathbf{v}_t$ is updated, mimicking a fully elastic impact. The training dataset consists of of 10000 random sequences with 20 time steps per sequence, to provide at least one nonlinearity in every sequence. This training dataset includes no noise.

Training Parameters

For training the model we used the Adam optimizer [29] with an $initial\,learning\,rate = 5^{-5}$, $beta_1 = 0.9$, $beta_2 = 0.999$ and $epsilon = 10^{-8}$. The batch size is fixed to 10 sequences over the whole training process, which consisted of roughly 100 epochs until the model converged. The total loss function of the DDN can be written as the sum of the VAE reconstruction loss, described by the edited Binary Cross-Entropy 4.12, the VAE regularization function 4.8 and the evidence lower bound of the Kalman Filter introduced in [14].

$$L_{total} = rec_{scale}\, L_{REC}(\lambda) + L_{BCE} + L_{KVAE} \tag{5.2}$$

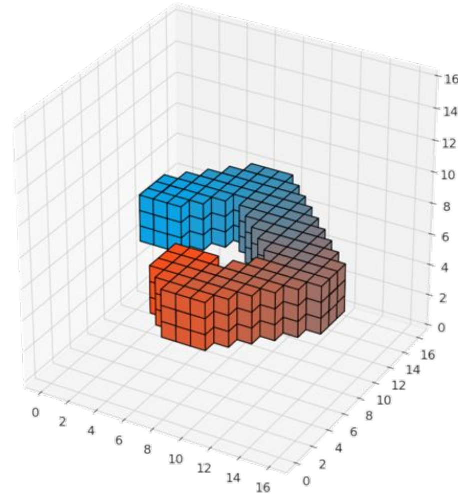


Fig. 5.1.: 3D bouncing ball heatmap of sample sequence movement over 20 time-steps

Adding the hyper-parameter rec_{scale} for weighting the reconstruction loss helps training the model. If rec_{scale} is set to a high value, the network focuses on adjusting the weights in order to minimize the VAE reconstruction loss. By setting the rec_{scale} to lower 1, the network updates the weights in favour of minimizing the KVAE loss and forces the latent space A to take the form of a unit Gaussian distribution. The λ parameter adjusts if the VAE is more likely to favor false-positive oder false-negative voxels. For this simulation setting lambda to 0.8 helped the network to learn the position of the ball. With this setting false-negative voxels are penalized stronger and learning to just reconstruct a static empty voxel space is avoided in the training process. Learning an empty voxel space is often a problem. In particular, for this experiment reconstructing an empty voxel space only results in 27 out of 16^3 false classified voxels.

Reconstruction

To measure the reconstruction performance of the DDN we have to compare the output sequence of the DDN to the original input sequence. One simple metric to measure the difference between these two sequences is to count the amount of wrongly classified voxels. This metric obviously doesn't actually capture how "wrong" the shape is displayed. It dosen't take into account where the voxel is wrongly classified. A wrongly classified voxel next to the surface of an object dosen't change its appearance, but a wrong voxel very far away from the object may do and so should be considered a worse mistake . Still this computational inexpensive and easy implementable metric gives a rough estimation of the network performance. Separating between false-positive and false-negative voxels also helps tune the parameters of the network.

The calculation of this reconstruction metric can be performed by the following python code:

```
def error_between_sequences(self, _orginal, _test):
        orginal = _orginal.astype(float)
        test = _test.astype(float)

        #only false voxels are marked
        wrong_voxels = np.absolute(orginal[:,:,:,:] - test[:,:,:,:])

        #calculate sum of all false negative voxels
        total_fn= np.sum(np.logical_and(wrong_voxels, orginal) )
        #calculate sum of all false positive voxels
        total_fp= np.sum(np.logical_and(wrong_voxels, test) )

        total_wrong = np.sum(wrong_voxels)

        print "Number Wrong Voxels: ", total_wrong
        print "Number False Positive Voxels: ", total_fp
        print "Number False Negative Voxels: ", total_fn

        return total_wrong, total_fp, total_fn
```

The reconstruction performance can also be seen in the reconstruction loss while training. In 5.2 the VAE reconstruction and regularization loss are plotted over the training period with rec_{scale} set to 0.5. Figure 5.2 illustrates three reconstruction samples at different reconstruction losses. Comparing the reconstruction loss to the average number of wrongly classified voxels per sequence in table 5.1 a common trend can be observed, despite the relationship between the two error functions not being linear.

Table 5.1.: Reconstruction error in comparision to average wrong voxels per sequence

Rec loss	78	39	21
Avg. wrong vox.	1207	821	16

The learning behavior seen in 5.2 is typical for the VAE. First the reconstruction loss decreases very fast, and the regularization loss increases. By increasing the regularization loss the VAE is able to specify more in the latent space and e.g. reduce the noise. By setting rec_{scale} very high the VAE structure can even converge to a normal auto-encoder, because the noise Σ would be able to nearly decrease to zero. When the reconstruction loss converges the VAE tries to reduce the reconstruction loss. This can be seen as generalization of the VAE.
For the bouncing ball experiment we compare different settings of network hyper-parameters. We found a good reconstruction error and smooth latent space for dim_a = 5. In [14], the authors adjusted the dimensions according to the problem scenario. In the 2D bouncing ball game, they set dim_a = 2 to encrypt information about the ball location x and y. They also set the dim_z = 4 to additionally store the information about the velocity of the ball. In the 3D bouncing ball game, we could not find these low dimensional latent space setting that directly describe the real world problem scenario.

An important property for latent space, which is the fundament for the modelling capability of the DDN, is the arrangement in the latent space. The smooth moving ball in the voxel space X should lead to an smooth space A. According to this, two very similar voxel configurations should produce similar multivariate Gaussian. A first impression can be obtained by plotting all dimensions of a single sequences latent space over time. In figure 5.3, we can clearly see the smooth movement in the latent space which corresponds to the smooth movement of the ball in the voxel space.
Another often used validation tool is to visualize the latent space by random sampling in the latent space

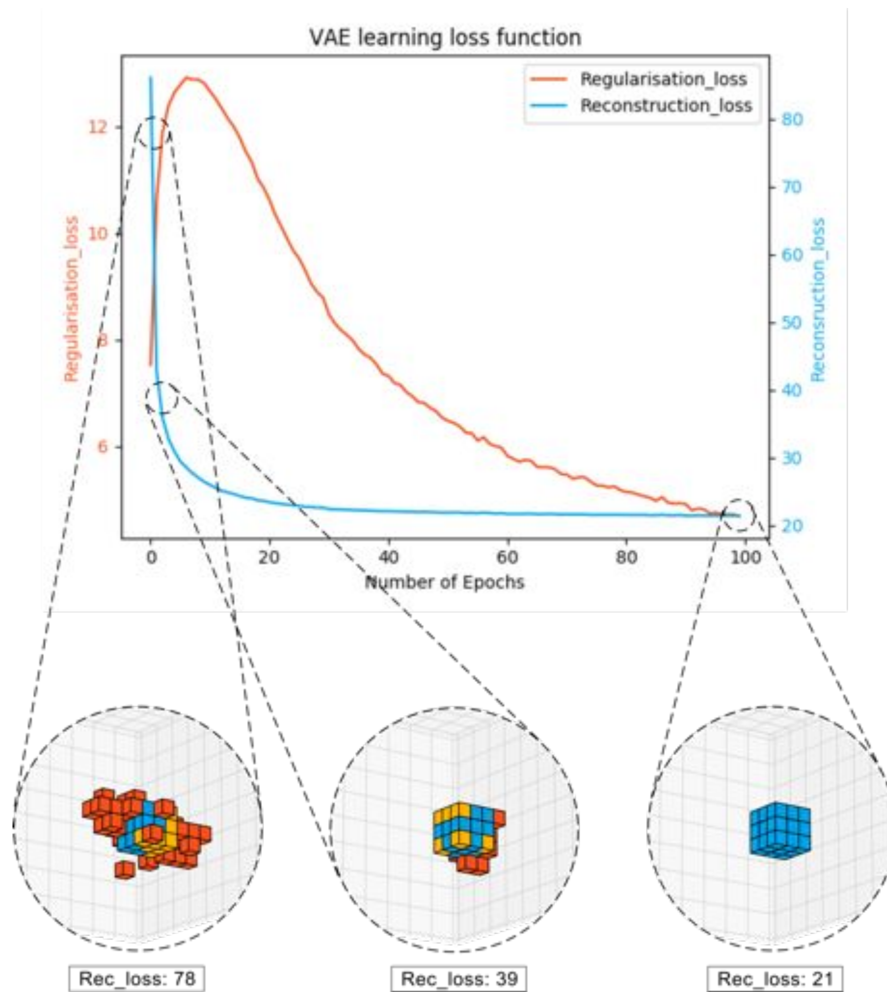


Fig. 5.2.: 3D bouncing ball reconstruction loss compared for different learning stages. In the voxel scene the orange voxels stand for false-positive and the yellow voxels for false negative reconstruction errors.

and plotting the reconstruction results to the corresponding 1D or optional 2D A-plane. With this method only two dimensions can be varied and evaluated at the same time. If existing, the correlation between A and X can be directly observed in the A-plane. Similar to this for low dimensional latent spaces we can cluster the training dataset to the A-plane. Still both methods are not attractive, because plotting a single view point of a 3D voxel space does not properly illustrate on paper the whole scene with all interesting features in it.
Linear interpolation of the latent space can be used for higher dimensions of A. For this we set a start and stop "voxel scene" and calculate the corresponding a-configuration. We can now generate a sequence of the linear interpolation between those two a-configurations. If, for the trained network, this results in a smooth movement, it suggests that the VAE learned to correctly sort the features into the latent space. A sample of this linear interpolation can be seen in figure 5.4. Here the ball is moving linearly from one side of the voxel space to the other. In the middle of the interpolation sequence is one jump at the wall, which still supports the assumption that the VAE learned a smooth latent space for this experiment. The reason for this is that the start and stop ball are on different heights and because we only trained for 2D movement, this is exactly what we would expect to see. A direct 3D movement of the ball cannot be expected since sorting the latent space in this way would not provide additional benefits to learn the appearing dynamics in the scene.

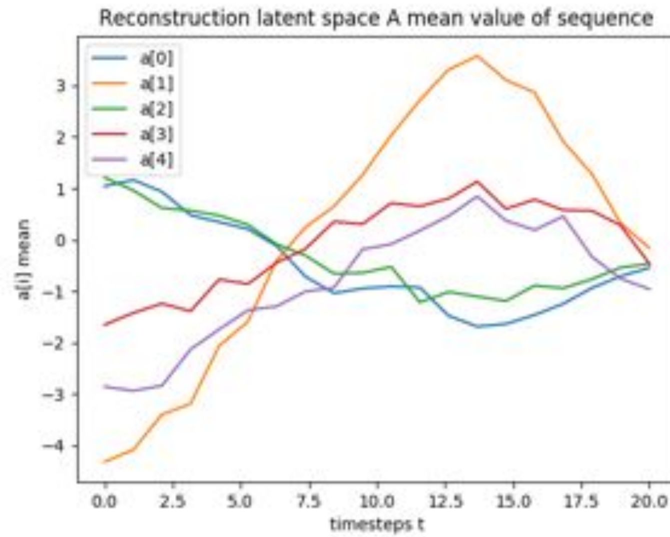


Fig. 5.3.: 3D bouncing ball latent space A sample over time

Although we will not apply Principal Component Analysis (PCA)in this thesis, it is often used in machine learning data visualization and worth mentioning. Shlens [53] describes PCA as "a simple, non-parametric method of extracting relevant information from confusing data sets". An implementation of this tool is directly provided in Googles Tensorboard deep learning visualization tool. The same methods and visualization tools introduced for the latent space A can be applied to the hidden space Z.

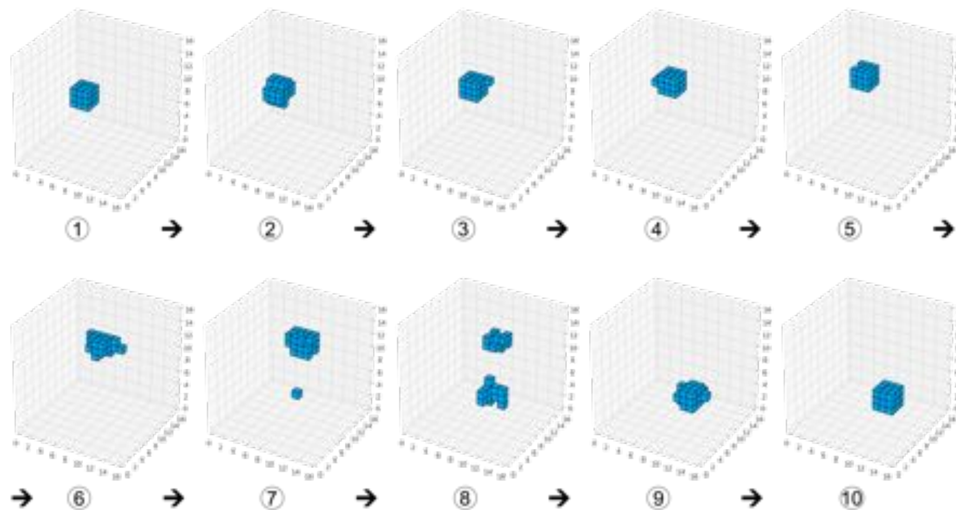


Fig. 5.4.: 3D bouncing ball linear interpolation in latent space A

Dynamics

After we are familiar with the VAE and its evaluation tools, we can have a closer look at the KVAE. Two important parts of the KVAE configuration have shown to determine whether the DDN is able to learn the hidden dynamics, and is able to utilize the full potential of the network. The first part is the dimension of Z, which can be evaluated with the same methods as A for the VAE. The correct configuration of the dynamical network, is the second major influence factor. As a short reminder, the dynamical network has the function to produce coefficients α_t^k. α_t^k sets the percentage of each individual transition matrix $A^k \in A^K$, $B^k \in B^K$, and $C^k \in C^K$, to be used for transition to the next time step. Learning the correct set of matrices and dynamical network configuration, can be interpreted as learning the underlying physical dynamics.
There are two scenarios where the DDN has to recourse to the learned dynamics: Imputation[1] and prediction. This network is especially useful for imputation, since the KVAE is able to perform inference on all observed time-steps. We can define imputation by its starting point t_{start} and the amount of unobserved frames t_{steps}. The same can be applied for prediction where t_{start} is the last observed frame and n_{gen} the total amount of frames to generate. This is graphically illustrated in 5.5.

Fig. 5.5.: Imputation [5-15] and prediction [5-20] schematic illustration

For the bouncing ball game setting dim_z to 7 or higher achieved the best results. For imputation the KVAE is able to apply Kalman smoothing or filtering on the hidden space Z. In our experiments applying filtering to the hidden space does not show better results than smoothing, since there are multiple nonlinearities which filtering only blurs. The setting of K shows high impact on the performance of the network. With the setting of K = 4 the DDN wasn‘t able to handle the non-linear dynamics in the scene. Providing a higher expressibility of the dynamical networks and allowing to learn nine different sets of matrices led to good overall performance. The network learned to choose the correct transition matrices for the linear movement appearing in the middle of the box and special constraints for the bounce of the wall. The resulting network performance can be seen in figure 5.6. In this figure 5.6, *REC* stands for the direct reconstruction result of the VAE. The average number of wrongly classified voxels per sequence needs to be set into perspective to the $16x16x16$ voxel space. There is a total of $16 * 16 * 16*$ time-steps $= 81920$ possible wrongly classified voxels in one sequence. 100 wrongly classified voxels corresponds to an error rate of 0.122%. The achieved results show that the VAE is able to reconstruct the scene. Since no noise and only little variety was included in the training data these results can be expected. Furthermore a 3D cube is very simple object which can be easily reconstructed by the highly expressive decoder and encoder function. When we later attempt to apply this network on real data, we have to concern ourselves with over-fitting issues and the performance difference of operating on test or training data.

[1]Imputation in statistics describes a method to reconstruct missing data.

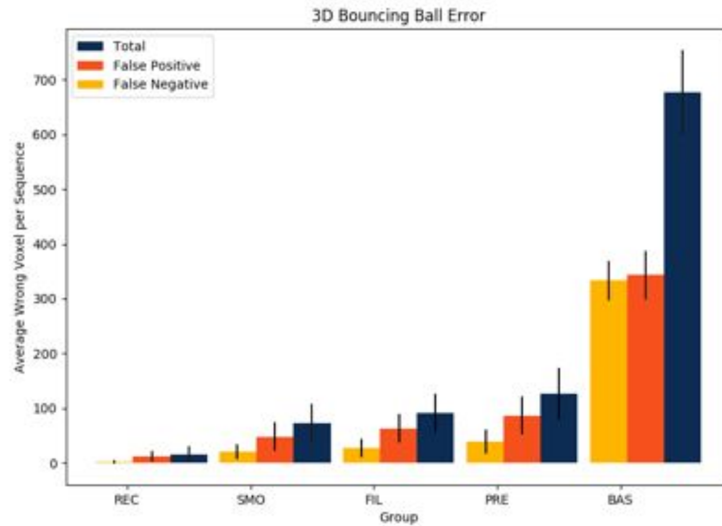


Fig. 5.6.: 3D bouncing ball imputation [5-15] and prediction [5-20] performance bar chart

The *BAS* bar provides an error baseline. This is computed by freezing the last observed time-step and comparing it, for the rest of the sequence, to the original input. Since the dynamical deformation network was able to learn the dynamics, prediction (*PRE* bar) results in a good result as well. Here we can see that filtering (FIL bar) in hidden space produces worse results than smoothing (*SMO* bar). This can be explained by the concepts of the Kalman-Filter. On the first few time-steps and after non-linear dynamics the error rate for filtering is higher than for smoothing. The blurred non-linearities in the hidden space, which influence the reconstructed voxel space, are the reason for bad performance after non-linear movements. The high initial uncertainty of the Kalman filter in the first few time-steps of every sequence is the reason for worse reconstruction results in the beginning when applying filtering. The prediction results are obviously worse than the imputation because less information is available for the DDN, but still outperforms the baseline by far.
As a next step for we tried to extend the movement of the ball to three dimensions. We could not learn these more complicated dynamics with the given network. Instead of applying further parameter tuning we created the next game, which is more related to deformation. Nevertheless we could show the abilities of the DDN to handle voxel data and learn dynamics even under non-linearities.

5.1.2. Box Deformation

We created the next game with the intention to clarify the ability of the DDN to handle local shape variety and learn the dynamics over time. We added a control signal which allows the network to incorporate additional information about the scene into the modelling process.

Dataset

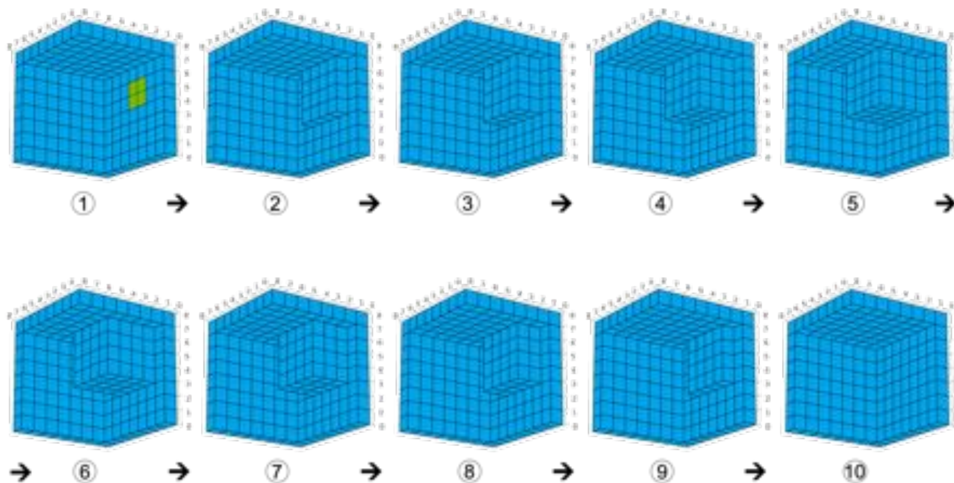


Fig. 5.7.: Box deformation sample sequence with touch point size of 2x2 and depth of 4. The green voxels in the first time-step mark the touch point.

For this simulation, we lowered the voxel-resolution to $8x8x8$. This allowed us to train the model faster. We created three surfaces of a cube and applied different deformations to it. The deformation shape applied should mimic a force pushing a certain amount of voxels next to each other into the object on one surface. The size of the touch point is randomly set between 1 and 5 for both coordinate axes and only refers to the amount of hollow voxels. This touch point can be localized anywhere on the cubes outer surface. The touch depth is randomly set between 1 and 4. After "pushing" into the box, the shape of the box recovers to its original state. One example sequence can be seen in figure 5.7. This time, to also utilize the full capabilities of the KVAE we applied an additional control signal to the network. We will discuss the setting for this control signal later. A total amount of 2000 sequences with 10 time-steps were generated for this simulation.

Network Parameters and Training

As previously mentioned, by changing the network voxel resolution adjustment of the number of convolution layers for the VAE framework is necessary. Changing the number of convolutions was the only major change applied to the DDN. To successfully train the model on the new data we have to perform hyper-parameter cross validations. Tuning the reconstruction cost function parameterized by λ and rec_{scale} to avoid finding static solutions of empty or completely filled voxel spaces, showed to be the fastest way of tuning the parameters. We previously adjusted $\lambda = 0.9$, to penalize the network if it did not reconstruct the ball, which only consisted out of $3*3*3 = 27$ voxels. Now with more voxels always being on in the scene, lowering this factor to 0.6 showed increase in learning performance. We also lowered the reconstruction error factor to 0.2 so the network focuses clearly on simplifying the latent

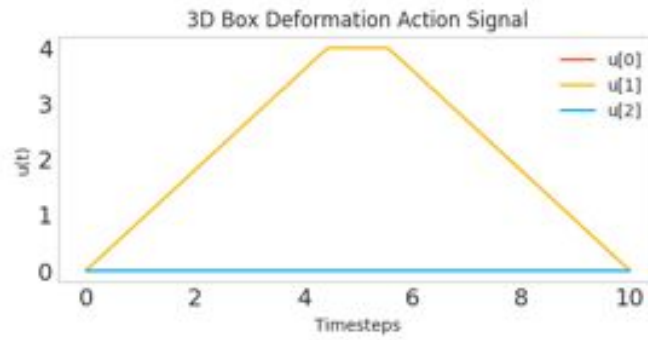


Fig. 5.8.: Box deformation control input U of sequence shown in 5.7

space and learning the best configuration for the DDN, while at the same time being able to lower the reconstruction cost continuously.
For dim_a = 10, dim_z = 16 and K = 9 we could achieve the best overall performance. The reduced number of network parameters allowed us to increase the Adam Optimizer learning rate to 0.0005. We trained the network for a total of 300 epochs.

Results

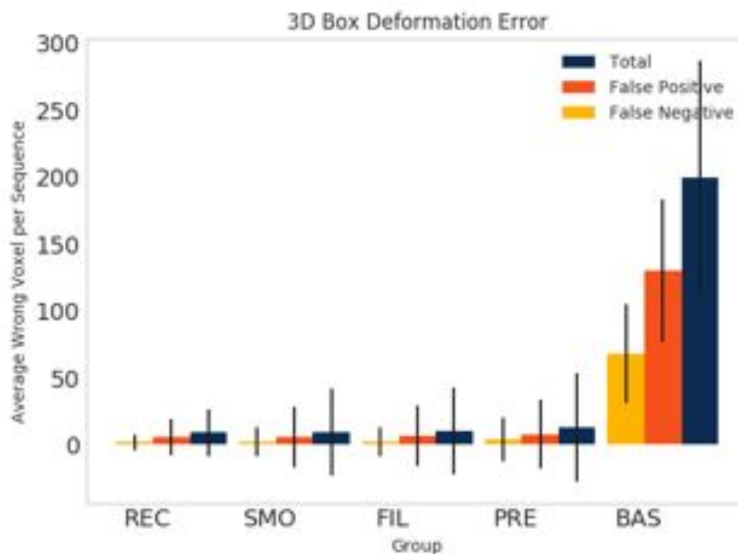


Fig. 5.9.: Box deformation imputation 3-8 and prediction 3-10 performance bar chart

We performed the same tests, previously introduced for the 3D bouncing ball game, on this dataset. We imputed the time-steps 3-8 and predicted time-steps 3-10. The control signal was provided at all time-steps to the DDN. The network learned to utilize the control signal 5.8 and could tell when the box begins to recover back to the initial state. By providing this additional knowledge about the scene good results that can be seen in 5.9 were achieved. The control signals had three dimensions (one for every surface of the half box). The value of each signal corresponded to the depth of deformation. Without applying the control signal, the network could only "guess" when the box would start to recover back to its initial state. So for a very good reconstruction result, it was essential for the DDN to learn how to utilize the given control signal.

By having a closer look at the VAE latent space reconstruction of the sequence 5.7, the property of symmetry can be found in the latent space 5.10 and the original sequence. This can obviously be suspected because of the symmetrical input sequence provided to the DDN.

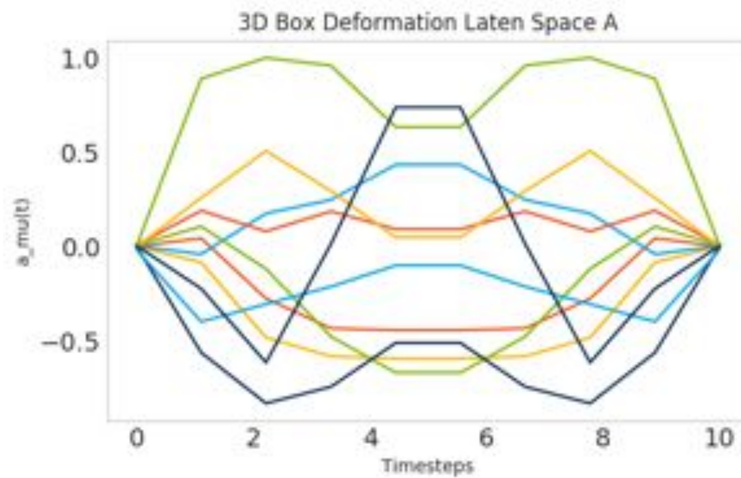


Fig. 5.10.: Box deformation latent space A of sequence shown in 5.7

By visualizing the hidden space in figure 5.11 for the same sequence, we can see that the property of symmetry is lost. This is exactly what we expect since the DDN has to encrypt information about the deformation movement into the hidden space. Nevertheless some lines in this sequence seem to be nearly symmetrical, and might encode the spatial symmetrical information of the scene. We cannot clearly interpret the hidden space to actual movement or the original voxel scene.

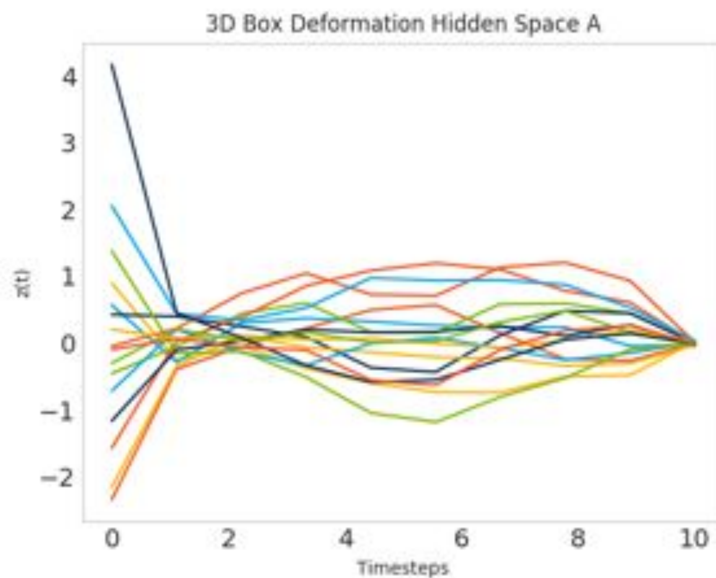


Fig. 5.11.: Box deformation hidden space Z of sequence shown in 5.7

5.1.3. Conclusion

In this simulation section we verified the capability of the DDN to reconstruct different object shapes. We showed that learning smooth and also non-linear dynamics can be achieved. The foundation for this was the capability of the convolutional voxel VAE to extract a meaningful latent space. Finding the right hyper-parameters for this network structure was not a simple task. We could not learn intuitively interpretable latent and hidden spaces. In the simulation phase only noise free data was applied and over-fitting the dataset wasn‘t investigated. To challenge this network on more complex data in the next section we will record real-world deformation data of multiple objects and show the capability of the DDN to capture real object deformation behavior.

5.2. Robot Experiments

To validate the DDN found in simulation, we perform real experiments on non-rigid objects. Since this is a completely data driven approach, large training and test datasets are required. To deform different object, we use the Universal Robot 5 (UR5) equipped with an Optoforce 3D force sensor. We choose to deform objects made out of materials, that recover there original shape within $3-6\,s$. To capture the spatial deformation, we use an Intel Realsense SR300 depth camera, with which we are able to capture a point cloud of the scene.

Fig. 5.12.: UR5 probing foam material to generate reference deformation data

To capture a large datasets, required for deep learning, integrating the robot and sensors into a high level abstraction framework helps to manage all the different tasks and data flow at the same time. For this we use the open source Robot Operating System (ROS), which allows us to easily deform the objects with UR5 and store the data of interest. For more information about ROS the reader is referred to the appendix, where a quick overview of the general structure is given.
To gain a better understanding of this experimental setup, a brief overview of every component and its specifications is provided in the following.

5.2.1. Robot Setup

Universal Robot 5

UR5 was developed by the Danish company Universal Robot. They specified on building collaborative robot arms, marketed as light weight, safe and easy to use. UR5 has 6 degrees of freedom through 6 revolute joints. Every joint can rotate from –360 to +360 degrees. The specified payload is 5 kg and more important for our use case is repeatability of +/- 0.1 mm. The main focus, while engineering this robot, was to develop a safe robot, which does not require to be separated from humans by a classic safety cage. This makes UR5 ideal for research and in particular for our experiment.
Controlling a robot arm can mainly be separated in two levels of control problems. On the one hand the low level controllers of the robot, which are implemented normally for every single joint separately. Their function is to set the right power for the motor to hold or move to a given angle or with certain required torque. This problem only concerns a individual actor unit and every controller operates at a high frequency between $100\,\mathrm{Hz}-1\,\mathrm{kHz}$. The implementation of these low level controllers is fully provided by the robot manufacturer. The high-level control has to calculate trajectories for the robot arm and find the right joint values needed to move to a given position. To plan trajectories for the UR5 the open source motion planning framework *MoveIt*! is used. It was developed in October 2011 for mobile manipulation,

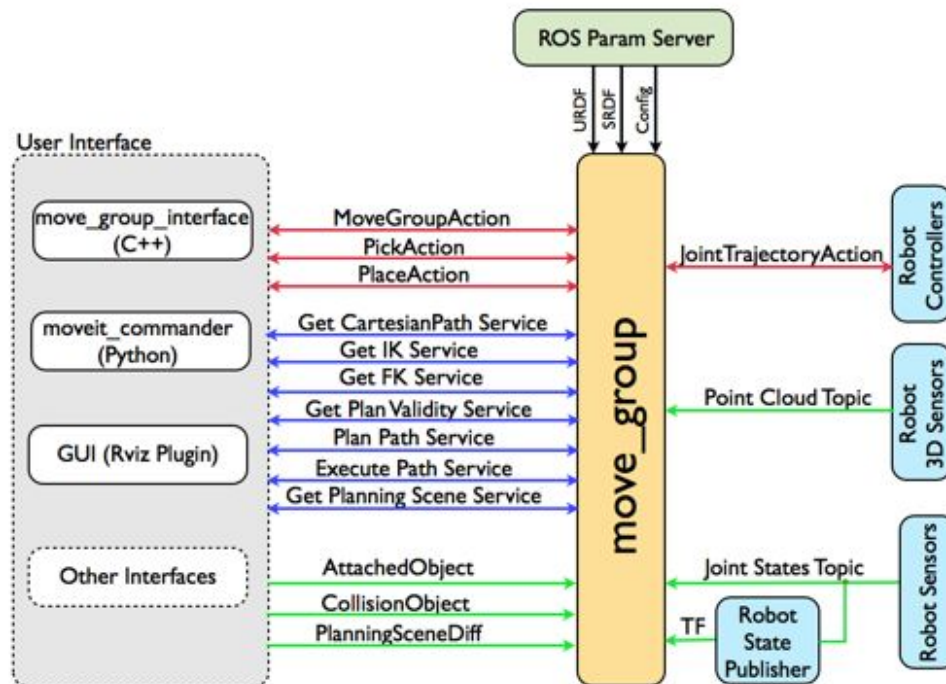
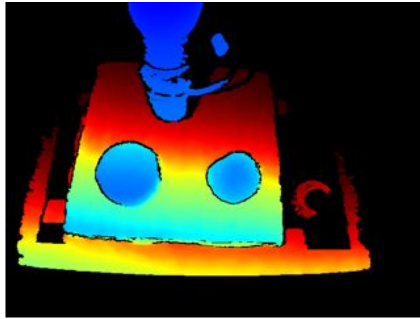


Fig. 5.13.: *MoveIt*! open source motion planing framework structural overview diagram. Data Source [26]

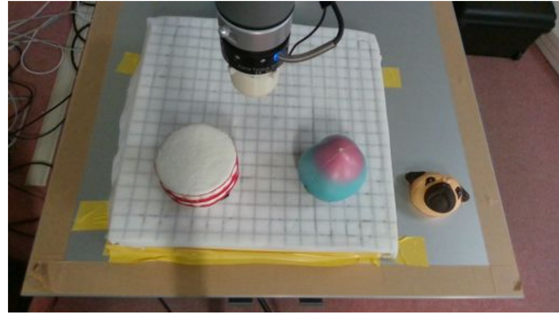
incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. [27]
In the *MoveIt*! overview figure 5.13, the robot is abstracted as a *move_group* and provides a wide range of user interfaces. The *RobotSensors* provide information about the robot joint values and joint torques. The *RobotController* is the interface to the manufacturer robot software. In our case we us the *ur_modern_driver* package to implement the bridge between the *move_group* and UR5. To find the relationship between the euclidean space, the robot is operating in, and the joint values (position or angle of each actor) we have to solve the inverse kinematics (IK) problem [9]. *MoveIt*! directly provides different IK-solvers, which are able to handle given constraints, and can be configured through the user interface. With this setup we are able to manipulate the shape of the probing object.

Intel RealSense Camera SR300

The SR300 is a low-cost active Infrared Radiation (IR) 2.5D depth-sensing camera. With the active-IR distance measurement this camera is able to capture highly precise depth images in the range of $0.2\,\mathrm{m} \sim 1.5\,\mathrm{m}$. This camera is designed for gesture tracking, facial expression recognition, 3D scanning and dynamic background segmentation. [46]
The basic principle of active-IR cameras is to project, with an IR-diode, a certain pattern to the scene. The reflected pattern can then be captured by the camera. With the knowledge of the projected pattern and the captured warped pattern, triangulation can be applied to calculate the depth image of the scene. These sensors are referred to as 2.5D depth sensors, because they only can gather limited information about the visible side of objects in their range of view. The operating range is mainly limited by the low emission power of the IR-diode to prohibited health risks for humans and animals. The depth image is acquired by the Intel Realsense SDK. [45]
A stable ROS compatible driver is directly provided by Intel. The depth data is available as an RGB point



(a) Depth image of robot workspace



(b) RGB image of robot workspace

Fig. 5.14.: SR300 side by side comparision of RGB and depth image

cloud captured with 30 fps. We use the Point Cloud Library (PCL)[50], a standalone large scale, open project 2D/3D image and point cloud processing library, to crop the provided point cloud to the region of interest and down sample the point cloud with a voxel grid in real-time. After this we off-line smoothed the point cloud over time and transformed the data to voxels.

OnRobot OMD - 3 Axes Force Sensor

This 3D force sensor was developed by Optoforce and is able to measure the direction and magnitude of external forces applied to the sensor's surface with a high resolution and up to 1kHz sample rate. They use a unique approach to measure the applied force, by emitting light to a deformable silicon surface and sensing the reflection. These sensors are very common in the field of robotics and are available in multiple types to measure different force ranges. For our experiment we have chosen the 40 Newton version.

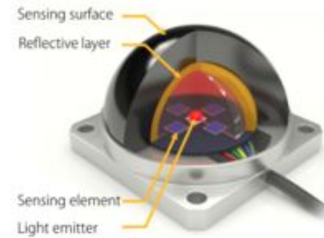


Fig. 5.15.: Function Principle OnRobot OMD - 3 Axes Forcesensor. Data Source [1]

5.2.2. Experimental Foam Deformation

The first experiment is designed to find simple dynamics and test the VAE performance on real data. The object is a small consumer pillow with dimensions 50 cm x 50 cm x 7 cm. We glued the pillow case to the interior foam material to capture the deformation of the foam. This pillow, since it is very big, allows us to capture local deformation. For this experiment we 3D printed the sphere-tool with the Optoforce sensor mounted in the middle 5.16. The sphere-tool is able to provide a bigger deformation area, which is easier to capture for the SR300.

Generated Data

We generate points of contact equally spaced in a 15x15 grid pattern with dimensions of 10 cm x 10 cm and repeated this procedure five times. After deforming the object softly, UR5 quickly releases the object and moves out of the scene, avoiding covering the pillow for the RealSense SR300. Focusing on the performance and generalization capabilities of the VAE, every touch was performed with the same depth and orientation onto the top surface. We generate for this object 1125 deformation sequences. We recorded the point cloud in the area of interest with

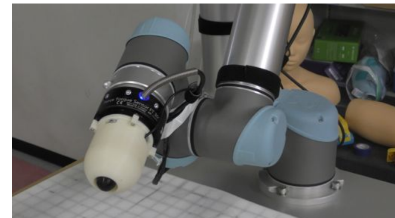


Fig. 5.16.: Universal Robot 5 equipped with Optoforce sensor and sphere tool

20 fps and rendered each deformation "action" into a sequence of 16x16x16 voxel data with ten total time-steps. A voxel was set on, when over four out of seven consecutive frames, more than three points were located in the voxel area. This can be seen as smoothing over time for data pre-processing and noise reduction. Additionally the Z-axis was scaled with a factor of 2.0 to see the depth of deformation clearly in the scene and to make use of the whole 16x16x16 voxel space. While the robot performed the experiment, the foam material was a few degrees tilted and in the end of every sequence a oblique plain can be observed in figure 5.17, which is not related to the deformation action applied to the material. We provide no additional force or position information to the network. The recorded data was randomly shuffled and split into 1000 training sequences and 125 test sequences. The training and test dataset did not differ in the actions applied to the object but include recording noise, volatilization errors and deviation, caused by the experiment itself. We have designed this dataset to test the real world performance of the DDN for a simple reconstruction task and to verify simple generalization capabilities.

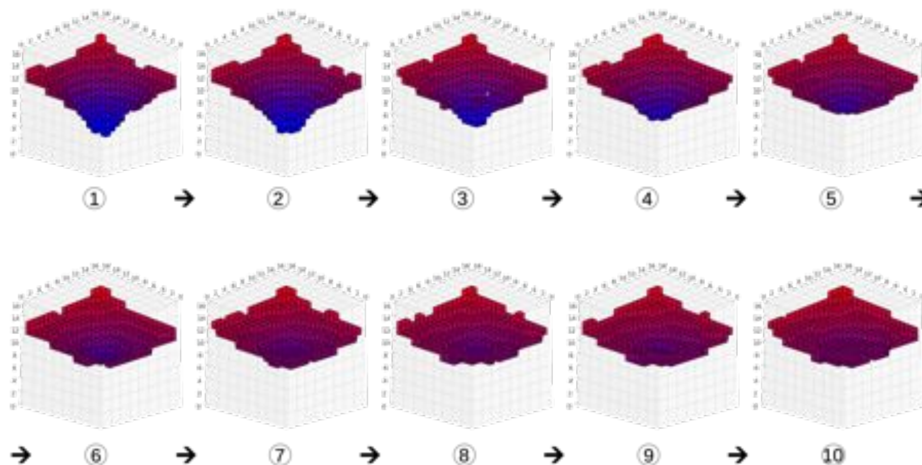


Fig. 5.17.: Foam robot experiment sample sequence bottom view point. The color is correlated to the height of the voxel in the scene, not directly to the degree of deformation.

Training

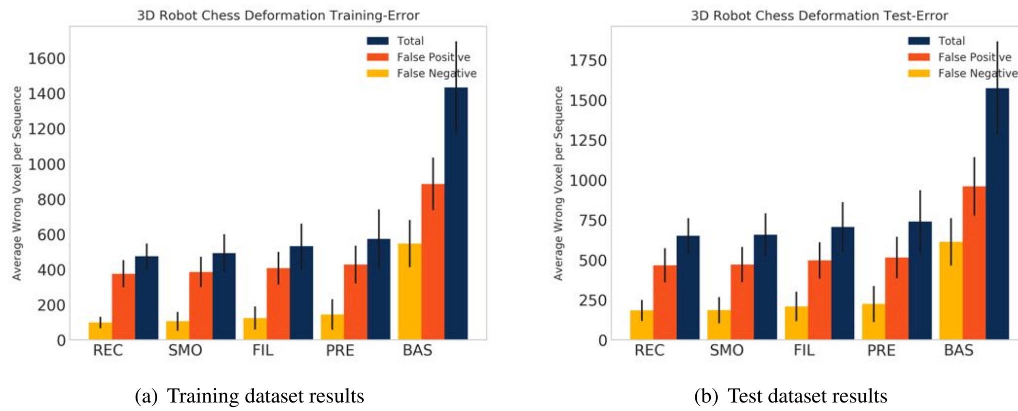
Adjusting the reconstruction loss function, dimension of A, Z and the number of matrices K helped to improve the overall performance of the network. The hyper-parameter setting can be found in table 5.2.

Table 5.2.: Foam deformation experiment hyper-parameter setting

Parameter	dim_a	dim_z	K	rec_{scale}	λ
Value	16	16	1	0.3	0.4

For training the model the Adam optimizer [29], which lead to good results in simulation, was used with an initial learning rate of 5^{-6}, $beta_1 = 0.9$, $beta_2 = 0.999$ and $epsilon = 10^{-8}$. Fraccaro et al. [14] suggest in the supplementary section, to first only update the VAE parameters, then the Kalman Filter parameters and afterwards all parameters at the same time. On real robot data no significant difference between this procedure and tuning all parameters at the beginning could be observed. The batch size for the foam experiment was set to 10. While training we exponentially decreased the learning rate in 40 steps by the factor of $e^{0.7}$. After a total amount of 500 epochs the model converged.

Results



(a) Training dataset results

(b) Test dataset results

Fig. 5.18.: Foam robot experiment imputation [3-8] and prediction [3-10] performance bar chart

At first we have a look at the reconstruction result (Figure 5.18). The mean wrong voxels per sequence of the VAE reconstruction are 475, with an standard deviation of 47, for the training data. This can be interpreted as a bad result since we wrongly classify over 3 % of the total voxels in the scene. In figure 5.19 we can see three different smoothed imputation sample sequences [3-8], with the false-positive and false-negative voxels highlighted. Despite this high reconstruction loss we can see the VAE was able to find multiple important features of the sequence. It localized the touch point and encoded the depth of the touch in the latent space. We don't expect the network to perfectly reconstruct the training data. This would only indicate over-fitting and learning the noise, that is included in the training data, instead of reducing the scene to its key hidden features.

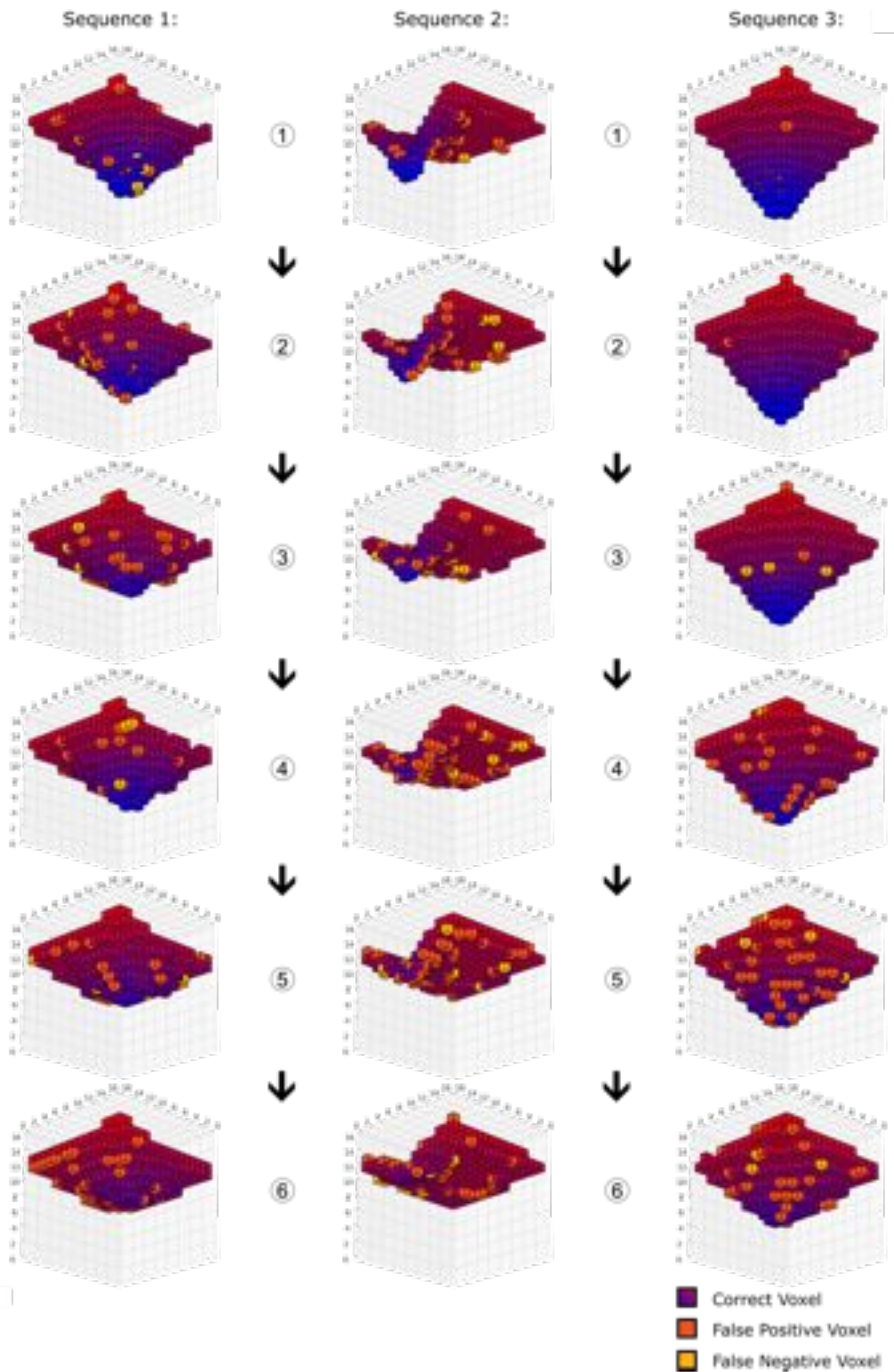


Fig. 5.19.: Foam deformation sample imputation [3-8] sequences over time-steps 1-6

In a sample of the first 5 VAE latent space dimensions (Figure 5.20), we can see, that the only way this model was able to reduce the reconstruction error, was by specialization in the latent space. Despite the regularization, the VAE consistently reduced the noise in favor to encode more information in the latent space. The standard deviation on average dropped to 0.05 for all latent variables. By increasing the penalty of the VAE, we could not achieve better performance on the test or training dataset. The

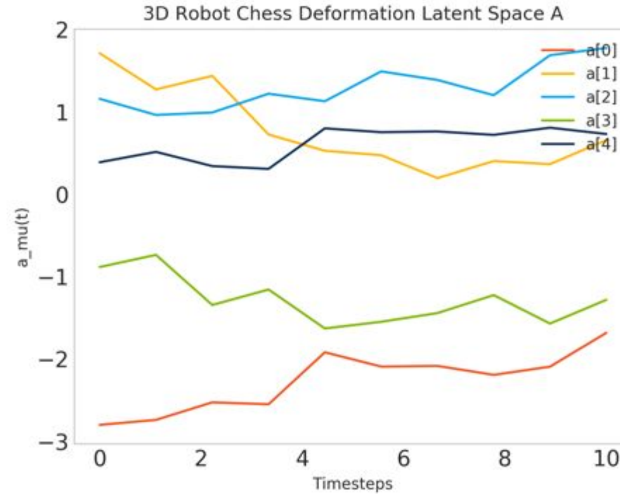


Fig. 5.20.: Foam robot experiment latent space *A* of sample sequence

previous simulation phase showed, while performing imputation tests, that smoothing the hidden space slightly outperformed filtering. The same behavior can be observed in the real robot experiment 5.18, but the performance difference is marginal. By comparing the test and training performance of the network in figure 5.18, similarly good results, can be observed. Previously mentioned the metric of the total amount of wrong voxels, which was a direct performance indicator in simulation, because no noise was included in the dataset, is not the best choice for evaluating the real noise robot data. For interpreting the modelling performance of the DDN, we have to more often take a look at the actual reconstruction data. By performing linear interpolation between the latent space of two different sequence starting points, we could also verify, that the VAE was able to encode the key features of the object.

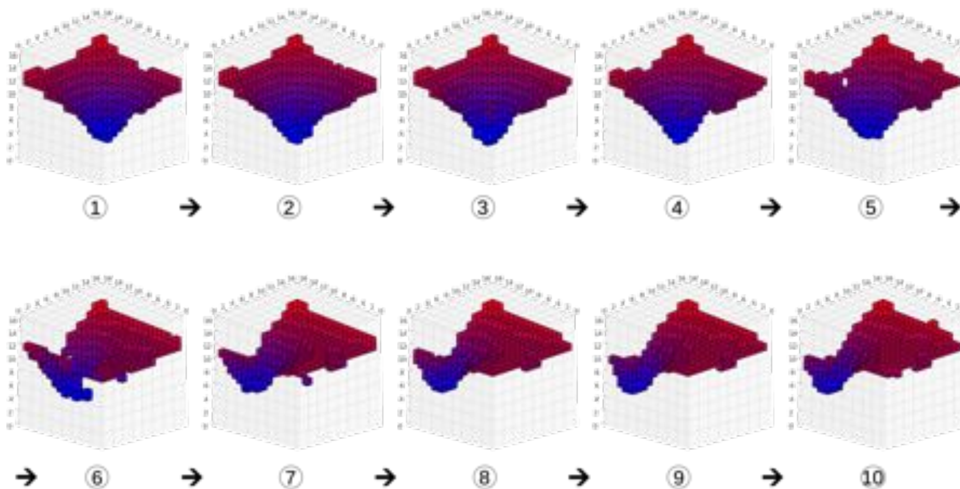


Fig. 5.21.: Foam robot experiment linear interpolation between two sequence starting positions

We can clearly see smooth movement of the touch area between the two different sequence starting positions (Fig. 5.21). This definitely, shows that we were able to reconstruct a robust, not completely over-fitted latent space which is able to encrypt information about the occurring deformation.
One unexpected result we found, while evaluating hyper-parameters was, that restricting the set of learnable metrices to K = 1 performed similar, compared to a more complex setting of K. In figure 5.22 the number of on average wrongly classified voxels per sequence, while training, is plotted for different settings of K. By setting K = 1 we "deactivated" the dynamical network and only allowed learning linear movement across the whole domain of the latent space. For this real robot experiment it seemed to be easier for the DDN to adjust the latent and hidden space, so only linear transitions, are necessary to reproduce the original sequence, even when more complex dynamical movements could be learned in the hidden space. A good dynamical performance can be seen by comparing the reconstruction loss to the imputation and prediction loss in the bar charts of figure 5.18. The bar chart shows good results for the test and also the training data. For a higher setting of K the network responded more less stable to other hyper-parameters variations. Lowering dim_a and dim_z resulted often in a static local minimum of the undeformed foam in the end of the sequence, for all time-frames, or in an overall higher reconstruction error. The network could not benefit from changing K to a higher setting and additional complexity often only lead to worse training results. So describing this deformation data by a single set of linear transmission matrices for the LGSSM just might be the simplest and most precise solution the network is able to learn.

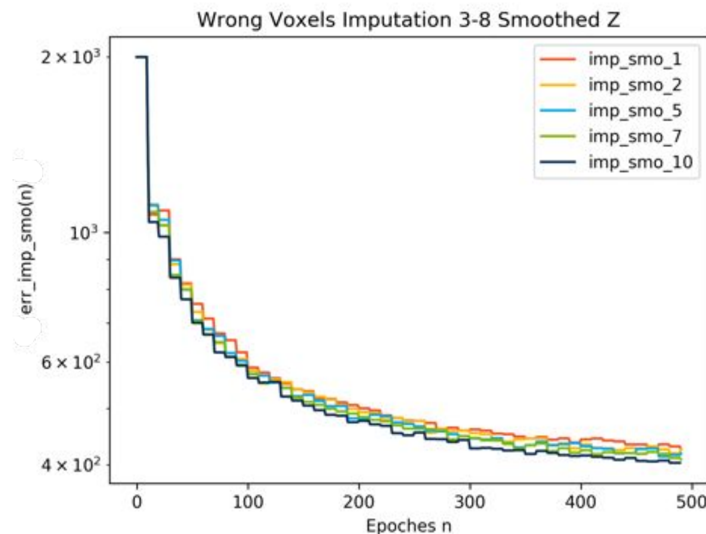


Fig. 5.22.: Foam robot experiment average number of wrong classified voxels per sequence over training process for smoothed imputation [3-8]. Different settings of K result in similar testing performance (log-scale).

At the beginning while creating the voxel data, we stretched the object in Z direction, with the factor of 2.0. This underlined the location and depth of the touch-point and reduced the noise in comparison to the features present in the scene. This preprocessing of the raw sensor data, can be seen as highlighting the deformation features in the scene. Similar training results could be achieved for the unstretched/original voxel data. We only presented the stretched version here, so the deformation features can be seen more clearly in the low resolution 16x16x16 voxel space.

Another important property we have not discussed, is the size of the training dataset. At first we tried to train the model with 225 touches, this resulted in higher over-fitting and worse reconstruction results for

the test data. Extending this dataset to 1125 sequences, resulted in overall better performance.
We tried to utilize the previously introduced control signal for this experiment. Two important informations were available. The applied force measured with the Optoforce sensor and the location of the touch. No matter how many different control matrices we learned influencing the hidden space by the control signal, did not lead to significant better results. We mainly tried to provide the (x,y) position of the touch to the network.

Conclusion

Despite the fact that we could not utilize the full functionality of the DDN, including the Dynamical Network to weigh mutiple learned LGSSM matrices and the additional control input, simple deformation could be captured, with high precision.
We showed in this foam deformation experiment, that the DDN, is able to extract meaningful features from the high dimensional voxel space. With the setting of K = 1 the network was forced to adjust its latent space, so only linear transmission is required in the hidden space to reconstruct a full deformation sequence. Learning this more complex sorted latent space of the VAE framework was only possible by the additional restriction given by the KVAE. We could also verify that the VAE network accomplished good generalization results, for a very similar test dataset.
One big disadvantage, that cannot be illustrated in this Bachelor's thesis, is that fine tuning the network structure has been extremely time consuming. If this would be required for all new objects, applying the proposed DDN to real world deformation scenarios would be absolutely intractable.
So for the last experiment we decided to apply the DDN to a different object, with the exact same parametrization and no additional data preprocessing and hope to find similar results.

5.2.3. Experiment Toy-Ball Deformation

For this experiment we have chosen to deform a smaller toy object. This object is referred to as a squeeze stress reliever toy and made out of polyurethane. This foam material is softer and recovers more smoothly than the previously evaluated pillow. The dimensions of this toy are approximately 8 cm x 8 cm x 8 cm. We try to verify in this experiment the ability of the DDN to learn similar objects, without fine tuning the hyper-parameters for the new training and test dataset.

Generated Data

To record data, we directly mounted the Optoforce sensor to the UR5, without increasing the surface by the sphere tool. We equally spaced 156 points on a sphere around the object, seen in figure 5.23. These point mark the start position, from where we began to push the end-effector, equipped with the optoforce sensor pointing towards the object, into the ball-toy. We repeated this procedure five times and generated in total 780 touch sequences. We randomly shuffled the data and split it into 156 test and 624 training sequences. This toy object recovers in approximately 3 s, so we could apply the same method to generate voxel data, used before, with smoothing over seven time-frames. One video sequence of every other frame can be compared to the resulting voxel data in figure 5.24.

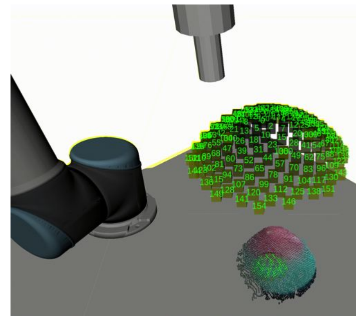


Fig. 5.23.: Screenshot ROS Rviz UR5 probing toy-ball. All starting positions marked by cube with green number. Real SR300 point cloud data of toy-ball.

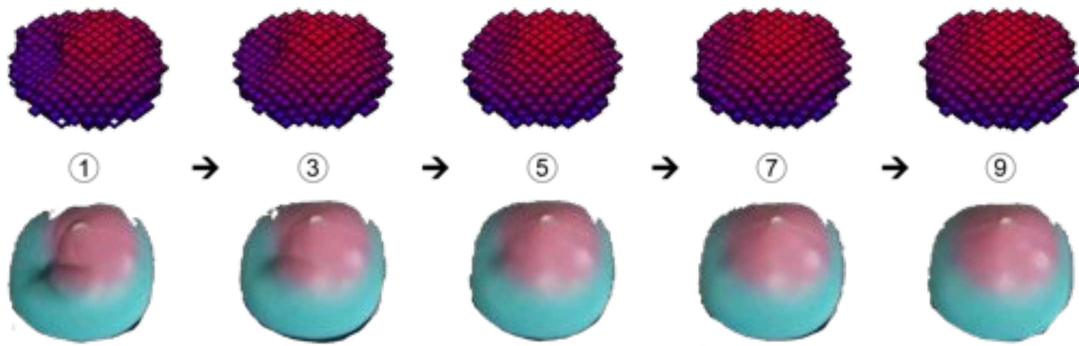
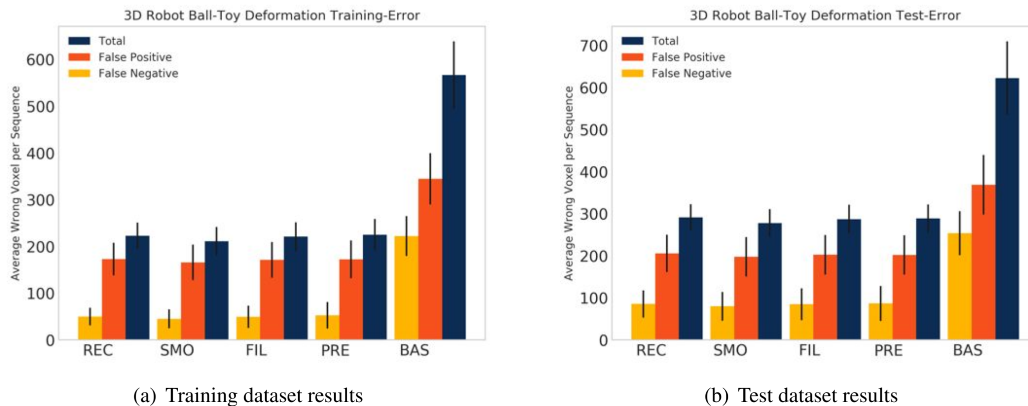


Fig. 5.24.: Ball-Toy SR300 image compared to generated voxel data for every other frame. The color is correlated to the height of the voxel in the scene not directly to the degree of deformation.

Training

We applied the exact same network hyper-parameters and training-parameters, we found in the previous experiment to this dataset.

Results



(a) Training dataset results

(b) Test dataset results

Fig. 5.25.: Ball-Toy robot experiment imputation [3-8] and prediction [3-10] performance bar charts

We can directly have a look at the reconstruction, imputation and prediction results in figure 5.25[a]. We can see a similar performance, compared to the previous foam deformation experiment. The DDN network could reduce the total amount of wrong voxels for the training data to under 250 per sequence for all evaluation methods. Since the imputation and prediction performance are very similar to the reconstruction result, we can clearly tell, that the network successfully learned all the dynamics, in the hidden space. Comparing the training 5.25[a] to the test 5.25[b] performance, we can note slightly worse results. On average the network misclassified 4 voxels per time-step more than in the original training data. This can be directly related to the data applied to the network. For every sequence in the foam experiment a clear deformation can be observed and the touch point can be perfectly observed in the voxel space. 5.19. For the ball experiment the deformation, when applied to the back side of the ball could not be clearly captured by the Realsense camera and the number of deformed voxels is noticeably smaller than for the

foam experiment. This leads to a lower "signal" to "noise" ratio, when we interpret the deformation as the signal appearing in the voxel space.

To understand what these error rates, occurring in the test and training data, represent we can have a look at a sequence with the time-steps 3-8 imputed, for the test and training case. In this data we are looking for features, that describe the shape deformation of the object, and want to evaluate where errors occur. For both cases 5.26 we can see that the object was deformed on the left side. We can also recognize in the data that this deformation on the left side slowly recovers over time. The occurring errors do not only affect the deformation area, they are nearly equally spread around the object. So the remaining error rate can be interpreted as noise, that does not influence the shape of the object.

Conclusion

With this second robot experiment we were able to validate the performance of the DDN to model simple object deformations, without the need of fine tuning the network parameters. The DDN performed on the new dataset comparably to the foam pillow experiment. This qualifies the network to learn a wider field of deformable objects, without additional parameter fine-tuning.

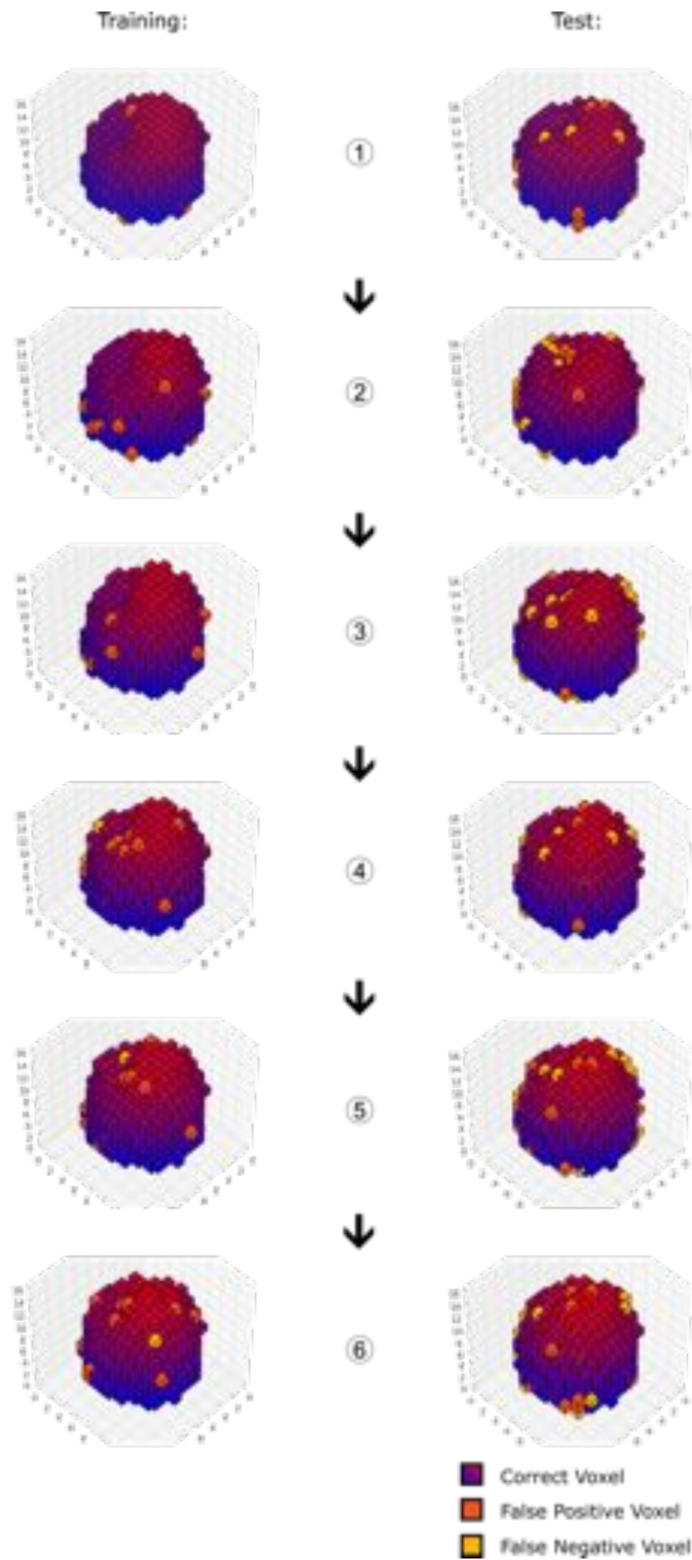


Fig. 5.26.: Toy-Ball deformation imputation [3-8] test and training sample sequence of time-steps 1-6

6. Conclusion and Outlook

The Dynamical Deformation Network is a completely novel data based approach to model deformation. Consequently, only basic problem scenarios were investigated. We successfully modeled simple dynamics in deformation by extending the KVAE framework to handle voxel data. Finding a low dimensional latent representation of the complex voxel space, was only possible by in-cooperating the recent advantages, achieved in 3D deep learning. In particular adopting the progress achieved in the context of the ModelNet classification challenge [67, 7, 18, 38] help to simplify the complex voxel representation. Mapping this low dimensional space over time by a LGSSM could predict the unobserved voxel scene, but only learning basic dynamics was verified. While implementing and training the network, we had to face different challenges and problems. The first problem was to acquire large datasets, which are necessary for deep learning. Generating these datasets to evaluate this new network structure is very time consuming. Another time intensive challenge we faced, consisted of evaluation the DDN hyper-parameters. Even equipped with a high-end computer graphic card (Nvidia GTX 1080) training the model took around 1 hour. This strongly limited the number of cross validated hyper-parameters. We suppose for future experiments to acquire higher resolution voxel data ($32x32x32$) to capture more deformation details. For our experiments we were limited to low resolution by the available training time and computational power given, with a single GPU. Even without using the full complexity of the DDN structure, to model the real deformation data, we acquired good results on the given deformation datasets.

In general applying stochastical deep learning to model deformation, did obviously not provide outstanding performance for the given task. Without any doubt classical FEM methods or mass spring damper models can outperform the DDN in terms of precision as well as simplicity. The provided experiment may be misleading, because the novel network was applied to a deformation problem, that already has been perfectly solved by other methods. Nevertheless we could verify the basic capability of the proposed network to model deformation. We could successfully perform prediction and imputation. Data based deformation modelling might be used in the field of medical engineering. Further the DNN can be used for interacting with objects of which the material parameters cannot be exactly discriminated or the object properties are not covered by "traditional" modelling approaches.

In the context of this bachelor‘s thesis only a simple evaluation of the proposed network was performed. To extend, and hopefully later apply this method to real world problem scenarios, future work is needed. At first further validation of the generalization capability of the network need to be performed. Therefore we propose to capture more complex deformation data. Instead of deforming a real-world object, a finite element simulation can be used to quickly generate a huge amount of training data. Evaluating the network on deformation data of multiple objects in different position, orientations or partial observed, can verify the capability to model more complex deformation data.
The resolution of the voxel data might be extended to 32x32x32 or higher, to capture more deformation details. Regarding the actual robot experiment, representing the 2.5D depth data of the Intel Realsense camera in voxel format is unusual for this application. The same data can be represented as a depth-image, on which already different deep learning methods have been successfully applied. Nevertheless, when we want to extend this method to other 3D data acquisition sensor types, sticking to voxel data can be justified and enhances the generalization capability of the network to other problem scenarios.

One reason why the DDN was not able to utilize its full potential, might be the difference in complexity between the VAE and LGSSM. The VAE is equipped with extremely expressive neural networks. In contrast to the limited LGSSM, which only can describe "simple" transitions in the hidden space. It might be worth investigating, if different statistical models are capable to better connect the latent space

over time. Beside this, the VAE could be extended with additional knowledge about the scene. This could be a picture of the scene, suggested in [18] or other data about the deformation. A "better" latent space to describe the deformation might be acquired, which then can be easier modeled over time. Further investigating the Dynamical Networks might increase the capability of the model to learn deformation in the hidden space. Fraccaro et al. [14] performed a simple evaluation of different network structures for the Dynamical Network. This evaluation can be extended to the real deformation data and might help to learn more complex dynamics in the hidden space.

References

[1] , ed. *3D Force Sensor OMD — Clearpath Robotics*. [Online; accessed 03-September-2018]. URL: https://cdn.shopify.com/s/files/1/1750/5061/products/image_1_300x237.png?v=1532626644.

[2] Holm [VerfasserIn] Altenbach. *Holzmann/Meyer/Schumpich Technische Mechanik Festigkeitslehre*. [Online; accessed 18-September-2018]. Wiesbaden, 2018. URL: https://doi.org/10.1007/978-3-658-22854-5.

[3] Arjan Westerdiep, ed. *Rendering of Voxel Rabbit Image*. [Online; accessed 03-September-2018]. URL: http://drububu.com/miscellaneous/legolizer/images/lego-object-bricks.jpg.

[4] Aron Martinez Romero, ed. *Concepts — Robot Operating System*. [Online; accessed 03-September-2018]. URL: http://wiki.ros.org/ROS/Concepts.

[5] K.J. Bathe. *Finite Element Procedures*. [Online; accessed 18-September-2018]. Prentice Hall, 2006. ISBN: 9780979004902. URL: https://books.google.de/books?id=rWvefGICfO8C.

[6] Gary Bishop, Greg Welch, et al. "An introduction to the Kalman filter". In: *Proc of SIGGRAPH, Course* 8.27599-3175 (2001), p. 59.

[7] André Brock et al. "Generative and Discriminative Voxel Modeling with Convolutional Neural Networks". In: *CoRR* abs/1608.04236 (2016). [Online; accessed 18-September-2018]. arXiv: 1608.04236. URL: http://arxiv.org/abs/1608.04236.

[8] S. Burion et al. "Identifying physical properties of deformable objects by using particle filters". In: *2008 IEEE International Conference on Robotics and Automation*. May 2008, pp. 1112–1117. DOI: 10.1109/ROBOT.2008.4543353.

[9] A. D'Souza, S. Vijayakumar, and S. Schaal. "Learning inverse kinematics". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 1. Oct. 2001, 298–303 vol.1. DOI: 10.1109/IROS.2001.973374.

[10] Dipl.-Ing.-Päd. Andreas Höfler, ed. *Tension test schematic*. [Online; accessed 03-September-2018]. URL: https://www.ahoefler.de/images/maschinenbau/werkstoffkunde/werkstoffpruefung/zerstoerende-werkstoffpruefung/zugversuch/spannungs-diagramm-einteilung.png.

[11] Carl Doersch. *Tutorial on Variational Autoencoders*. [Online; accessed 03-September-2018]. 2016. URL: http://arxiv.org/abs/1606.05908.

[12] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. [Online; accessed 18-September-2018]. 2016. URL: http://arxiv.org/abs/1603.07285.

[13] Fei-Fei Li, Justin Johnson, Serena Yeung, ed. *CS231n Backpropagation - Stanford University*. [Online; accessed 03-September-2018]. URL: http://cs231n.stanford.edu/syllabus.html.

[14] Marco Fraccaro et al. "A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. [Online; accessed 18-September-2018]. Curran Associates, Inc., 2017, pp. 3601–3610. URL: http://papers.nips.cc/paper/6951-a-disentangled-recognition-and-nonlinear-dynamics-model-for-unsupervised-learning.pdf.

[15] B. Frank et al. “Learning the elasticity parameters of deformable objects with a manipulation robot”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2010, pp. 1877–1883. DOI: 10.1109/IROS.2010.5653949.

[16] Barbara Frank et al. “Learning object deformation models for robot motion planning”. In: *Robotics and Autonomous Systems* 62.8 (2014). [Online; accessed 18-September-2018], pp. 1153–1174. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2014.04.005. URL: http://www.sciencedirect.com/science/article/pii/S0921889014000797.

[17] P. Güler et al. “Estimating the deformability of elastic materials using optical flow and position-based dynamics”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. Nov. 2015, pp. 965–971. DOI: 10.1109/HUMANOIDS.2015.7363486.

[18] Rohit Girdhar et al. “Learning a Predictable and Generative Vector Representation for Objects”. In: *CoRR* abs/1603.08637 (2016). [Online; accessed 18-September-2018]. arXiv: 1603.08637. URL: http://arxiv.org/abs/1603.08637.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Book in preparation for MIT Press. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[20] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. [Online; accessed 18-September-2018]. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[21] Karol Gregor et al. “DRAW: A Recurrent Neural Network For Image Generation”. In: *CoRR* abs/1502.04623 (2015). [Online; accessed 18-September-2018]. arXiv: 1502.04623. URL: http://arxiv.org/abs/1502.04623.

[22] N. Haouchine et al. “Image-guided simulation of heterogeneous tissue deformation for augmented reality during hepatic surgery”. In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. Oct. 2013, pp. 199–208. DOI: 10.1109/ISMAR.2013.6671780.

[23] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). [Online; accessed 18-September-2018]. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[24] Carl T. Herakovich. *A Concise Introduction to Elastic Solids : An Overview of the Mechanics of Elastic Materials and Structures*. [Online; accessed 18-September-2018]. Cham, 2017. URL: https://doi.org/10.1007/978-3-319-45602-7.

[25] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997). [Online; accessed 03-September-2018], pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[26] Ioan A. Sucan and Sachin Chitta, ed. *MoveIt!* [Online; accessed 03-September-2018]. URL: http://moveit.ros.org/documentation/concepts/.

[27] Ioan A. Sucan and Sachin Chitta, ed. *MoveIt!* [Online; accessed 03-September-2018]. URL: https://moveit.ros.org/about/.

[28] Peter Kaiser et al. “Experimental Evaluation of a Perceptual Pipeline for Hierarchical Affordance Extraction”. In: *2016 International Symposium on Experimental Robotics*. Springer International Publishing, 2017, pp. 136–146.

[29] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). [Online; accessed 18-September-2018]. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[30] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes.” In: *CoRR* abs/1312.6114 (2013). [Online; accessed 18-September-2018]. URL: http://arxiv.org/abs/1312.6114.

[31] Diederik P. Kingma et al. "Semi-Supervised Learning with Deep Generative Models". In: *CoRR* abs/1406.5298 (2014). [Online; accessed 18-September-2018]. arXiv: 1406.5298. URL: http://arxiv.org/abs/1406.5298.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. [Online; accessed 18-September-2018]. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[33] Tejas D Kulkarni et al. "Deep Convolutional Inverse Graphics Network". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. [Online; accessed 18-September-2018]. Curran Associates, Inc., 2015, pp. 2539–2547. URL: http://papers.nips.cc/paper/5851-deep-convolutional-inverse-graphics-network.pdf.

[34] I. E. Lagaris, A. Likas, and D. I. Fotiadis. "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations". In: *Trans. Neur. Netw.* 9.5 (Sept. 1998). [Online; accessed 18-September-2018], pp. 987–1000. ISSN: 1045-9227. DOI: 10.1109/72.712178. URL: https://doi.org/10.1109/72.712178.

[35] Rynson W.H. Lau et al. "LARGE a Collision Detection Framework for Deformable Objects". In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. VRST '02. [Online; accessed 18-September-2018]. Hong Kong, China: ACM, 2002, pp. 113–120. ISBN: 1-58113-530-0. DOI: 10.1145/585740.585760. URL: http://doi.acm.org/10.1145/585740.585760.

[36] Matteo Lionello. "A Variational Autoencoder Approach for Representation and Transformation of Sounds". [Online; accessed 03-September-2018]. MA thesis. 2018. URL: https://projekter.aau.dk/projekter/files/281073844/thesis_matteolionello.pdf.

[37] Vincenzo Lippiello, Fabio Ruggiero, and Bruno Siciliano. "Floating Visual Grasp of Unknown Objects Using an Elastic Reconstruction Surface". In: *Robotics Research*. Ed. by Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 329–344. ISBN: 978-3-642-19457-3.

[38] Daniel Maturana and Sebastian Scherer. "3D Convolutional Neural Networks for Landing Zone Detection from LiDAR". In: *International Conference on Robotics and Automation*. Mar. 2015.

[39] Warren S. McCulloch and Walter Pitts. "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. [Online; accessed 18-September-2018]. Cambridge, MA, USA: MIT Press, 1988. Chap. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27. ISBN: 0-262-01097-6. URL: http://dl.acm.org/citation.cfm?id=65669.104377.

[40] , ed. *Number of Smartphone Useres worldwide — Statista*. [Online; accessed 03-September-2018]. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[41] Stanley Osher. "Level Set Methods". In: *Geometric Level Set Methods in Imaging, Vision, and Graphics*. [Online; accessed 18-September-2018]. New York, NY: Springer New York, 2003, pp. 3–20. ISBN: 978-0-387-21810-6. DOI: 10.1007/0-387-21810-6_1. URL: https://doi.org/10.1007/0-387-21810-6_1.

[42] A. Petit, V. Lippiello, and B. Siciliano. "Real-time tracking of 3D elastic objects with an RGB-D sensor". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 3914–3921. DOI: 10.1109/IROS.2015.7353928.

[43] Hanspeter Pfister et al. "Surfels: Surface Elements as Rendering Primitives". In: (May 2000).

[44] Prof. Ted Belytschko, ed. *Northwestern University, Civil Engineering Department, Advanced Finite Elements*. [Online; accessed 03-September-2018]. URL: http://www.tam.northwestern.edu/tb/D26/side.gif.

[45] , ed. *Realsense SDK — Intel Corporation*. [Online; accessed 03-September-2018]. URL: https://software.intel.com/en-us/realsense/sdk.

[46] , ed. *Realsense SR300 Infrared RGB-D camera — Intel Corporation*. [Online; accessed 03-September-2018]. URL: https://software.intel.com/en-us/realsense/sr300.

[47] , ed. *Rendering of Mesh Rabbit Image*. [Online; accessed 03-September-2018]. URL: https://cdn-images-1.medium.com/max/1200/1*_Pkoy3Yu8xLjdEaqte3RvQ.jpeg.

[48] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. [Online; accessed 18-September-2018]. Bejing, China: PMLR, 2014, pp. 1278–1286. URL: http://proceedings.mlr.press/v32/rezende14.html.

[49] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[50] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[51] Tim Salimans, Diederik Kingma, and Max Welling. "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. [Online; accessed 18-September-2018]. Lille, France: PMLR, 2015, pp. 1218–1226. URL: http://proceedings.mlr.press/v37/salimans15.html.

[52] , ed. *Schematic Stress Strain Graph — steemKR*. [Online; accessed 03-September-2018]. URL: https://steemkr.com/education/@ghostgtr/how-stuff-works-structures.

[53] Jonathon Shlens. "A Tutorial on Principal Component Analysis". In: *CoRR* abs/1404.1100 (2014). [Online; accessed 03-September-2018]. arXiv: 1404.1100. URL: http://arxiv.org/abs/1404.1100.

[54] Simon Kamronn, ed. *Kalman Variational Auto-Encoder Online Resources — GitHub*. [Online; accessed 03-September-2018]. URL: https://github.com/simonkamronn/kvae.

[55] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. [Online; accessed 18-September-2018]. Curran Associates, Inc., 2015, pp. 3483–3491. URL: http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf.

[56] Qingyang Tan et al. "Mesh-based Autoencoders for Localized Deformation Component Analysis". In: *CoRR* abs/1709.04304 (2017). [Online; accessed 18-September-2018]. arXiv: 1709.04304. URL: http://arxiv.org/abs/1709.04304.

[57] Bilal Tawbe and Ana-Maria Cretu. "Acquisition and Neural Network Prediction of 3D Deformable Object Shape Using a Kinect and a Force-Torque Sensor †". In: *Sensors*. 2017.

[58] Robert F. Tobler and Stefan Maierhofer. *A Mesh Data Structure for Rendering and Subdivision*. 2006.

[59] Hassan Ugail and Eyad Elyan. "Efficient 3D data representation for biometric applications". In: (Jan. 2007).

[60] Waldir Pimenta, ed. *Rendering of Point Cloud Rabbit Image*. [Online; accessed 03-September-2018]. URL: http://waldyrious.net/learning-holography/img/stanford-bunny-points.png.

[61] Jacob Walker et al. "An Uncertain Future: Forecasting from Static Images using Variational Autoencoders". In: *CoRR* abs/1606.07873 (2016). [Online; accessed 18-September-2018]. arXiv: 1606.07873. URL: http://arxiv.org/abs/1606.07873.

[62] Wikipedia contributors, ed. *Artificial neural network — Wikipedia, The Free Encyclopedia*. [Online; accessed 03-September-2018]. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.

[63] Wikipedia contributors, ed. *Deformation — Wikipedia, The Free Encyclopedia*. [Online; accessed 03-September-2018]. URL: https://en.wikipedia.org/wiki/Deformation_(engineering).

[64] Wikipedia contributors, ed. *Deformation (engineering) — Wikipedia, The Free Encyclopedia*. [Online; accessed 03-September-2018]. URL: https://en.wikipedia.org/wiki/Deformation_(engineering).

[65] Wikipedia contributors, ed. *Robot Operating System — Wikipedia, The Free Encyclopedia*. [Online; accessed 03-September-2018]. URL: https://en.wikipedia.org/wiki/Robot_Operating_System.

[66] Jiajun Wu et al. "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. [Online; accessed 18-September-2018]. Curran Associates, Inc., 2016, pp. 82–90. URL: http://papers.nips.cc/paper/6096-learning-a-probabilistic-latent-space-of-object-shapes-via-3d-generative-adversarial-modeling.pdf.

[67] Zhirong Wu et al. "3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction". In: *CoRR* abs/1406.5670 (2014). [Online; accessed 18-September-2018]. arXiv: 1406.5670. URL: http://arxiv.org/abs/1406.5670.

[68] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. [Online; accessed 18-September-2018]. Nov. 2015. arXiv: 1505.00853. URL: http://arxiv.org/abs/1505.00853.

[69] Kai Xu et al. "Data-driven Shape Analysis and Processing". In: *SIGGRAPH ASIA 2016 Courses*. SA '16. [Online; accessed 18-September-2018]. Macau: ACM, 2016, 4:1–4:38. ISBN: 978-1-4503-4538-5. DOI: 10.1145/2988458.2988473. URL: http://doi.acm.org/10.1145/2988458.2988473.

[70] M. E. Yumer and N. J. Mitra. "Learning Semantic Deformation Flows with 3D Convolutional Networks". In: *European Conference on Computer Vision (ECCV 2016)*. Springer. 2016, pp. -.

[71] Mehmet Ersin Yumer and Levent Burak Kara. "Co-constrained Handles for Deformation in Shape Collections". In: *ACM Trans. Graph.* 33.6 (Nov. 2014). [Online; accessed 18-September-2018], 187:1–187:11. ISSN: 0730-0301. DOI: 10.1145/2661229.2661234. URL: http://doi.acm.org/10.1145/2661229.2661234.

[72] Mehmet Ersin Yumer et al. "Semantic Shape Editing Using Deformation Handles". In: *ACM Trans. Graph.* 34.4 (July 2015). [Online; accessed 18-September-2018], 86:1–86:12. ISSN: 0730-0301. DOI: 10.1145/2766908. URL: http://doi.acm.org/10.1145/2766908.

[73] L. Zaidi et al. "Interaction modeling in the grasping and manipulation of 3D deformable objects". In: *2015 International Conference on Advanced Robotics (ICAR)*. July 2015, pp. 504–509. DOI: 10.1109/ICAR.2015.7251503.

A. Appendix

A.1. Deep Learning Frameworks

With the constantly increasing popularity of deep learning and the new fields of applications, a variety of big technology companies like Google, Amazon, Facebook and Microsoft, to name a few, released their own deep learning frameworks. The purpose of these frameworks is to simplify, the highspeed parallel implementation of complex networks with millions of computational operations. Utilizing modern GPU acceleration, allows the programmer to create and train big models in a reasonable time using cheap consumer hardware (mainly Nvidia graphic cards).

The most popular Frameworks Tensorflow (Google), Caffee (Amazon), PyTorch (Facebook), CNTK (Microsoft) are based on building a computational graph. This directed graph, where nodes correspond to operations, can be utilized to compute a high number of operations. This structure allows the calculation of the loss function's gradient with respect to every learnable parameter by the means of backpropagation. Variables can feed their value into operations, and operations provide their output data, to other operations. The output and inputs of these nodes are multi-dimensional arrays, commonly called tensors.

To find the optimal network parameters we need to minimize the cost function of the network. This function corresponds to the inverse of the performance of the network e.g. the classification error. To achieve this, the network parameters (weights) can be modified by an optimizer function. This optimizer function adjusts the weights according to the negative gradient of the cost function in relation to every network-parameter. To effectively calculate this gradient these frameworks, define for every operation a forward and backward step. The forward step is the calculation of the actual function of the operation. This can be a weighted sum of the input variables with additional bias term, a nonlinear function or a more complex function. When the calculation is finished the output variables of the operation are set to the calculated value, which then are provided to the next operation or are the actual output of the neural network. Successively calculating the forward step in direction of the graph for the whole neuronal network is called the forward pass. A complete forward pass is performed when to a given input data the neuronal network discriminated all outputs.

The backward pass calculates the gradient of the loss function in reverse order for the whole neuronal network starting from the output of the graph. To calculate this gradient the chain rule can be applied for every operation. The partial derivatives of the operation outputs in respect to the operation inputs are known for every operation.

When the gradient of the loss function is known in respect to the input, only simple multiplication with the partial deviations is necessary to calculate the gradient in respect to the inputs of the operation. To complete a full backward pass and calculate the gradient of the loss function in respect to every learnable parameter in the neural network, this procedure is repeated for all operations in the graph starting from the output in the reverse graph order.

To illustrate this, a single node is shown in figure A.1. This node performs the operation **f** on the input tensors **x** and **y** to calculate the output tensor **z**. When the output gradient

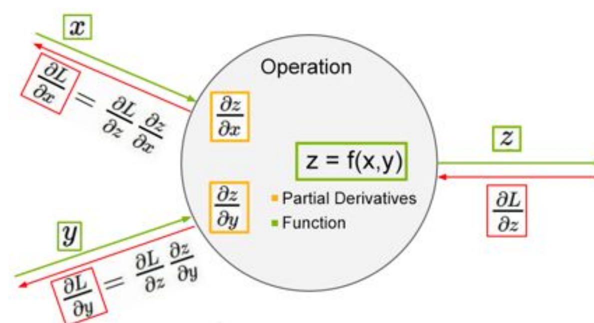


Fig. A.1.: Computational graph, gradient backward step (red), forward step (green). Based on [13]

is passed to the node, it can calculate the output gradient in respect to the input tensors by a simple multiplication with the stored partial deviations $\frac{\delta z}{\delta x}$ and $\frac{\delta z}{\delta y}$. The training of a neural network can be described by the repetition of feeding training data to the network, calculating the gradient of the network and adjusting the weights until a given minimal cost is accomplished. In Figure A.2 the schematic interaction between the different components for a supervised training process is provided.

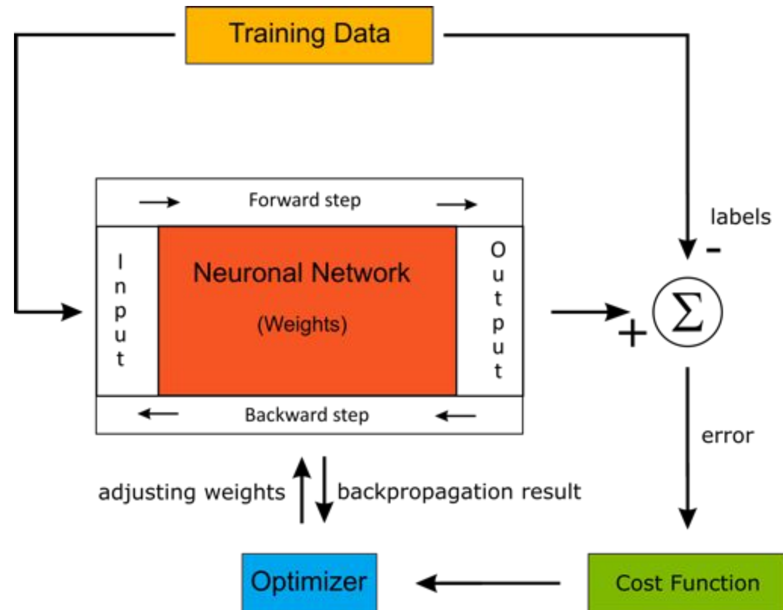


Fig. A.2.: Supervised neural network training process. Overview of interaction between optimizer and neural network.

A.2. Robot Operating System

"Robot Operating System (ROS) is robotics middleware (i.e. collection of software frameworks for robotsoftware development). Although ROS is not an operating system, it provides services designed for heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. " [65]
We previously had a look at the computational graph in TensorFlow. The purpose of this was to separate a big neural network into simpler operations and layers. ROS is providing a similar structure to manage different task of a robot system.
"The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways." [4]
Nodes are processes that perform computational operations. Every node must register to the ROS Master. The ROS Master can be seen as the root of this computational graph and provides look-up-service to the rest of the graph. Communication between nodes can be established on two ways. First a node can publish or subscribe to a topic. A topic has a unique name and a static message type (data type). A topic can be subscribed and published by multiple nodes at the same time. When new data arrives to a topic, all subscribed nodes are notified that new information is available. The second way is provided by request and reply structure, called services. They allow a node to offer a service to other nodes. A service is defined by a message to request a response and by the responding message provided by the node offering the service. The connections in the computational graph are provided mainly by an implementation of TCP/IP sockets. This allows ROS to run over multiple machines in the same network. ROS is supported

for selected Linux distributions. It also comes with additional useful tools like the visualization toolbox Rviz or simulation environment Gazebo.
One of the biggest advantages of ROS is the big research community and the open source policy. There are many packages available, that help to accelerate your robot software development. Despite numerous efforts to bring ROS to industrial environments the lack of real-time reliability and security concerns hindered this so far. With ROS 2.0 the community tries to tackle the open real-time problems, but the safety issues will stay a future obstacle for the advance of ROS into industrial environments. In conclusion ROS is a great framework for research and development, with minor lacks that are in no relation to its usefulness for fast development.

A.3. Decoder and Encoder Tensorflow Implementation

A.3.1. Encoder

```
def encoder(self, x):
  print "\nEncoder Layer Structure:"
  with tf.variable_scope('vae/encoder'):
    def conv3d(x, W):
      # ksize = size of filter (2,2,2);
      # strides = filter movement (2,2,2);
      return tf.nn.conv3d(input=x, filter=W, strides=[1,1,1,1,1],
          padding='SAME')
    def maxpool3d(x):
      # ksize = size of filter (2,2,2);
      # strides = filter movement (2,2,2);
      return tf.nn.max_pool3d(x, ksize=[1,2,2,2,1], strides
          =[1,2,2,2,1], padding='SAME')
    def avgpool3d(x):
      # ksize = size of filter (2,2,2);
      # strides = filter movement (2,2,2);
      return tf.nn.avg_pool3d(x, ksize=[1,2,2,2,1], strides
          =[1,2,2,2,1], padding='SAME')

    x = tf.layers.batch_normalization(x)
    conv_input = tf.reshape(x, shape=[-1, self.d1, self.d2, self.d3,
        1])
    print conv_input
    #conv_input dimension [bs,16,16,16,1]

    conv1 = tf.nn.leaky_relu(conv3d(conv_input, self.enc_weights['
        W_conv1']) + self.enc_biases['b_conv1'])
    conv1 = tf.layers.batch_normalization(conv1)
    conv1_b = maxpool3d(conv1)
    conv1_a = avgpool3d(conv1)
    concat_1= tf.concat([conv1_b, conv1_a],4)
    print conv1,"\n", conv1_a,"\n",conv1_b,"\n",concat_1
    #concat_1 dimension [bs,8,8,8,16]

    conv2 = tf.nn.leaky_relu(conv3d(concat_1, self.enc_weights['W_conv2
        ']) + self.enc_biases['b_conv2'])
    conv2 = tf.layers.batch_normalization(conv2)
    conv2_b = maxpool3d(conv2)
    conv2_a = avgpool3d(conv2)
    concat_2= tf.concat([conv2_b, conv2_a],4)
```

```
    print conv2,"\n", conv2_a,"\n",conv2_b,"\n",concat_2    #concat_2
        dimension [bs,4,4,4,64]

    _ , x , y, z, f_o = concat_2.get_shape()
    concat_2_reshape = tf.reshape(concat_2 ,shape=[-1,x*y*z*f_o] )
    print concat_2_reshape  #concat_2 dimension [bs,4096]

    fully0 = slim.fully_connected(concat_2_reshape, 512, activation_fn=
        tf.nn.leaky_relu)
    print fully0            #fully0 dimension [bs,512]

    fully1 = slim.fully_connected(fully0, 512, activation_fn=tf.nn.
        leaky_relu)
    print fully1            #fully1 dimension [bs,512]

    fully2 = slim.fully_connected(fully1, 256, activation_fn=tf.nn.
        sigmoid)
    print fully2            #fully2 dimension [bs,512]

    a_mu = slim.fully_connected(fully2, self.config.dim_a,
        activation_fn=None)
    print a_mu              #a_mu dimension [bs,dim_a]

    a_var = slim.fully_connected(fully2, self.config.dim_a,
        activation_fn=tf.nn.sigmoid)
    print a_var             #a_var dimension [bs,dim_a]

    a_var = self.config.noise_emission * a_var
    epsilon = tf.random_normal(tf.shape(a_var), name="epsilon")
    a = a_mu + tf.sqrt(a_var) * epsilon
    print a                 #a dimension [bs,dim_a]

    a_seq = tf.reshape(a, tf.stack((-1, self.ph_steps, self.config.
        dim_a)))

  return a_seq, a_mu, a_var
```

```
Output:

Encoder Layer Structure:
Tensor("vae/encoder/Reshape:0", shape=(?, 16, 16, 16, 1), dtype=float32
    )
Tensor("vae/encoder/batch_normalization_2/batchnorm/add_1:0", shape=(?,
     16, 16, 16, 8), dtype=float32)
Tensor("vae/encoder/AvgPool3D:0", shape=(?, 8, 8, 8, 8), dtype=float32)
Tensor("vae/encoder/MaxPool3D:0", shape=(?, 8, 8, 8, 8), dtype=float32)
Tensor("vae/encoder/concat:0", shape=(?, 8, 8, 8, 16), dtype=float32)
Tensor("vae/encoder/batch_normalization_3/batchnorm/add_1:0", shape=(?,
     8, 8, 8, 32), dtype=float32)
Tensor("vae/encoder/AvgPool3D_1:0", shape=(?, 4, 4, 4, 32), dtype=
    float32)
Tensor("vae/encoder/MaxPool3D_1:0", shape=(?, 4, 4, 4, 32), dtype=
    float32)
Tensor("vae/encoder/concat_1:0", shape=(?, 4, 4, 4, 64), dtype=float32)
Tensor("vae/encoder/Reshape_1:0", shape=(?, 4096), dtype=float32)
```

```
Tensor("vae/encoder/fully_connected/LeakyRelu/Maximum:0", shape=(?,
    512), dtype=float32)
Tensor("vae/encoder/fully_connected_1/LeakyRelu/Maximum:0", shape=(?,
    512), dtype=float32)
Tensor("vae/encoder/fully_connected_2/Sigmoid:0", shape=(?, 256), dtype
    =float32)
Tensor("vae/encoder/fully_connected_3/BiasAdd:0", shape=(?, 16), dtype=
    float32)
Tensor("vae/encoder/fully_connected_4/Sigmoid:0", shape=(?, 16), dtype=
    float32)
Tensor("vae/encoder/add_2:0", shape=(?, 16), dtype=float32)
```

A.3.2. Decoder

```
def decoder(self, a_seq):
  print "\nDecoder Layer Structure"
  with tf.variable_scope('vae/decoder'):
    a = tf.reshape(a_seq, (-1, self.config.dim_a))

    dec_fully0 = slim.fully_connected(a,512, activation_fn=tf.nn.
        leaky_relu)
    print dec_fully0
    #dec_fully0 dimension [bs,512]
    dec_deconv1_input = slim.fully_connected(dec_fully0,4* 4 * 4 * self
        .df0, activation_fn=tf.nn.leaky_relu)
    dec_deconv1_input = tf.reshape(dec_deconv1_input, (-1, 4 ,4, 4,
        self.df0))
    print dec_deconv1_input
    #dec_deconv1_input dimension [bs,4,4,4,32]

    batch_size = tf.shape(dec_deconv1_input)[0]

    deconv1 = keras.layers.UpSampling3D(size=(2,2,2),data_format="
        channels_last")(dec_deconv1_input)
    deconv_shape1 = tf.stack([batch_size,8,8,8,self.df1])
    deconv1 = tf.nn.conv3d_transpose(dec_deconv1_input,
      self.dec_weights['W_deconv1'],
      output_shape=deconv_shape1,
      strides=[1,2,2,2,1],
      padding="SAME")
    deconv1 = tf.nn.bias_add(deconv1, self.dec_biases['b_deconv1'])
    deconv1 = tf.nn.leaky_relu(deconv1)
    deconv1 = tf.layers.batch_normalization(deconv1)
    print deconv1
    #dec_deconv1_input dimension [bs,8,8,8,16]

    deconv2 = keras.layers.UpSampling3D(size=(2,2,2),data_format="
        channels_last")(deconv1)
    deconv_shape2 = tf.stack([batch_size,16,16,16,self.df2])
    deconv2 = tf.nn.conv3d_transpose(deconv1,
      self.dec_weights['W_deconv2'],
      output_shape=deconv_shape2,
      strides=[1,2,2,2,1],
      padding="SAME")
```

```
    deconv2 = tf.nn.bias_add(deconv2, self.dec_biases['b_deconv2'])
    deconv2 = tf.nn.leaky_relu(deconv2)
    deconv2 = tf.layers.batch_normalization(deconv2)
    print deconv1         #deconv1 dimension [bs,16,16,16,1]
    deconv2 = tf.reshape(deconv2,(-1, self.d1 * self.d2 * self.d3))

    dec_fully_out0 = slim.fully_connected(deconv2, self.d1 * self.d2 *
        self.d3, activation_fn=tf.nn.leaky_relu)
    dec_fully_out1 = slim.fully_connected(dec_fully_out0, self.d1 *
        self.d2 * self.d3, activation_fn=tf.nn.leaky_relu)
    dec_fully_out2 = slim.fully_connected(dec_fully_out1, self.d1 *
        self.d2 * self.d3, activation_fn=tf.nn.leaky_relu)
    print dec_fully_out2
    #dec_fully_out2 dimension [bs,16,16,16,1]

    x_mu = slim.fully_connected(dec_fully_out2, self.d1 * self.d2 *
        self.d3, activation_fn=tf.nn.sigmoid)

    x_mu = tf.reshape(x_mu, (-1, self.d1, self.d2, self.d3, 1))
    x_var = tf.constant(self.config.noise_pixel_var, dtype=tf.float32,
        shape=())
    x_hat = x_mu
    print x_mu
    #x_mu dimension [bs,16,16,16,1]

  return tf.reshape(x_hat, tf.stack((-1, self.ph_steps, self.d1, self.
      d2, self.d3))), x_mu, x_var
```

```
Output:

Decoder Layer Structure
Tensor("vae/decoder/fully_connected/LeakyRelu/Maximum:0", shape=(?,
    512), dtype=float32)
Tensor("vae/decoder/Reshape_1:0", shape=(?, 4, 4, 4, 32), dtype=float32
    )
Tensor("vae/decoder/batch_normalization/batchnorm/add_1:0", shape=(?,
    8, 8, 8, 16), dtype=float32)
Tensor("vae/decoder/batch_normalization/batchnorm/add_1:0", shape=(?,
    8, 8, 8, 16), dtype=float32)
Tensor("vae/decoder/fully_connected_4/LeakyRelu/Maximum:0", shape=(?,
    4096), dtype=float32)
Tensor("vae/decoder/Reshape_3:0", shape=(?, 16, 16, 16, 1), dtype=
    float32)
```