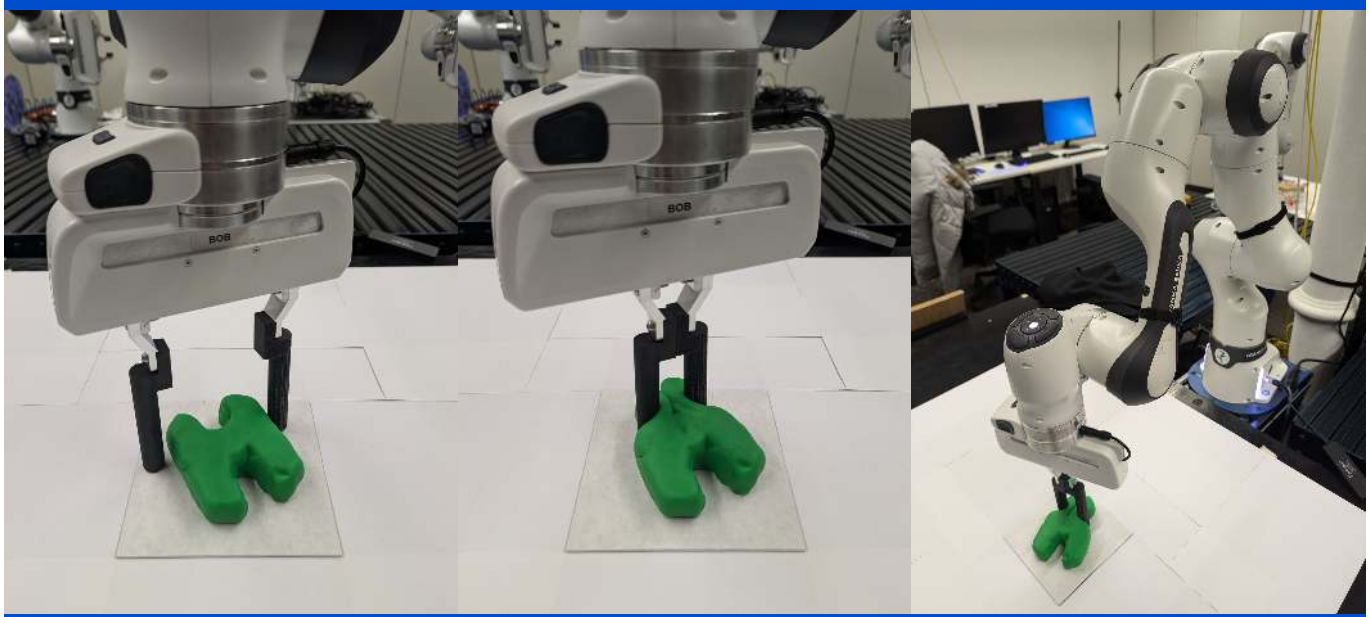
Real2Sim for Play-Doh Manipulation

Bachelor thesis in the department of Computer Science by Marius Glaser
Date of submission: December 4, 2024

1. Review: Paul Jansonnie
2. Review: Prof. Dr. Frank Jäkel
3. Review: Prof. Dr. Jan Peters
4. Review: Dr. Oleg Arenz

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Marius Glaser, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 4. Dezember 2024

Marius Glaser

Contents

Abstract

Robots have made significant progress in handling rigid objects, but manipulating soft materials like dough and fabric presents unique challenges due to their constantly changing shapes. Enabling robots to form target shapes out of dough effectively requires planning frameworks that can handle the nearly infinite state representations of such complex soft objects. Such frameworks must accurately capture the complex dynamics of materials like dough. In our work, we develop a dynamics model that predicts how dough changes shape when grasped by a 2-finger gripper. To train this model, we introduce a method for transforming raw camera data into precise 3D point cloud representations of dough shapes. We also provide a dataset illustrating how various dough shapes deform under different grasping actions. For this dataset, we investigate the effect of volumetric point clouds versus shell point clouds, which typically only capture the outer layer of objects. A crucial aspect of our approach involves using autoencoder neural networks designed to extract essential geometric features and encode them into lower-dimensional vector representations. Based on these learned features, our dynamics model can predict the subsequent states of the dough. We not only analyze single grasps but also assess how well our model predicts changes over a series of multiple grasps. We compare the performance of three different autoencoder architectures, PointNet, PointNet++, and Point Transformer, which are capable of directly processing 3D point clouds, and their impact on the learned dynamics model. Our experiments reveal a solid performance of both PointNet++ and Point Transformer models. The presented dynamics models can make decent predictions for subsequent dough manipulation steps. However, our results indicate that there is still room for improvement. This research aims to gain new insights into the intricacies of learning complex soft body dynamics to create accurate planning frameworks that ultimately enable robots to handle soft dough-like materials with high precision.

Zusammenfassung

Roboter haben im Umgang mit festen Körpern erhebliche Fortschritte gemacht, aber die Handhabung weicher Materialien wie Knete oder Stoffen stellt aufgrund ihrer sich ständig ändernden Formen eine besondere Herausforderung dar. Um es Robotern zu ermöglichen, effektiv Zielformen aus Knete zu formen, sind Planungssysteme erforderlich, die mit den nahezu unendlichen Freiheitsgraden solcher komplexen weichen Objekte umgehen können. Solche Systeme müssen die komplexe Dynamik von Materialien wie Knete genau erfassen können. In unserer Arbeit entwickeln wir ein Dynamikmodell, das vorhersagt, wie sich die Knete verändert, wenn sie von einem Zweifingergreifer verformt wird. Wir stellen eine Methode vor, wie aus Aufnahmen von Tiefenkameras präzise 3D-Punktwolkendarstellungen der verschiedenen Knetformen erzeugt werden können. Außerdem stellen wir einen Datensatz zur Verfügung, der zeigt, wie sich verschiedene Knetformen bei unterschiedlichen Greifvorgängen eines Roboters verformen, und wenden diesen für das Training unseres Modells an. Dabei untersuchen, wir welche Effekte volumetrische Punktwolken und welche Effekte Punktwolkenhüllen, die nur die äußere Schicht der Objekten darstellen, auf unser Modell haben. Ein entscheidender Aspekt unseres Ansatzes ist die Verwendung neuronaler Netze, die in der Lage sind, die wesentlichen geometrischen Merkmale der Punktwolken zu extrahieren und in Vektordarstellungen zu kodieren. Auf Grundlage dieser gelernten Merkmale kann unser Dynamikmodell die nächsten Zustände der Knete vorhersagen. Wir analysieren nicht nur einzelne Greifbewegungen, sondern untersuchen auch, wie gut unser Modell Veränderungen über eine Serie von mehreren Greifbewegungen vorhersagen kann. Wir vergleichen drei verschiedene Autoencoder-Architekturen – PointNet, PointNet++ und Point Transformer – die in der Lage sind, 3D-Punktwolken direkt zu verarbeiten, und ihre Auswirkungen auf das gelernte Dynamikmodell. Unsere Experimente demonstrieren eine gute Performanz der beiden PointNet++ und Point Transformer Modellen. Die vorgestellten Dynamikmodelle können solide Vorhersagen für resultierende Knetformen nach mehreren aufeinander folgende Greifbewegungen treffen. Unsere Ergebnisse weisen jedoch darauf hin, dass für noch präzisere Dynamikmodelle weitere Forschung nötig ist. Diese Forschung zielt

darauf ab, neue Einblicke in das Lernen weicher Materialien zu gewinnen, um genaue Planungssysteme zu schaffen, die es Robotern letztendlich ermöglichen, weiche Materialien wie beispielsweise Knete mit hoher Präzision zu handhaben.

1. Introduction

Robot automation has seen rapid improvements across various industries in recent years. However, this success is mainly limited to domains involving rigid body objects. Enabling robots to handle and process soft body materials has applications in many daily tasks, such as cooking/baking with dough or dealing with soft fabrics like clothes or ropes when doing laundry. It can also be found in the medical domain when operating on soft tissue or in big industrial manufacturing processes that involve materials with dough-like, foam-like, or even rubber-like properties. Especially in the industry, robotic automation of soft body manipulation tasks could significantly improve safety standards, for example, when dealing with dangerously hot materials or in the medical field when conducting operations on a microscopic scale. In addition, robotic automation could help us with our day-to-day chores by completing tasks like laundry and cooking. This research field experienced increasing interest in recent years mainly due to the success of works such as RoboCraft [1], RoboCook [2], or SculptBot [3].

A prevalent task in soft body manipulation is enabling a robot to deform soft materials like dough or foam into complex target shapes. This task can require multiple consecutive manipulation actions to reach the soft body's desired state. For such operations a model-based planning framework is essential. In our work, we are interested in learning a dynamics model that can predict the resulting shape given the grasping action of a 2-finger parallel gripper and the initial state of a dough object. We provide a dataset and additionally present our methodology and insights into developing such a high-quality dataset. The dataset provides a collection of samples of how different shapes of dough deform under various grasping actions of a 2-finger parallel gripper. Acquiring precise state representations of soft objects like dough is a challenging task. The nearly infinite degrees of freedom of soft bodies make it hard to model their state representations. Moreover, because of their complex dynamics, conventional particle simulators can only approximate the system's real-world physics, leading to significant deviations between simulation and real-world behavior, especially in long-horizon planning tasks [4]. Additionally, these methods always assume a complete state observation, which is usually hard to obtain in

real-world applications, especially if systems only rely on observed visual sensory data, for example. Our approach presents a procedure for transforming raw RGB-D data of the observed dough shapes into accurate 3D point clouds. Our primary interest lies in training a dynamics model with such a simple state representation format and still being able to make complex predictions of the outcomes of the resulting state of the dough. A key component of the dynamics model is an autoencoder deep neural network architecture. We use our provided dataset to train three different autoencoder networks, which can extract the spatial features of the complex dough shapes by learning a latent embedding of the dough's states. Due to point clouds' unordered data structure, conventional convolutional feature learning networks struggle to handle the 3D point cloud format. Therefore, we propose using a particular form of deep neural network architecture, PointNet [5], for the autoencoders to directly process 3D point clouds. To be more precise, we look closer at a PointNet-based, a PointNet++-based [6], and a Point Transformer [7] architecture for the autoencoders. We then integrate the autoencoders into the latent dynamics model, which uses their learned state embeddings to infer the complex dynamics of a soft object like dough and ultimately can make an assumption about the future state of the dough object. Finally, we compare the performance of the three resulting dynamics models. Not only are we interested in predicting the resulting shape from a single grasp action, but we also conduct long-horizon prediction tests over a sequence of multiple grasp actions.

1.1. Contributions

In this thesis, we aim to learn a latent dynamics model that can predict how a soft object like dough deforms under the grasping action of a parallel 2-finger robot gripper only based on observed RGB-D data. To train our dynamics model, we collect data on various dough shapes (see Figure 5.5) and their behavior under a sequence of different 2-finger grasp actions. This dataset forms the backbone of our research. The model receives inputs in the form of a 3D point cloud representation of the dough's observed shape P_t (right before the start of the next manipulation action) at time t and the grasp action of the robot gripper. We define the grasp action applied to the dough's initial shape at time t as A_t. A_t consists of the robot state at the start of the grasp action and the robot state during the grasp action, where the robot fingers reach their closest point before beginning to open again. For more details on the action input, see Section 4.3.2. Given the initial dough state P_t and grasp action A_t, the dynamics model learns to predict the resulting point cloud of the next dough shape $\hat{P}_{t+1}$ at time $t+1$.

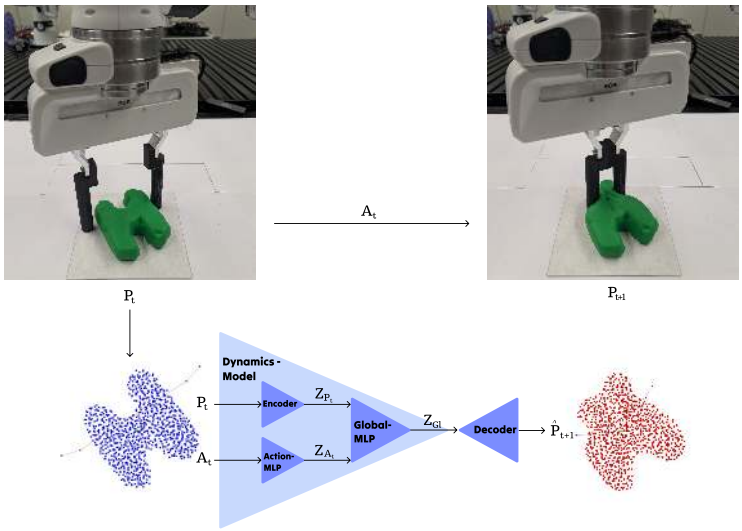


Figure 1.1.: Schematic representation of the latent dynamics model with a prediction sample (top-down view on the dough). The dynamics model gets the 3D point cloud representation of a dough shape P_t (blue point cloud) and the robot action A_t (fingers are visualized as vertical purple lines, and the grasp trajectory is visualized as an orange horizontal line connecting the two fingers). The encoder network extracts the essential features and outputs the latent representation Z_{P_t} of the inputted point cloud P_t. Additionally, a multi-layer perceptron (Action-MLP) transforms the action input A_t into the latent representation Z_{A_t}. Another bigger multi-layer perceptron (Global-MLP) processes both Z_{P_t} and Z_{A_t} and learns how to predict the effect of the robot action on the point cloud shape. The output of the Global-MLP is a latent representation of the resulting point cloud shape Z_{GL}, which is the output of the latent dynamics model. The decoder decodes the latent space vector Z_{GL} back to a 3D point cloud $\hat{P}_{t+1}$ (red point cloud).

A schematic view of the latent dynamics model with one prediction example can be viewed in Figure 1.1. Our experiments show that our dataset can be utilized to learn accurate 3D representations of various complex dough shapes, and the presented dynamics models can make solid assumptions about subsequent dough states.

2. Motivation

2.1. Learning soft body manipulation

We are particularly interested in predicting manipulations for (Play-Doh) dough. Studies on 3D objects with high plasticity, like dough, are a relatively new research field with already promising conducted works [1, 2, 3, 8]. Still, related work of RoboCraft has shown that their learned dynamics model can generalize to predict manipulations of memory foam with decent success despite initially being trained with dough. Learning the dynamics of dough is only the first step in learning how to predict manipulations of other elasto-plastic objects, such as dough for cooking/baking or various types of clay or foam in manufacturing processes. While our work involves learning the dynamics of dough, our primary focus lies on gaining critical insights and techniques in soft body manipulation in general.

Our work aims to provide a dataset of various dough manipulation actions. This dataset is unique because it introduces many new and different shapes made of dough. Additionally, it contains not only isolated manipulation samples but also a set of consecutive manipulation actions performed on the dough. With these sequences of subsequent dough manipulations, we are able to conduct long-horizon prediction tests for our model to further assess its performance. We hope this dataset can be used to train different kinds of dynamics models in the future that use 3D point clouds as input. For future works, our goal is to integrate the learned dynamics model into a planning framework that enables robots to plan their actions to achieve a target dough shape.

2.2. Challenges of soft body manipulation

While robot interaction with rigid bodies primarily focuses on controlling their position and orientation, manipulating deformable objects like dough adds the challenge of managing the objects' changing shape [9]. Moreover, the shape does not only change once per interaction. When under stress, these materials continuously change their shape. The nearly infinite Degrees of Freedom (DoF) of elasto-plastic objects present a complex challenge in modeling their dynamics and finding adequate object state estimations [1]. As a direct result of the complex state estimation, the task of robot action selection to manipulate and interact with the dough also becomes inevitably more difficult [10]. Additionally, partial observability becomes a significant problem due to the object's nonexistent inherent shapes. While it is easy to make assumptions about the shape and state of a rigid body with a limited number of viewing angles or even one viewing angle, this is not possible for a deformable object that constantly changes its shape in new and possibly never-before-seen constellations. A constant view from all angles is necessary to estimate the object's current shape. Even with a view from multiple perspectives, we are confronted with the challenge of self-occlusion caused by a concave shape of the object itself or caused by the robot gripper when directly interacting with the dough. There are efforts to make it easier to model the dynamics of complex scenes and objects with particle-based simulators [11]. However, these kinds of simulators often rely on approximation methods and focus on visual realism but differ from the real-world non-linear properties [10] of the object, making them impractical for long-term predictions [4]. Our initial goal is not only to predict the resulting shape of different grasping actions (with a parallel gripper) on more complex dough shapes but also to provide the basis and first steps for a model-based planning framework similar to RoboCraft or SculptBot that enables a robot to manipulate the dough over multiple steps into a given target shape. Therefore, we take on a real2sim approach to learn an accurate dynamics model of a complex, non-rigid object like dough for this task.

2.3. Real2sim approach

Training robots in the real world is time-consuming and expensive. Imagine having to wait for a robot to undergo thousands or maybe tens of thousands of trials of action state exploration to learn how to form dough into a specific target shape. Not only do you have

to wait, but you also have to make sure to take the dough and form it back to its initial shape after each trial.

Therefore, the only feasible alternative would be to train the robots in simulation. Robots can be safely trained in simulation before being deployed in the real world. Additionally, they can explore more diverse states and can be tested in different scenarios that can be dynamically changed and augmented. This process of transferring learned skills or knowledge from simulation to the real world is what we understand under sim2real. However, no simulation can capture the real world in all its detail, richness of information, and diversity. Simulation is only an approximation of reality, sometimes only a simple one. This can lead to big discrepancies between the simulations and the real world (especially in long-horizon prediction and planning) since the simulations cannot capture complex real-world dynamics. It is important to note that while valuable for training, simulators have limitations. They frequently fail to capture the many fine-grained details of the robot and its environment [12]. In our specific case, this is mainly due to camera noise and the dough's complex non-linear behavior. The consequence of these discrepancies is that successful robots in simulation often fail when deployed in the real world [13]. This phenomenon is often called the sim2real gap in literature [12, 14]. To diminish this gap, we use an alternative approach that reverses the concept of sim2real. This approach, known as real2sim, offers a promising path forward in robot training [15]. We aim to capture real-world RGB-D data and transform the captured scene into a 3D point cloud representation. These point clouds are used to train a latent dynamics model that can capture the complex dynamics of dough and, given a particular robot grasping action, predict the resulting shape.

2.4. Related work

Our work follows in the footsteps of a few stand-out research projects, Robocraft [1] and SculptBot [3], for soft body manipulation of 3D elasto-plastic objects. Similar to our work, RoboCraft and SculptBot focus on learning a dynamics model that can predict the next state of the elasto-plastic object given a grasping action of a 2-finger parallel gripper that deforms the object solely based on raw visual observation of the scene. Naturally, partial obstruction makes it hard to obtain a complete representation of the recorded object only based on raw sensory data. In the works of RoboCraft and its follow-up work RoboCook [2], researchers present a method for constructing a particle graph out of raw RGB-D data and use it to learn a graph neural network dynamics model. Their approach

is unique because it is not reliant on observing the exact particle-to-particle state of the elasto-plastic object. RoboCraft provides a pipeline for transforming raw RGB-D data into point clouds, allowing the dough object to be extracted from the rest of the scene. They use this 3D point cloud representation of the dough to construct a particle graph to train their dynamics model. In total, they collect 6000 frames of recorded data in 50 episodes. Each episode records three consecutive grasping actions on the dough with the robot. With this data, RoboCraft's dynamics model can learn the dynamics of the dough and, given the dough's current state and a grasping action, predict its next state. Our approach is similar to RoboCraft's approach in extracting the 3D point cloud of the elasto-plastic object, but because we are only interested in the state of the dough right before and after a grasping action, we do not have to deal with heavy self-occlusion caused by the gripper fingers like it is the case for RoboCraft. However, that means our dataset needs to contain many more grasping actions (one sample equals one grasping action). Therefore, we conduct 103 episodes in total with three to five consecutive grasping actions on the dough. This leads to a dataset with many diverse dough manipulation samples that our and future models can be trained with. In our work, we are also interested in learning a dynamics model that can directly process a 3D point cloud without constructing a particle graph. Instead, we present a method of learning a latent dynamics model that learns with 3D point cloud representations of the dough's state, similar to SculptBot.

Learning representations of 3D data usually requires a lot of data, which can be time-consuming. To work around this challenge, SculptBot proposes to use a pre-trained model, Point-BERT [16], to learn 3D representations for their dynamics model. Point-BERT is trained on ShapeNet [17], a large dataset containing many different 3D shapes. However, SculptBot as well as RoboCraft, use a mold to reset the dough shape before manipulating it again. RoboCraft uses a square shape, and Sculptbot uses a cylinder shape as an initial shape for the dough. This resetting method is more convenient and much faster than reshaping the dough by hand, which is our approach to resetting the dough. Resetting the dough by hand allows us to create many different and more complex initial shapes (see Figure 5.5).

Both RoboCraft and ScultpBot focus on learning one fairly sophisticated dynamics model and employing it in a planning framework to sculpt target dough shapes with impressive results. In contrast, our work utilizes a relatively simple approach for the dynamics model. We employ a multi-layer perceptron that learns to predict the resulting shape in a lower-dimensional latent space on a learned latent representation (for more details see Section 3.2) of the initial dough shape. We set the focus on learning 3D point cloud representations with three different promising network architectures (PointNet 3.2.1, PointNet++ 3.2.2, Point Transformer 3.2.3). Similar to SculptBot's utilized Point-BERT

feature learner, Point Transformer is also based on self-attention mechanisms. However, in contrast to SculptBot, we train this network directly on our own provided dataset. We are interested in the networks' capabilities of learning point cloud representations and compare their effects when employed in the dynamics model.

3. Foundations

3.1. 3D data representation

To represent the captured RGB-D data, we use point clouds. They are a widely used format for 3D scanning and modeling due to their capability of capturing complex real-world 3D objects with a relatively high precision [18]. Another commonly used format for representing 3D data is meshes, which represent 3D data by vertices, edges, and faces. Compared to point clouds, a mesh-based representation makes assumptions of surfaces and can mitigate noise this way. These assumptions can be beneficial to close holes in the shape, caused by occlusion. This leads to better connectivity of the segments of the whole geometric shape, allowing for a more precise representation of the 3D object [19]. However, mesh interpolation can lead to an oversimplification of the geometric shape, and meshes sometimes incorrectly connect segments or close gaps. Point Clouds are better suited for representing the raw 3D data because they avoid modifying the data by interpolation [19]. Point clouds are also comparably much simpler data structures than object meshes. Object meshes are reliant on connectivity information and represent 3D data by vertices, edges, and faces. In contrast, a point cloud is defined as a discrete set of points in Euclidean space. Each point consists of its three cartesian coordinates (x, y, z), which define its position in a 3D coordinate system. Point clouds provide a simple representation of 3D objects, but unlike meshes or graphs, they lack information about how individual points connect to their neighbors [18]. Instead, the Euclidean distance metric can be used to determine neighboring points that form meaningful subsets this way. In addition to their coordinates, points may contain more features like RGB color values or normals. A known characteristic of point clouds is their unstructured nature. Point clouds are sets of unordered points in contrast to the ordered structure of pixels in 2D images. Hence, they are permutation invariant. That means the resulting point cloud will always look the same regardless of the permutation of the points. Due to the high success of deep convolutional neural networks (CNNs) in 2D image processing [20], it

is a common approach to translate these types of neural network architectures from 2D deep learning to 3D deep learning [21, 22, 23]. However, CNNs work on structured data. Therefore, a common approach is to structure point clouds by transforming them into 3D voxel representations. Nevertheless, there are significant drawbacks to 3D voxelization. Space in voxel grids is represented as a 3-dimensional array, which makes them really memory inefficient and leads to a cubical increase of memory and computational time with an increasing number of voxels [24, 21]. Because point clouds result from 3D scans of depth cameras in this research, they only contain the outer layer of an object. Inside the 3D-scanned object representation, there are no points. That means besides the already high memory requirements of voxel grids, there is an additional factor of data sparsity. Voxel grid representation of sparse point clouds is inefficient since this process transforms the whole object into voxels regardless of sparse point density or even completely empty areas. This relatively inefficient transformation from point clouds to 3D voxels limits the resolution of voxel grids, which inevitably leads to information loss. While there have been efforts to break this stereotype of high computational cost and memory inefficiency [22], we want to create a representation of the dough with a high precision while also keeping a high information density to be able to learn complex shapes and small details of various kinds of dough shapes. Another advantage of using point clouds to represent geometric data is that it is straightforward to perform simple transformations on them like rotations or translation [8]. Such transformations can be applied to each point separately to transform the whole point cloud without changing its global shape or the relationships between neighboring points. This feature of point clouds is particularly useful for augmenting the dataset later by applying rotations to the point cloud samples, thus increasing its size. Recent works of PointNet and PointNet++ have shown remarkable results in performing deep learning directly on 3D point cloud input data. Thus, we keep point clouds as a representation format for the 3D data instead of 3D voxels. Directly using point clouds is a much more straightforward and practical approach since it spares us one additional data processing step, making the data processing pipeline from RGB-D image to final input for the neural network more transparent and faster.

3.2. Deep Learning for 3D point clouds

We need neural network architectures that can handle point clouds as inputs in order to learn a visual representation model of the different dough shapes. As mentioned, we focus on PointNet and its follow-up work, PointNet++. These two relatively simple network architectures are often used to classify 3D objects and segment unstructured 3D point

clouds in whole scenes or parts of them. Additionally, we explore Point Transformer, a network based on self-attention mechanisms. For our work, we are only interested in the network capabilities to identify and extract detailed local features of complex 3D objects, such as different dough shapes. In the following sections, we briefly explain the architectures of PointNet, PointNet++, and Point Transformer, focusing mainly on their feature extraction modules.

3.2.1. PointNet

The intuitive idea of PointNet is to learn a compact set of important points that still captures the shape and most essential features of the initial point cloud [5]. PointNet achieves this by first applying feature transformations to the three coordinates of each input point with multi-layered perceptrons (MLP). Each point is transformed from a 3-dimensional vector into a much higher-dimensional feature vector. In the next and most crucial step, a symmetrical function (max pooling) aggregates the learned point features into a single global feature vector. This feature vector encodes the most essential key features of the input point cloud. Because every point is processed the same way and because of this function's symmetry, the order of the points in the point cloud does not matter.

PointNet is an easily applicable but promising approach to understanding and processing 3D point clouds. Still, while it shows remarkable results, it struggles to capture more minor and more detailed features of complex 3D shapes [6]. Because of this loss of finer geometric details, we also utilize PointNets follow-up work, PointNet++, which can perform a more fine-grained feature extraction than its predecessor.

3.2.2. PointNet++

While the methods used in PointNet++ are fundamentally different from conventional CNNs for 2D image learning, they build upon the same intuitive idea of hierarchical local feature learning. First, small local features and patterns are learned by applying mini-PointNets on a regional scale. Then, these learned local features are combined and processed into larger units in later layers from which another mini-PointNet can learn more complex features. PointNet++ contains multiple set abstraction layers that progressively capture more abstract point cloud features. Each layer processes input points to less and less outputted elements. A set abstraction layer consists of the following three layers:

- **Sampling layer**: In CNNs, a filter (kernel) slides over the ordered 2D input array and progressively captures local features, but this is impossible on unordered 3D point clouds. Instead, this layer utilizes iterative furthest point sampling (FPS) to determine points that will later be the center of local feature regions (centroids). These local feature regions will be necessary in the next steps for hierarchical feature learning.
- **Grouping layer**: In this layer, a ball-query algorithm with a certain radius is applied to each previously sampled point to determine their local neighborhood points. This way, the input point cloud is segmented into many local point neighborhoods, and due to FPS, they evenly cover the whole point cloud.
- **PointNet layer**: This layer utilizes the same architecture as PointNet to encode the inputted local point neighborhood into a feature vector. PointNet's ability to flexibly handle point clouds of any given input size is handy since the local point neighborhoods can contain a variable number of points.

The set abstraction layer outputs fewer points than the number of points of the input point cloud. Additionally, each point is enriched with a local feature vector produced by the PointNet layer to summarize the local context. The outputted points of the set abstraction layer can again be used as inputs for the next set abstraction layer to progressively capture more abstract point cloud features. Similar to PointNet, in its final set abstraction layer, PointNet++ applies a max pooling function to aggregate the features of all remaining points to get a global feature vector representation of the whole point cloud. The crucial difference is that PointNet++ has learned smaller local features, enabling it to capture more complex shapes in greater detail than PointNet.

3.2.3. Point Transformer

Inspired by the success of transformer and self-attention networks in 2D image learning, Point Transformer [7] is an approach to learning features of 3D point clouds by attention mechanisms and hierarchical processing. This architecture allows Point Transformer to learn local and global point cloud features. The feature encoder in Point Transformer networks consists of five stages. Each stage operates on point sets that are progressively downsampled, and the feature dimension increases in every stage. Each stage consists of a point transformer block and the transition down block.

- **Point transformer block**: The point transformer block performs self-attention in the local regions of the input point cloud. It gets a set of points with their 3D

coordinates and features as input. The transformer block consists of multiple layers, including a self-attention layer. Additionally, a multi-layer perceptron creates a position encoding to capture spatial relationships between points. This encoding is integrated into the self-attention layer, allowing the self-attention mechanism to consider the points' features and their positional information.

- **Transition down block**: The transition down block is important to reduce the number of features. It gets a set of points $P1$ with their 3D coordinates and their features as input. First, the number of points is reduced by applying FPS. We call this downsampled point set $P2$. FPS ensures that the downsampled point set $P2$ maintains a good coverage of the original set of points $P1$. Then, a kNN search is applied to each point of $P2$ to find its nearest neighbors in the original point set. For each point in $P2$, max pooling is applied to its neighborhood to aggregate the point features of all points within this neighborhood. These point neighborhoods contain not only the local points from the downsampled point set $P2$ but also points from the original point set $P1$. This way, information from the previous higher-resolution point set $P1$ can be aggregated to the lower-resolution point set. This way, the transition down block effectively reduces the number of points while simultaneously still encoding all the features of the global point cloud. A point transformer block then processes the output of the transition down block again in the next stage.

After the last transition down block, the resulting feature vector is processed by a MLP before applying global average pooling to aggregate the features of all remaining points to a global feature vector.

3.3. Autoencoders

An important type of deep neural network architecture utilized in our work is the autoencoder (AE). These networks learn to compress an input into a compact latent representation while simultaneously learning how to decode this latent representation back to the original input. AEs consist of an encoder and a decoder. In our setting, the encoder receives a 3D point cloud as input and learns how to transform it into a lower-dimensional space representation. This representation space is often called latent space, and the compressed form of the original point cloud in this space is called latent vector Z or sometimes, in literature, the bottleneck size. By learning to compress a high-dimensional input like a point cloud into its low-dimensional representation, the encoder learns how to extract the most important features of the point cloud. The decoder of the AE learns how to

reconstruct the original point cloud from the latent vector. AEs are trained by minimizing the error between the input point clouds P and the decoded output point clouds $\hat{P}$. This approach makes it very convenient to train AEs because they are not reliant on labeled data. Furthermore, AEs ability to reconstruct a latent space vector back to a point cloud is essential because we are interested in learning a generative dynamics model. By generative model, we mean a model that can output point clouds. These point clouds represent the dough's shape after it has been modified by the robot's grasping action. An essential challenge in AE architecture is finding the right latent space size. If the latent size is too small, the decoder might be unable to capture all the different features of a complex point cloud shape. Due to this extreme compression, some critical features could be lost. Otherwise, a large latent space might lead to weaker feature extraction because the model is not forced to focus only on the most important key features of the point cloud.

4. Method

Our method consists of three modules. Module 4.1 is the vision system. It describes our pipeline for recording the raw RGB-D and transforming the data into meaningful 3D point clouds. In the next module 4.2, we describe how we use this point cloud data to train different architecture variants of autoencoders to learn 3D data representations of the dough shapes. Finally, in module 4.3, we elaborate on how we integrate the pre-trained autoencoders of module 4.2 to train a dynamics system that, given the robot's grasping action and initial dough state, learns to predict the resulting shape.

4.1. Vision system

4.1.1. Sensing visual data with a multi-camera setup

Camera calibration

We use 4 Intel® RealSense™ D435i [25] cameras in stationary positions around the dough to record the RGB-D data. Additionally, we use two Intel® RealSense™ cameras D405 cameras as wrist cameras for the robot arm, which capture RGB-D data of the dough from above. The ideal depth sensing range for the D405 camera model is between 7cm and 50cm, making it suitable as a wrist camera that is relatively close to the dough right before and during the dough manipulation. These two wrist cameras aim to provide a closer viewing angle and a unique perspective on the dough during manipulation, reducing the heavy self-occlusion caused by the robot gripper and the dough itself. We set static and wrist cameras to their highest RGB-D resolution (1280×720) to capture the data in the highest quality possible. Since we are using stationary cameras and constantly moving wrist cameras, we are opposed to the two following pose estimation problems. The first one is the eye-on-base pose estimation for the static cameras. And the second

one is the eye-in-hand pose estimation for the two wrist cameras. For our experiments, we use the easy-handeye calibration tool [26], an automated hardware-independent pose estimation tool for ROS1 that relies on ArUco markers [27]. Square fiducial markers like ArUco make it easy to extract the camera poses if the camera's intrinsics are correctly calibrated. We place the marker on the robot's flat workspace for the dynamic pose estimation of the wrist cameras. For the static pose estimation, we use a reusable adhesive putty to fix an ArUco marker on the robot hand. We ensure that the ArUco marker is always near the dough's manipulation center. During the hand-eye calibration, maximizing the rotation while minimizing the translation between poses is crucial [26]. With our hardware configuration, we found that 12 samples are the best number of samples for the pose estimation calculator. Before conducting the data collection, the calibration results are visually inspected in RViz [28] by closely examining the alignment of all partial point clouds. This is a solid initial guess for the camera poses. The calibration results in an error of estimated 5 to 8 millimeters (except for one static camera that had a poor calibration; for more details, see Chapter 7), which is close to the best achievable result according to the authors of easy-handeye [26]. The merged point cloud using the hand-eye calibration results can be viewed in Figure 4.1a. We use the estimated extrinsic matrices for each camera to merge all six point clouds.

Calibration fine-tuning

Nevertheless, the calibration results still need to be more precise for high-quality point clouds. In Figure 4.1a, on the top left of the dough is a visible hole resulting from the inaccurate calibration. The diameter of the 3D-printed fingers we use for manipulation is 18 millimeters, but changes in the shape of the dough caused by a grasping action can still be nuanced within a range of a few millimeters. Additionally, we argue that the calibration error accumulates over six separate cameras.

To further fine-tune the calibrations, we use the iterative closest point (ICP) algorithm [29] to perform point cloud registration on our merged point clouds. ICP needs a relatively "accurate" first guess to avoid getting stuck in possible local minima. Therefore, the initial calibration results from the previous hand-eye calibration are helpful. For this calibration, we record an asymmetrical 3D object (Figure A.1) before the data collection. Because the position of this object is unknown to us in real life, we use the best calibration result of one static camera as the reference position of the 3D object. Fine-tuning each camera pose with this reference point cloud significantly improves the calibration quality. To view the results of the ICP fine-tuning, see Figure 4.1b. After applying the calibration correction,

the hole in the top left of the dough is closed, and the overall shape appears more clearly. We recognize the limitation that this approach does not provide a direct calculation of the estimation error for the reference camera. Using the object's actual pose instead of only an estimation would result in a more accurate ICP calibration for all cameras.

Camera synchronization

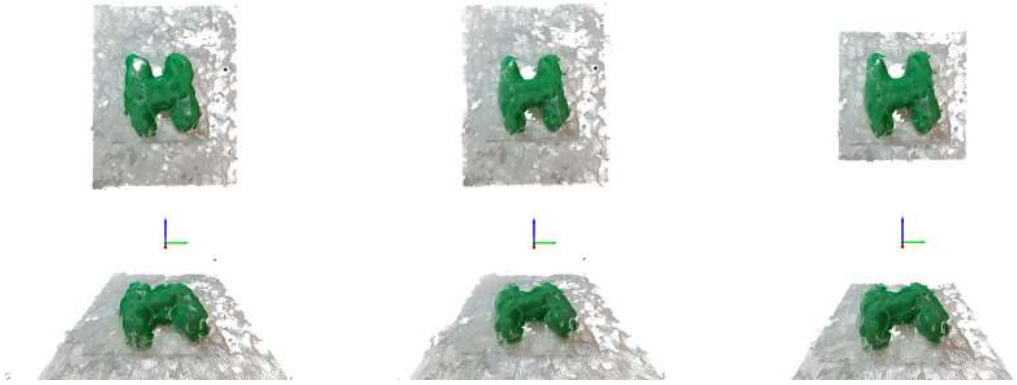
While it is possible to hardware-synchronize multiple Intel® RealSense™ D435i cameras by connecting them with a sync cable [30], the Intel® RealSense™ D405 cameras do not have this option. However, we argue that synchronizing the cameras is not necessary if their publishing rate is high enough. We set each camera to 15Hz (i.e., an image every 66.7 millisecond) for the recording. This constraint means that two recorded frames from two cameras can be at max 66.7 milliseconds apart. However, after the recording, we match each frame from one camera with the closest corresponding frame from the other cameras. This matching process results in a time difference of only $66.7/2$ milliseconds between frames. This significant time window could still be problematic, especially for fast movements. The used cameras also allow for a recording at 30Hz [25], effectively cutting the time window for asynchronous frames in half, but this would also double the storage consumption. We are only interested in specific key frames of the recordings. Particularly important are the moments before and after the manipulation, when the robot's grasping action ends. This consideration means that we do not have to deal with fast movements, so recording at 15 Hz is accurate enough even without a hardware-synchronized camera setup.

4.1.2. Finding key frames

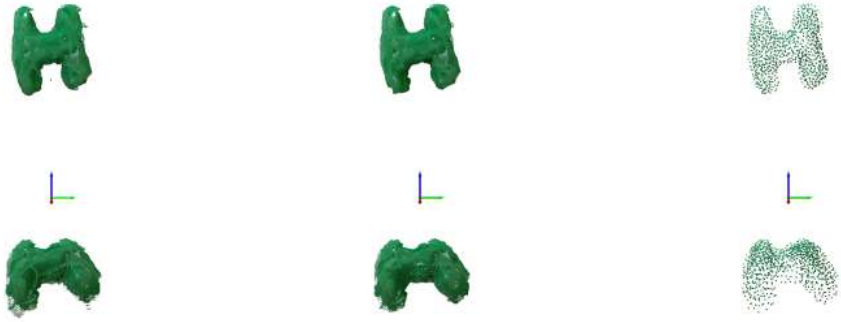
We want to predict the dough's resulting shape, given the robot's grasping action and the dough's initial state. This focus means we only need specific key frames during the dough manipulation. For the robot state, we are interested in the state right before the start of the manipulation. To find this specific starting time frame, we search for the moment when the velocity of the parallel gripper equals zero and its acceleration is nonzero (this is the last frame before the fingers start to move). The next robot state we want to collect is right at the end of a grasping action when the two fingers are closest together and are about to open again. We determine this state by searching for the shortest distance between the fingers right after the start of a grasping action. Because we want to avoid as much obstruction as possible, we do not use this grasping action ending frame as the time frame for the dough shape. We aim to take frames with as little obstruction as possible caused by the gripper fingers. To do that, we move the robot arm a few centimeters above the dough so that each camera can see the dough clearly without any obstruction. We also ensure that the distance between the dough and the two wrist cameras on the robot is still within their range to capture the dough. However, this method has the flaw that it captures the dough in a state where the grasping action has ended, and the fingers are pulled apart again. Additional deformations of the dough can occur during the process of opening the gripper, caused by its sticky properties. The dough can stick to the fingers and stretch or tear as the gripper opens, potentially altering its shape. Despite being aware of the potential problem, we prioritize a clear, unobstructed view of the dough. The impact of these deformations on the dough is mostly minor.

4.1.3. From RGB-D to 3D point cloud

The Intel® RealSense™ cameras have a built-in tool for directly transforming the recorded color and depth images into colored 3D point clouds. A frame-by-frame recording of such point clouds would become memory-intensive compared to recording color and depth images separately. Colored point clouds have to store the (x, y, z) coordinates and the RGB values of each point, resulting in six stored values for each recorded point. In contrast, a color image stores only three RGB values per pixel, and a depth image stores only one depth value, resulting in a total of four stored values for each recorded pixel. Thus, recording color and depth image data is more memory efficient than directly recording colored point cloud data. We do not use any compression formats because we want to maintain the high image quality of the RGB-D data. Instead, we record the data in the RAW image format [31]. A RAW image comes directly from the camera's



(a) **Merged raw point cloud** (b) **ICP fine-tuning** (c) **Bounding box cropping**



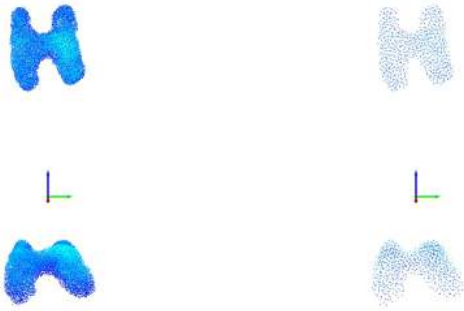
(d) **HSV color filtering** (e) **Outlier removal** (f) **Downsampling to 1024 points**

Figure 4.1.: Data processing pipeline of the whole vision system (Part 1: RGB-D to filtered point cloud). (a) Original point cloud after using the hand-eye calibration results to merge all partial point clouds. (b) The point cloud after fine-tuning the calibration using ICP. (c) The point cloud after applying position-cropping to remove more of the surrounding scene. (d) The point cloud after applying a HSV color filter. (e) The point cloud after applying outlier removal to filter out noise of the depth cameras. (f) The point cloud after applying FPS to evenly downsample to 1024 points. These point clouds create the foundation for the *point cloud shell* dataset.

(g) **Added lower plane (blue) to original point cloud (red)**

(h) **Poisson surface reconstructed mesh**

(i) **Volumetric point cloud after SDF sampling**



(j) **Shape refinement**

(k) **Downsampling to 1024 points**

Figure 4.1.: Data processing pipeline of the whole vision system (Part 2: point cloud volumetrization). (g) Using the 2D projection of the point cloud as floor (blue) for the rest of the point cloud (red). (h) Creating a watertight mesh with Poisson surface reconstruction. (i) Sample points inside the mesh with SDF. (j) Remove over smoothed points to fit the original shape. (k) Evenly downsample the volumetrized point cloud to 1024 points. These point clouds create the foundation for the *volumetric point cloud* dataset.

image sensor and is not preprocessed or compressed in any form. Besides the high image resolution, the RAW format provides flexibility for the data processing later. We use each camera's intrinsic matrix to transform the RGB image data and the depth image data into partial, colored 3D point clouds. This matrix contains camera properties like focal length, principal point, and skew. These properties define the camera's internal geometry and are used to convert the RGB image data and the depth image data into colored 3D point clouds. The camera intrinsic matrices for the applied cameras can easily be obtained by a built-in function of the ROS1 wrapper for Intel® RealSense™ cameras [32]. However, each partial point cloud is located in the local coordinate frame of the camera it was recorded with. Thus, we apply the camera extrinsic matrix to each partial point cloud to merge all six point clouds into one full point cloud. These extrinsic matrices are obtained in Section 4.1.1, where we estimate each camera's pose with respect to the robot base. Figure 4.1a shows the point cloud after merging all partial point clouds using the extrinsic matrices.

4.1.4. Point cloud pre-processing

Extracting the dough object

Because the exact location of the dough manipulation platform is known, we perform a bounding box cropping using its coordinates. This process significantly reduces the memory size of the point clouds and makes further pre-processing much faster. The cropped point cloud can be viewed in Figure 4.1c.

After cropping the bounding box, we apply a color-based filter to extract the green dough from its surroundings. The color segmentation is done in the HSV color space, better suited for this task than the RGB color space [33]. Another commonly used color space for color segmentation is the L*a*b* color space that SculptBot utilizes. However, studies show that the HSV color space yields even better results [34]. The results of the color filtering can be viewed in Figure 4.1d. The surface properties of the dough's manipulation platform and lighting are crucial for good color segmentation. A dark color for the platform results in a worse segmentation quality for the green dough. A smooth, reflective surface, often resulting from 3D-printed objects, also results in worse color segmentation. Additionally, a smooth surface can introduce noise to the depth images of the applied cameras. Therefore, we use white adhesive tape to create a matte, non-reflective, and bright surface for the dough manipulation platform (see Figure 5.2). The robot arm and the dough's self-occluding properties can create shadows on the dough. While only

minimal, the shadows can still introduce noise, especially on the outer edges of the dough, so besides the natural room lighting, we use an additional static light source to create a more even scene lighting (see Figure 5.3a. We recommend using even more lighting sources. For example, in the direction of each camera, one lighting source.

Filtering the point cloud

Due to noise from the depth cameras, we still have to deal with many outlier points, which we remove using Open3D's statistical outlier removal algorithm [35]. Figure 4.1e shows the point cloud after applying statistical outlier removal. This algorithm removes points further away from their neighbors than the average neighbor-to-neighbor distance for the point cloud. It is essential to ensure that this filter is not set too strictly. Otherwise, it is common to lose important points of the dough shape, especially at the outer edges of the lower parts of the dough close to the manipulation platform.

We then downsample the point clouds to 1024 points using farthest point downsampling [36] (FPS). These filtered and downsampled point cloud shells, only containing the surface layer of the dough's shape, create the first initial dataset. Because two cameras (the wrist cameras) capture the dough from above, the recorded point clouds often have a higher point density at the top and a much lower point density at the lower parts of the dough closer to the manipulation platform. This downsampling method ensures an even distribution of points in the downsampled point clouds. Figure 4.1f shows the downsampled point cloud.

After downsampling, we remove the color of the point clouds. PointNet and PointNet++ can handle 3D data with additional features like RGB values, but this would lead to higher computational complexity. For our research, we are only interested in predicting the dough's resulting shapes, not its colors. The colored point clouds are only necessary for the object extraction of the dough.

Obtaining a watertight mesh

Because of the multiple viewing angles we capture during the recording with the six-camera setup, we can avoid severe self-occlusion caused by the concavity of more complex shapes. However, the more cameras we use, the more apparent the inaccuracy of the camera calibration becomes. This issue results in point clouds with many separate layers that do not perfectly fit together, making the surface of the 3D point clouds of the

dough shapes uneven and noisy. Therefore, we create a second dataset from volumetric point clouds with a smoother outer surface. In addition to point cloud shells, we are also interested in the effects of volumetric point clouds on the dynamics model, which include points located within the shapes. To reduce this noise and to smooth the surface of the multiple point cloud layers, we apply Poisson surface reconstruction [37] and alpha surface reconstruction [38]. Poisson surface reconstruction is especially robust to noisy data, like the multiple point cloud layers of our point clouds, and creates smooth surfaces that perfectly fit the smooth and unstructured surface of the dough. We use RoboCraft's [1] implementation for the Poisson surface reconstruction algorithm and the alpha reconstruction algorithm. However, the point clouds are missing the dough's lower surface, which is connected to the manipulation platform. With such a big hole in the point clouds, both surface reconstruction algorithms struggle to reconstruct a mesh of more complex shapes accurately. Alpha surface reconstruction struggles to create a watertight mesh (visualized in Figure 4.3c), and Poisson reconstruction tends to "over smooth" the shapes, losing many details of the shapes in the process (visualized in Figure 4.3b). The solution to this problem is to estimate the lower surface of the shapes by simply projecting all points of a point cloud onto the 2D plane (XY-plane) like a shadow where the dough is connected to the manipulation platform (visualized in Figure 4.2). We argue that this approach is sound because the concave properties of the dough shapes only unfold in the x and y directions, hardly in the z-direction. However, this assumption is not completely true and shapes can be "rugged" and can have concave properties from above to some extend. Figure 4.1g shows the point cloud (red) with the newly added lower plane (blue).

After estimating the dough's lower surface, we can finally apply Poisson- and alpha surface reconstruction. While only an approximation, this lower surface significantly improves the quality of the reconstructed and watertight mesh. Each point cloud has to be individually downsampled to $700 - 1024$ to get optimal results for the mesh reconstruction. Using a higher number of points generally results in higher computational complexity but does not necessarily lead to a better mesh. By default, we use a point size of 1024 with decent results, but sometimes, a lower point size of 700 can also be beneficial if a point cloud is extra noisy to create a smoother surface. We mainly apply Poisson surface reconstruction to the dataset, which can create meshes that are not too smooth and keep the details of the initial shape after enclosing the point cloud shell with the approximated lower surface. Figure 4.1h shows a watertight mesh obtained with Poisson surface reconstruction. The over smoothing effect of Poisson surface reconstruction is still visible in the bottom right and left of the dough, but it is mostly minor. However, for some shapes where Poisson surface reconstruction still loses too much detail, we apply alpha surface reconstruction. Because the point clouds no longer contain big holes, alpha surface reconstruction can

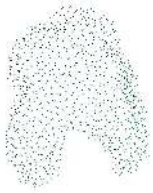
also produce accurate and watertight meshes in some particular cases. Overall, we found that Poisson surface reconstruction is still superior to alpha surface reconstruction for most point clouds. Additionally, Poisson surface reconstruction creates rounder meshes that fit the overall structure of the dough shapes much better.

(a) **Original point cloud** (b) **2D projection of the point cloud**

Figure 4.2.: Sampling process of the bottom surface of the point cloud. The point cloud (a) is projected on the XY-plane (b). This projection serves as an estimation for the bottom surface of the dough. The original point cloud (a) and the lower surface estimation of the dough (b) can be merged to create an enclosed point cloud shell.

Creating volumetric point cloud shapes

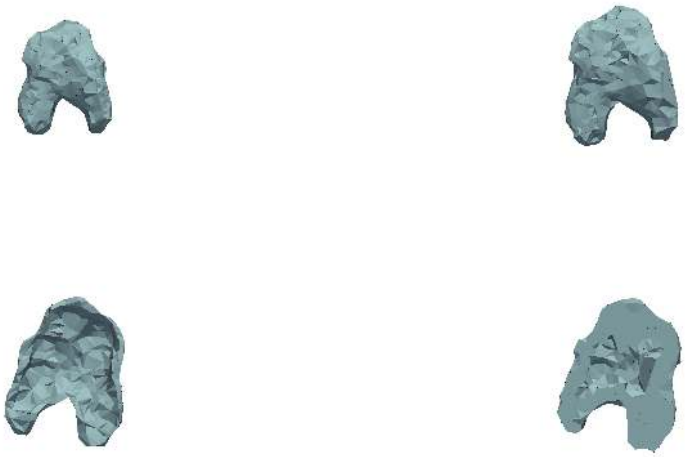
In the next step, we use the obtained watertight meshes of the point clouds to sample the points inside the shapes, giving the point clouds volume. We randomly sample 100.000 points inside the bounding box containing the dough and then use the signed distance function (SDF) to remove all points outside the reconstructed watertight mesh of our shapes. We sample such a high number of points to create a dataset with dense point cloud values to be later able to downsample the point clouds to a flexible number of points. After this step, the resulting 3D point clouds not only have volume but also have a smooth surface, as can be viewed in Figure 4.1i. However, in some instances, Poisson surface reconstruction still tends to over smooth the original shape significantly (see Figure 4.4b). To mitigate this effect, we use a heuristic approach involving the lower plane we obtained at step 4.1g. We remove all points with x and y coordinates that lie outside this plane that represent the base shape of the dough. We argue that this approach is viable because, as seen in Figure 5.5, the dough shapes we use can be projected on a 2D plane (XY-plane) and still keep their original shape. We are aware of this approach's limitations, but we argue it is still viable because the results of the Poisson surface reconstruction are mostly accurate. We only apply this method in severe cases (like in Figure 4.4b) to remove local parts that the Poisson algorithm struggled with. The results of this point cloud refinement can be viewed in Figure 4.4d.



(a) **Original empty point cloud shell**

(b) **Poisson on open point cloud shell**

(c) **Poisson on enclosed point cloud shell**



(d) **Alpha on open point cloud shell** (e) **Alpha on enclosed point cloud shell**

Figure 4.3.: Surface reconstruction comparison between open point cloud shell and enclosed point cloud shell with the reconstructed lower plane. (b) Poisson surface reconstruction massively oversmooths the mesh of the open point cloud shell, thus losing much detail. (c) Alpha surface reconstruction captures the base shape of the open point cloud shell well but is not able to reconstruct the lower floor resulting in an open mesh. (d) After enclosing the point cloud shell by adding the lower plane Poisson accurately reconstructs a watertight mesh of the dough shape. (e) Alpha surface reconstruction nearly constructs a perfect mesh but it is not watertight, because of a small hole on the back side of the mesh.

We use FPS to downsample the point clouds to 1024 points while ensuring an even distribution of points. The more points a point cloud contains, the more accurately it can represent the actual dough shape, but this comes with a higher computational cost during the training process of our dynamics model. This point cloud size is also commonly used in PointNet and PointNet++. Furthermore, point clouds containing 1024 provide sufficient detail to represent the dough shapes accurately. The final volumetrized point cloud can be viewed in Figure 4.1k. At this point, we want to mention that we not only provide the volumetrized point clouds with 1024 points 4.1k but also the dense volumetrized point clouds (see Figure 4.1j) in a dataset. This could particularly be useful for future work that wants to learn 3D point cloud shapes with a much higher resolution than 1024 points.

(a) **Original point cloud shell with lower plane added**

(b) **Poisson mesh reconstruction over smoothing effect**

(c) **Resulting SDF sampled point cloud**

(d) **Refining the volumetric point cloud based on the lower plane of the dough**

Figure 4.4.: Volumetric point cloud refinement process. This figure visualizes how our refinement method can improve the original shape of a point cloud that has been over-smoothed by Poisson surface reconstruction.

4.2. Learning dough representations

4.2.1. Autoencoders

To make meaningful predictions of how the dough is deformed by the robot first it is essential to bring it into a form that is suitable for deep learning. In this research, we propose to use three different autoencoder network architectures that learn the key features of the input point cloud P by converting it into a lower dimensional latent vector Z_P. The encoder network performs this task. We compare different encoder networks and their ability to learn point cloud features. The decoder network for all three autoencoders remains the same. We implement a simple MLP structure for the decoder architecture utilizing linear layers with ReLU activation functions. We compare a simple PointNet architecture 3.2.1, a PointNet++ architecture 3.2.2, and a Point Transformer 3.2.3 architecture for the decoder. Furthermore, we compare the different autoencoders' abilities to extract point cloud features and reconstruct the original point cloud from them. We are also interested in the effects of the different learned point cloud representations on the dynamics model.
See Appendix A.2.1 for more specific autoencoder network details.

4.2.2. Loss Metrics

Previous studies [1, 2, 3, 8] have successfully employed two prominent loss functions for comparing output point clouds with ground truth point clouds. These loss functions have proven to be highly effective in assessing the similarity of point clouds. The first is the Chamfer distance (CD), and the second is the earth mover's distance (EMD). EMD loss is more sensitive to more fine-grained details. Additionally, EMD does not tend to create blurred details like the CD loss when comparing point clouds, especially on point clouds with varying density [8, 39]. However, CD is much less computationally expensive than EMD, and our dataset only contains evenly sampled point clouds. While EMD loss would be better overall since it clearly outperforms the CD loss in terms of point cloud quality [8], we still argue that CD loss is better suited for the model training because it is significantly less computationally expensive. This decision is due to hardware limitations, and it is still possible to get good results with CD loss [8]. During training, we found that a large batch size is recommended on average (see Section 5.4). Small batch sizes result in unstable training due to high fluctuations in the loss. Due to this batch size constraint and the much slower training progress, we choose the CD loss. In [8], EMD and CD loss are used to

learn surface shapes only, whereas one of our datasets is volumetric. We are interested in whether the CD loss is suitable for learning volumetric point clouds. Figure 4.5 illustrates this process. The predicted point cloud $\hat{P}$ (red) is evaluated against the ground truth point cloud P (blue) using the Chamfer distance. However, we would also like to mention that there are efforts to combine CD and EMD that not only outperform both loss metrics but are also more computationally efficient than EMD [1, 8]. The potential impact of using better loss functions is undoubtedly an exciting topic for future work.

4.3. Learning dough dynamics

This section explains the building blocks of the generative prediction model. A schematic visualization of the dynamics model and one prediction example of manipulating dough in the shape of an "H" can be viewed in Figure 1.1. Our dynamics model predicts and outputs the resulting dough shape $\hat{P}_{t+1}$ (in Figure 1.1 illustrated in red), given the dough's initial shape P_t and the robot's grasping action A_t (in Figure 1.1 illustrated in blue). A fundamental architecture for the dynamics model is the encoder network and the decoder network of the previously mentioned autoencoder deep neural networks 4.2. The main idea of this model is to first transform the initial input point cloud P_t into the lower dimensional latent representation Z_{P_t} of shape $(1, n)$ by employing one of the encoders. This representation contains only the most essential features of the initial point cloud structure. Additionally, a multi-layer perceptron (in Figure 1.1 the "Action-MLP") learns to translate the input grasping action into a latent representation Z_{A_t}. Another bigger MLP (in Figure 1.1 the "Global-MLP") takes the latent representations of both the robot action and the initial point cloud as input and processes them into a global latent vector Z_{Gl} of shape $(1, n)$ (the same shape as the latent representation of the input point cloud). Intuitively explained, the dynamics model is trained to learn the effect of the robot's action on the initial point cloud shape in the latent space and creates a latent representation of the resulting manipulated point cloud. The decoder network then translates this latent space encoding of the predicted point cloud Z_{Gl} back into a 3D point cloud $\hat{P}_{t+1}$.
For the training of the dynamics model we are interested in the effects of freezing the layers of the integrated encoder and decoder modules vs. fine tuning their layers (see Section 5.4). Appendix A.2.3 contains more specific network details like layer sizes.

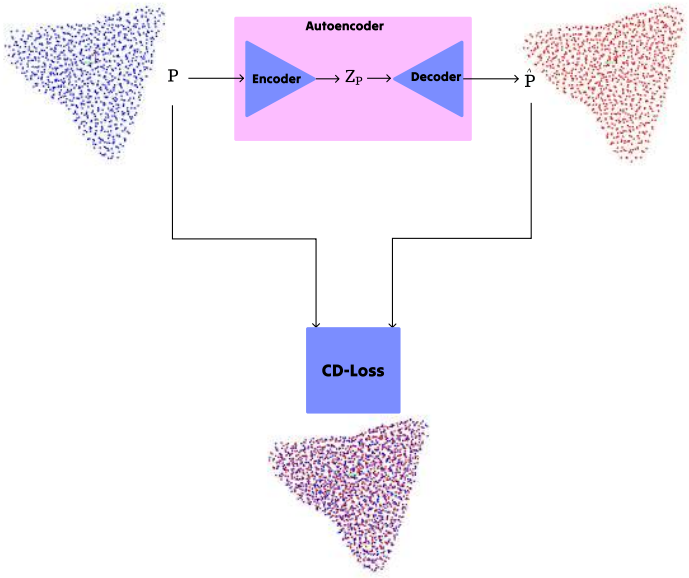


Figure 4.5.: Training of the autoencoder with Chamfer distance (CD). The encoder receives a 3D point cloud P (blue) as input and compresses it into a lower-dimensional representation Z_P. Z_P is a latent vector of shape $(1, n)$, with n being the latent space size (in our case 256). The decoder reconstructs the point cloud (red) from the latent vector. The reconstructed point cloud $\hat{P}$ is compared to the original point cloud P in geometrical space using CD.

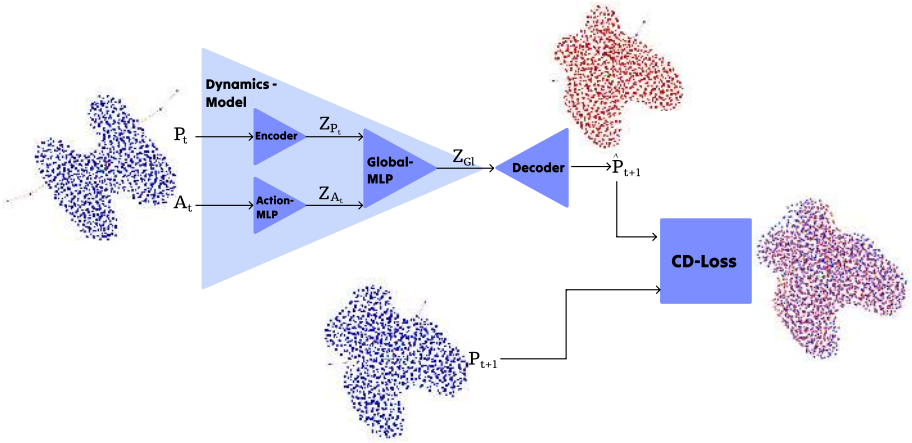
4.3.1. Loss Metrics

To compare the output point cloud generated by the dynamics model to the ground truth point cloud, we use the Chamfer distance for the same reasons previously explained in Section 4.3. A schematic visualization of the training of the dynamics model with CD can be viewed in Figure 4.6a.

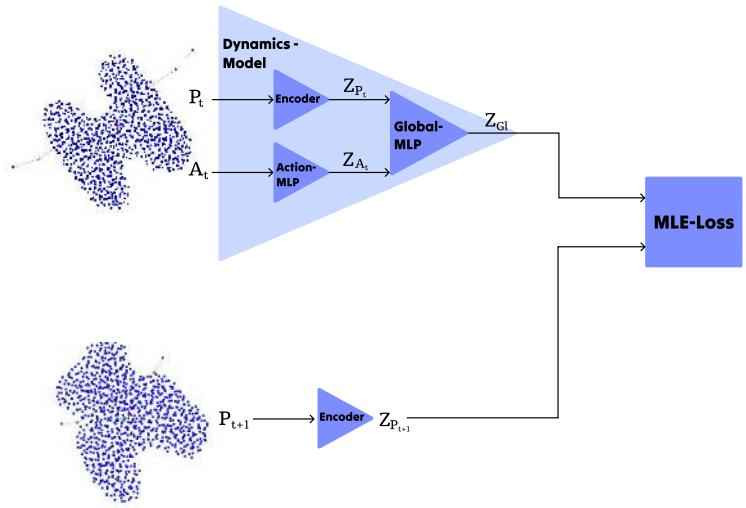
Additionally, we utilize maximum likelihood estimation (MLE) as an alternative method to assess the dynamics model's predictive capability. MLE allows us to directly compare the predicted latent vector with the latent vector obtained from encoding the ground truth point cloud using the encoder network. We are particularly interested in examining the effects of predictions in geometric space using Chamfer distance versus assessing them in latent space based on learned point cloud representations. A schematic visualization of the training of the dynamics model with MLE can be viewed in Figure 4.6.

4.3.2. Action input

We define the action input A_t for the dynamics model (the robot's grasp action) similar to SculptBot. It consists of three components: the x, y, and z coordinates of the end-effector position, the rotation matrix to express the end-effector orientation, and the distance between the two fingertips. The robot end-effector is the center between the robot's fingertips. We define the robot base as the base coordinate frame. Because the position and rotation of the end-effector stay the same during a grasp action, we can represent the movement of the fingers with just the distance between the fingertips before and after a manipulation action. To obtain these transformational values, we record the robot state during the data collection and apply forward kinematics to them. Forward kinematics is a technique that computes the position and orientation of the end-effector based on the robot's joint angles. To further reduce redundant features of this grasp action representation, we only take the first two columns of the rotation matrix. Dropping the last column gives us a 6D representation for the rotation matrix that is continuous [40]. Not only does this representation of a 3D rotation reduce the size of the action input vector, but it is also proven that this representation is better suited for learning with deep neural networks [40]. It is possible to further reduce the size of the grasp action by using 4D or even 3D representations for the rotation, like quaternions or Euler-angles. However, such rotation representations are discontinuous and can lead to significant errors in deep neural network learning [40]. In the end, we are able to represent a complete grasp action as an 11D vector. The first three elements are the end-effector's x, y, and z coordinates.



(a) Training the dynamics model with Chamfer distance (CD). The dynamics model outputs the latent representation of the predicted point cloud shape Z_{GL}. The decoder decodes the latent space vector Z_{GL} to a 3D point cloud $\hat{P}_{t+1}$ (red point cloud). Then, the predicted point cloud $\hat{P}_{t+1}$ is compared to the ground truth point cloud P_{t+1} (blue point cloud) in geometrical space using CD.



(b) Training the dynamics model with maximum likelihood estimation (MLE). The dynamics model outputs the latent representation of the predicted point cloud shape Z_{GL}. The encoder encodes the ground truth point cloud into latent space $Z_{P_{t+1}}$. These two latent representations, Z_{GL} and $Z_{P_{t+1}}$, are then compared using MLE to evaluate the model's performance in capturing the underlying data distribution.

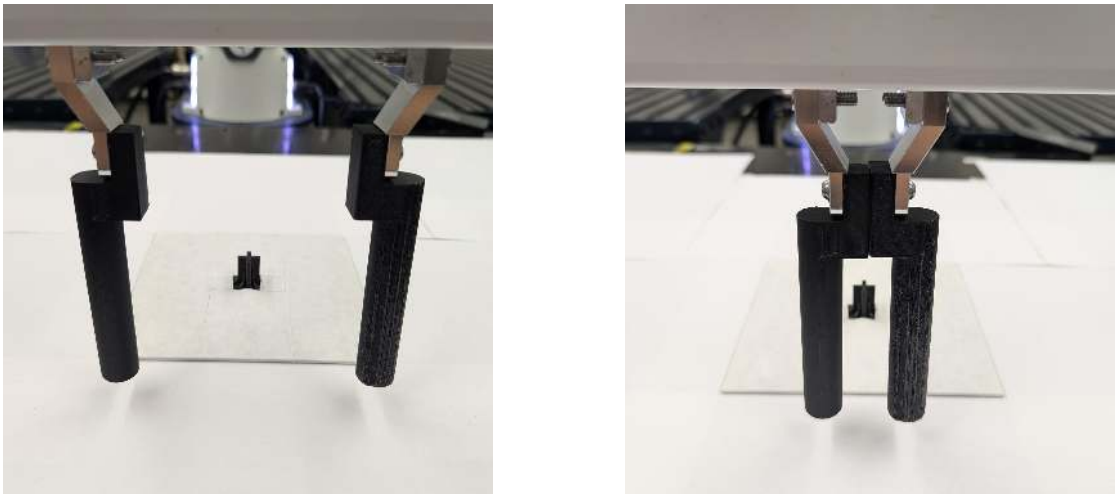
Figure 4.6.: Schematic comparison of training the dynamics model using maximum likelihood estimation.

Elements four to nine are the first two columns of the rotation matrix. The tenth element is the finger distance before the grasp action, and finally, the eleventh element is the finger distance during the grasp action when the fingers are fully closed.

5. Experiments

5.1. Physical setup

For our experiments and data collection, we manipulate green Play-Doh using a physical 7-axis Franka Emika Panda robot arm with a 2-finger parallel gripper. Play-Doh is a soft and bendable modeling compound that can easily be shaped into various forms. The dough used for the data collection weighs about 168 grams (the contents of two conventional Play-Doh buckets). For a detailed look at the physical setup, see Figure 5.3. The fingers are 3D printed in black and shaped like a cylinder. We use RoboCraft's finger mesh [1]. Figure 5.1 shows the gripper fingers in more detail.

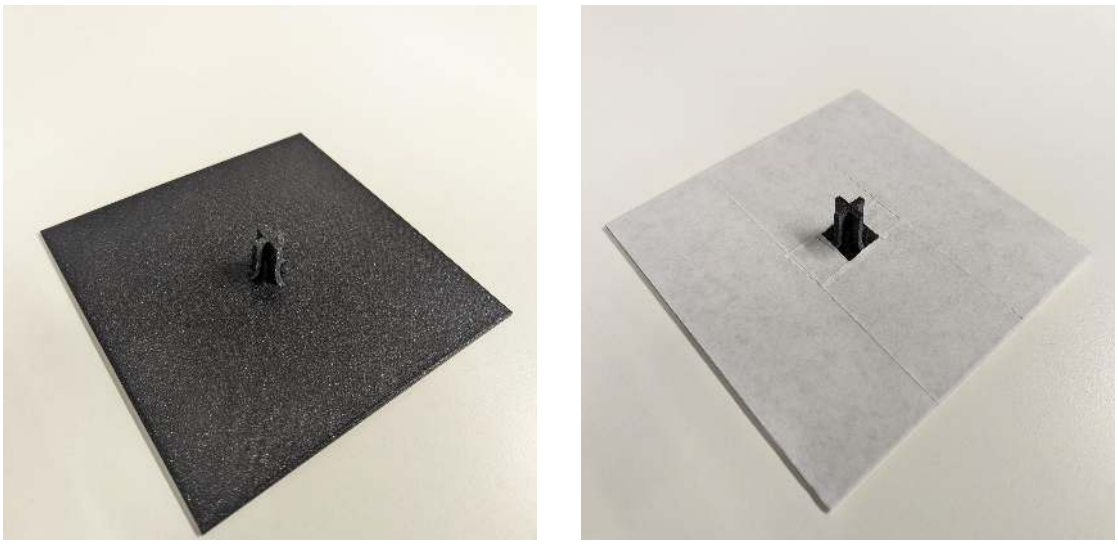


(a) Fingers opened

(b) Fingers fully closed

Figure 5.1.: 3D printed robot fingers to manipulate the dough.

The whole workspace is located on a table 90cm$\times 80$cm table surface. The robot base is at the edge of this workspace. At the center of this workspace is the location of the 3D printed manipulation platform. The platform is a 15cm$\times 15$cm square and 3mm in height. At the center of this platform, a small, cross-shaped plastic rod sticks out (see Figure 5.2). This 3D-printed mount is designed to securely hold the dough during manipulation, ensuring consistent positioning and reducing the risk of the dough being pushed off the platform during a grasp action. During manipulation by the gripper, the dough is placed on top of this plastic rod to hold it in place. To enable better color filtering in the data processing pipeline, we cover the entire workspace, including the manipulation platform, with white A4 paper and white adhesive tape.



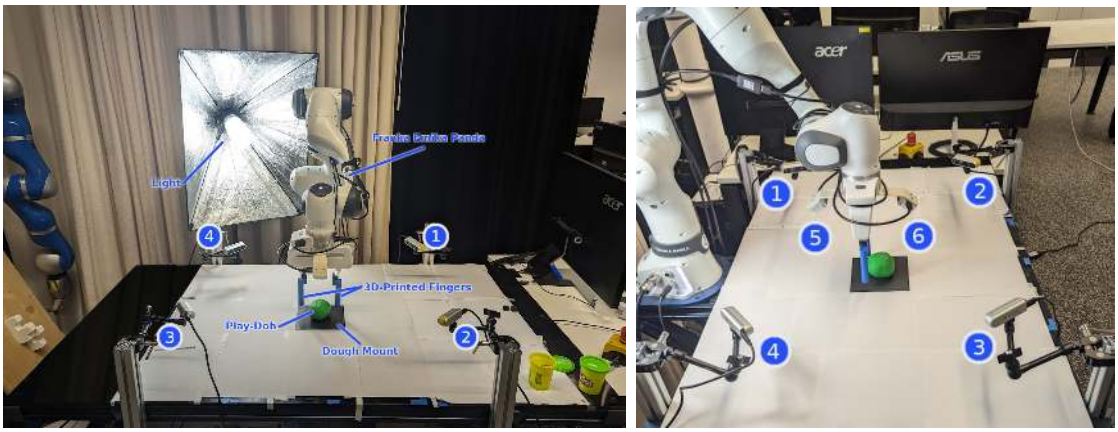
(a) 3D printed dough mount with a plastic rod in the center

(b) Dough mount covered with white adhesive tape for better color contrast

Figure 5.2.: 3D printed dough mount to loosely fix the dough in place during manipulation.

In each of the four corners of the workspace, we position a static Intel® RealSense™ D435i camera [25]. The optimal range of the D435i cameras lies within 30cm to 3m [25]. We ensure each camera is within a 40cm range from the manipulation platform. All four cameras are positioned to allow the dough to be captured evenly from all angles. For the wrist cameras, we use two Intel® RealSense™ D405 cameras attached to a 3D-printed mount at the end-effector link of the panda hand. This way, one camera is on each side of

the hand (see Figure 5.3b). We make sure to create even illumination in the room but to reduce the effects of shadows caused by the robot arm when it hovers above the dough, we additionally position a stationary light source behind the robot arm (see Figure 5.3a). Because of limited bandwidth on one computer, we use two computers for the data recording. We connect three cameras to each computer by cable of 5m. See Appendix A.1 for a more detailed computer setup and synchronization description.
While using cables with a short length as possible to reduce noise caused by the data transfer is usually recommended, we do not find a significant quality difference in the recorded depth images between a 1m cable and a 5m cable. The longer cables are necessary to connect every camera to the computers because of the distance between the robot's workstation and the position of the two computers.

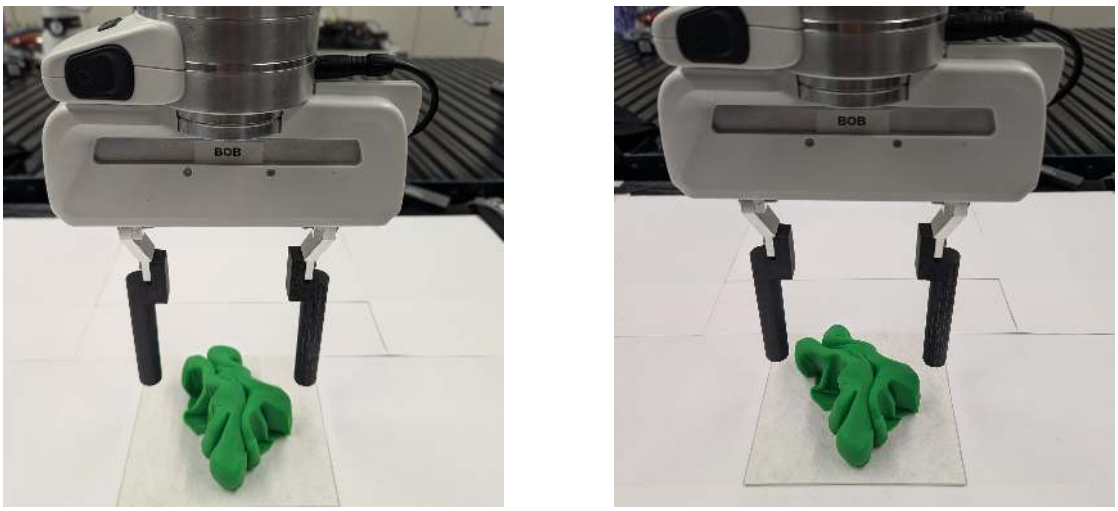


(a) front view (b) side view

Figure 5.3.: Hardware setup for data collection. Numbers **1-6** mark the cameras. **1-4** are the static cameras, and **5/6** are the two wrist cameras attached to the Franka Emika Panda robot. We position a static light source behind the robot to mitigate the effect of its shadow. The dough mount is a small quadratic $15cm \times 15cm$ platform in black, but during the recording, we put adhesive tape on it for better color filtering.

5.2. Data collection

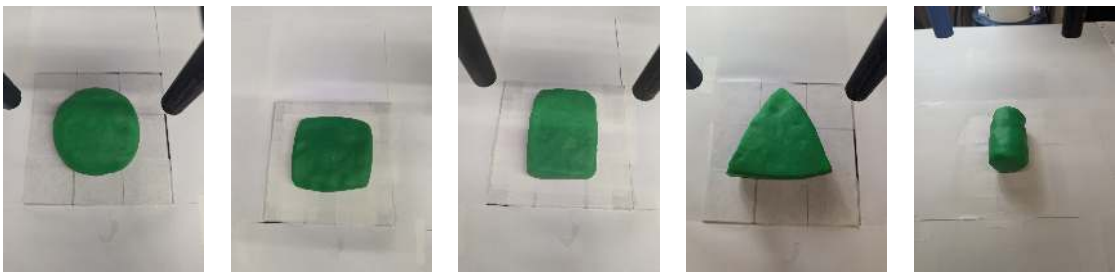
There is a tradeoff with the size of the dough. If the dough is too small compared to the size of the gripper fingers, it is difficult to manipulate the dough meaningfully. If the dough is too big, the robot gripper cannot grab around the dough because the dough is wider than the maximum distance between the two fingers. During data collection, it is essential to replace the dough every three to four hours with fresh dough because it loses moisture, changing its texture and firmness. Automated robot movement is not used during the data collection. Instead, the robot is set to interaction mode and guided by hand. Manual guidance of the robot means that not only the end-effector pose but also the grasp action of the robot fingers is controlled by a human hand. We make sure to close the fingers at a constant speed. While it is more time-consuming, we argue that by doing the manipulation actions of the dough guided by hand, we can create more meaningful and diverse shapes. Automated manipulations would create a lot of redundant manipulations and changes in the shape of the dough that are hardly visible. The robot gripper is freely movable around all axes between each grasp action. During a grasp action, the robot's joints are locked, only allowing the fingers to close and open. In addition, push-down manipulations could be performed to grasp actions by pushing the tip of the fingers down into the dough, further enriching our dataset with more diverse interactions with the dough. However, in this research, such push-down manipulations are not implemented because the dough is only loosely attached to its mount. Due to the dough's stickiness, the dough may stick to the gripper's fingers when pulling the gripper up after such a push-down manipulation.
Similar to SculptBot, we define a *grasp trajectory* as three to five consecutive grasping actions to the dough. The general idea behind stopping earlier than after five grasping actions is that the dough is frequently deformed into the same generic, meaningless shape after three grasp actions. Figure 5.4 illustrates an example of manipulating such a generic shape. The changes in the dough's shape are hardly visible after applying an additional grasp action to it.



(a) Dough at state P_t (b) Dough at state P_{t+1}

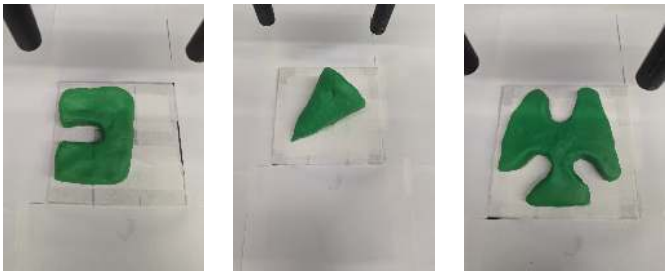
Figure 5.4.: This figure shows the dough after applying a grasp action without resetting the shape. The dough has a generic and undefined shape. The changes in the shape after additional grasp actions are hardly visible.

We want to set the focus on a dataset with more meaningful shapes and manipulations. Each *grasp trajectory* is recorded separately, and after it has ended, the dough shape is reset. For each recording, the resolution of all cameras is set to 1280×720, and the frame rate is set to 15 fps. Depending on the complexity of the initial dough shape, we conduct eight to twelve grasp trajectories for each shape. The idea is that more complex shapes offer more diverse manipulations and resulting shapes, so we can naturally perform more grasp trajectories. The initial shapes are created by hand without any pre-made tools or molds compared to the works of SculptBot or RoboCraft. During the forming process of the shapes, we ensure that there are no wrinkles in the dough. This process is time-consuming but, in contrast to other related studies [1, 3], enables us to create multiple diverse and complex starting shapes (see Figure 5.5). During each *grasp trajectory*, we ensure that all parts of the dough stay inside the $15cm \times 15cm$ square of the manipulation platform because we use these measurements later for the bounding box cropping in the data processing pipeline (Section 4.1). After the data collection and processing, our complete dataset contains 103 grasp trajectories of various shapes.



(a) **Disc** (b) **Square** (c) **Rectangle** (d) **Triangle** (e) **Cylinder**

(f) **C** (g) **F** (h) **H** (i) **S** (j) **Fish**



(k) **Bridge** (l) **Cone** (m) **Special**

Figure 5.5.: Initial dough shapes. More complex shapes like (h) or (m) allow for more diverse manipulations; thus we conduct more *grasp trajectories* than for simpler shapes like (b) or (e)

In total, 438 different manipulation samples. Not only can this dataset be used to train a dynamics model to predict the state of the dough after one grasp action, but also to predict the dough after a sequence of three to five grasping actions.

5.3. Data augmentation

During the training of the models, we propose two augmentation strategies to increase the size of the dataset. First is to apply different rotations to the dough and the action input about the z-axis. As a rotation center, we use the center of the manipulation platform where the dough is mounted. This augmentation strategy is viable because we can assume that the dough always stays fixed on the dough mount. Similar to SculptBot, we augment the samples by applying random rotations in intervals of $6°$. Applying this rotation augmentation effectively increases the size of the dataset by a factor of 60. The second augmentation strategy is to mirror dough and action input against a plane that contains the z-axis. We mirror against the YZ-plane. The mirroring augmentation effectively doubles the size of the dataset. This way, we can increase the original size of the dataset, which is 438 samples, to 52560 samples. Before training, we normalize the point clouds and the action input into a unit sphere [5].

5.4. Model training

We use the Adam optimizer for model training and set the test size to 0.1 and the training size to 0.9. Before we train our dynamics model, we perform a parameter grid search to find the best architecture configuration and training configuration for each autoencoder model (PointNet, PointNet++, Point Transformer). Generally, a greater batch size for training results in less significant loss fluctuations and faster and better learning, so we keep the batch size as high as possible. For the normal PointNet autoencoder, we use a batch size of 1024 samples. Because PointNet++ and Point Transformer models are more computationally expensive the highest possible batch size for training is only 128. While significantly smaller, this size is still adequate and does not lead to unstable learning. In the grid search, we search for each autoencoder model's most optimal learning rate (0.001, 0.0005, 0.0001) and latent space size (128, 256, 512). Due to time constraints, we limit the grid search to one random seed for the weight initialization and stop non-promising parameter configurations earlier. See Appendix A.2 for more details on the grid search.

After obtaining the most optimal autoencoder configuration, we want to find the best configuration to train the dynamics model optimally. We propose three strategies.

A. Initialize an untrained autoencoder network and dynamics model and train the dynamics model on the dataset from scratch.

B. Pre-train the autoencoder network and integrate the pre-trained autoencoder into the dynamics model. Freeze all encoder and decoder layers to update only the dynamics models Global-MLP and Action-MLP.

C. Pre-train the autoencoder network and integrate the pre-trained autoencoder into the dynamics model. However, do not freeze the encoder and decoder layers to enable further fine-tuning of these autoencoder layers during the training of the dynamics model.

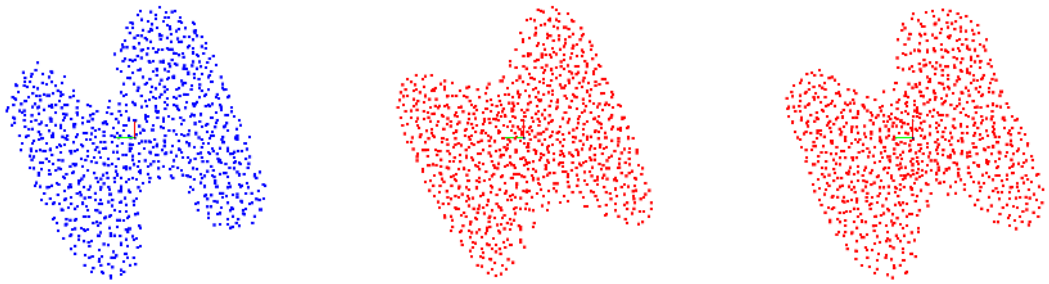
For autoencoder pre-training, we train each autoencoder model only on the input point clouds before a grasp action. We perform three random weight initializations and take the model with the lowest loss for each autoencoder model. Besides training the dynamics model using CD loss, we are also interested in whether MLE loss yields better results, so we additionally train strategy **B** with the MLE loss.

6. Results

6.1. Representation model

6.1.1. Grid search

For the training of the autoencoders, a better CD loss on the test set overall also means a better reconstruction quality of the point clouds (see Figure 6.1). While a higher learning rate of 0.001 initially leads to much faster learning, the decrease in the test-loss plateaus later in the training, and models trained with a learning rate of 0.0001 eventually outperform the models with the higher learning rate. Overall, the model performances between all three latent sizes (128, 256, 512) are only marginal regarding CD-loss. In direct comparison, latent size 128 performs the worst for all three models. A latent size of 512 performs nearly identical to a latent size of 256 for the PointNet++ and Point Transformer models. However, a latent size of 256 for the PointNet model outperforms a latent size of 512 in the later stages of the training. Therefore, and for better comparability, we take a latent size of 256 as the latent size for all three autoencoders.



(a) Ground truth

(b) CD-loss: 0.002389 after 5000 epochs of training

(c) CD-loss: 0.002271 after 25000 epochs of training

Figure 6.1.: Comparison between the PointNet autoencoder during training (b) and fully trained (c)

6.1.2. Autoencoder results

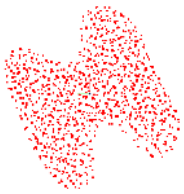
The results of thoroughly training each autoencoder architecture are shown in Table 6.1. In terms of loss, the PointNet autoencoder performs the best. The second best model is the PointNet++ autoencoder, and the worst performance in terms of loss is the Point Transformer autoencoder. However, the variation in loss values across all models is minimal, with only marginal differences observed between the different architectures.

Model	Epoch	Batch size	Test loss
PointNet	25000	1024	0.0022714
PointNet++	5700	128	0.0022724
Point Transformer	4400	128	0.0023225

Table 6.1.: This table shows the number of epochs to thoroughly train each autoencoder architecture on the volumetric point cloud dataset and the achieved test loss.

Table 6.2 shows a side-by-side comparison of a collection of reconstructed point clouds of all three autoencoder architectures. All three models demonstrate high reconstruction quality for the volumetric point clouds. However, despite exhibiting the lowest loss, the

PointNet reconstruction performs marginally inferior to PointNet++ and Point Transformer. The reconstructed point clouds show a loss of detail, particularly for concave shapes. PointNet++ shows the best reconstruction quality.

Ground truth	**PointNet**	**PointNet++**	**Point Transformer**
			
			
			
			

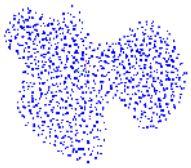
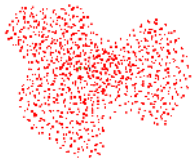
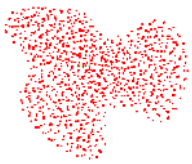
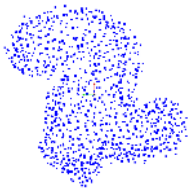
Ground truth	PointNet	PointNet++	Point Transformer
			
			
			

Table 6.2.: Comparison of the reconstruction quality of autoencoders using PointNet, PointNet++, and Point Transformer architectures on the volumetric point cloud dataset (on the test set).

Table 6.3 illustrates the reconstruction quality of the two best-performing models (PointNet++ and Point Transformer) on the point cloud shell dataset. Edges are better visible for the point cloud shells. This property makes a qualitative assessment of the reconstruction quality more straightforward. PointNet++ and Point Transformer exhibit high reconstruction quality for the point cloud shells. Although the difference is subtle, PointNet++ marginally outperforms Point Transformer regarding reconstruction quality.

Ground truth	PointNet++	Point Transformer
		
		
		
		
		

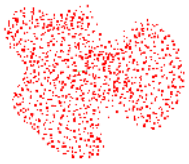
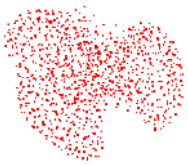
Ground truth	PointNet++	Point Transformer
		
		

Table 6.3.: Comparison of the reconstruction quality between PointNet++ and Point Transformer on the point cloud shell dataset (on the test set).

Regarding the CD loss assessment, PointNet++ marginally outperforms Point Transformer, similar to the performance on the volumetric dataset. However, the number of training steps required to train both models fully increases for the point cloud shell dataset (see Table 6.4).

Model	Epoch	Batch size	Test loss
PointNet++	6200	128	0.0022063
Point Transformer	5800	128	0.0023537

Table 6.4.: This table shows the number of epochs to fully train the PointNet++ and Point Transformer autoencoder models and the achieved test loss on the point cloud shell dataset.

Table 6.5 shows a side-by-side comparison of the in terms of reconstruction quality best-performing model, PointNet++, between the volumetric point cloud dataset and the

point cloud shell dataset. The qualitative assessment of the point cloud shells is more straightforward, and these representations contain more detail. However, this perception is primarily due to the limited viewing angle in the 2D images. To fully comprehend the shape of the volumetric point clouds, rotation in 3D space is necessary. After extensive evaluation of the point clouds using a 3D image viewer, it becomes apparent that PointNet++ performs marginally better on the point cloud shells than Point Transformer.

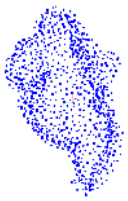
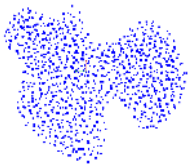
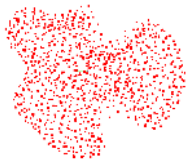
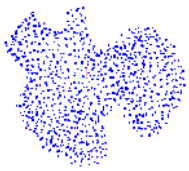
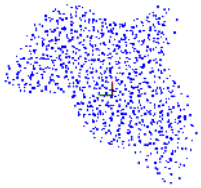
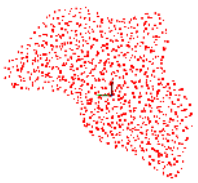
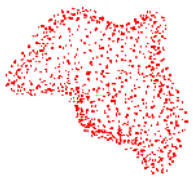
Ground truth (volumetric)	PointNet++ (volumetric)	PointNet++ (shell)	Ground truth (shell)
			
			
			
			

Table 6.5.: Comparison of the reconstruction quality between the point cloud shell dataset and the volumetric dataset with PointNet++. This table shows point clouds that are better represented as point cloud shells than volumetric point clouds (on the test set).

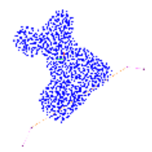
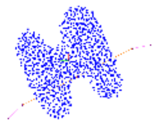
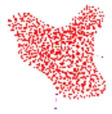
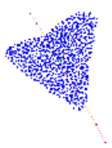
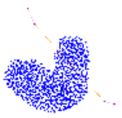
6.2. Dynamics models

6.2.1. Grid search

Initializing the dynamics model with an untrained autoencoder (training strategy **A**) overall results in significantly higher test loss and poor reconstruction and prediction quality of the dynamics model for all three autoencoder architectures. Integrating a pre-trained autoencoder in the dynamics model significantly improves the test loss as well as the reconstruction quality, however (training strategies **B** and **C**. Fine-tuning the autoencoder (strategy **C**) initially improves performance but leads to overfitting after a certain point in the training process for dynamics models with integrated PointNet++ and Point Transformer autoencoders. In contrast, freezing the autoencoder's encoder and decoder layers during the dynamics model training (strategy **B**) results in slower learning but eventually yields better test loss. The PointNet dynamics model does not seem to overfit with strategy **C** and overall performs best in terms of loss and point cloud quality with this strategy.

6.2.2. Dynamics model results

Table 6.6 demonstrates the predictive ability of the dynamics model with integrated pre-trained PointNet, PointNet++, and Point Transformer models using MLE loss on the volumetric point cloud dataset. Overall, all models often fail to predict the next dough state. The models are unable to predict the impact of the grasp action, and the outputted shapes often exhibit a significant loss of detail. Qualitative assessment indicates that PointNet++ performs the best among the other tested models. Despite the Point Transformer autoencoder's high reconstruction quality, this architecture performs markedly worse than the other two models in predicting dough state transitions.

Initial state	Ground truth next state	PointNet	PointNet++	Point Transformer
				
				
				
				
				
				

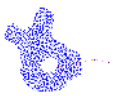
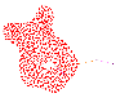
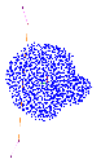
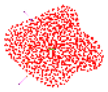
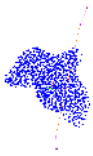
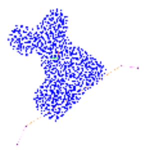
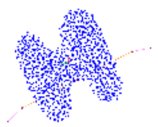
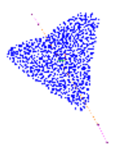
Initial state	Ground truth next state	PointNet	PointNet++	Point Transformer
				
				
				

Table 6.6.: Comparison of the prediction quality between PointNet, PointNet++ and Point Transformer dynamics model on the volumetric point cloud dataset trained with MLE loss (on the test set).

In contrast to the MLE-trained dynamics models, training with CD loss significantly improves the model's ability to predict subsequent dough states given a grasp action. We find that lower test loss also means a better prediction of the model, similar to the finding of better reconstruction quality for lower loss demonstrated in Figure 6.1. Table 6.7 shows all three dynamics model variants trained with CD loss on the volumetric point cloud dataset. Noticeably, each model can more accurately reconstruct the original dough shape when trained with CD loss. While the models still occasionally struggle to predict minor impacts of grasp actions accurately, they more frequently capture the overall changes in dough shapes. The models demonstrate better performance in handling grasp actions with subtle effects on the dough's shape than actions that substantially alter its global structure. All models demonstrate similar predictive abilities. However, the dynamics model with integrated PointNet autoencoder performs notably worse than the other two

models. The predicted point clouds lack detail and sharpness, similar to the results of the reconstructed point clouds of the PointNet autoencoder.

Initial state	**Ground truth next state**	**PointNet**	**PointNet++**	**Point Transformer**
				
				
				
				
				

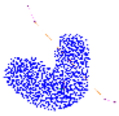
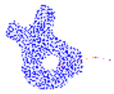
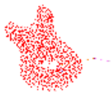
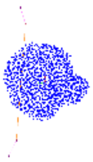
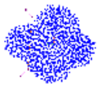
Initial state	Ground truth next state	PointNet	PointNet++	Point Transformer
				
				
				
				

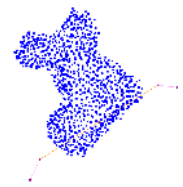
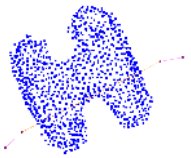
Table 6.7.: Comparison of the prediction quality between PointNet, PointNet++, and Point Transformer dynamics model on the volumetric point cloud dataset trained with CD loss (on the test set).

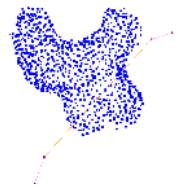
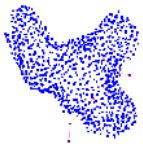
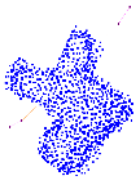
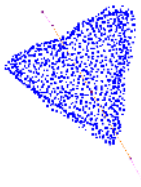
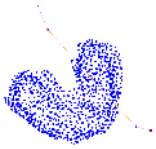
Table 6.8 compares three dynamic model architectures regarding their test loss. The PointNet++ dynamics model marginally outperforms the other two models.

Model	Epoch	Batch size	Loss
PointNet	11700	1024	0.0027121
PointNet++	11500	128	0.0026270
Point Transformer	5400	128	0.0026466

Table 6.8.: Training epochs and test loss for dynamics models on volumetric point cloud dataset.

For a better qualitative assessment, we look closer at the PointNet++ and Point Transformer dynamics models performances on the point cloud shell dataset (illustrated in Table 6.9). While subtle variances in the predicted dough states exist, neither model has a clear performance advantage. However, overall, the performance of both models seems to slightly increase on the point cloud shell dataset compared to the volumetric point cloud dataset.

Initial state	Ground truth next state	PointNet++	Point Transformer
			
			

Initial state	Ground truth next state	PointNet++	Point Transformer
			
			
			
			

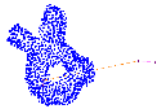
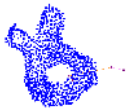
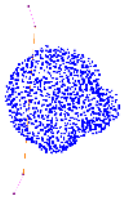
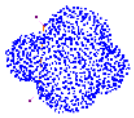
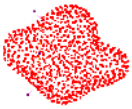
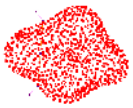
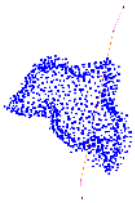
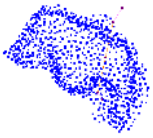
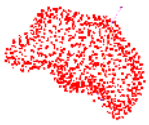
Initial state	Ground truth next state	PointNet++	Point Transformer
			
			
			

Table 6.9.: Comparison of the prediction quality between PointNet++ and Point Transformer dynamics model on the point cloud shell dataset trained with CD loss (on the test set).

6.2.3. Long-horizon prediction

The qualitative assessment of the dynamics model is complemented by a comparison of their predictive abilities in long-horizon prediction tasks. Table 6.10, Table 6.11, and Table 6.12 present a comparison of the two best-performing models, PointNet++ and Point Transformer, based on three recorded grasp trajectories from the provided dataset. This analysis involves evaluating the model outputs across multiple grasp actions. Table 6.10

illustrates long-horizon predictions over five grasp actions applied to an H-shape. Table 6.11 showcases long-horizon predictions over five grasp actions on a more complex fish-shape. Finally, Table 6.12 showcases long-horizon predictions over five grasp actions on a C-shape. In all three scenarios, both models perform similarly well. It is apparent that both models struggle with the initial state predictions of the C-shape but perform accurately on the fish-shape and the H-shape. Similar to the single prediction results, subtle variances in the predicted subsequent dough states exist, but neither model has a clear performance advantage. Both models can make accurate predictions of up to three subsequent grasp actions. However, after each prediction, the dough's resulting shape loses a little detail. This effect influences the next state prediction and, over the course of multiple grasp actions, leads to a loss of accuracy. The models are still able to predict the subsequent shape after five grasp inputs with a decent resemblance to the ground truth point cloud states.
It is important to note that the initial starting shapes of this experiment could have been included in the training set for the dynamics models. The models potentially have already been trained with the first grasp action.

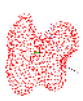
Dough states	P_0	P_1	P_2	P_3	P_4	P_5
Ground truth						
Point-Net++						
Point-Trans-former						

Table 6.10.: Long-horizon prediction comparison between PointNet++ and Point Transformer with a H-shape (first manipulation sample was part of train set).

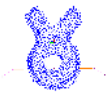
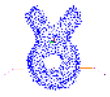
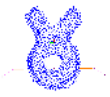
Dough states	P_0	P_1	P_2	P_3	P_4	P_5
Ground truth						
Point-Net++						
Point-Trans-former						

Table 6.11.: Long-horizon prediction comparison between PointNet++ and Point Transformer with a fish-shape (first manipulation sample was part of train set).

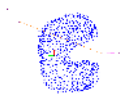
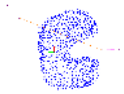
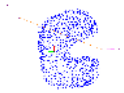
Dough states	P_0	P_1	P_2	P_3	P_4
Ground truth					
Point-Net++					
Point-Trans-former					

Table 6.12.: Long-horizon prediction comparison between PointNet++ and Point Transformer with a C-shape (first manipulation sample was part of train set).

To further assess the model's long-horizon predictive abilities, we conduct a long-horizon prediction sequence with the same starting shapes, but we apply random grasp actions to these shapes. This means the models have never seen the effect of the given grasp action on the initial dough shape. Because these manipulation scenarios have never existed, there is no ground truth dough state to compare the model predictions against. We argue this is still a valid approach to showcase the performance of our dynamics model because the grasp actions are fairly simple, and a human examiner can assess whether the subsequent state predictions are meaningful to some extent. Table 6.13, Table 6.14 and Table 6.15 showcase the long-horizon predictions over five consecutive grasps on the previously presented shapes (H-shape, fish-shape, C-shape). While a fair direct comparison of the two models without ground truth states is not possible, it is still apparent that both models can make meaningful predictions about the dough while preserving the most important features of the initial shapes. After five subsequent state predictions for the H-shape and the fish-shape, the final shapes have a close resemblance between both models. However, for the C-shape, the predictions of Point Transformer and PointNet++ begin to differ after the first two state predictions, resulting in slightly different shapes in the end. It is important to note that while both models fail to make accurate predictions for the first

grasp action, the predictions for all other consecutive shapes are still meaningful.

Dough states	P_0	P_1	P_2	P_3	P_4	P_5
Point-Net++						
Point-Trans-former						

Table 6.13.: Long-horizon prediction comparison between PointNet++ and Point Transformer with a H-shape (prediction with never-before-seen action states on the initial shape).

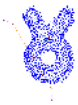
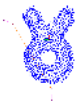
Dough states	P_0	P_1	P_2	P_3	P_4	P_5
Point-Net++						
Point-Trans-former						

Table 6.14.: Long-horizon prediction comparison between PointNet++ and Point Transformer with a fish-shape (prediction with never-before-seen action states on the initial shape).

Dough states	P_0	P_1	P_2	P_3	P_4	P_5
Point-Net++						
Point-Transformer						

Table 6.15.: Long-horizon prediction comparison between PointNet++ and Point Transformer with a C-shape (prediction with never-before-seen action states on the initial shape).

7. Conclusion

Our work presents a pipeline to represent different dough shapes as quality 3D point clouds based solely on visual sensory data. We compare a volumetric point cloud representation with the conventional point cloud shell representation. Using this pipeline, we record how the grasping actions of a 2-finger parallel gripper deform the dough shapes and create a dataset with the collected data. To demonstrate the quality of the provided dataset, we employ three different autoencoder architectures (PointNet 3.2.1, PointNet++ 3.2.2, Point Transformer 3.2.3) to learn 3D representations of the dough shapes. All three models can accurately extract essential point cloud features and learn complex dough-shape representations. A lower CD loss generally correlates with better point cloud reconstruction during model training, so the best training strategy is to train the models until the loss no longer significantly decreases. However, it's not always true that a model with lower loss than another model has better point cloud reconstruction quality. For example, while the PointNet autoencoder achieved the best loss by a small margin, the qualitative assessment shows that both PointNet++ and Point Transformer can reconstruct point clouds in more detail, especially in concave areas. PointNet requires substantially more data to perform on par with PointNet++ and Point Transformer, as demonstrated in Table 6.1. In our qualitative assessment, PointNet++ slightly outperforms Point Transformer in terms of point cloud reconstruction quality. When comparing the learned representations of the volumetric dataset to the point cloud shell dataset, we found that the volumetric approach doesn't perform better. It might even perform slightly worse. Overall, a point cloud shell representation seems visually better suited to represent the dough shapes because it's easier to see details in the dough surface due to the edges.

Next, we demonstrate how our volumetric point cloud dataset can be used to train a dynamics model that predicts subsequent dough states given the initial state and a grasping action. The dynamics model makes these predictions based on learned 3D point cloud representations from the three employed networks. We found that training the dynamics model with CD loss leads to much more accurate predictions than using MLE loss across all models. The qualitative assessment shows that all three variants of the dynamics

model can reasonably predict the dough's next state. However, all models often struggle to capture the full impact of a single grasping action, especially for more complex shapes. Predicted dough states can lose some of the shape's original details. The results indicate that better feature representation translates to more accurate predictions of the next dough state. The dynamics model using PointNet performs notably worse in terms of point cloud quality than those using PointNet++ and Point Transformer. When comparing PointNet++ and Point Transformer dynamics models, they produce visually slightly different predictions, but neither consistently outperforms the other. PointNet++ and Point Transformer perform similarly well on the point cloud shell dataset. The point cloud shells make it easier to see finer details, making the predictions on this dataset seem more meaningful and accurate.

Regarding long-horizon prediction over multiple grasping actions, similar to the single grasp prediction results, PointNet++ and Point Transformer produce visually slightly different predictions, but neither consistently outperforms the other. Both models can make accurate predictions for up to three consecutive grasp actions. The state predictions after five consecutive grasp actions are still reasonable and closely resemble the ground truth state. However, the original shapes begin to lose more detail. A potential reason for both models performing worse on the C-shape (see Table 6.12) could be a flawed latent representation of the initial shape. In Table 6.3, we can see that Point Transformer and PointNet++ autoencoders cannot perfectly reconstruct the C-shape. To some extent, the reconstructed point cloud loses a little detail compared to the ground truth C-shape. In comparison, the H-shape and the fish-shape are reconstructed more accurately. The slightly less accurate latent representation of the C-shape could lead to the worse performance of both models on the C-shape compared to the H-shape and the fish-shape.

Our experiments demonstrate that the provided dataset can be effectively used to learn 3D representations of complex dough shapes and train dynamics models to make reasonable predictions about the dough's next state. However, a volumetric point cloud representation might not be advantageous. It could even slightly negatively impact the prediction quality of the dynamics models compared to a point cloud shell representation with the same number of points for the representation. All three models we presented, PointNet, PointNet++, and Point Transformer, perform similarly well on the dataset. However, a dynamics model using PointNet++ or Point Transformer point cloud representations is the most promising. These two models can even make meaningful long-horizon predictions for a dough's state to some extent.

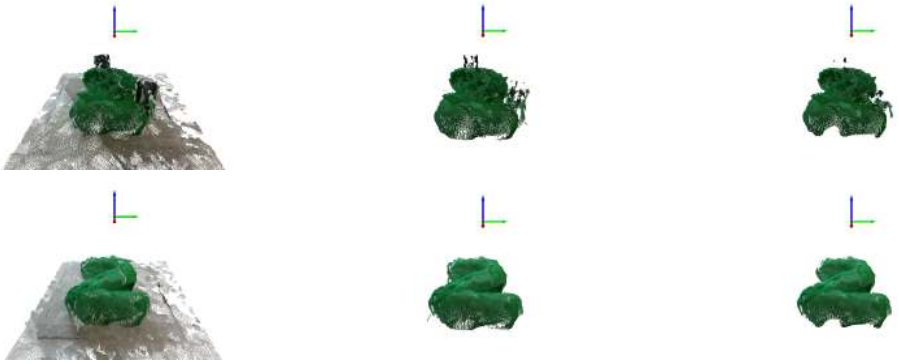
7.1. Discussion & Limitations

7.1.1. Camera noise

One of the critical limitations of our work is the quality of the depth images perceived by the cameras. Except for the usual noise of the cameras, we experienced an issue with the cameras having problems assigning the correct depth value around edges if there is an object (the robot finger) in front of another object (the green dough). What frequently happens is that some pixels that belong to the dough are assigned the same depth values as the values of the robot finger closer to the camera. Not only does this introduce a lot of noise and inaccuracies in the recorded data, but it also complicates filtering the point clouds in the pre-processing pipeline. Our primary filtering method is applying the HSV color filter to the green color. However, with the noisy depth images, significant parts of the robot fingers now contain green points at their edges. Our applied filters are not able to remove these outlier points altogether. We can choose to leave them in the dataset or apply more aggressive outlier removal filters. However, this method would also remove many details about the dough point cloud object, especially in parts with sparse point density. Figures 7.1 and 7.2 illustrate this noise issue. We compare a frame where the gripper fingers obstruct parts of the dough (top row) versus a frame where the robot gripper hovers over the dough so as not to cause any obstruction with the fingers (bottom row).

(a) **Original point cloud** (b) **HSV color filtering** (c) **Outlier removal**

Figure 7.1.: Top-down perspective: Comparison of point clouds with gripper finger obstruction (top row) and without obstruction (bottom row)



(a) **Original point cloud** (b) **HSV color filtering** (c) **Outlier removal**

Figure 7.2.: Side-angle perspective: Comparison of point clouds with gripper finger obstruction (top row) and without obstruction (bottom row)

Our initial plan was to learn a dynamics model that can predict the deformation of dough frame by frame. However, due to depth camera noise we experienced if the robot fingers are touching the dough during manipulation we decided to only take the frames before and after each manipulation and we decided to completely remove the gripper out of the view of the cameras so that there is no obstruction caused by the gripper at all (see Figures 7.1 and 7.2). However this leads to another issue that is that during a grasping action of the robot if the robot fingers are fully closed and are about to open again it can occur that the fingers are tearing and scretching the dough when pulled out again due to it's stickiness.
A possible solution to the noisy depth image quality could be to remove the points of the fingers with a different method. By using forward kinematics and the known object mesh of the 3D printed fingers we could specifically remove the points of the robot fingers in the point cloud. This would be similar to RoboCraft's approach by refining the point cloud with physics and shape prior using robot fingers' SDF [1].

7.1.2. Malfunctioning camera

Besides the cameras' general noise, we also experienced an exceptionally bad quality of one of the static cameras. We suspect a malfunction or wrong intrinsical calibration of the camera (maybe caused by a hard blow to the camera before the recording) because

the depth image quality was notably worse than the depth image quality of the other static cameras. The malfunctioning camera's issue impacts the quality of the recorded depth images and the quality of its calibration results, making it significantly worse than the calibrations of the other cameras we use. Its recorded depth images are still usable, especially the ICP fine-tuning improves the final merged point cloud quality.

7.1.3. Inaccurate robot state

Additionally, we experienced inaccuracies in determining the exact robot state that affected the whole data collection process (including the camera calibration). The inaccuracies are caused by the fact that the internal URDF model of our Franka Emika Panda robot slightly differed from its physical counterpart. If the robot is moved by hand, its cartesian position is calculated using the joint states of the robot and the slightly differing URDF model. This difference causes errors in calculating the exact end effector position in the range of a few millimeters. This issue inevitably affects the camera calibration with easy-hand-eye and the exact location of the grasping actions in our dataset.
While there was no available solution to this issue (at the time of the data collection), there could have been other possible ways to improve the quality of the camera calibration by choosing the ArUco marker size. Our initial idea was to use big ArUco markers ($13\text{cm}\times13\text{cm}$) to make them more easily identifiable by our cameras. The bigger size limited the variety of positions and rotations required for accurate calibration results by easy-handeye. We recommend using smaller ArUco markers (the cameras should still be able to identify the markers correctly) to enable a greater number of different poses during calibration.
The inaccuracies in the camera calibration lead to a slight but notable offset of every partial point cloud, which causes multiple object surfaces in the 3D point clouds. To remove this noise in the dough surface, we introduce Poisson surface reconstruction [37] and alpha surface reconstruction [38]. We think they have a positive impact (mainly Poisson surface reconstruction) on the quality of the point clouds. However, it is still worth mentioning that these algorithms are only approximations of the dough's actual surface, and both algorithms always tend to over-smooth fine-grained details of especially concave shapes as showcased in Figure 4.4. Additionally, the surface reconstruction process is time-consuming because the reconstructed mesh of every point cloud has to be manually supervised by a human due to the algorithms often failing to create a watertight mesh.

7.1.4. Diverse grasping actions

Another limitation of our work is the 3D-printed fingers we use. While they can perform simple grasping actions similar to SculptBot, we found that they lead to predictable and monotonous manipulation actions. When fully closed, there is still a relatively large gap between the two fingers (see Figure 5.1. One of the key advantages of operating the gripper by hand and not by automated robot movement is that it enables the human operator to choose specific grasping actions to deform the dough into a target shape. For example, the initial shape of the dough can be a disc, and the task is to deform it into a square. However, during the data collection process, creating such target shapes is often impossible due to the limitations of the 2-finger parallel gripper. Additionally, sometimes, the grasping actions to reach such target shapes have a minimal effect on the resulting shape. For our work, we want to avoid such fine-grained manipulation actions because the current data collection setup cannot capture these kinds of small changes in the shape of the dough due to the inaccuracies of the camera calibration and the noise in the depth cameras. Instead, we aim for grasp actions that have a notable and diverse effect on the shape of the dough. However, this means scraping the idea of trying to create target shapes that would often require only fine-grained grasping actions.
The dough mount itself is another limitation in collecting as many diverse manipulation actions as possible. While the mount ensures that the dough stays in place during manipulation, it also leads to the issue that the fingers cannot push into the location of the dough where it is mounted on the small plastic rod. During the grasping actions, we have to avoid pushing the fingers into the rod of the dough mount, further limiting the number of different grasping actions.
The experiments initially include manipulation actions where the gripper fingers push down onto the dough from above. This approach creates holes in the surface of the dough, resulting in more diverse manipulation actions in the dataset. But these actions are removed for the final dataset because even though the dough is mounted on the plastic mount, it often happens that the dough sticks to the robot fingers when it is moved up after a manipulation (see Figure 7.3).

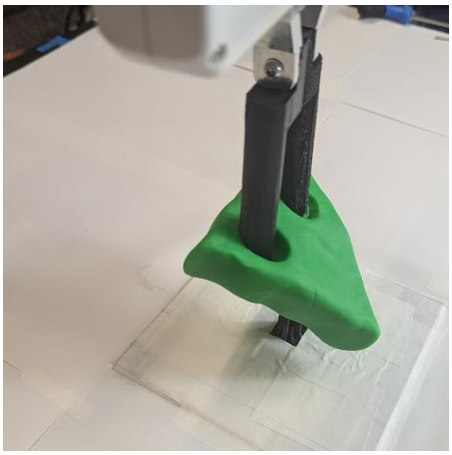


Figure 7.3.: Sticking dough during push-down manipulation. When moving the gripper up the dough sticks to the fingers and leaves its original position making state estimation impossible.

7.1.5. Gripper fingers

During data collection, we found that the fingers are too flexible, resulting in the fingers bending under stress if they pinch a lot of dough mass. Our forward kinematics (using the robot's SDF and robot state) assume the robot fingers to be rigid and do not account for any finger flexibility during manipulation. While the deformation of the fingers under stress is minimal, we think it is still notable and can potentially affect the exact location of the fingertips and the distance between them.
The high flexibility of the fingers can also cause them to break during manipulation, which happened two times during the data recording process. This accident led to losing time recording more dough manipulation samples because we had to wait until backup fingers were reprinted. Additionally, the new fingers were printed out of black plastic. Initially, we planned on using blue fingers. Initially, we planned to use blue fingers. However, since our work does not involve frames where the robot fingers obstruct the view of the dough, the color of the fingers is not significant. However, for future work that aims to capture every frame of the dough manipulation, we recommend not using a black color for the fingers. The reasoning behind this is similar to our choice of color for the manipulation platform, which we explain in Section 4.1. Black is too similar to some of the color shades created by shadows from the gripper or human operator during the grasping action.

7.1.6. Dataset size

Learning representations on 3D data requires a lot of data [3]. Data augmentation helps increase the size of the provided dataset. Still, only having 438 unique grasping action samples might not be enough to accurately learn the dynamics of a complex soft object like dough. In our work, we focus on collecting as many diverse shapes and grasping actions as possible by guiding the robot arm by hand. This type of data collection is tedious and time-consuming, leading to a smaller dataset than using automated robot movements for data collection, but we think it is necessary. Besides reprinting the fingers while recording the data, another bottleneck of the data collection is the limited disc size of the recording PCs. We have to transfer the recorded data from the PCs onto an external hard drive at regular intervals. Due to the size of the recordings (one minute of recording equals 40GB to 45GB), this takes a substantial amount of time, during which we have to interrupt the data collection process until the data transfer is finished to clear the disc space on the PCs.

7.1.7. Gridsearch

Due to time constraints, the training runs for the grid search are kept shorter. However, many hyperparameters often reveal their impact on the model performance late in the training. This makes it challenging to choose the best hyperparameter configuration for the models accurately. Additionally, different configurations of the models, like the number of layers and layer size, might not be sufficiently explored.

7.1.8. Point cloud representation

The finding that the point cloud shell dataset leads to slightly better results (even though the point clouds contain more noise and have no lower surface) might not necessarily mean that it is a representation form unsuited for the presented deep learning networks. We suspect the reason to be caused by the utilized Chamfer distance metric. Volumetrizing the point clouds but keeping the same point size of 1024 points leads to a lower surface point resolution than the point cloud shells. By lower resolution, we mean the average distance between the nearest points is lower. The Chamfer distance is sensitive to discretization and is naturally better at capturing details if the resolution is higher. This means that comparing the model performances on the volumetric and the point cloud shell dataset with the same point size seems unfavoured for the volumetric point cloud representation. It is worth noting that for most next-state prediction tasks, representing shapes as point

cloud shells, focusing only on their outer surfaces, is usually sufficient. Points inside the dough may not be necessary, and it would be more beneficial to concentrate on a high-resolution outer surface rather than sampling the interior volume of the point cloud.

7.1.9. Encoder bottleneck

A possible explanation for why all three autoencoder architectures we tested had similar reconstruction results could be the decoder network we used. Since it's a simple MLP, it might not be able to accurately reconstruct the point cloud, even if the encoded latent representation is good. This means the decoder could be limiting the overall reconstruction ability. The simpler decoder network might be holding back the potentially better encodings from the PointNet++ and Point Transformer models.

7.2. Outlook

Our research lays the groundwork for developing an accurate dynamics model that can capture the complex behavior of soft objects like dough. Similar to projects like Sculptbot or RoboCraft, our ultimate goal is to integrate this dynamics model into a planning framework, enabling a robot to create complex, never-before-seen target shapes out of dough. We provide a pipeline to create a quality dataset of dough manipulation based on raw RGB-D data. Our results suggest room for improvement in the reconstruction quality and prediction capabilities of the networks we use. Repeating the data collection process with our new insights could significantly improve the overall point cloud quality, especially since one camera we used in this research had some issues. To further assess the quality of the provided dataset, we suggest using a pre-trained Point-BERT model similar to SculptBot's work and comparing its reconstruction ability on our point cloud dataset.
We recommend using higher-resolution point clouds, particularly for the volumetric point cloud representation. To support future research, we provide the downsampled volumetric point cloud dataset and the dense version from an earlier data processing step 4.1.4 that can be easily downsampled to any size. Another approach for the point cloud shell dataset could be to sample the lower surface of the dough and see the potential impact of a fully enclosed point cloud shell. Furthermore, it might be worth exploring the impact of different loss metrics for model training. While computationally expensive, the Earth Mover's Distance or variations [39] could improve model training.

Instead of only training the dynamics model to predict subsequent dough states given the initial dough state and a grasp action as input, it might also be interesting to give the model both the initial state and the next state as input and let it learn to predict the grasp action necessary to get to the next dough state. This ability is essential to fully integrating the dynamics model into a planning framework.
One limitation of our work is that the dataset only captures the dough states before and after manipulation. For future data collection experiments, it might be worth focusing on a method to accurately extract the frame-by-frame states of the dough manipulation. A model that can predict changes in the dough's shape in every frame could lead to a more robust planning framework that can accurately change its policy even during a manipulation action. When manipulating the dough in the real world and encountering a change in the dough's shape that significantly differs from the predicted shape, the model could update its behavior during the grasp action and not only until after the complete grasp action has finished. Our dynamics models using PointNet++ and Point Transformer can perform decent long-horizon predictions of subsequent dough states for five subsequent grasp actions. However, there is still a lot of room for improvement in precision. Future research might benefit from exploring more sophisticated dynamics models, similar to SculptBot, instead of using a simple MLP for predictions in the latent space.
Another direction for future work could be exploring learning alternative 3D data representations instead of latent vector representations and comparing their effects on learned dynamics models. For example, researchers created a model, DeepSDF [41], that learns a continuous signed distance function to represent a shape's surface as a continuous volumetric field.

Bibliography

[1] H. Shi, H. Xu, Z. Huang, Y. Li, and J. Wu, "Robocraft: Learning to see, simulate, and shape elasto-plastic objects with graph networks," 2022.

[2] H. Shi, H. Xu, S. Clarke, Y. Li, and J. Wu, "Robocook: Long-horizon elasto-plastic object manipulation with diverse tools," *arXiv preprint arXiv:2306.14447*, 2023.

[3] A. Bartsch, C. Avra, and A. B. Farimani, "Sculptbot: Pre-trained models for 3d deformable object manipulation," 2023.

[4] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," 2019.

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016.

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[7] H. Zhao, L. Jiang, J. Jia, P. H. S. Torr, and V. Koltun, "Point transformer," *CoRR*, vol. abs/2012.09164, 2020.

[8] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," 2018.

[9] J. Sanchez, J. A. Corrales Ramon, B. C. BOUZGARROU, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, pp. 688 – 716, 06 2018.

[10] C. Matl and R. Bajcsy, "Deformable elasto-plastic object shaping using an elastic hand and model-based reinforcement learning," 2021.

[11] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, July 2014.

[12] J. Truong, S. Chernova, and D. Batra, "Bi-directional domain adaptation for sim2real transfer of embodied navigation agents," *IEEE Robotics and Automation Letters*, vol. 6, p. 2634–2641, Apr. 2021.

[13] Y. Chen, X. Li, S. Guo, X. Y. Ng, and M. Ang, "Real2sim or sim2real: Robotics visual insertion using deep reinforcement learning and real2sim policy adaptation," 2022.

[14] P. Huang, X. Zhang, Z. Cao, S. Liu, M. Xu, W. Ding, J. Francis, B. Chen, and D. Zhao, "What went wrong? closing the sim-to-real gap via differentiable causal discovery," 2023.

[15] M. Torne, A. Simeonov, Z. Li, A. Chan, T. Chen, A. Gupta, and P. Agrawal, "Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation," 2024.

[16] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, "Point-bert: Pre-training 3d point cloud transformers with masked point modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19313–19322, June 2022.

[17] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," *CoRR*, vol. abs/1512.03012, 2015.

[18] C. Cao, M. Preda, and T. Zaharia, "3d point cloud compression: A survey," in *Proceedings of the 24th International Conference on 3D Web Technology*, Web3D '19, (New York, NY, USA), p. 1–9, Association for Computing Machinery, 2019.

[19] M. Bassier, M. Vergauwen, and F. Poux, "Point cloud vs. mesh features for building interior classification," *Remote Sensing*, vol. 12, no. 14, 2020.

[20] I. Ulku and E. Akagündüz, "A survey on deep learning-based architectures for semantic segmentation on 2d images," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2032924, 2022.

[21] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[22] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[23] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, 2015.

[24] F. Engelmann, T. Kontogianni, J. Schult, and B. Leibe, "Know what your neighbors do: 3d semantic segmentation of point clouds," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[25] V. Tadic, "Intel realsense d400 series product family datasheet," 2019.

[26] M. Esposito, R. O'Flaherty, Y. Li, S. Virga, R. Haschke, Y. Urakami, R. Joshi, B. A. Cakal, and H. Xiao, "easy_handeye: automated, hardware-independent hand-eye calibration for ros1," 2021.

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

[28] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "Rviz: a toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, pp. 337–345, 10 2015.

[29] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures* (P. S. Schenker, ed.), vol. 1611, pp. 586 – 606, International Society for Optics and Photonics, SPIE, 1992.

[30] A. Grunnet-Jepsen, P. Winer, A. Takagi, J. Sweetser, K. Zhao, T. Khuong, D. Nie, and J. Woodfill, "Using the intel® realsensetm depth cameras d4xx in multi-camera configurations," *New Technologies Group, Intel Corporation, Rev*, vol. 1, 2018.

[31] P. Yao, "The advantages of raw format image post-processing," in *Communications and Information Processing* (M. Zhao and J. Sha, eds.), (Berlin, Heidelberg), pp. 625–631, Springer Berlin Heidelberg, 2012.

[32] IntelRealSense, "ROS Wrapper for Intel ®RealSense™Devices," 2024.

[33] S. Sural, G. Qian, and S. Pramanik, "Segmentation and histogram generation using the hsv color space for image retrieval," in *Proceedings. International Conference on Image Processing*, vol. 2, pp. II–II, 2002.

[34] D. J. Bora, A. K. Gupta, and F. A. Khan, “Comparing the performance of l*a*b* and hsv color spaces with respect to color image segmentation,” 2015.

[35] Q.-Y. Zhou, J. Park, and V. Koltun, “Open3d: A modern library for 3d data processing,” 2018.

[36] J. Li, J. Zhou, Y. Xiong, X. Chen, and C. Chakrabarti, “An adjustable farthest point sampling method for approximately-sorted point cloud data,” in *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, 2022.

[37] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[38] H. Edelsbrunner and E. P. Mücke, “Three-dimensional alpha shapes,” *ACM Trans. Graph.*, vol. 13, p. 43–72, Jan. 1994.

[39] M. Liu, L. Sheng, S. Yang, J. Shao, and S.-M. Hu, “Morphing and sampling network for dense point cloud completion,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 11596–11603, Apr. 2020.

[40] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, “On the continuity of rotation representations in neural networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[41] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, “Deepsdf: Learning continuous signed distance functions for shape representation,” 2019.

[42] M. Görner, R. Haschke, H. Ritter, and J. Zhang, “Moveit! task constructor for task-level motion planning,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 190–196, 2019.

[43] M. Fey and J. E. Lenssen, “Fast Graph Representation Learning with PyTorch Geometric,” May 2019.

[44] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *CoRR*, vol. abs/1903.02428, 2019.

A. Appendix

A.1. Data recording setup

To record the data, we use two separate computers. The main machine runs on Ubuntu 20.04 with ROS1 and is responsible for launching Rviz to inspect the point clouds and for launching the panda moveit stack [42], necessary for recording the robot state. The second machine runs on Ubuntu 22.04, thus we use an Ubunutu 20.04 docker image with ROS1 installation). The robot and the second machine are connected via a network to the main machine. Before every *grasp trajectory*, the two machines are synchronized using the chrony tool. Every machine is connected to three cameras. For the robot state, we are subscribing to the $/tf$, $/tf\_static$ and $/joint\_state$ topics in ROS1. Figure A.1 shows the SL-block used for the ICP registration.

Figure A.1.: SL-block used for ICP registration.

A.2. Grid search

We do not implement learning rate reduction nor random input dropout in the training of the networks. All models were trained with a learning rate of 0.0001. We use the following notations to describe our architectures:

1. FC($input, output$): is a fully connected (linear) layer with input size $input$ and output size $output$
2. latent_size: is 256 for all our models
3. point_size: refers to the point cloud size for our models which is 1024
4. MLP($l_1, ..., l_d$): is a multi-layer perceptron with l_1 being the input layer size and l_d the output layer size. The layers are fully connected linear layers with ReLu functions as activation functions except for the output layer.
5. SAModule($ratio, r$, MLP($l_1, ..., l_d$)): is a set abstraction layer with $ratio$ being the downsampling ratio and r being the ball query radius.
6. GlobalSAModule(MLP): is the last set abstraction layer that performs additional feature transformation with a MLP and then applies global max pooling resulting in the latent vector.

The decoder for all employed autoencoder architectures is based on the same architecture:

$$
\begin{aligned}
\text{FC}(\text{latent\_size}, \text{latent\_size}) &\rightarrow \text{ReLU} \\
&\rightarrow \text{FC}(\text{latent\_size}, 512) \\
&\rightarrow \text{ReLU} \\
&\rightarrow \text{FC}(512, \text{point\_size}) \\
&\rightarrow \text{ReLU} \\
&\rightarrow \text{FC}(\text{point\_size}, \text{point\_size} \times 3) \\
&\rightarrow \text{Reshape}(3, \text{point\_size})
\end{aligned}
$$

A.2.1. Autoencoder grid search

A.2.2. PointNet

Our PointNet encoder is a modified version of [8].
Input is input point cloud of shape $(3, 1024)$
Encoder:

$$\begin{aligned} \text{Input} &\to \text{FC}(3, 64) \\ &\to \text{FC}(64, 128) \\ &\to \text{FC}(128, 256) \\ &\to \text{FC}(256, \text{latent\_size}) \end{aligned}$$

Figure A.2 compares the test losses for the PointNet autoencoder using latent sizes of $128, 256,$ and 512.

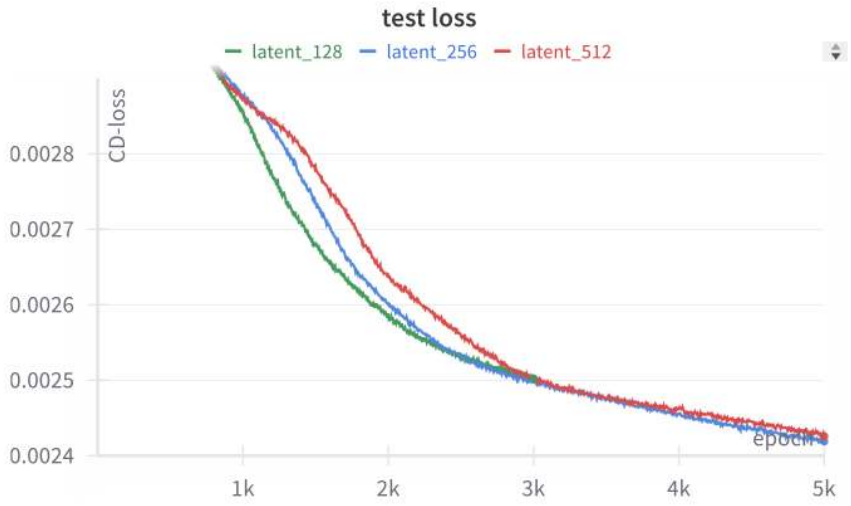


Figure A.2.: Optimal latent size search for the PointNet autoencoder. The run for latent size 128 ends early, however it is still apparent that the other two latent sizes perform better. Latent size 256 performs best.

PointNet++

Our PointNet++ encoder is a modified version of PyG's PointNet++ networks [43]. Input is is a pytorch geometric [44] graph object (pos, x). pos refers to the three coordinates of each point of the input point cloud and x to the point features. In our usecase we set $pos = x$ because our point clouds do not contain additional features.
Encoder:

$$\begin{aligned}\text{Input} &\rightarrow \text{SAModule}(0.5, 0.1, \text{MLP}(3+3, 32, 32, 64, 64)) \\ &\rightarrow \text{SAModule}(0.4, 0.2, \text{MLP}(64+3, 64, 64, 128, 128)) \\ &\rightarrow \text{SAModule}(0.1, 0.4, \text{MLP}(128+3, 128, 128, 256, 256)) \\ &\rightarrow \text{GlobalSAModule}(\text{MLP}(256+3, 256, 512, 512, \text{self.latent\_size}))\end{aligned}$$

Figure A.2 compares the test losses for the PointNet++ autoencoder using latent sizes of $128, 256$, and 512.

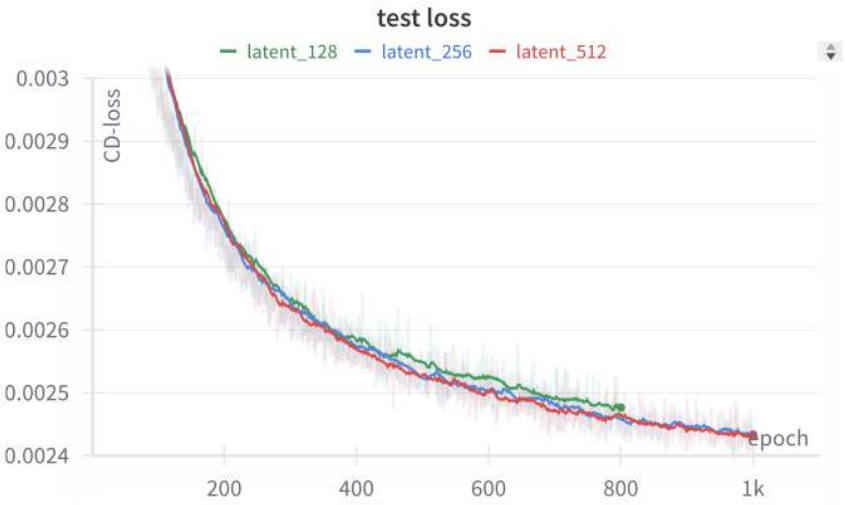


Figure A.3.: Optimal latent size search for the PointNet++ autoencoder. The run for latent size 128 ends early, however it is still apparent that the other two latent sizes perform better. The latent sizes of 256 and 512 yield comparable performance. For better comparability with the other two models we choose latent size 256.

Point Transformer

Our Point Transformer encoder is a modified version of PyG's Point Transformer networks [43]. Input is is a pytorch geometric [44] graph object (pos, x). pos refers to the three coordinates of each point of the input point cloud and x to the point features. In our usecase we set $pos = x$ because our point clouds do not contain additional features. We use the following notations to describe our architectures:

1. TransitionDown$(a, b, ratio, k)$: a describes the number of input channels (features), b describes the number of output channels. $ratio$ describes the point downsampling ratio and k is the number of k number of nearest neighbors to consider in this layer.
2. TransformerBlock(a, b): a describes the number of input channels (features), b describes the number of output channels.

Encoder:

$$
\begin{aligned}
\text{Input} &\rightarrow \text{MLP}(3, 32) \\
&\rightarrow \text{TransformerBlock}(32, 32) \\
&\rightarrow \text{TransitionDown}(32, 64, 0.25, 16) \\
&\rightarrow \text{TransformerBlock}(64, 64) \\
&\rightarrow \text{TransitionDown}(64, 128, 0.25, 16) \\
&\rightarrow \text{TransformerBlock}(128, 128) \\
&\rightarrow \text{TransitionDown}(128, 256, 0.25, 32) \\
&\rightarrow \text{TransformerBlock}(256, 256) \\
&\rightarrow \text{TransitionDown}(256, \text{latent\_size}, 0.25, 64) \\
&\rightarrow \text{GlobalMeanPooling}
\end{aligned}
$$

Figure A.2 compares the test losses for the Point Transformer autoencoder using latent sizes of $128, 256,$ and 512.

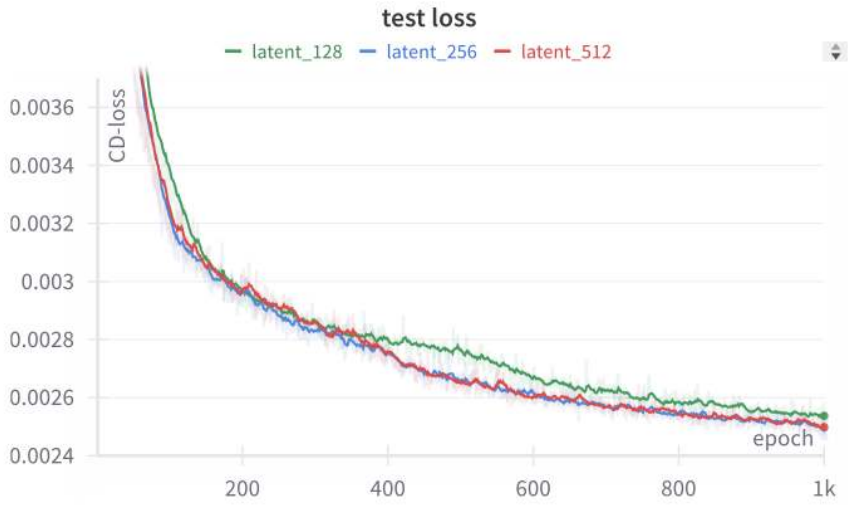


Figure A.4.: Optimal latent size search for the Point Transformer autoencoder. The run for latent size 128 ends early, however it is still apparent that the other two latent sizes perform better. The latent sizes of 256 and 512 yield comparable performance. For better comparability with the other two models we choose latent size 256.

A.2.3. Dynamics model grid search

The dynamics models only differ in the encoder and decoder layers. Action-MLP and Global-MLP are the same. The Action-MLP gets the robot state of shape $(1, 11)$ as input: $Input \rightarrow MLP(11, 32, 32)$
The Global-MLP gets a the latent representation of the input point cloud and the latent representation of the action input as input:

$$\begin{aligned} \text{Input} &\rightarrow \text{Global-MLP}(\text{latent\_size} + 32, 64, 128, 256) \\ &\rightarrow \text{decoder} \\ &\rightarrow \text{Output}(3, \text{point\_size}) \end{aligned}$$

Figure A.5, Figure A.6 and Figure A.7 show the test loss for the three employed training strategies for each of the presented network architectures:

A. Encoder/decoder layers and dynamics layers initialized from scratch.

B. Freeze encoder/decoder layers during dynamics model training.

C. Fine-tune encoder/decoder layers during dynamics model training.

PointNet

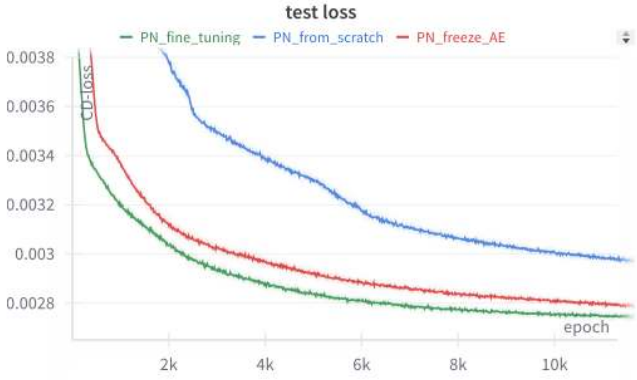


Figure A.5.: Comparison of training strategies **A**, **B** and **C**. Strategy **B** (fine-tuning encoder/decoder layers) yields the best test loss.

PointNet++

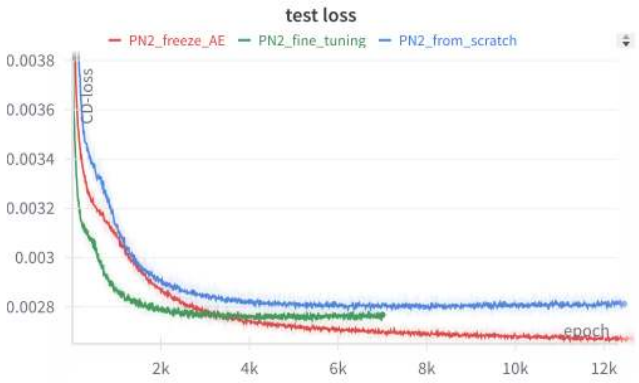


Figure A.6.: Comparison of training strategies **A**, **B** and **C**. Strategy **B** (fine-tuning encoder/decoder layers) leads to an increase of test loss after a short time and was stopped early. Strategy **C** (freezing encoder/decoder layers) yields the best test loss.

Point Transformer

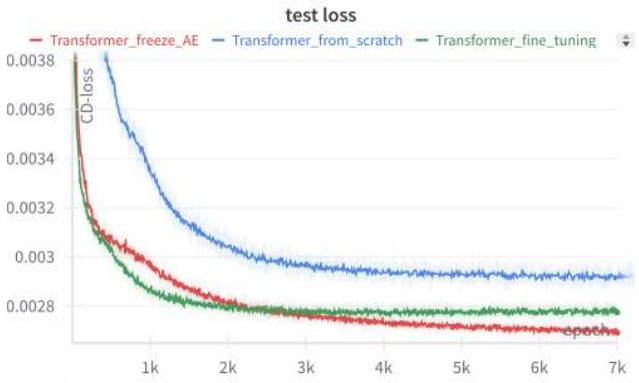


Figure A.7.: Comparison of training strategies **A**, **B** and **C**. Strategy **B** (fine-tuning encoder/decoder layers) leads to an increase of test loss. Strategy **C** (freezing encoder/decoder layers) yields the best test loss.