Learning Object Stress and Deformation With Graph Neural Networks

Lernen von Spannungs- und Verformungsfeldern von elastischen Objekten mit Graph-Neuronalen Netzwerken
Bachelor thesis in the field of study “Computational Engineering” by Frederik Heller
Date of submission: August 26, 2024

1. Review: Alap Kshirsagar, Ph.D.
2. Review: Tim Schneider, M.Sc.
3. Review: Guillaume Duret, M.Sc.
4. Review: Prof. Jan Peters, Ph.D.

Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Hiermit erkläre ich, Frederik Heller, 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.

Darmstadt, August 26, 2024

F. Heller

Abstract

Deformable objects are ubiquitous in our daily lives, and we humans handle and manipulate them instinctively. Robots lack this intuitive judgement, and their capabilities today are far behind human skill. Traditional approaches to robotic grasping predominantly focus on rigid objects or used questionable simplifications of object deformation dynamics. Recently, robotic simulators predict object deformation based on the realistic finite element method, and learning-based approaches can predict deformation and stress more accurately.

In this work, we re-implement the *DefGraspNets* model: It is a graph neural network capable of predicting object deformation and stress fields resulting in robotic grasp experiments. Our implementation allows training directly on ground truth data files output by a FEM-based simulator, which was not possible in the released baseline. Additionally, we propose two key modifications: The first allows inference on datapoints without needing to launch a simulation. The second extends the graph neural network architecture, informing about tetrahedral elements in the mesh. Stress values are then predicted at tetrahedra, which is a physically more sensible way. We provide a detailed evaluation of a multitude training runs conducted. Our training results suggest the capability of our implementation, but also limitations of the model. We release our codebase and generated dataset, and are confident that it will accelerate future research in this exciting field.

Acknowledgments

I would like to thank my supervisors Alap Kshirsagar, Tim Schneider and Guillaume Duret for their guidance and support during the creation of this thesis. I am especially thankful to Alap for welcoming me at the IAS groud, his mentorship and the close collaboration in our second project together, and for his helpful comments on the draft of this thesis. I thank Tim for help in shaping the high-level goal of this thesis, and his help in debugging the implemented network in a critical phase of the project. I am grateful to Guillaume for setting up the Docker containers and the help in understanding and reconstructing the pre-processing methods.

We express our gratitude to Dr. Isabella Huang for providing her dataset of tetrahedral meshes to be used as input for the *DefGraspSim* simulator.

Contents

Algorithms, Figures and Tables

List of Figures

List of Tables

Abbreviations and Symbols

Abbreviations

CNN Convolutional Neural Network
FEM Finite Element Method
GNN Graph Neural Network
MAE Mean Absolute Error
MLP Multi-Layer Perceptron
MSE Mean Squared Error
ODE Ordinary Differential Equation
PDE Partial Differential Equation
PINN Physics-informed Neural Network
RL Reinforcement Learning
w.r.t. With Respect To

Symbols

Symbol	Description	Pages
E	Young's modulus as stiffness measure of material.	6, 8, 14, 40, 45
$\mathbf{u}$	Deformation vector field of a continuous 3D object.	8–10
ε_C	Strain tensor describing relative deformation of material in a point.	8, 9
$\mathbf{D}$	Elasticity tensor relating strain to stress tensors in linear material.	8, 9
ν	Poisson's ratio of a material, relating lateral to axial strain.	8, 40
$\boldsymbol{\sigma}$	Stress tensor describing internal elastic forces of material in a point.	9, 11, 15
$\mathbf{C}$	Elasticity matrix relating strain to stress vectors using Voigt's notation.	9, 11

Symbol	Description	Pages
$\ddot{\mathbf{u}}$	Acceleration vector field of a continuous 3D object.	9, 11
ρ	Density of a material.	9, 11, 40
$\mathbf{b}$	Vector field of external forces acting on the continuous object.	9, 11
$\mathbf{u}_i$	Ground truth deformation of a node i.	10, 11
N_i	Shape function of a single point i, used to interpolate point values on a tetrahedron.	11
$\mathbf{u}_{\text{int}}$	Deformation field $\mathbf{u}$ interpolated on a tetrahedron.	11
$\mathbf{N}_{\mathbb{T}}$	Matrix assembled of shape functions of a tetrahedron.	11
$\mathbf{u}_{\mathbb{T}}$	Vector of deformation values at the points of a tetrahedron.	11
$\ddot{\mathbf{u}}_{\text{int}}$	Acceleration field $\ddot{\mathbf{u}}$ interpolated on a tetrahedron.	11
$\mathbf{b}_{\text{int}}$	External forces $\mathbf{b}$ interpolated on a tetrahedron.	11
$\ddot{\mathbf{u}}_{\mathbb{T}}$	Vector of acceleration values at the points of a tetrahedron.	11, 12
$\mathbf{b}_{\mathbb{T}}$	Vector of external force values at the points of a tetrahedron.	11
$\mathbf{w}$	Arbitrary test function in the weak form of a PDE.	11
$\mathbf{w}_{\text{int}}$	Arbitrary test function interpolated on a tetrahedron.	11
$\mathbf{w}_{\mathbb{T}}$	Vector of test function values at the points of a tetrahedron.	11
$\mathbf{M}_{\mathbb{T}}$	Mass matrix for a single tetrahedral element.	11
$\mathbf{K}_{\mathbb{T}}$	Stiffness matrix for a single tetrahedral element.	11
$\mathbf{M}_{\mathbb{G}}$	Global mass matrix for the tetrahedral mesh.	12
$\mathbf{K}_{\mathbb{G}}$	Global stiffness matrix for the tetrahedral mesh.	12
$\mathbf{u}_{\mathbb{G}}$	Values of the deformation field $\mathbf{u}$ sampled at all points in the tetrahedral mesh.	12
$\ddot{\mathbf{u}}_{\mathbb{G}}$	Values of the acceleration field $\ddot{\mathbf{u}}$ sampled at all points in the tetrahedral mesh.	12
$\mathbf{b}_{\mathbb{G}}$	External forces on each point in the tetrahedral mesh.	12
$\sigma_{v,i}$	Scalar von Mises stress value at a tetrahedron i.	15, 18, 25, 36
F^G	Gripper force at a given datapoint.	16, 18, 20, 25
d_{max}	Maximal distance between a gripper and a surface node to be considered a world edge.	18

Symbol	Description	Pages
$\bar{\sigma}_{v,i}$	Stress measure for a node i, computed as average of the von Mises stresses of tetrahedra it is part of.	18, 25, 36, 37
$\dot{\mathbf{u}}_i$	Virtual velocity of a node i, which is the gripper surface normal for gripper nodes, and zero for object nodes.	19
$\mathbf{c}_i$	Categorical type of a node i as one-hot vector.	19
$\mathbf{v}_i$	Feature vector of a node.	19, 21
$\mathbf{p}_i^1$	Deformed position of node i at the first frame of a simulated trajectory.	19, 20
$\mathbf{p}_i^2$	Deformed position of node i at the second frame of a simulated trajectory.	19, 20
$\mathbf{e}_M^{ij}$	Feature vector of a mesh edge between nodes i and j.	19, 21, 24, 25
$\lvert\mathbf{e}_W\rvert$	Number of world edges between gripper and object surface nodes.	19, 20, 25
$\mathbf{e}_W^{ij}$	Feature vector of a world edge between nodes i and j.	20, 21, 24, 25
$\tilde{\mathbf{v}}_i$	Feature vector of a node in latent space.	21
$\tilde{\mathbf{e}}_M^{ij}$	Feature vector of a mesh edge in latent space.	21
$\tilde{\mathbf{e}}_W^{ij}$	Feature vector of a world edge in latent space.	21
$(\tilde{\mathbf{e}}_M^{ij})'$	Latent mesh edge feature vector, updated after a round of message passing.	21
$(\tilde{\mathbf{e}}_W^{ij})'$	Latent world edge feature vector, updated after a round of message passing.	21
$(\tilde{\mathbf{v}}_i)'$	Latent node feature vector, updated after a round of message passing.	21, 22
$\hat{\mathbf{v}}_i$	Decoded node output feature vector.	22
$\hat{\mathbf{u}}_i$	Predicted deformation of a node i.	22
$\hat{\sigma}_{v,i}$	Predicted stress measure for a node i.	22
$\mathbf{p}_i^{\text{undeformed}}$	Original undeformed position of a node i.	25
$\mathbf{p}_i^{\text{deformed}}$	Deformed position of node i at given frame of a simulated trajectory.	25
$\mathbf{t}_i$	Feature vector of a tetrahedron i.	26

Symbol	Description	Pages
$\tilde{\mathbf{t}}_i$	Feature vector of a tetrahedron in latent space.	26
$(\tilde{\mathbf{t}}_i)'$	Latent tetrahedron feature vector, updated after a round of message passing.	26
$\hat{\mathbf{t}}_i$	Decoded tetrahedron output vector.	26
$\hat{\sigma}_{v,i}$	Stress prediction for a tetrahedron i.	26

1. Introduction

Deformable objects are omnipresent to us humans, and we instinctively know how to grasp and handle them, whether we pick a ripe fruit from a tree, squeeze ketchup out of a bottle, or shape clay in our hands. Robots lack this intuitive judgement, and approaches to robotic grasping for a long time were restricted to rigid objects or made invalid assumptions about deformation dynamics [1]. As robots move more and more in our everyday lives, whether in caregiving [2] or fruit picking [3], it is apparent that their conception of the consequences of robotic interaction through grasping with deformable objects must improve.

Recent developments in robotic simulators [4, 5] have enabled accurate simulation of the dynamics of deformable objects through the finite element method (FEM). The simulator *DefGraspSim* [5] is especially designed to predict deformation and stress values in soft objects grasped with a robotic gripper. However, this simulation runs much slower than real time, preventing it from being used in e.g. control schemes or reinforcement learning (RL) approaches to robotic grasping. Given the success of machine learning for prediction of physical dynamics in many domains [6], several learning-based approaches to robotic grasping are proposed [1]. Many of these approaches, however, do not consider deformation and stress values densely or at all within the grasped object. Huang et al. [7] implemented *DefGraspNets*, a graph neural network (GNN) model trained on grasp outcomes simulated by *DefGraspSim*. The model can accurately predict resulting deformation and stress at a number of nodes within the object. Huang et al. report that their trained model generalizes to unseen objects in different hardnesses and shapes.

At first, the aim of this work was to evaluate the *DefGraspNets* model, whose code was published, on a self-generated *DefGraspSim* dataset of grasp experiments. We planned to later use *DefGraspNets* as model in a model-based RL approach to robotic grasping of deformable objects.
Unfortunately, we realized multiple issues with its implementation: While Huang et al.

released a pre-processed dataset with their published codebase, the pre-processing module they used to convert simulation output data to network input data is missing. Training the model on the provided dataset, we found its performance to be unsatisfactory with default parameters. Improving the implementation would be challenging. Since it was forked from the work [8], a large part of the code is unused, and only identifying the relevant parts takes some time. Interestingly, we find that in the published form, the network takes the deformed object state at the first and second step of a simulated grasp as inputs when predicting the deformation at a later step. This means that inference is only possible for grasp poses which have been simulated.

Given these challenges, we decide on a complete re-implementation of the *DefGraspNets* model using the *PyTorch* deep learning library. The main goals of this are to

1. Enable training on a self-generated dataset from *DefGraspSim*, meaning the pre-processing module would have to be reconstructed;
2. Allow inference for arbitrary grasp poses without simulation, meaning the input state would have have to be changed to only include the undeformed state; and
3. Provide a quickly understandable and extendable codebase for future research, through clear code and documentation.

We first implement a baseline model replicating *DefGraspNets* as close as possible, while fulfilling goals 1 and 3. Afterwards, we implement an alternative input feature construction to achieve goal 2 as modification to the baseline. Additionally, we propose and implement a modification that lets the model learn stress values at tetrahedra in the object instead of at nodes, which is more in line with the ground truth data obtained and the physical reality.

This thesis is structured as following: After this introduction, we cover relevant related work in the areas of robotic simulation and learning object deformation in Chapter 2. Afterwards, in Chapter 3, we investigate the modus operandi of the finite element method to improve understanding of how ground truth deformation and stress data of grasp experiments is obtained in simulation. In Chapter 4, we give a detailed description of our methodology used within the scope of this thesis. We show how we use *DefGraspSim* to generate the ground truth dataset and present the implementation of our GNN model along the *DefGraspNets* baseline, their modifications and our training process. Chapter 5 gives detailed results of a multitude of conducted training runs. In chapter 6, we critically discuss and interpret these results. Chapter 7 concludes our work and gives an outlook to future work possible based on our findings.

2. Related Work

In this chapter, we discuss related work in two areas key to our approaches. For the area of robotic grasping in simulation, we give an overview of robotic simulators and their suitability for grasp experiments on deformable objects, and motivate the choice of *DefGraspSim* for the generation of our dataset. In the area learning for deformable objects, we look at how recent machine learning approaches succeeded in modeling and predicting the dynamics of physical processes, especially object deformation, and relate *DefGraspNets* to these approaches.

2.1. Robotic Grasping in Simulation

In robotics research, simulation environments play a crucial role. They enable experiments for researchers without access to expensive robots, and are leveraged in early stages of research where simulation offers risk-free testing and development of control strategies. Significant speedups can be achieved since set-ups (and tear-downs) of a scene become a matter of seconds, and simulation environments can be run in parallel and faster than real time. In their survey [9], Collins et al. define criteria that robotics simulators fulfill: Models for popular robot joints, actuators and sensors are available. A robot and its environment are represented as a scene, and the user may import objects given as 3D meshes to the scene. A programming interface lets the user set up scenes and control the robot. A physics engine is used to model physical phenomena, especially collision and friction between the robot and its environment. Currently, the robotics simulators *MuJoCo* [10], *PyBullet* [11], *Gazebo* [12] and *CoppeliaSim* [13] enjoy high popularity [9].

As grasping is one of the most important modes of interaction of a robot with its environment, a realistic simulation of grasps is of high importance. In order to grasp and

manipulate an object, a robot is equipped with a gripper. Grippers may be detailed imitations of the human hand [14]. A simpler case is a parallel jaw gripper, which only consists of two fingers that can be squeezed around an object. Early robotic simulators already specialized on grasping. *GraspIt* [15] and *OpenGrasp* [16] modeled several available robot gripper types, and provided a framework for grasp evaluation on rigid objects. Nowadays, the mentioned simulators [10–13] model robotic grippers and allow simulation of grasps, and are popular tools for research around grasping. For instance, [17] used *MuJoCo* for a RL approach to grasp planning.

However, we are interested in grasping deformable objects. This application of robotics is also called *soft robotics* (parts of the robot or environment may be soft, deformable) in [9]. For soft robotics, the landscape of simulators changes. In the survey [9], Collins et al. note that research for soft robotics often employs full-fledged industry multiphysics suites, which allow simulation of several physical aspects from heat transfer to fluid dynamics. Deformable object dynamics are modeled by realistic FEM simulation. However, these suites are not tailored towards robotics and do not qualify as robotic simulators due to the lack of predefined robot models. Requiring additional setup, they come at the cost of less ease of use compared to robotic simulators [9]. Moreover, industry standard suites such as ANSYS are closed-source and expensive. Of the popular robotic simulators, *Gazebo* and *CoppeliaSim* are limited to rigid bodies. *MuJoCo* uses a mass-spring system to model deformation, a simplification from the physical reality which can approximate soft objects, but does not solve for realistic deformations and can not model stresses in objects [18]. Furthermore, it suffers from numerical instability. Especially when attempting to bridge the Sim-to-real gap, the limitations of spring-mass models come to light [19]. *PyBullet* uses a FEM model for object deformation, however, stress values are not accessible to the user.
Developed with the objective of accelerating research of RL approaches in physical systems, the robotic simulator *IsaacGym* [4] encapsulates the powerful multiphysics engine *FLEX*, offering FEM simulation of rigid robotic grippers in contact with deformable objects [20]. Resulting deformation and stress tensors are computed on the GPU and accessible via programming interfaces. Deep RL approaches can learn without CPU bottleneck, and parallel simulation of several scenes is possible for generating large datasets [21, 22].

Building on *IsaacGym*, the work *DefGraspSim* [5] is purpose-built for the simulation of grasp experiments and their outcomes. *DefGraspSim* focuses on a *Franka Panda* parallel jaw gripper, in a pose as specified by the user. The rest of the robot arm is not active part of the simulation. A deformable object is also specified with its physical properties. The gripper closes with increasing force, and interaction between gripper and object is simulated by *FLEX*'s FEM solver. Deformation and stress values are saved to disk. In their

study, Huang et al. additionally derive grasp performance metrics for deformable objects, which *DefGraspSim* computes. As *DefGraspSim* plays a role in our work, we describe its usage and data formats in Section 4.1.

2.2. Learning for Deformable Object Dynamics

Though accurate FEM simulation of the physics of deformable objects in a multiphysics suite or suitable robotic simulator is the "gold standard" [7], it comes at the cost of a high computational load. Given the recent successes of machine learning and deep learning in multiple applications, research interest has been become high in the topic of *learning* physical processes, i.e. fitting a function between inputs and desired outputs, and optimizing this function by minimizing a loss term on a number of examples. Using learned models can cut the computational effort in fractions compared to simulation in e.g. a multiphysics suite, at the cost of requiring a large training dataset. Furthermore, machine learning approaches can be employed on processes where the exact physical model underlying is unknown [23]. In the field of fluid dynamics, successful approaches [24, 25] employed convolutional neural networks (CNNs). CNNs require the division of the computational domain into a regular grid. Other approaches model rigid bodies [26] or particle-based fluids [27] as graph structure, and predict dynamics through GNNs.
Wang et al. [28] were able to accurately model the springback of bent metal tubes, a process hard to model via FEM, using GNNs. In their paper [8], Pfaff et al. proposed learning on mesh-based representations. Many physical simulations, e.g. the finite element method for object deformation, approximates the solution on discretized mesh geometries. They constructed graph features from mesh representations of different simulated processes, deforming metal plate, a waving cloth, airflow over an airfoil and turbulent water flow around a cylinder. Key in graph construction was to define two different edge sets, where one comes from the mesh geometry, and the other one is constantly remeshed, connecting nodes which are close to each other in the world space. Using the same GNN block for the different processes, they showed that accurate predictions of the next time step could be learnt, given a mesh input state.

When it comes to robotic grasping, several studies [29, 30] focused on learning object parameters such as Youngs modulus E from exploration, and then model the deformable object in a classic way. In [29], FEM is used, while [30] uses a mass-spring model. These studies were conducted on real robots, with the object geometry being assessed by depth cameras. Also using depth camera input, the work [31] used a GNN to learn object

deformation on a particle-based model. The learnt model was accurate enough to be used in a model-based RL approach to deform objects into target shapes.
In the study [7], Huang et al. build on the architecture of [8], and train their model *DefGraspNets* on the mesh-based FEM simulation of robotic grasps, obtained using the grasp simulator *DefGraspSim*. The model predicts object deformation and stress measures at the nodes of the object mesh. Unlike in [8], this model does not perform predictions of the next time step, but instead predicts the resulting state when squeezing with a given force. Inspired by [31], Huang et al. used the model as a differentiable surrogate simulator. This allowed them to perform gradient-based optimization of input parameters like gripper rotation. Grasp plannng on deformable objects is a topic of high interest in recent research, and approaches such as [32, 33] do not model the object as detailed as *DefGraspNets*. For a detailed overview, we refer to the survey [1].

3. Preliminaries

Before diving into the efforts of implemented methodology of this work, this section motivates and introduces two key techniques used—the linear finite element method, used in similar form in the backbone of the *DefGraspSim* simulator generating ground truth data, and graph neural networks as the network architecture enabling us to learn physics of deformable objects on simulation data.

3.1. Finite Element Method

Many of the physical phenomena which we see and experience every day can be modeled by partial differential equations (PDEs). The flow of heat in a room, the elastic budging of a chair when we sit on it, or the flow of air around an airplane are described by complex equations relating measures such as air velocity and temperature, stress and strain in an object, or air velocity and pressure with their derivatives. These differential equations, which contain derivatives of functions with respect to (w.r.t.) multiple dimensions, e.g. time and space, are called PDEs. When we restrain degrees of freedom by giving values for a function or one of its derivatives at some position, the PDE and these boundary conditions describe a boundary value problem. Given that it is well posed, a unique solution of functions that satisfy the PDE as well as the boundary conditions exists. Unfortunately, closed-form solutions are feasible only for specific types of PDEs on simplest geometries with specific boundary conditions. For many problems, such as viscuous fluid dynamics described by the Navier-Stokes equations, not even the existence of a solution could be proven to date. In practice, boundary value problems can be solved using numerical methods, which discretize the function space to a finite amount of points on which function values are sampled, and solve for these finite degrees of freedom. These methods can give accurate results, and in the past decades, have proven as an invaluable tool in engineering and research.

Following this motivation, we derive the PDE governing the physics of object deformation following [34, 35]. We describe how the object geometry is discretized in a tetrahedral mesh, and sketch the principle of how a FEM solver approximates a solution of the PDE along [36].

3.1.1. Continuum Mechanics and Physics of Deformable Bodies

Contrary to the physical reality of molecules and atoms, the study of continuum mechanics models a solid body as material continuously filling out a 3D field. The state and properties of the body may then be described through continuous functions. This is an assumption close to reality for applications on a sufficiently large domain, and the continuous formulation allows mathematically consistent use of differentiation and integration of functions over the object. A central function in continuum mechanics is the displacement field $\mathbf{u}$, a 3D vector field which gives the 3D deformation of each point in the continuous body. It is the primary unknown quantity we would like to solve for in FEM.

From the displacement vector field, we derive the strain tensor $\boldsymbol{\varepsilon}_C$ as measure for relative deformation of the material at a point. A strain component ij gives the relative elongation in direction i in response to a deformation along direction j. Diagonal entries of the strain tensor are normal strains ε, and off-diagonal entries are shear strains γ, which measure angular distortion. Assuming small deformations of the object, the linearized, symmetric CAUCHY strain tensor is derived from the displacement field as

$$\boldsymbol{\varepsilon}_C = \begin{bmatrix} \varepsilon_{xx} & \gamma_{xy} & \gamma_{xz} \\ \gamma_{yx} & \varepsilon_{yy} & \gamma_{yz} \\ \gamma_{zx} & \gamma_{zy} & \varepsilon_{zz} \end{bmatrix} = \frac{1}{2} \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right). \tag{3.1}$$

Strains present in material result in internal elastic forces acting against the deformation. In the continuum, we consider infinitesimal volumes, and convert to the notation of stress, which has the same dimension as pressure (force per area). The relationship between strain and stress is governed by properties of the material. In HOOKEAN material, its properties are captured in the fourth-rank elasticity tensor $\mathbf{D}$. The elasticity tensor mainly depends on the material parameters E and ν. YOUNGS modulus E is a measure of material stiffness, while POISSONS ratio ν relates material deformation in lateral and axial direction resulting from axial stress. These properties may be functions over the continuum. In isotropic material, E and ν are constant within the body. Having derived the elasticity

tensor, the relationship between strain and stress is linear with

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{bmatrix} = \mathbf{D}\,\boldsymbol{\varepsilon}_C. \tag{3.2}$$

In the resulting stress tensor $\boldsymbol{\sigma}$, similar to the strain tensor, diagonal entries σ_{ii} give normal stresses, and off-diagonal entries τ_{ij} give shear stresses. Normal stresses inform whether the material is compressed or expanded. Shear stresses are non-zero if the material is experiencing distortion or shearing.
Like the strain tensor, the stress tensor is symmetric. The six degrees of freedom can be encoded in VOIGTS notation as six-dimensional vector each. In this notation, the elasticity tensor **D** reduces to a 6×6 elasticity matrix **C**:

$$\left[\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \tau_{xy}, \tau_{xz}, \tau_{yz}\right]^T = \mathbf{C}\left[\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \gamma_{xy}, \gamma_{xz}, \gamma_{yz}\right]^T$$

The introduced concepts let us now calculate the equilibrium of forces for an infinitesimal volume element. The sum of elastic forces over an element is the divergence of the stress tensor

$$\nabla \cdot \boldsymbol{\sigma} = \begin{bmatrix} \partial_x \sigma_{xx} + \partial_y \tau_{xy} + \partial_z \tau_{xz} \\ \partial_x \tau_{yx} + \partial_y \sigma_{yy} + \partial_z \tau_{yz} \\ \partial_x \tau_{zx} + \partial_y \tau_{zy} + \partial_z \sigma_{zz} \end{bmatrix}.$$

NEWTONS law of motion postulates that the acceleration $\ddot{\mathbf{u}}$ of the infinitesimal element mutliplied with its density ρ is equal to the sum of forces present over it. Additionally to the elastic forces, arbitrary external forces **b** may be applied to the object. The equilibrium valid for each infinitesimal volume (and thus point) in the continuum follows as

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \rho\,\ddot{\mathbf{u}}. \tag{3.3}$$

The derived PDE (3.3) governs the dynamics of object deformation. We note that it contains a second derivative of the displacement field **u** w.r.t. the temporal dimension on the right side, and first derivatives of the stress tensor w.r.t. the three spatial dimensions in the divergence operator on the left side. Since we derived the stress tensor in Equation (3.2) from the CAUCHY strain tensor (Equation (3.1), the left side of the equation contains spatial derivatives of second order of **u**.

3.1.2. Finite Element Discretization

Given the complexity of Equation (3.3) and the arbitrary complexity of the analyzed object and its boundary conditions, a solution for the vector field $\mathbf{u}$ is possible only numerically. For this, we step away from considering the displacement field as continuous function. Instead, we sample its values $\mathbf{u}_i$ for a finite number of points i in the object. These points are the product of *discretization* of the spatial domain of the object into a set of volume primitives, for each of which a closed-form approximated solution of the PDE becomes possible. As volume primitives, tetrahedra are chosen, each spanning the volume between four points. By convention, a tetrahedron is interpreted to have a positive volume if the fourth point is in the direction of the surface normal implied by the three other points. Tetrahedra do not intersect or leave gaps between each other.

The object geometry is approximated by the union of these tetrahedra, and this approximation is called tetrahedral mesh. Depending on the number and resolution of elements used, the geometry approximation may become arbitrarily accurate and complex. Especially inside the object, the manner where the volume gets divided into elements is ambiguous. Shewchuk [37] examined the influence of several mesh properties to the accuracy of FEM results. Best simulation results can be achieved with elements that have similar edge lengths and thus large interior angles. It was found that a single badly conditioned element in a mesh, or elements with physically nonsensical negative volume, introduce numerical instability and can significantly degrade the simulation accuracy.

Given the high importance of mesh quality to FEM, *meshing* as the process of converting object geometry to a tetrahedral mesh is an active research field [38, 39]. Adaptive meshing is a strategy where element resolution is finer in sections of the object with small details, and coarser in homogeneous areas [40]. While traditional meshing algorithms only accept surface meshes as input, and struggle with imperfections in the surface, newer approaches [41] can construct high-quality tetrahedral mesh even from sets of triangles not structured as surface mesh, allowing fast meshing from e.g. 3D scans.

3.1.3. Weak Formualation and Solution

Given the discretized object geometry as tetrahedral mesh, we now sketch the solution of PDE (3.3) with the linear FEM. First, we introduce the role of *shape functions*: They allow to come back to a continuous displacement function within a tetrahedron given the displacement values $\mathbf{u}_i$ at its points i. The shape function N_i of a point i is given as

piecewise polynomial. Its coefficients depend on the geometry of the tetrahedron. With $\mathbf{u}_i$ given at the points, a continuous interpolation $\mathbf{u}_{\text{int}}$ can be constructed as

$$\mathbf{u}_{\text{int}}(\mathbf{x}) = \sum_i N_i(\mathbf{x})\, \mathbf{u}_i.$$

Assembling the shape functions of a tetrahedron $\mathbb{T}$ in a matrix $\mathbf{N}_{\mathbb{T}}$, and stacking the point deformations $\mathbf{u}_i$ as degrees of freedom for the interpolation in a vector $\mathbf{u}_{\mathbb{T}}$ allows the interpolation to be computed as the product $\mathbf{u}_{\text{int}}(\mathbf{x}) = \mathbf{N}_{\mathbb{T}}(\mathbf{x})\, \mathbf{u}_{\mathbb{T}}$.

Similarly, we obtain interpolations of the acceleration field $\ddot{\mathbf{u}}_{\text{int}}$ and the external forces $\mathbf{b}_{\text{int}}$ from their discrete values $\ddot{\mathbf{u}}_{\mathbb{T}}$ $\mathbf{b}_{\mathbb{T}}$. Given these interpolations, spatial derivatives within the tetrahedron can now be computed in closed form. However, using linear shape functions (polynomials of first degree) raises the issue that the second derivatives in Equation (3.3) disappear. To aid this, we convert PDE (3.3) to its weak formulation by multiplication with an arbitrary test function $\mathbf{w}$, leading to

$$\mathbf{w}^T \left(\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} - \rho\, \ddot{\mathbf{u}}\right) = 0.$$

We also obtain an interpolation $\mathbf{w}_{\text{int}}$ of the test function from its discrete values at the tetrahedrons points $\mathbf{w}_{\mathbb{T}}$ as $\mathbf{w}_{\text{int}}(\mathbf{x}) = \mathbf{N}_{\mathbb{T}}\, \mathbf{w}_{\mathbb{T}}$. Inserting Equation (3.2) and Equation (3.1) and the interpolations in the weak form yields

$${\mathbf{w}_{\text{int}}}^T \left(\nabla \cdot \frac{1}{2}\mathbf{C}\left(\nabla \mathbf{u}_{\text{int}} + (\nabla \mathbf{u}_{\text{int}})^T\right) + \mathbf{b}_{\text{int}} - \rho\, \ddot{\mathbf{u}}_{\text{int}}\right) = 0.$$

By introducing the test function into the parentheses and integrating each term over the tetrahedron, it is shown in [36] that the equation is equivalent to

$${\mathbf{w}_{\mathbb{T}}}^T \left(\mathbf{M}_{\mathbb{T}}\, \ddot{\mathbf{u}}_{\mathbb{T}} + \mathbf{K}_{\mathbb{T}}\, \mathbf{u}_{\mathbb{T}} - \mathbf{b}_{\mathbb{T}}\right) = 0,$$

where the *mass matrix* $\mathbf{M}_{\mathbb{T}}$ depends on the density and the shape functions, and the *stiffness matrix* $\mathbf{K}_{\mathbb{T}}$ contains spatial derivatives of the shape functions and depends on the elasticity matrix $\mathbf{C}$ of the material. Since the test function was defined as arbitrary, its degrees of freedom also are, and we find that

$$\mathbf{M}_{\mathbb{T}}\, \ddot{\mathbf{u}}_{\mathbb{T}} + \mathbf{K}_{\mathbb{T}}\, \mathbf{u}_{\mathbb{T}} - \mathbf{b}_{\mathbb{T}} = 0.$$

This equation relates the discrete deformations and accelerations in the tetrahedron with its geometry, material properties and external forces. The spatial derivatives have been computed on the geometric primitive, and it only contains the second temporal derivative $\ddot{\mathbf{u}}_{\mathbb{T}}$.

The mass and stiffness matrices can be constructed for each tetrahedron in the object mesh. Through reindexing of local to *global* points and superposition of the local matrices, global mass and stiffness matrices $\mathbf{M}_{\mathbb{G}}$, $\mathbf{K}_{\mathbb{G}}$ are constructed that relate the global discrete deformations $\mathbf{u}_{\mathbb{G}}$ and accelerations $\ddot{\mathbf{u}}_{\mathbb{G}}$ to the global geometry, material and external forces $\mathbf{b}_{\mathbb{G}}$:

$$\mathbf{M}_{\mathbb{G}}\,\ddot{\mathbf{u}}_{\mathbb{G}} + \mathbf{K}_{\mathbb{G}}\,\mathbf{u}_{\mathbb{G}} - \mathbf{b}_{\mathbb{G}} = 0. \tag{3.4}$$

Through discretization of the spatial domain in volume primitives, we transformed the PDE (3.3) on the continuum into a system of ordinary differential equations (ODEs) on its discretized function values. If we are only interested in the static case, we can constrain the acceleration $\ddot{\mathbf{u}}_{\mathbb{G}}$ in Equation (3.4) to zero. We then obtain a system of linear equations, which we can solve for the static deformations in equilibrium with the external forces. In the dynamic case, Equation (3.4) represents a system of ODEs. The spatially discretized displacement field is then a function of time $\mathbf{u}_{\mathbb{G}}(t)$, and its solution is approximated by classical numerical methods for ODEs by discretization of the temporal domain into timesteps. For each timestep, a system of linear equations must be solved, which has three degrees of freedom per node in the tetrahedral mesh. Though the system matrix typically is sparse, the computational effort is high, since a stable and accurate simulation requires sufficiently small timesteps. The given boundary conditions to PDE are encoded in the mass matrix for movement constraints of nodes, and in the external forces for applied force.

4. Methodology

In this section, we give a detailed description of the approaches used in this thesis. The first key component was to obtain ground truth deformation and stress for a variety of grasp experiments through simulation with *DefGraspSim*. We describe the setup and execution of the simulations and the obtained dataset. Second, we used this data as the basis for implementing and training a GNN designed to predict deformation and stress outcomes based on input grasp parameters. We provide a detailed description of the necessary data pre-processing, feature construction and GNN architecture, as well as the training process. Finally, we motivate and present our two proposed modifications to the baseline model.

4.1. Simulating Grasp Experiments Using *DefGraspSim*

To compute the resulting deformation and stress for a grasped object, for a variety of different gripper positions and objects, the robotic simulator *DefGraspSim* [5] was used. As discussed in Section 2.1, *DefGraspSim* is based on the *IsaacGym* simulator and uses the *FLEX* GPU-based multiphysics engine. Using this engine, *DefGraspSim* simulates a deformable object in contact with rigid robotic grippers using FEM. Following, we will first define the experiments provided by *DefGraspSim* and its usage. Then, we describe the generation and scope of our ground truth dataset.

4.1.1. Grasp Experiment

DefGraspSim provides multiple experimental settings. All simulate the Franka Emika Panda robot arm equipped with a parallel jaw gripper. The deformable object to be grasped is provided by the user as tetrahedral mesh. Also, a list of 7D *grasp poses* for the gripper hand must be specified. Each grasp pose gives the 3D position of the hand as

Flag name	Interpretation
`--object`	Selection of one of the deformable objects provided by user
`--grasp_ind`	Selection of the grasp pose to be used
`--density`	Density ρ of the deformable object
`--youngs`	YOUNGS modulus E of the deformable object
`--poissons`	POISSON's ratio ν of the deformable object
`--friction`	Kinematic friction μ between gripper and deformable object
`--ori_start`	Start index of vector directions to run experiments in parallel for
`--ori_end`	End index of vector directions to run experiments in parallel for
`--mode`	Experiment setting to simulate

Table 4.1.: Command line arguments for the *DefGraspSim* simulator.

well as the rotation of the gripper as 4D quaternion. The object mesh and grasp poses are input as files. Physical properties of the object such as stiffness and density are given as command line arguments when launching the simulation. Table 4.1 lists the command line arguments available to the user and their interpretation.

The setting `pickup` starts with the deformable object laying on a support plane. The gripper is transformed to the given grasp pose and closes its fingers around the object with increasing force. Then, the support plane is slowly lowered. The experiment ends as soon as the object loses contact with the support plane (indicating a successful pickup), or the gripper loses contact with the object (unsuccessful pickup). During the experiment, the simulator records measures such as contact points between gripper and object or strain energy within the object. `pickup` can be used to determine whether a given grasp pose is sufficient to establish a firm grasp around the object to pick it up, and compute quality measures for the grasp pose [5]. Three other settings extend the `pickup` scenario: The `reorient` experiment moves the gripper such that the object turns 180 degrees around a specified vector direction, as soon as the support plane loses contact. `shake` accelerates the gripper along the specified direction with constant jerk. `twist` performs angular acceleration around the vector direction with constant jerk. 16 direction vectors are provided by *DefGraspSim* and selected via command line arguments.

We used an undocumented mode, `squeeze-no-gravity`, to obtain the data used in the scope of this work. `squeeze-no-gravity` is the only mode that saves object deformation and stress within the entire object. In this setting, the gripper hand is in a static position, and the effect of gravity is not simulated. With the gripper in its grasp pose, the fingers are closed with constant velocity, until contact with the sample object is established. Then, the fingers slowly increase their force on the object, until a force

of $15\,\mathrm{N}$ is reached. The soft body physics are simulated at a frequency of $1500\,\mathrm{Hz}$. Since the computation is slower, the experiment does not run in real time. Results of the FEM solver are saved to disk: For a number of 50 frames between initial contact and maximal force, the deformed positions for all mesh nodes are saved. For each mesh tetrahedron i, from its stress tensor $\boldsymbol{\sigma}$ obtained from its deformation, the VON MISES stress

$$\sigma_{v,i} = \sqrt{\frac{1}{2}\left[(\sigma_{xx} - \sigma_{yy})^2 + (\sigma_{yy} - \sigma_{zz})^2 + (\sigma_{zz} - \sigma_{xx})^2\right] + 3(\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2)}$$

is computed and saved. The VON MISES stress gives a scalar value for the extent of the stress within the object [42]. It is an important measure since comparing it to the yield strength of the material gives an estimate if it would yield, i.e. permanently deform, at this position [43]. Huang et al. demonstrated the accuracy of these simulated values in a sim-to-real experiment on tofu blocks of various firmness, which yielded approximately at the positions where *DefGraspSim* predicted the largest VON MISES stress value within the mesh [5]. The simulator does not model permanent (plastic) deformation, as objects are assumed to be ideally elastic. All output data is saved in a `.h5` dictionary-style file. An overview of all data output in a *DefGraspSim* experiment is given in Table A.3. The parallel computing capability of the *IsaacGym* framework is not exploited in this experiment setting, as in a launched scene only one grasp pose can be evaluated.

4.1.2. Simulator Input Dataset

The performance of our later trained GNN models depends on the variety of training data. We are grateful that we have been provided the `dgn_dataset` by Huang et al., which they used for their work [7]. For this dataset, Huang et al. created both synthetic simple object geometries such as cylinders and spheres, as well as fruit and vegetable pieces from 3D scans [7]. The obtained surface meshes were converted to tetrahedral meshes using fTetWild [41]. For each of the objects, using an antipodal grasp sampler [44], 100 grasp poses were generated. A list describing the various object meshes is given in Table A.1. We chose to simulate the grasp experiments with a YOUNGS modulus of $5 \times 10^5\,\mathrm{Pa}$ for all objects, which is approximately the hardness of a ripe tomato [5]. This value is consistent with the hardness of the fruit objects in reality, and gave visible deformations within simulated trajectories.

On a local machine with i9-11990k CPU and RTX 3090 GPU, simulating one grasp in *DefGraspSim* took around 7 to 10 minutes. We have utilized the institute's internal high-performance computing cluster to simulate all grasp experiments for each object. The

simulator was set up in a *Docker* image. For each of the 71 objects in the `dgn_dataset`, a SLURM script, passed to the cluster's workload manager, launched a container for the simulator. The input dataset as well as an output folder was mounted. Inside the container, all grasp experiments were simulated sequentially for the object. Utilizing the several machines available to the cluster, data generation was conducted spread over the project's duration. Since each grasp saved measures for 50 frames, the dataset contains $71 \cdot 100 \cdot 50 = 355\,000$ individual frames as data points available for training and testing of the models.

4.2. Graph Neural Network Implementation

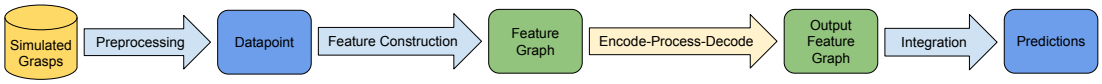

Figure 4.1.: Structure of the forward pass in the implemented GNN model.

After understanding the use of the *DefGraspSim* simulator, as well as its input and output data formats, and generating a large dataset, we implemented the graph neural network model as the main effort of this thesis. Given an input state including positions of gripper and object nodes and a closing force F^G, resulting 3D deformation and stress should be predicted for each object node. The implementation can be, as sketched in Figure 4.1, divided in four main steps: First, a pre-processing module combines all relevant information of a given frame from *DefGraspSim* output data in a datapoint. A pre-processed datapoint, which includes the input state as well as ground truth deformation and stress values, serves as input to the network implementation. Second, the network constructs a multigraph representation of the data point, where relevant features are encoded in node and edge feature vectors. Third, an *Encode-Process-Decode* GNN block, which includes all learnable parameters, encodes the feature graph in a latent representation, performs message passing rounds, and decodes an output feature graph. Fourth, the decoded features are interpreted as deformation and stress at each graph node, and the deformation is integrated on the input positions to obtain predicted position and stress. On this prediction, a loss value is computed, allowing gradient descent optimization of the GNN parameters.

In the following subsections, we describe the implementation in detail.

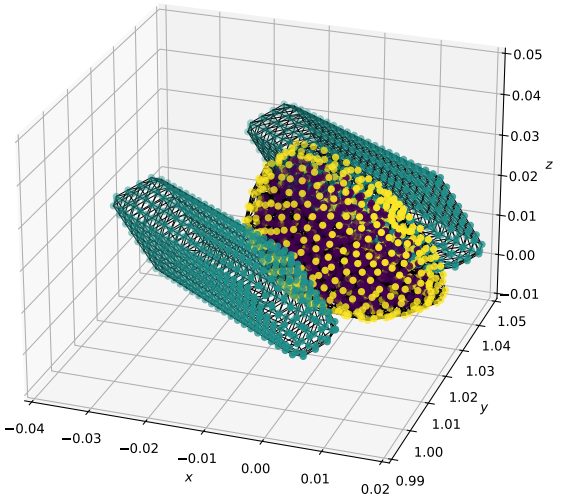

Figure 4.2.: Part of the input state of a pre-processed datapoint. Object and gripper nodes have been transformed into a common coordinate system. The gripper has been translated to its grasp pose and closed. The nodes belonging to the object surface were identified. Nodes are displayed in a scatterplot and colored according to whether they belong to the gripper, the object surface, or the interior. Mesh edges connecting the object and gripper nodes were obtained from the input geometry files and drawn.

4.2.1. Pre-processing Training Data

Huang et al. have used *DefGraspSim* data to train a graph neural network predicting object deformation and stress [7]. Table A.4 summarizes the data available in their pre-processed dataset for each data point. Understanding the interpretation and usage of these fields has been challenging due to a lack of documentation in their implementation. Comparing the pre-processed data to the outputs from a *DefGraspSim* simulation, given in Table A.3, it became clear that additional steps were required to obtain pre-processed data from simulation output data. For example, the simulator output did not contain information about the undeformed geometry of the object or its tetrahedrons, nor about the gripper geometry, both of which are present in pre-processed data. Huang et al. did not publish their pre-processing method along with their GNN implementation. In the following paragraphs, we describe how we filled the gap with our pre-processing routine.

Each simulation frame part of a dataset is pre-processed as following:

First, the geometry of the gripper fingers is loaded. We utilize a simplified geometry of the fingers from [7], which are given as `.stl` surface meshes for each finger. These meshes are parsed into a tensor of gripper nodes and edges. The node positions are given in a coordinate system relative to the robot end effector, where the left finger is in its open position 4 cm from the origin, and the right finger 4 cm in the other direction. These are the original positions of the open gripper fingers. From the `.tet` file, we load the nodes, edges and tetrahedra of the undeformed object.

The gripper fingers are transformed to the specified grasp pose. Due to the variety of conventions existing for 3D transformations and the lack of documentation, significant effort went into reconstructing the correct order and application of the translation and rotation. Additionally, *DefGraspSim* uses a different world coordinate system than the given object and gripper meshes. We translate the gripper and undeformed object into this coordinate frame. By projection of the object nodes on the gripper surface, we determine the distance that the gripper fingers close until first contact with the object. Closed gripper nodes and undeformed object nodes are concatenated and saved as tensor in `undeformed_pos`. We also add deformed positions at the first and second frame of the trajectory as `first_pos` and `second_pos` to the datapoint. For these, the deformed object node positions and the closing distance of the gripper are obtained from the simulation output. Similarly, `gt_pos` gives closed gripper and deformed gripper positions for the selected frame. `mesh_edges` save all edges present in both the gripper and object meshes. To be able to create a 3D visualization of datapoints, faces belonging to the gripper are added as `mesh_faces`, and object tetrahedra as `mesh_tets`. The field `node_type` informs whether each node belongs to gripper or object. Additionally, from the tetrahedral mesh, it is computed which nodes belong to the object surface. These surface nodes make a third category in `node_type`.

We save `world_edges` as all edges that are between gripper and surface nodes, and cover a scalar distance smaller than a hyperparameter d_{max}. The restriction to surface nodes reduces the computational load, since less candidate node pairs must be considered. We naively compute distances between all gripper and surface nodes and threshold them. Using *PyTorch*, this is faster than using e.g. a KD-tree. From the simulator output, we add the force at the given frame F^G to the scalar tensor `force`. We also add the target von Mises stress $\sigma_{v,i}$ at each tetrahedron i as `gt_tet_stress`. Additionally, as a target measure of stress at each node i, we calculate the average stress of the tetrahedra that the node is part of as $\bar{\sigma}_{v,i}$. These nodal stress representations are added to the datapoint as `gt_stress`. Finally, the pre-processing method returns all defined input and target fields in a dictionary-style object.

4.2.2. Feature Construction

We encode the input state from pre-processed features into a *multigraph* representation. This enables us to utilize powerful GNN layers to predict deformation and stress at nodes. The multigraph is given as a set of feature vector representations $\mathbf{v}_i$ for nodes, as well as a number of edge sets, which give connections between nodes and enable message passing between them. Each edge belonging to an edge set k, connecting nodes i and j has a feature vector $\mathbf{e}_{ij}^k$.

The nodes of the geometric mesh of gripper fingers and deformable object serve as nodes in our multigraph, since we wish to predict deformation and stress at the object nodes, and gripper nodes are an important part of the input state. Following the work of Huang et al. [7], we decide on a simple feature representation for nodes: Given a virtual velocity $\dot{\mathbf{u}}_i$ of a node, and its node type in one-hot encoding $\mathbf{c}_i$, we obtain the node feature vector $\mathbf{v}_i$ as

$$\mathbf{v}_i = [\dot{\mathbf{u}}_i{}^T, \mathbf{c}_i{}^T]^T.$$

The virtual velocity is a 3D unit vector informing in which direction the gripper closes (the gripper normal), if the node belongs to a finger. For object nodes, it is the zero vector. Since the possible node types are gripper, object surface and object interior, $\mathbf{c}_i$ is three-dimensional, and with the three-dimensional virtual velocity, we obtain the six-dimensional node feature $\mathbf{v}_i$.

We use the mesh edges (edges in gripper and object geometry) as the first edge set. This allows information to be distributed within object and gripper. For the feature representation, we decide that the most prominent property of an edge is the 3D distance it covers, as well as the scalar distance. We consider these distances at both the first frame of the simulation (node positions $\mathbf{p}_i^1$), as well as at the second frame (positions $\mathbf{p}_i^2$). With two 3D distances as well as two scalar distances, we arrive at the eight-dimensional mesh edge feature vectors

$$\mathbf{e}_M^{ij} = \left[(\mathbf{p}_i^1 - \mathbf{p}_j^{(1)})^T, \left|\mathbf{p}_i^1 - \mathbf{p}_j^{(1)}\right|, (\mathbf{p}_i^2 - \mathbf{p}_j^{(2)})^T, \left|\mathbf{p}_i^2 - \mathbf{p}_j^{(2)}\right|\right]^T.$$

For the second edge set, we use the computed world edges to allow important information to flow between gripper and object nodes. For the feature vector of these edges, we encode 3D and scalar distance (for simplicity, only from the second frame), and the normalized force f. The normalized force is computed as the squeezing force f between gripper and object, divided by the number of world edges $|\mathbf{e}_W|$. We motivate this physically as a measure related to pressure: For a constant force, a large contact area between

object and gripper leads to less pressure on the object surface. A large contact area also means there are less world edges, leading to smaller normalized force. The feature vector for each edge is

$$\mathbf{e}_W^{ij} = \left[(\mathbf{p}_i^1 - \mathbf{p}_j^{(2)})^T, \left| \mathbf{p}_i^2 - \mathbf{p}_j^{(2)} \right|, \left(F^G / |\mathbf{e}_W| \right) \right].$$

With these node feature and edge feature representations for mesh and world edges, we construct the multigraph structure to act as input to the GNN. We will now turn to the implementation of the GNN block which contains all learnable parameters of the network.

4.2.3. Encode-Process-Decode Block

In their work [8], Pfaff et al. proposed the architecture of the *Encode-Process-Decode* GNN block, which is employed in our implementation. We sketch its structure in Figure 4.3, and in the next paragraphs, describe each of the steps in detail.

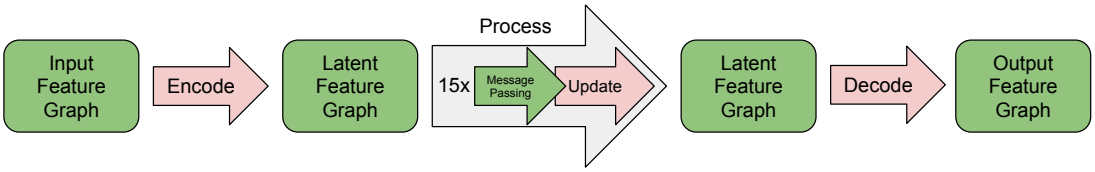

Figure 4.3.: Structure of the Encode-Process-Decode GNN block. Node and edge features of an input multigraph are encoded in latent space. For 15 rounds, a message passing scheme followed by an update function propagates information through the graph. The latent node features are decoded back into interpretable node features. Encoder, update function and decoder are MLPs with learnable parameters.

Encoder: From the input multigraph, node and edge features are encoded into a 128-dimensional latent space. This dimensionality was chosen as a trade-off between model complexity and performance. The increase in dimensionality is hoped to allow the processor to capture complex and possibly nonlinear relationships in the physical process that the model should learn. We define the MLPs f_{enc}^v for node encodings, and f_{enc}^M and f_{enc}^W for mesh edge respectively world edge encodings. The input width of these MLPs is the dimensionality of the constructed features, which we discussed in Section 4.2.2.

The output width is the latent feature dimensionality of 128. The latent node features $\tilde{\mathbf{v}}_i$, latent mesh edge features $\tilde{\mathbf{e}}_M^{ij}$ and $\tilde{\mathbf{e}}_W^{ij}$ are then computed as

$$\tilde{\mathbf{v}}_i = f_{\text{enc}}^v(\mathbf{v}_i),\ \tilde{\mathbf{e}}_M^{ij} = f_{\text{enc}}^M(\mathbf{e}_M^{ij}),\ \tilde{\mathbf{e}}_W^{ij} = f_{\text{enc}}^W(\mathbf{e}_W^{ij}).$$

Processor: The processor block consists of 15 rounds of message passing followed by an MLP update. One such step consists of feature aggregation, and an MLP update. For mesh edges and world edges, the message vector consists of the original latent feature of the edge, concatenated with the latent features of its start and end node. This concatenated feature is three times the latent size. We define update MLPs f_{upd}^M respectively f_{upd}^W that process and combine this feature containing message information back to a single latent mesh or world edge feature. To fight vanishing gradients and improve training performance, we add a residual connection. Altogether, the updated mesh edge features $(\tilde{\mathbf{e}}_M^{ij})'$ and world edge features $(\tilde{\mathbf{e}}_W^{ij})'$ are obtained as

$$(\tilde{\mathbf{e}}_M^{ij})' = f_{\text{upd}}^M(\tilde{\mathbf{e}}_M^{ij}, \tilde{\mathbf{v}}_i, \tilde{\mathbf{v}}_j) + \tilde{\mathbf{e}}_M^{ij},\ (\tilde{\mathbf{e}}_W^{ij})' = f_{\text{upd}}^W(\tilde{\mathbf{e}}_W^{ij}, \tilde{\mathbf{v}}_i, \tilde{\mathbf{v}}_j) + \tilde{\mathbf{e}}_W^{ij}.$$

For nodes, the message vector is its own latent feature $\tilde{\mathbf{v}}_i$, concatenated with the sum of latent features of incoming mesh edges and the sum of incoming world edge latent features. The node update MLP f_{upd}^v processes the message vector back to a single, updated node feature $(\tilde{\mathbf{v}}_i)'$, and we add a residual connection:

$$(\tilde{\mathbf{v}}_i)' = f_{\text{upd}}^v(\tilde{\mathbf{v}}_i, \textstyle\sum_j \tilde{\mathbf{e}}_M^{ij}, \sum_j \tilde{\mathbf{e}}_W^{ij}) + \tilde{\mathbf{v}}_i$$

We observe that the update rule for edges allows for information encoded in nodes to be represented in the edges' features. The update rule for nodes aggregates edge information in the nodes. As such, when performing several rounds of message passing, information can pass to neighboring nodes and their neighbors, and implicit encodings of deformation and stress can be propagated through the object. It is to be noted that the number of message passing rounds has influence on both network performance (it must be high enough such that useful information arrives everywhere in the network), and computational load (the MLP gradients must be computed for each round). In [7], Huang et al. found 15 rounds to give a good trade-off.

Decoder: With optimally trained encoder and update MLPs, the processor will have transformed the input features into meaningful latent features. After the message passing rounds, these are expected to encode implicit information of deformation and stress in the object. The decoder processes each updated latent node feature into a 4D output

feature. We interpret this output feature $\hat{\mathbf{v}}_i$ for each node as its predicted 3D deformation $\hat{\mathbf{u}}_i$ stacked on the averaged (see Section 4.2.1) VON MISES stress $\hat{\sigma}_{v,i}$. The decoding

$$[\hat{\mathbf{u}}_i{}^T, \hat{\sigma}_{v,i}]^T = \hat{\mathbf{v}}_i = f_{\text{dec}}((\tilde{\mathbf{v}}_i)')$$

again is performed by an MLP f_{dec} with learnable parameters.

All described MLPs are, with reference to [7] and [8], constructed similarly. Their first layer maps from the input dimensionality to the latent size 128 with *ReLU* activation. A hidden layer maps from 128 to 128, again with *ReLU* activation. The final linear layer maps from 128 to the output dimensionality.

4.2.4. Feature Normalization, Integration and Loss Function

To improve training performance, we normalize the constructed node, mesh edge, and world edge features over all nodes and edges in a training dataset. We compute the mean and standard deviation of each component in the feature vectors. Then, before calling the Encode-Process-Decode block in the forward pass of the network, we normalize each feature by subtracting the mean of each components, and dividing by the standard deviation of each component. We also compute the mean and standard deviation of ground truth outputs of the network. After obtaining graph output features as normalized predictions, we obtain unnormalized predictions by multiplying with standard devoation and adding the mean.

Since the pre-processing gives us the ground truth deformed node positions, we add the decoded, unnormalized node deformations to the input positions to get a predicted positions instead of only the deformation. We note a similarity to numerical methods for ODEs: If we interpret the predicted deformation as a deformation velocity, and assume a timestep of 1, then this is equal to the explicit EULER method, which has the update rule $y(t + \Delta t) = y(t) + t \cdot \partial_t y(t)$ for an arbitrary ODE of y. The stress outputs are not integrated, since the network is trained to directly output the stress value, instead of an increment.

We compute the training loss on the normalized graph outputs. This is necessary because the magnitudes of deformation and stress outputs differ significantly, with deformation (often less than 1 mm) being much smaller than stress (often larger than 100 kPa. Using normalized loss allows learning deformation and stress in the same network, with a reasonable learning rate. We obtain the normalized graph output features, and normalize the difference between `gt_pos` and `second_pos` and the ground truth stress with the

target normalizer to get a normalized target. We compute a mean squared error (MSE) loss on both the normalized deformation and normalized stress predictions and targets, and return the loss as sum of these. For the loss function, only the values for object nodes are considered. To be able to judge the training performance with physically sensible error values, we also compute the MAE between unnormalized target and ground truth deformation and stress.

4.2.5. Training Data Split and Training

We recall that data points were generated and pre-processed in files encompassing 100 different trajectories per object, and each trajectory contains 50 frames as datapoints. For training and evaluation, a dataset must be split into a training and a test data set. Training data is used to compute the gradient of the loss function with respect to the models parameters, and update per gradient descent. A set of test data is used to evaluate the performance of the model and judge its capability to generalize on unseen data points. We find three possible strategies to split into training and test data:

1. Split across trajectories—some frames of one trajectory are part of training data, some are part of test data;
2. Split trajectories across an object—some full trajectories are training data, some full trajectories are test data;
3. Split across objects—all trajectories of some objects are part of training data, all trajectories of other objects are part of test data.

The first strategy serves primarily as a proof of concept for the network, as both the training and test data points come from the same trajectories. We use it to tune hyper-parameters and debug the network during the implementation. The second strategy is more interesting: The 100 trajectories available for one object offer a high variety of grasp poses, e.g. in a split of 80 training and 20 test trajectories. Though it is to be expected that the network overfits on the objects geometry, it should generalize to arbitrary grasp poses, and would allow e.g. grasp sampling or refinement on the object. Training and evaluating with the third data split strategy reveals the capability of generalization to different object geometries. A such trained model is hoped to allow accurate inference on unseen geometries, provided that they are meshed in similar fashion as the objects trained on.

We provide a script that implements the training and evaluation loop. Through flags, the user is able to decide on the data split strategy, hyperparameters of the network, learning rate and training epochs, enable or disable the later described modifications, and set a random seed. The network is trained using the Adam optimizer [45] using a decaying learning rate. We save the model with the best performance on the test data set, and report training and physical loss values over training epochs using *TensorBoard*. We provide an evaluation script to test a model on other objects. The evaluation also creates files including model predictions and ground truth on test data, which can be visualized in *ParaView*, allowing a more intuitive judgement of performance than through the abstract loss values.

4.3. Modifications to the Baseline Model

The previously described implementation replicates the baseline given by [7], and closes the pre-processing gap, allowing training on our own generated dataset. In this section, we motivate and propose our two modifications that extend the baseline. The first modification lets the network train on undeformed inputs, allowing inference on grasp poses that were not simulated. The second modification extends the GNN architecture to learn stress values as tetrahedral features.

4.3.1. Inference Without Need to Simulate

In Section 4.2.2, we notice that the input state given to the network includes information that was *output* by the simulator, since mesh edge features $\mathbf{e}_M^{ij}$ encode node positions at the first respectively second frame of the simulated trajectory, where the object has deformed and the gripper closed. World edge features $\mathbf{e}_W^{ij}$ also encode second frame node positions. This gives raise to three issues: First, we must launch the simulator and simulate a trajectory just to obtain the input state of the data points. This is not a problem for training and evaluation, where we use the outputs at later frames to compute loss values, however this prevents the network from productive use, e.g. in grasp pose refinement or to infer on an unseen object. Second, with the features encoded as such, it is possible that the network learns to extrapolate the small deformation between the frames, instead of learning the actual physical dynamics. Third, obtaining the distances the gripper closed at the input state from *DefGraspSim* makes us lose differentiability of

the model with respect to the gripper pose, since the simulator is not differentiable [7]. This property, however, is important for grasp pose refinement.

We propose to only encode information present in the undeformed positions of the object and closed gripper nodes instead. These node positions $\mathbf{p}_i^{\text{undeformed}}$ were already constructed in the pre-processing (Section 4.2.1). Using the undeformed position instead of the first and second frame positions means mesh edge features $\mathbf{e}_M^{ij}$ only encode one 3D and scalar displacement, and its dimensionality is reduced to four:

$$\mathbf{e}_M^{ij} = \left[(\mathbf{p}_i^{\text{undeformed}} - \mathbf{p}_j^{\text{undeformed}})^T, \left| \mathbf{p}_i^{\text{undeformed}} - \mathbf{p}_j^{\text{undeformed}} \right| \right]^T$$

World edges are meshed based on the undeformed gripper and object surface nodes. World edge features $\mathbf{e}_W^{ij}$ are also constructed from undeformed positions

$$\mathbf{e}_W^{ij} = \left[(\mathbf{p}_i^{\text{undeformed}} - \mathbf{p}_j^{\text{undeformed}})^T, \left| \mathbf{p}_i^{\text{undeformed}} - \mathbf{p}_j^{\text{undeformed}} \right|, \left(F^G / |\mathbf{e}_W| \right) \right].$$

With these feature representations, the input state of a datapoint passed to the network is now independent of the outputs of the simulator. Ground truth deformation and stress targets are constructed from simulator data, and the network can be trained using the modified inputs. We further implement a script that constructs an input state without ground truth targets only from a provided object geometry and 7D gripper pose, allowing gradient-based grasp pose refinement to find a grasp minimizing e.g. maximal stress.

4.3.2. Learning Stress as Tetrahedral Value

In FEM simulation, displacements are computed at each node, and stress tensors at each tetrahedron in the mesh of the deformable object. From *DefGraspSim*, we obtained the deformed positions $\mathbf{p}_i^{\text{deformed}}$ at nodes, as well as the scalar VON MISES stress values $\sigma_{v,i}$ at tetrahedra for each frame in a simulated trajectory. The Encode-Process-Decode module, as employed by [7], [8] and our baseline implementation, however, can only decode node features and has no knowledge of tetrahedra in the mesh. To enable predictions of stress values, pre-processing converted the stress to a nodal feature: A stress value $\bar{\sigma}_{v,i}$ at each node was computed by averaging the stress values of neighboring tetrahedra. Given limited mesh resolution especially in small parts of an object, e.g. the handle of a cup, it is questionable whether this simplification is reasonable.

We propose instead learning the original stress values at tetrahedra, and extend the Encode-Process-Decode framework. For this, we add a tetratedron set to the multigraph

structure. The tetrahedron set gives all tetrahedra $\mathbb{T}_i$ in the object mesh by the indices of its four nodes. Unlike for the edge sets, no tetrahedra in the gripper or between gripper and object exist. The tetrahedron set also holds a feature vector $\mathbf{t}_i$ for each tetrahedron. The feature vector may include information related to the tetrahedra, such as its volume or interior angles. For simplicity, however, we construct scalar zero feature vectors

$$\mathbf{t}_i = 0.$$

Similar to node and edge feature representations (Section 4.2.3), latent tetrahedron features are obtained using a tetrahedron encoder MLP f^t_{enc}:

$$\tilde{\mathbf{t}}_i = f^t_{\text{enc}}(\mathbf{t}_i)$$

Since the features input to the encoder are zero, the latent features can not yet contain meaningful information. Nevertheless, we used this feature encoder instead of initializing the latent features with all-zero vectors to allow future work to experiment with different feature construction. We expect relevant information to enter the tetrahedron features during the message passing rounds. We define the update as following: The latent features of all nodes that make a tetrahedron are concatenated with the latent feature vector $\tilde{\mathbf{t}}_i$ of the tetrahedron. An update MLP f^t_{upd} processes these features back into a single latent vector. Finally, a residual connection is added:

$$(\tilde{\mathbf{t}}_i)' = f^t_{\text{upd}}(\tilde{\mathbf{t}}_i, \{\tilde{\mathbf{v}}_j\}_{j \in \mathbb{T}_i}) + \tilde{\mathbf{t}}_i$$

Per this update rule, the latent tetrahedron feature representations will encode information both from latent node features and latent edge features, which are part of the update for node features. As with node and edge updates, tetrahedron features are updated for several rounds, allowing relevant information to be propagated through the entire object.

Finally, the decoder structure is altered: We use a decoder MLP f^t_{dec} to decode an output feature vector $\hat{\mathbf{t}}_i$ for each tetrahedron. This decoded feature is one-dimensional and we interpret it as the prediction of the scalar von Mises stress $\hat{\sigma}_{v,i}$ at the tetrahedron,

$$[\hat{\sigma}_{v,i}] = \hat{\mathbf{t}}_i = f^t_{\text{dec}}((\tilde{\mathbf{t}}_i)').$$

We alter the training script, loss function, and normalization accordingly to account for the tetrahedral stress outputs. We note that our modification is implemented flexible in the input space, where meaningful tetrahedral features could be constructed, as well as in the output space, where additional values at tetrahedra could be learnt.

5. Results

Given the large generated dataset of outcomes of grasp experiments in 100 grasp poses on 71 objects, we evaluated the performance of the implemented GNN model both in its baseline form and with our proposed modifications. Multiple training runs were conducted with different combinations of model implementation and training dataset selection.

The baseline model was trained on a large dataset of 34 different objects, allowing us to roll it out on novel objects. Additionally, we trained a number of baseline models on a single object each to observe how the model capability depends on the choice of training objects. Finally, for both of our proposed modifications (Section 4.3), we train a number of models on these single objects. In the following chapter, we present the results of these runs. We provide evaluation metrics for each training run along with visualizations of model predictions, allowing for an intuitive judgement of the performance.

5.1. Baseline Model

5.1.1. Training on a Large Dataset

To evaluate the ability of the baseline model to generalize to unseen objects, we trained it on approximately half of the objects in our generated dataset. The 34 training objects for this training run are listed in Table A.5. The training dataset included $170\,000$ single frames as datapoints. The training was run for 20 epochs with a learning rate of 8×10^{-5}. Following each training epoch, the model was evaluated on the 5000 datapoints of the `strawberry01` object.

Figure 5.1 shows the mean training loss and the mean test loss on the `strawberry01` object at each training epoch. Except for a peak in epoch 12, the training loss steadily decreases throughout the training. Its final value is 0.3579. The test loss starts at a

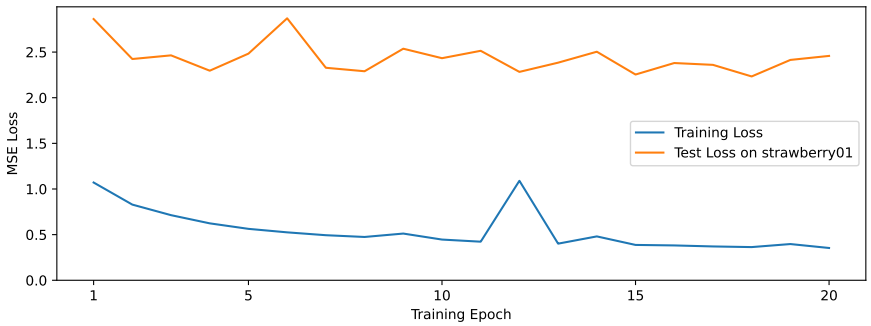

Figure 5.1.: Mean training loss and mean test loss on the test object `strawberry01` over training epochs for the baseline model trained on a large dataset.

Test Object	MSE Loss	Deformation MAE [mm]	Stress MAE [kPa]
`cuboid02`	0.032	0.177	0.602
`cup02`	1.100	0.633	5.501
`cucumber1`	1.291	0.746	4.153
`strawberry01`	2.176	1.001	5.832

Table 5.1.: Performance of the baseline model trained on a large dataset on unseen objects.

significantly higher value than the test loss and shows more noise during training. It marginally declines, with its lowest value being 2.176.

The model parameters were saved as checkpoint after epoch 18, where the lowest test loss on `strawberry01` was observed. We then evaluated this model on the objects `cuboid02`, `cup02` (primitive geometries), `cucumber1`, and `strawberry01` (3D scans). Table 5.1 gives the mean loss value and mean deformation and stress errors on these objects. The table shows that the loss is smallest on `cuboid02`. On `cup02` and `cucumber1`, the loss values are significantly higher, while `strawberry01` shows the highest test loss among these test objects. The mean deformation and stress errors are smallest for `cuboid02`, and of similar scale for the other objects.

Since the loss and error values are abstract and do not allow intuitive judgement of the accuracy of predictions, we provide a visualization of model outputs in Figure 5.2. For the last frame of a selected trajectory of the four test objects, deformation and stress values predicted by the model are shown next to the ground truth values. The 3D deformed

objects are rendered, and a colormap informs about the stress values. The loss values on these trajectories have been annotated. We observe that the predicted stress values are very close to the ground truth. The largest stress error is visible on `strawberry01`: High stress is predicted only relatively close to the grippers, while the ground truth stress actually is higher in an area between the grippers. The predicted deformed positions are mostly accurate, however, for each object except `cuboid02`, we see a rough, "noisy" surface which is not present in ground truth data.

5.1.2. Training on Single Objects

To investigate how well the baseline model can fit to single objects, we trained a model for each of the objects `8polygon06`, `cylinder07`, `lemon01`, `potato2`, `sphere03` and `strawberry01`. These runs used our second data split strategy, where the frames of 80 trajectories make the training dataset of 4000 frames, and 20 trajectories make a test dataset of 1000 frames. The models were trained with a learning rate of 8×10^{-5} for 20 epochs.

Training and Test Object	MSE Loss		Deformation MAE [mm]	Stress MAE [kPa]
	Training	*Test*		
`8polygon06`	0.517	0.719	0.169	0.255
`cylinder07`	1.278	1.179	0.108	0.146
`lemon01`	0.376	0.518	0.822	5.650
`potato2`	0.253	0.421	0.409	0.975
`sphere03`	0.123	0.147	0.266	0.712
`strawberry01`	0.248	0.324	0.539	2.921

Table 5.2.: Performance of baseline models trained on single objects. Each line gives the metrics for the model's checkpoint with lowest test loss. MAE of deformation and stress are given as interpretable error measures for test data.

Figure A.1 shows how for all of the runs, the training and test loss lowers significantly after only four epochs. The test loss converges after around ten epochs, though it is noisy. The training loss slowly keeps declining until the end of the training. Table 5.2 gives metrics for the model checkpoints with lowest test loss each. We observe that even though the losses of the models on different objects converge in a similar shape, the loss values between different objects are dissimilar. Training and test losses of single objects are of similar scale.

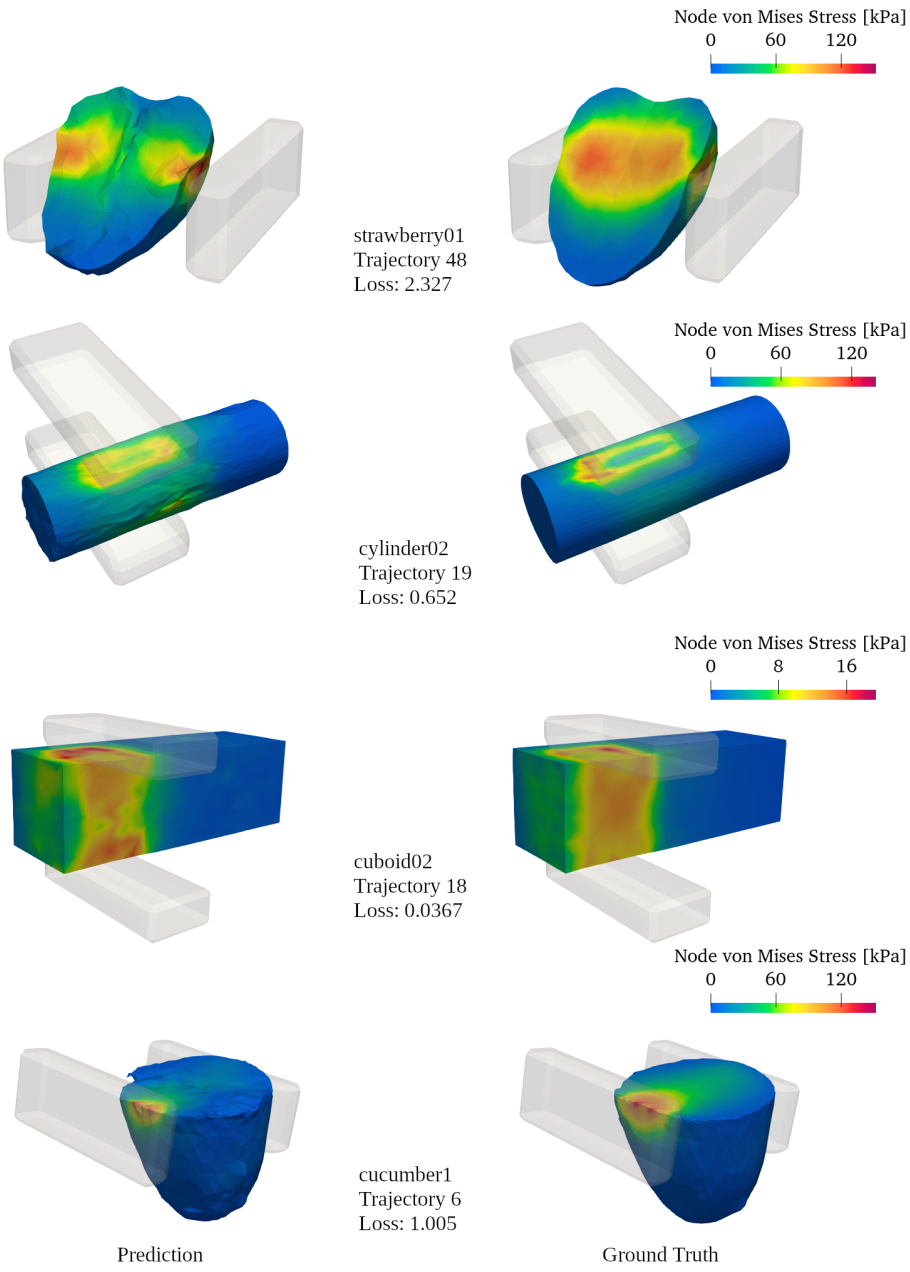

Figure 5.2.: Predicted and ground truth deformation and stress for frame 49 of one trajectory of each test object for the baseline model trained on a large dataset.

For each of the models, we visualized predictions and ground truth values of four trajectories from the test set. These visualizations can be found in the appendix. On `strawberry01` (Figure A.7), we observe high similarity between predicted and ground truth values. Especially stress values show nearly no visible difference as per the color coding. The predicted deformation is very accurate in trajectories 4 and 5. Loss values are accordingly small with approximately 0.05. In trajectories 18 and 33, which portray a stronger deformation of the object, the overall deformed shape is accurate. However, the object surface partially shows a rougher, "noisier" texture which is not present in ground truth. The loss values of these two trajectories are 0.265 respectively 0.568.
The models trained on `8polygon06` and `cylinder07` show very high accuracy in the visualizations in Figure A.2 and Figure A.3. The computed loss values, however, are high.
The model trained on `lemon01` (Figure A.4) does not perform well on trajectory 0, where the gripper is only in contact with the lemon in a very small edge region. The model performs much better, both visually and as measured by the loss values, for the other three visualized trajectories. However, all trajectories show a rough, "noisy" object surface.
The predictions on `potato2` (Figure A.5) are very accurate, and their loss values are reasonably small. Interestingly, the ground truth stress for trajectory 31 shows high stress on the right side of the visualization, where no gripper is in contact. This may be an error in simulation data, since we intuitively do not expect high stress values in that region. The prediction for trajectory 31 does not show significant stress values there.
Finally, the visualization of predictions on `sphere03` shows no visible error, and the loss values accordingly are minuscule.

5.2. Learning on Undeformed Input State

As defined in Section 4.2.2, the baseline model constructs the input state of a datapoint from node positions at the first two simulated frames. In Section 4.3.1, we proposed to modify the feature construction to only encode undeformed node positions, which can be obtained without accessing simulator outputs in our pre-processing.

For this modified model, we trained six runs in the second data split strategy. The training object for each run is the same as in the runs for the baseline model. Again, 80 trajectories are chosen as training and 20 trajectories as test data, however, due to different seeds for each run, the random split of trajectories is different to the runs on the baseline model. Each training run completed 35 epochs with a learning rate of 8×10^{-5}.

Training and Test Object	MSE Loss		Deformation MAE [mm]	Stress MAE [kPa]
	Training	*Test*		
`8polygon06`	0.821	0.687	0.177	0.238
`cylinder07`	1.551	1.229	0.113	0.151
`lemon01`	0.329	0.239	0.665	3.746
`potato2`	0.426	0.454	0.432	1.010
`sphere03`	0.282	0.320	0.392	1.112
`strawberry01`	0.162	0.329	0.638	1.949

Table 5.3.: Performance of models with undeformed input trained on single objects.

Figure A.8 shows different convergence of training and test losses during training compared to the baseline model (Figure A.1). For the runs on all objects, the training losses start at much higher values. During the first ten epochs, training losses decline quickest. The test losses also decline quickest during the first ten epochs and converge around the 20^{th} epoch. The model training and testing on `sphere03` has the highest training and test loss during the first epochs, through the training loss drops rapidly. For `cylinder07`, the test loss increases again towards the last epochs.

Metrics for each model checkpoint with lowest test loss are given in Table 5.3. It can be seen that training and test loss values are of similar scale for each run on an object, but again between the different object runs, loss values vary highly. Compared to the baseline results in Table 5.2, the loss and error metrics for the models with undeformed input are slightly higher. For the model trained on `lemon01`, the last frame of a trajectory is visualized in Figure 5.3. Though the loss value is quite high with 0.7842, the predicted deformations and stresses are accurate. The object surface is noisy again, as we already observed on the baseline models.

The implemented pre-processing returns the input state as function of the grasp pose, among others. Since the pre-processing is implemented in *PyTorch*, we can obtain the gradient of an arbitrary measure, e.g. the maximal stress value observed in the object w.r.t. the grasp pose, and refine the grasp pose accordingly to grasp with minimum resulting stress. We did not yet implement a method that refines grasp poses based on a trained model.

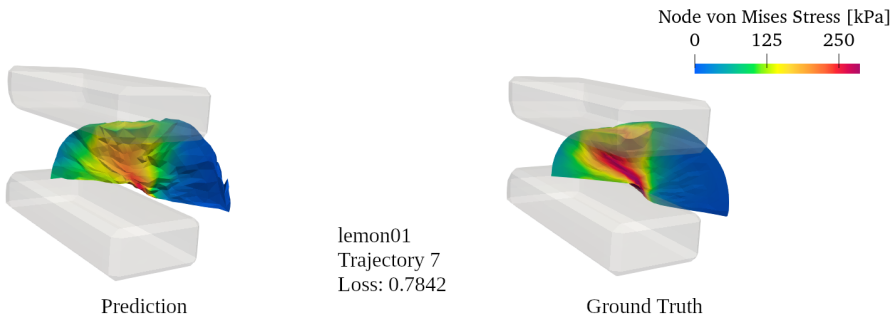

Figure 5.3.: Predicted and ground truth deformation and stress for frame 49 of test trajectory 7 for the model trained on undeformed inputs on `lemon01`.

5.3. Learning Tetrahedral Stress

Finally, we trained models to evaluate our proposed modification on the structure of the stress predictions and targets. As introduced in Section 4.3.2, for these models, the GNN block is informed about tetrahedra present in the mesh. The construction of tetrahedral features, and according encoder, update and decoder functions allows prediction of stress values for each tetrahedron, instead of for nodes. As for the baseline model and model with undeformed inputs, we trained six runs, each on an object with 80 training and 20 test trajectories randomly selected.

In Figure A.9, we see training and test loss of the trained models over completed training epochs. The training loss quickly declines for most runs within the first five epochs, and keeps declining slower until the end of the training. The training loss declines quickest until the fifth epoch for all runs and appears to have converged after approximately the tenth epoch. The scale of the loss values differs per object, as we already observed for the baseline models and the models with undeformed input. Table 5.4 shows the best training and test losses as well as average deformation and stress errors for each run. The losses and errors are slightly lower compared to the metrics for the baseline models trained on single objects. Figure 5.4 visualizes the last frame of a test trajectory for the `lemon01` model. The loss is low with 0.222. The predicted deformations and stresses look accurate. Compared to the models predicting stress values at each node, the stress field has more distinct borders in its rendering, and the largest, deep red values are only present in a few tetrahedrons. In the predicted trajectories, we observe less noise on the object surface as compared to predictions of the models which predict nodal stress.

Training and Test Object	MSE Loss		Deformation MAE [mm]	Stress MAE [kPa]
	Training	*Test*		
`8polygon06`	0.465	0.491	0.156	0.274
`cylinder07`	1.193	1.126	0.106	0.162
`lemon01`	0.310	0.301	0.756	4.393
`potato2`	0.174	0.327	0.368	0.995
`sphere03`	0.124	0.117	0.236	0.877
`strawberry01`	0.197	0.342	0.569	2.504

Table 5.4.: Performance of models learning tetrahedral stress trained on single objects.

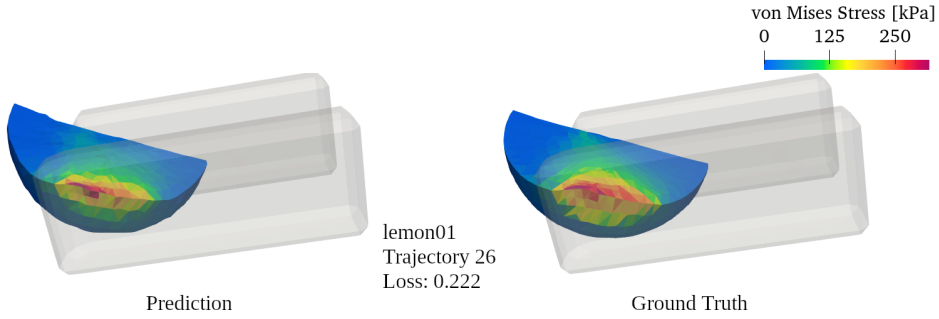

Figure 5.4.: Predicted and ground truth deformation and stress for frame 49 of test trajectory 26 for the model learning tetrahedral stress trained on `lemon01`.

6. Discussion

In the previous chapter, we presented key metrics of our conducted training runs, and visualized their predictions compared to ground truth. We summarize the results of these runs, interpret them and briefly discuss their implications.

Baseline trained on a large dataset: In Figure 5.1, we observed the convergence of the training loss, indicating that the baseline model is learning and fitting well to the training data. The GNN is able to learn feature representations that allow decoding into accurate deformation and stress, generalizing to many different mesh geometries and grasp poses. However, the relatively high test loss only marginally declined during training.
We attribute the scale of the test loss to a poor choice of the test object, `strawberry01`. Its shape, characterized by a flat side and a thin tip, as seen in Figure 5.2 and Figure A.7, is unique among the the dataset. This interpretation is supported by the lower loss values we observed during the rollout on `cuboid02`, `cup02` and `cucumber1`. The relatively good performance on `cuboid02` and `cup02` can be explained by the fact that the training set includes these geometric shapes, though in other sizes and differently meshed. The geometry of `cucumber1` (Figure 5.2) is more regular compared to `strawberry01`, lacking sharp edges and having an approximately equal thickness in every direction, which explains the good performance.
Regarding the slow decline of the test loss over the training epochs, we speculate that it is rooted in the large size of the training dataset. Since the model already completed $170\,000$ training steps before computing the loss values in the first epoch, it is conceivable that the model already learnt parameters in the first epoch which give a test performance close to its best capability. Subsequent epochs then would only further fit on the training set, without giving much improvement on the test set. Given the unfortunate choice of the single test object during the training process, it would be interesting to assess the performance per epoch on more test objects. Unfortunately, due to the long duration of the training on such a large dataset, we could not run additional experiments to investigate this.

Baseline trained on single objects: For the baseline models trained on single objects, we have seen pleasing convergence of the training and test losses in Figure A.1. Table 5.2 shows that for each of these models, the training and test loss is similar. This leads us to conclude that the baseline model does not tend to overfit. The difference in the scale of training and test loss values may be explained by them being computed on normalized targets. Since the normalization statistics can be expected to vary highly between the single objects, the normalized loss of an equally well performing model may also vary. The claim that normalized loss values do not consistently measure model performances on different datasets is supported by the observations on `cylinder07`: While its normalized loss values are the highest among the six objects, its performance judged by the visualizations in Figure A.3 is practically perfect. Mean deformation and stress errors are computed on the unnormalized targets, however, the different geometric dimensions of objects and different stress ranges during their trajectories (Figures A.2–A.7) result in the variance of these errors in Table 5.1.

Model with undeformed inputs: With undeformed mesh and closed gripper positions available from our pre-processing, we proposed changing the input state to encode these undeformed positions, instead of the first and second frames from the simulated trajectory (Section 4.3.1). The training results show slower convergence (Figure A.8) and higher training and test loss values (Table 5.3) compared to the baseline models trained on single objects. We interpret this difference is rooted in that encoding the first two simulation frames hints the model to some extent about the resulting deformation, and lets the model extrapolate the deformation between the first frames to infer the deformation at force. The convergence of the models trained on undeformed inputs shows that the architecture is suitable for learning the physical dynamics. Inspecting the visualization of the prediction compared to ground truth of the `lemon01` model trained on undeformed inputs (Figure 5.3) shows that even predictions with higher loss values are accurate in predicting stresses, and accurate in predicting the rough deformation of the object, though the coarse texture on the surface remains. The models trained with undeformed inputs are fully differentiable w.r.t. its inputs, e.g. the grasp pose.

Model learning tetrahedral stress: Our second proposed modification to the model involved the output side: We introduced tetrahedral features to the message passing, which are decoded into tetrahedral stress predictions (Section 4.3.2). This allows learning the tetrahedral VON MISES stresses $\sigma_{v,i}$ as output by *DefGraspSim*, instead of learning averaged stress measures $\bar{\sigma}_{v,i}$ at nodes.
The results obtained with models learning tetrahedral stress on single objects were satisfactory. Compared to the baseline models, the training and test losses converged similarly fast (Figure A.9). The training and test loss values, captured at the best checkpoints, as

shown in Table 5.4 are consistently lower than the ones of the baseline models trained on the same objects (Table 5.2). This good performance was not wholly expected. The meshes of the objects trained on have approximately four times as many tetrahedra as they have nodes, as shown in Table A.2. Thus, the number of tetrahedral stress values the modified model has to predict is also around four times higher than the number of predicted node stress values of the baseline model. We did not construct meaningful features for tetrahedra, yet only from the modified message passing phase, where tetrahedral features accumulated information from its nodes, the tetrahedral stress could be learnt with high accuracy. We hypothesize that the reason might be rooted in the higher similarity of the modified message passing with FEM, where the stress tensor of a tetrahedral element is computed based on the deformations of its nodes. The implemented message passing rule for tetrahedrons may implicitly learn this process in its update MLP. Finally, we note that additionally to the improved training performance, learning tetrahedral stresses is physically more accurate than learning the average node stresses. This can be underlaid by a comparison of visualized ground truth node stress (e.g. in Figure 5.3) to ground truth tetrahedral stress (Figure 5.4). The node stress field looks more blurry compared to the tetrahedral stress field, whose values can differ significantly even in neighboring tetrahedral elements. Especially for objects meshed in low resolution or with fine details, the assumption that the averaged node stress measures $\bar{\sigma}_{v,i}$ would be physically accurate is violated. Models training on such inaccurate ground truth can not be expected to fully learn sensible physical dynamics.

Finally, we attempt to explain the noise observed on predicted deformations on several trained models. This apparent noise, which in visualizations looks like a rough surface of the predicted deformed object, compared to the rather smooth surface of the ground truth deformed object, was observed in the rollout of the large model on `strawberry01` and `cucumber01`, as seen in Figure 5.2. For the models trained on undeformed inputs, it was observed in outputs of the models trained on `strawberry01` (Figure A.7) and `lemon01`. The model trained with undeformed inputs on `lemon01` (Figure 5.3) also showed this noise. The model learning tetrahedral stress on `lemon01` showed less of such noise (Figure 5.4).
Examining several animated trajectories of these models, we observed that the noise is largest in trajectories where the object does not only deform in areas close to the gripper, but where parts of the object show a rigid body motion as response to the deformation close to the gripper. Rigid body motion means that points of tetrahedra translate or rotate in similar manner, without deforming relative to each other. For instance, this is the case in the top-most trajectory in Figure A.4, where only a small part of the lemon gets squeezed, making the lemon oscillate during the trajectory. On the geometric primitives

and e.g. `potato2` (Figure A.5), such behavior is not observed, and the predicted deformed object surface lacks this noise. We conclude that the implemented models have difficulty in learning the dynamics of these rigid body motions. The stress fields predicted by our models do not suffer from this problem, since the stress field is invariant to rigid body motion.
Interestingly, the models learning stress at tetrahedra performed slightly better, showing less noise on the surface, e.g. in Figure 5.4. It is conceivable that the construction of features for each tetrahedron, which accumulate information from its nodes features, helped the model to better relate low present stress to low local deformation.

7. Conclusion

In the scope of this work, we generated a large, diverse training dataset of resulting deformation and stress values in grasp experiments utilizing the *DefGraspSim* simulator. We implemented a complex GNN model along the *DefGraspNets* baseline by Huang et al. in PyTorch. We reconstructed the pre-processing routine, allowing training directly on self-generated simulation output files, which was not possible with the *DefGraspNets* model. Additionally, we proposed and implemented two modifications which face key issues present in Huang et al.s implementation: First, changing the input feature construction to encode the undeformed object geometry instead of the first two frames of a simulated trajectory allows trained models to infer on datapoints for which no simulation results are available. These models are fully differentiable w.r.t. its inputs, and a gradient-based grasp pose refinement could be implemented. Second, we extended the Encode-Process-Decode architecture to learn features for mesh tetrahedra, which lets the model learn tetrahedral stress values as output by FEM simulation, instead questionably averaging stresses at nodes.

We trained numerous models in baseline form and with our modifications on several different training objects, and a baseline model on 34 objects. We presented detailed performance metrics and visualized predictions and ground truth for a multitude of datapoints. We performed a comprehensive interpretation of these training results, investigating the capabilities of the baseline and modified models, and the influence of the choice of the training object to the performance. All trained models showed overall good performance on test objects and trajectories similar to the training objects, and show the capability of the GNN model. Our discussion revealed that rigid body motion of parts of objects is a weak point both in baseline and modified form.
The modified models training on undeformed input showed both that the baseline model depends on the simulation output, and that learning the physical dynamics is possible with this input state. Its full differentiability w.r.t. the input grasp pose could be used for gradient-based grasp pose optimization.
Our implementation learning tetrahedral stress showed both better performance than the

baseline as measured by the loss values, as well as handling rigid body motions slightly better. We attribute this to the modification in architecture, which by being informed about mesh tetrahedra becomes more similar to the FEM model of object deformation physics.

7.1. Future Work

We are confident that our re-implementation *TorchGraspNet* is easier to understand, use and extend than the original *DefGraspNets* model through clear documentation, renamed fields, methods and files and the removal of unused code, and included tools for the visualization of predictions. We hope for future research to leverage our implementation and are confident that it can significantly accelerate it. Based on our experiments and findings, we believe research in the following directions would be most promising.

First, additional training runs could be conducted. Our largest training run used only around half of the objects available in our dataset. Since already this took multiple days of compute, we only completed this run training on multiple objects and rolling out on unseen objects. Other than training on the entire dataset, interesting experiments would be to train on all primitive geometries and test on all fruit and vegetable 3D scans and vice versa. Of course, the dataset could also be extended with additional everyday objects. Additionally, our dataset includes only isotropic objects with equal properties E, ν and ρ. The model in its implemented form is not informed about these material properties. Generating a dataset with variance in this domain, and extending the model to encode material properties would be viable.

Second, we found that the normalized MSE loss is suitable to learn the network, but lacks interpretability and due to different normalization statistics, the performance of models trained on different datasets can not be compared easily. The MAE values of unnormalized deformation and stress also are influenced by different scale and meshing strategy between objects. Deriving an easily interpretable, intuitive loss term for deformation and stress would be helpful.

Third, we identified rigid body motion of parts of objects to be a weakness of the implemented model in its current form. Addressing this in further research would be very interesting. Extending the GNN architecture with tetrahedral features already showed slight improvement in this domain. Starting points might be to explicitly inform nodes

about their neighbors movement in the message passing phase, and adjust their movement depending on the predicted stress. In areas with high stress, deformation of tetrahedra would be allowed, while in areas with no stress the tetrahedron should at most show rigid body movement. This relation may be possible using physics-based loss functions. For instance, the difference between predicted stress and the stress consistent with the deformation of its points could be placed as additional loss term on each tetrahedral element. The consistency of solutions predicted by neural networks with the physical model of the underlying process is an active research field named physics-informed neural network (PINN) [46, 47], and the models presented in this thesis could be refined with methods used in PINNs.

Fourth, it would be interesting whether the model can be altered to predict not the final deformed state of an object grasped with force, but to let it predict the deformed state at a next timestep. This could enable the model to be used in a model-based RL approach, where an agent could learn careful handling of deformable objects, with negative reward for excessive object deformations or stresses. This question was the initial aim of this thesis. Finally, of course, a validation of predicted deformations with experiments involving a real robot (sim-to-real) would be interesting.

Bibliography

[1] Kilian Kleeberger et al. “A survey on learning-based robotic grasping.” In: *Current Robotics Reports* 1 (2020), pp. 239–249.

[2] Liman Wang and Jihong Zhu. “Deformable object manipulation in caregiving scenarios: A review.” In: *Machines* 11.11 (2023), p. 1013.

[3] Rinto Yagawa et al. “Learning Food Picking without Food: Fracture Anticipation by Breaking Reusable Fragile Objects.” In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 917–923.

[4] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO].

[5] Isabella Huang et al. “DefGraspSim: Physics-Based Simulation of Grasp Outcomes for 3D Deformable Objects.” In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 6274–6281. DOI: 10.1109/LRA.2022.3158725.

[6] Giuseppe Carleo et al. “Machine learning and the physical sciences.” In: *Reviews of Modern Physics* 91.4 (2019), p. 045002.

[7] Isabella Huang et al. *DefGraspNets: Grasp Planning on 3D Fields with Graph Neural Nets*. 2023. arXiv: 2303.16138 [cs.RO].

[8] Tobias Pfaff et al. *Learning Mesh-Based Simulation with Graph Networks*. 2021. arXiv: 2010.03409 [cs.LG].

[9] Jack Collins et al. “A review of physics simulators for robotic applications.” In: *IEEE Access* 9 (2021), pp. 51416–51431.

[10] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control.” In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[11] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016.

[12] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator.” In: *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. Ieee. 2004, pp. 2149–2154.

[13] E. Rohmer, S. P. N. Singh, and M. Freese. “CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework.” In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[14] A. Bicchi and V. Kumar. “Robotic grasping and contact: a review.” In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 348–353 vol.1. DOI: 10.1109/ROBOT.2000.844081.

[15] Andrew T Miller and Peter K Allen. “Graspit! a versatile simulator for robotic grasping.” In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122.

[16] Beatriz León et al. “Opengrasp: a toolkit for robot grasping simulation.” In: *Simulation, Modeling, and Programming for Autonomous Robots: Second International Conference, SIMPAR 2010, Darmstadt, Germany, November 15-18, 2010. Proceedings 2*. Springer. 2010, pp. 109–120.

[17] Asad Ali Shahid et al. “Learning continuous control actions for robotic grasping with reinforcement learning.” In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2020, pp. 4066–4072.

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

[19] Sam Kriegman et al. *Scalable sim-to-real transfer of soft robot designs*. 2019. arXiv: 1911.10290 [cs.RO]. URL: https://arxiv.org/abs/1911.10290.

[20] Jacky Liang et al. “Gpu-accelerated robotic simulation for distributed reinforcement learning.” In: *Conference on Robot Learning*. PMLR. 2018, pp. 270–282.

[21] Youngsung Son, Hyonyong Han, and Joonmyun Cho. “Usefulness of using Nvidia IsaacSim and IsaacGym for AI robot manipulation training.” In: *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*. 2023, pp. 1725–1728. DOI: 10.1109/ICTC58733.2023.10393380.

[22] Shangding Gu et al. “A review of safe reinforcement learning: Methods, theory and applications.” In: *arXiv preprint arXiv:2205.10330* (2022).

[23] Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. “Deep learning for physical processes: Incorporating prior scientific knowledge.” In: *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124009.

[24] Xiaoxiao Guo, Wei Li, and Francesco Iorio. “Convolutional neural networks for steady flow approximation.” In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 481–490.

[25] Saakaar Bhatnagar et al. “Prediction of aerodynamic flow fields using convolutional neural networks.” In: *Computational Mechanics* 64 (2019), pp. 525–545.

[26] Alvaro Sanchez-Gonzalez et al. “Graph Networks as Learnable Physics Engines for Inference and Control.” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 4470–4479. URL: https://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

[27] Benjamin Ummenhofer et al. “Lagrangian fluid simulation with continuous convolutions.” In: *International Conference on Learning Representations*. 2019.

[28] Zili Wang et al. “Towards high-accuracy axial springback: Mesh-based simulation of metal tube bending via geometry/process-integrated graph neural networks.” In: *Expert Systems with Applications* 255 (2024), p. 124577.

[29] Barbara Frank et al. “Learning object deformation models for robot motion planning.” In: *Robotics and Autonomous Systems* 62.8 (2014), pp. 1153–1174.

[30] Veronica E Arriola-Rios and Jeremy L Wyatt. “A multimodal model of object deformation under robotic pushing.” In: *IEEE Transactions on Cognitive and Developmental Systems* 9.2 (2017), pp. 153–169.

[31] Haochen Shi et al. “RoboCraft: Learning to see, simulate, and shape elasto-plastic objects in 3D with graph networks.” In: *The International Journal of Robotics Research* 43.4 (2024), pp. 533–549.

[32] Tran Nguyen Le et al. “Deformation-aware data-driven grasp synthesis.” In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3038–3045.

[33] Seungyeon Kim et al. “DSQNet: a deformable model-based supervised learning algorithm for grasping unknown occluded objects.” In: *IEEE Transactions on Automation Science and Engineering* 20.3 (2022), pp. 1721–1734.

[34] Matthias Müller et al. “Real Time Physics Class Notes.” In: *SIGGRAPH ’08: ACM SIGGRAPH 2008 classes*. Aug. 2008, pp. 1–90. DOI: 10.1145/1401132.1401245.

[35] Daniel Weber, Stephanie Ferreira, and Johannes Mueller-Roemer. *Physikalisch-basierte Simulation und Animation, Lecture 06 – Physik deformierbarer Körper*. June 2023.

[36] Daniel Weber, Stephanie Ferreira, and Johannes Mueller-Roemer. *Physikalisch-basierte Simulation und Animation, Lecture 09 – Finite Elemente Methode für deformierbare Körper*. June 2023.

[37] Jonathan Shewchuk. "What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint)." In: *University of California at Berkeley* 2002 (2002).

[38] Timothy J Baker. "Mesh generation: Art or science?" In: *Progress in aerospace sciences* 41.1 (2005), pp. 29–63.

[39] Jonathan Richard Shewchuk. "Unstructured mesh generation." In: *Combinatorial Scientific Computing* 12.257 (2012), p. 2.

[40] SH Lo. "Finite element mesh generation and adaptive meshing." In: *Progress in Structural Engineering and Materials* 4.4 (2002), pp. 381–399.

[41] Yixin Hu et al. "Fast Tetrahedral Meshing in the Wild." In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392385. URL: https://doi.org/10.1145/3386569.3392385.

[42] W. H. Yang. "A Generalized von Mises Criterion for Yield and Fracture." In: *Journal of Applied Mechanics* 47.2 (June 1980), pp. 297–300. ISSN: 0021-8936. DOI: 10.1115/1.3153658. eprint: https://asmedigitalcollection.asme.org/appliedmechanics/article-pdf/47/2/297/5878449/297_1.pdf. URL: https://doi.org/10.1115/1.3153658.

[43] Stephen P Timoshenko and James Norman Goodier. *Theory of elasticity*. Vol. 3. McGraw-hill New York, 1982.

[44] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. "A billion ways to grasp: An evaluation of grasp sampling schemes on a dense, physics-based grasp data set." In: *The International Symposium of Robotics Research*. Springer. 2019, pp. 890–905.

[45] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[46] Salvatore Cuomo et al. "Scientific machine learning through physics–informed neural networks: Where we are and what's next." In: *Journal of Scientific Computing* 92.3 (2022), p. 88.

[47] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.” In: *Journal of Computational physics* 378 (2019), pp. 686–707.

A. Appendix

The appendix to this thesis contains detailed tables and figures for further reference.

A.1. Generated Dataset and DefGraspSim Output Format

Object Class	Type	Number of Objects
`06polygon`	Geometric primitive	8
`08polygon`	Geometric primitive	8
`annulus`	Geometric primitive	8
`apple`	3D scan	3
`cuboid`	Geometric primitive	6
`cucumber`	3D scan	1
`cup`	Geometric Primitive	6
`cylinder`	Geometric primitive	8
`eggplant`	3D scan	1
`ellipsoid`	Geometric primitive	4
`lemon`	3D scan	4
`potato`	3D scan	3
`sphere`	Geometric primitive	6
`strawberry`	3D scan	3
`tomato`	3D scan	2
Total	71	

Table A.1.: Tetrahedral mesh objects available to *DefGraspSim* as provided by Huang et al.

Object	Number of Nodes	Number of Tetrahedra
8polygon06	1389	6152
cylinder07	925	3658
lemon01	1309	5163
potato2	1429	6494
sphere03	1393	6096
strawberry01	1126	4851

Table A.2.: Objects used in training for validation of baseline and modified models.

Key	Interpretation
initial_desired_force	Desired force to grasp with
pre_contact_positions	3D position of nodes in undeformed mesh
pre_contact_se	Strain energy over the entire undeformed mesh
pre_contact_stresses	Von Mises stress of each tetrahedron in undeformed mesh
left_contacted_nodes_under_gravity_initial	Nodes first in contact with left finger
right_contacted_nodes_under_gravity_initial	Nodes first in contact with right finger
squeeze_no_gravity_max_force	Maximum force achieved in grasp
stacked_forces	Squeezing force at given frame
stacked_forces_on_nodes	Force on each node at given frame
stacked_gripper_positions	Translation of each gripper along its normal at given frame
stacked_positions	3D position of nodes in deformed mesh
stacked_left_node_contacts	Mesh type category each gripper node is in contact with
stacked_right_node_contacts	Mesh type category each gripper node is in contact with
stacked_left_gripper_contact_points	Positions of nodes in contact with left gripper finger
stacked_right_gripper_contact_points	Positions of nodes in contact with right gripper finger
stacked_stresses	Von Mises stress of each tetrahedron in deformed mesh

Table A.3.: Data generated in one simulated `squeeze_no_gravity` experiment in *DefGraspSim*.

A.2. *DefGraspNets* Pre-processed Input Data

Key	Interpretation
name	Name of 3D object grasped on
cutoffs	Parameter for rejecting trajectories in which contact is lost?
cells	List of tetrahedron cells in object and gripper mesh.
node_type	Node type category
node_mod	Youngs modulus E at each node
tfn	6D transformation of gripper from neutral position
mesh_pos	3D position of nodes of undeformed mesh
mesh_edges	Edges in object and gripper meshes
world_edges	Edges between close object and gripper nodes
force	Force gripper closes with
gripper_pos	Distance of gripper fingers along their closing direction
world_pos	3D position of nodes of deformed mesh
stress	Von Mises stress given at node positions
pd_stress	Alternative computation of nodal stress

Table A.4.: Fields present in a *DefGraspNets* pre-processed data point.

A.3. Training Process and Results

This appendix contains additional content describing the results of the trained models.

6polygon02
6polygon03
6polygon08
6polygon05
8polygon01
8polygon04
8polygon07
annulus03
annulus04
annulus06
apple1
apple2
cuboid01
cuboid04
cuboid03
cup04
cup01
cup06
cylinder02
cylinder05
cylinder07
ellipsoid02
ellipsoid03
lemon01
lemon02
potato1
potato3
sphere01
sphere03
sphere05
sphere06
strawberry02
strawberry03
tomato1

Table A.5.: Objects in the training dataset for the large training run of the baseline model.

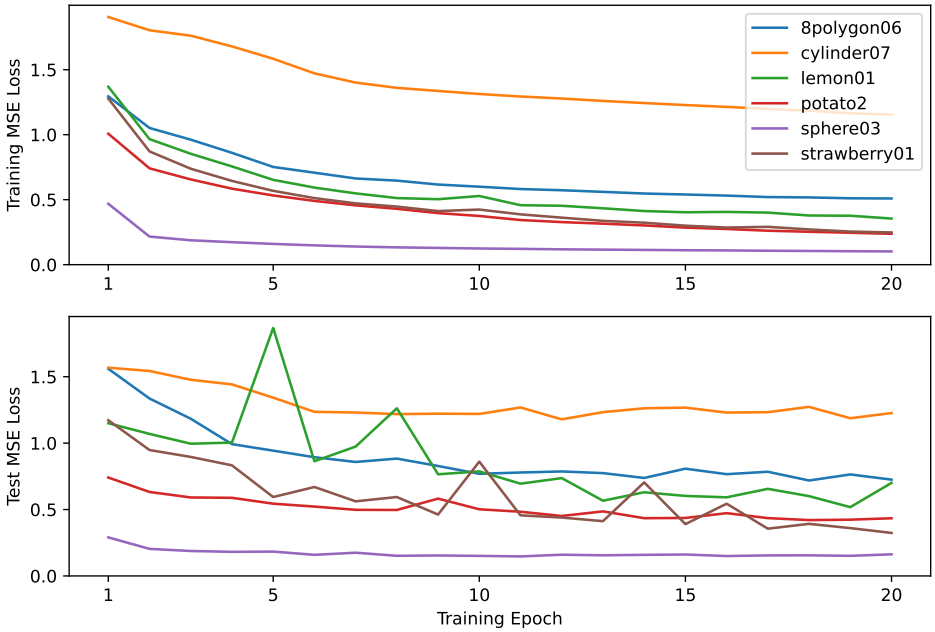

Figure A.1.: Mean loss values for training and test set over training epochs for baseline models trained on a single object each.

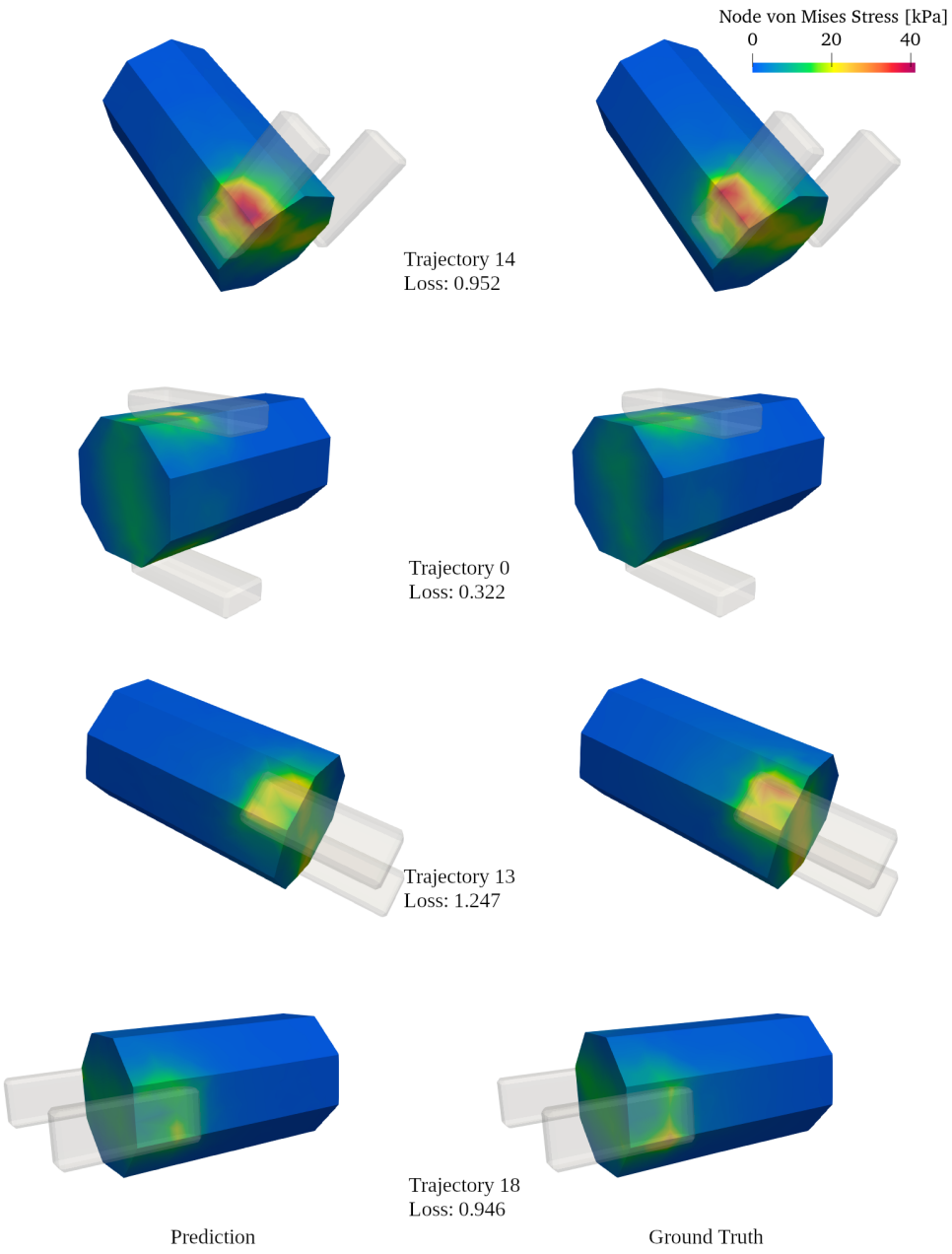

Figure A.2.: Predicted and ground truth deformation and stress of selected test trajectories for the baseline model trained on `8polygon06`.

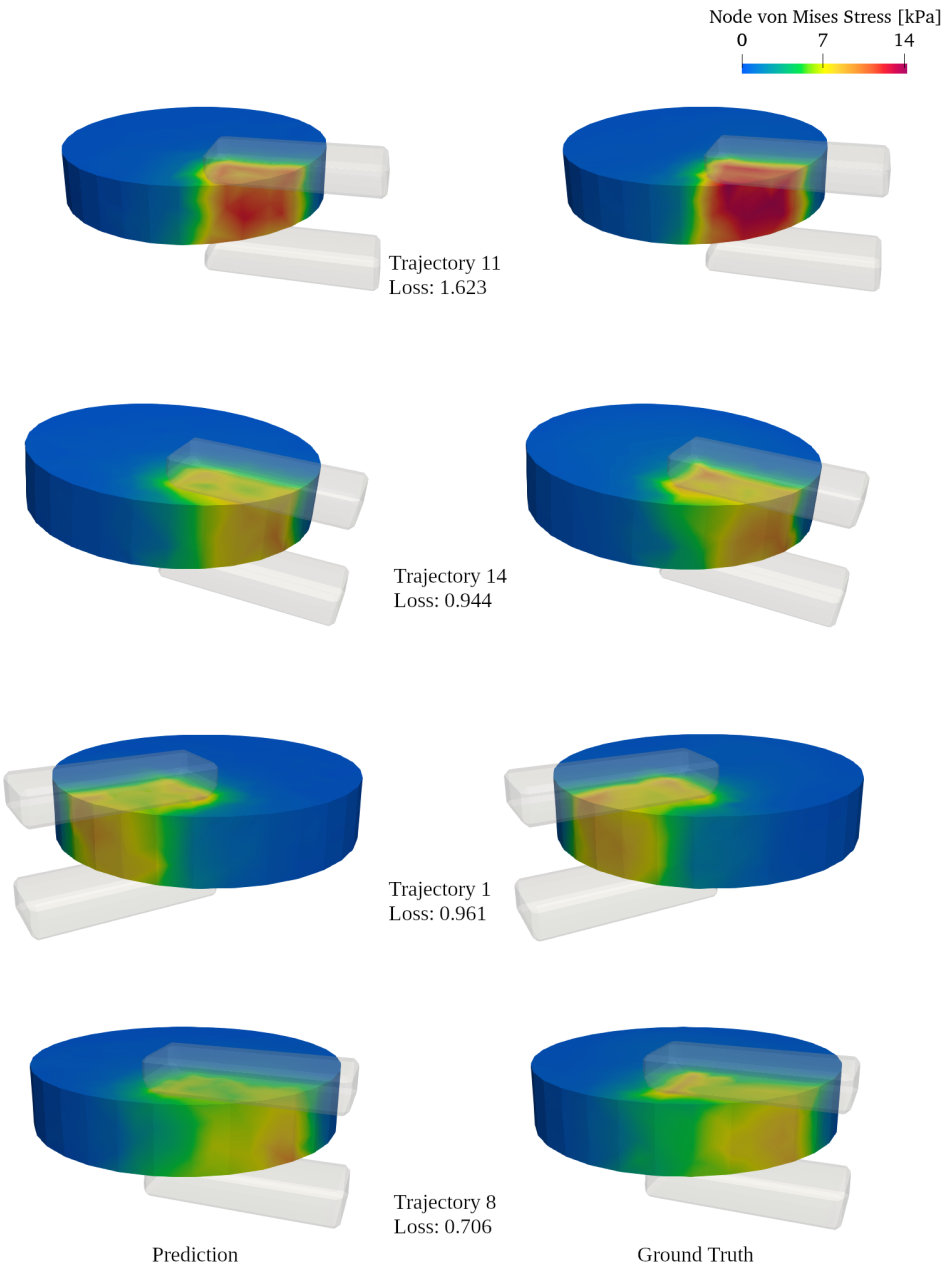

Figure A.3.: Predicted and ground truth deformation and stress of selected test trajectories for the baseline model trained on `cylinder07`.

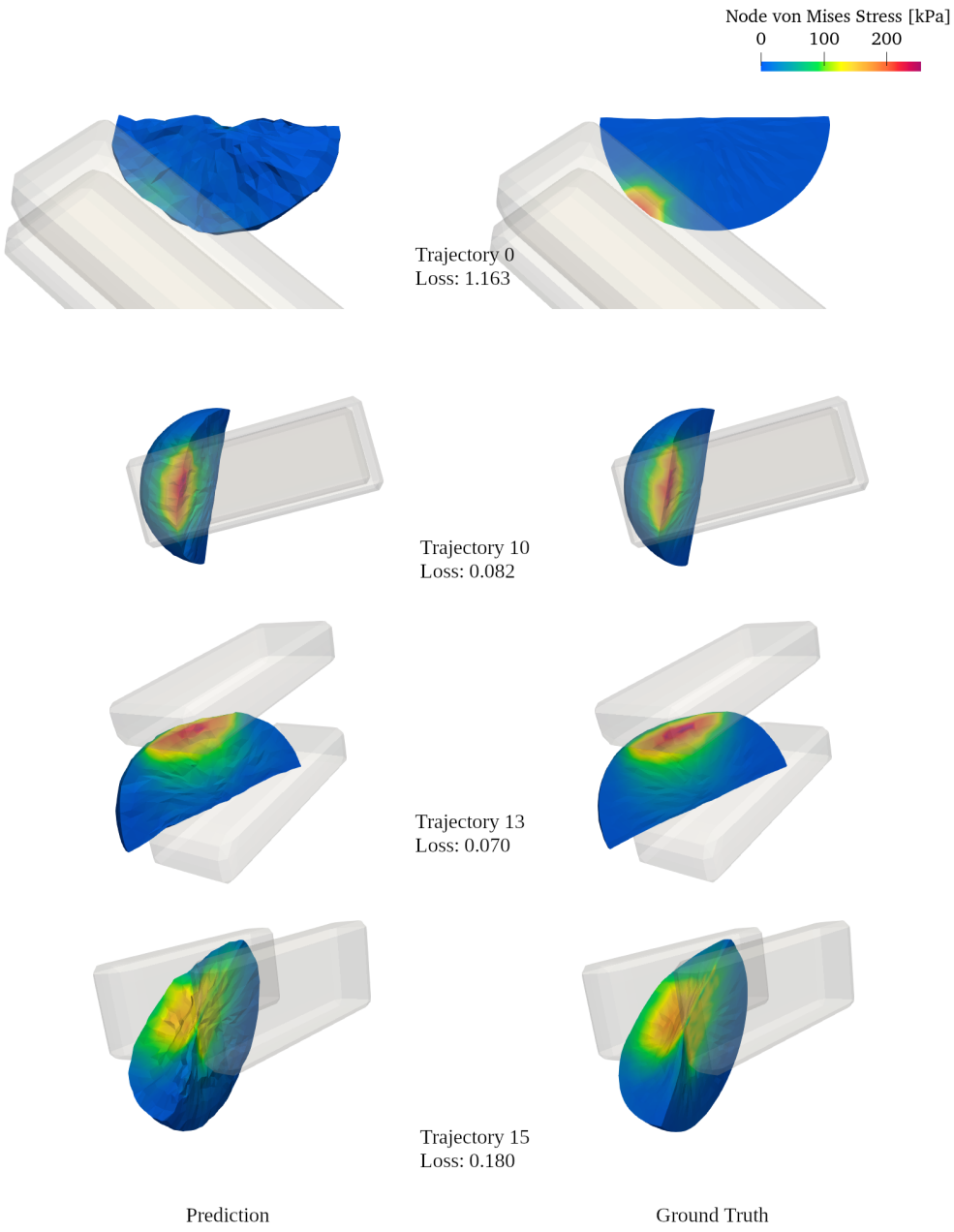

Figure A.4.: Predicted and ground truth deformation and stress of selected test trajectories for the baseline model trained on `lemon01`.

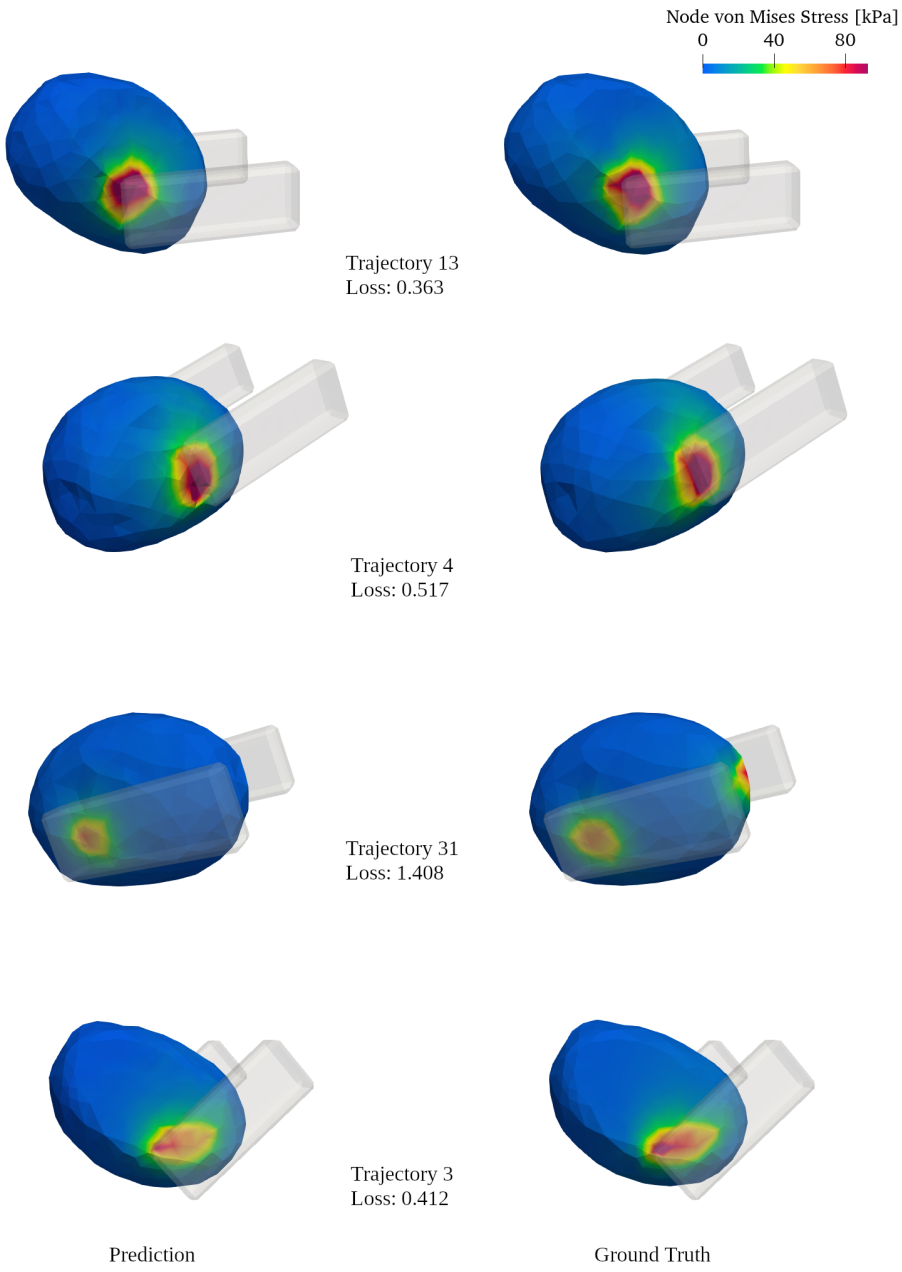

Figure A.5.: Predicted and ground truth deformation and stress of selected test trajectories for the baseline model trained on `potato2`.

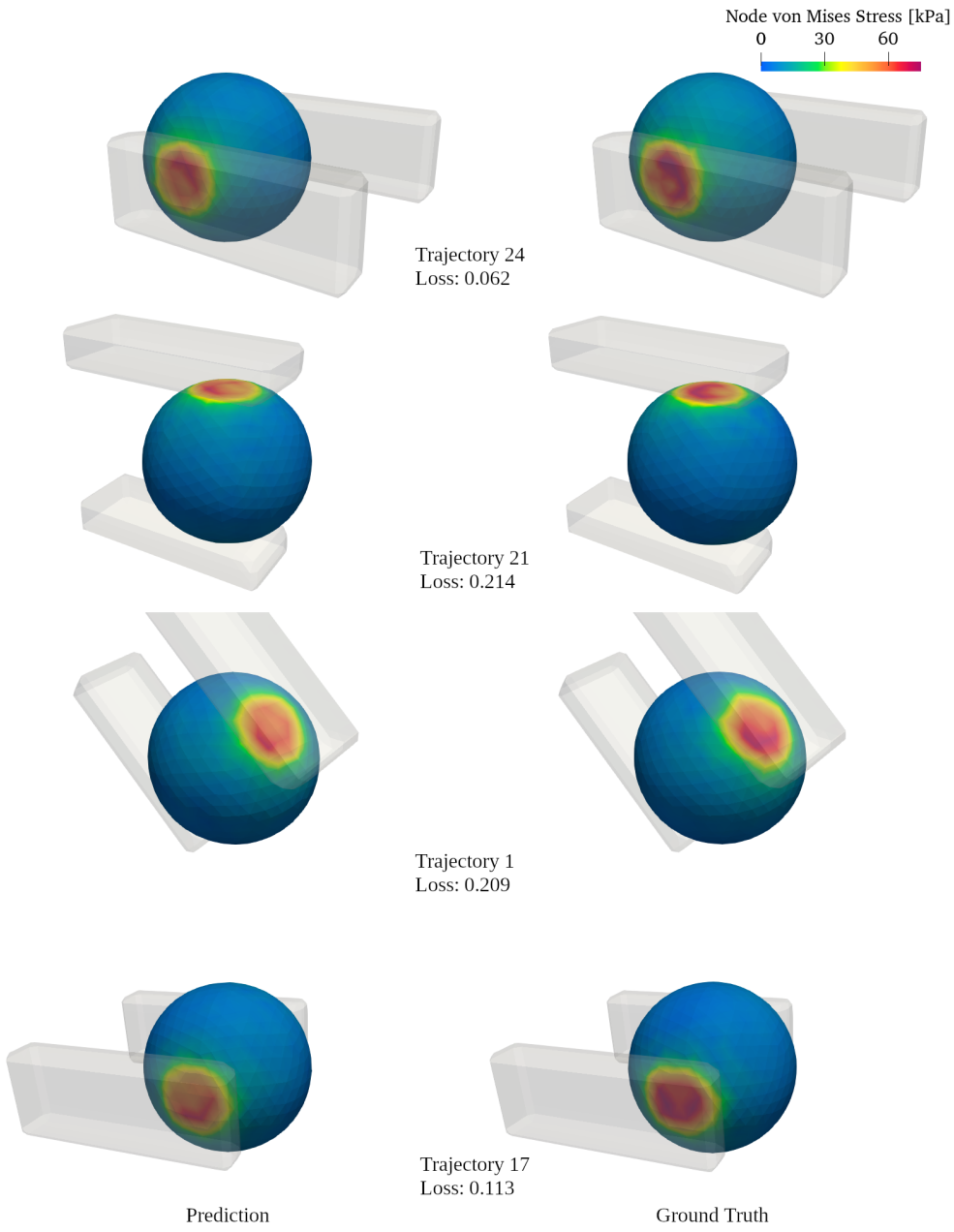

Figure A.6.: Predicted and ground truth deformation and stress of selected test trajectories for the baseline model trained on `sphere03`.

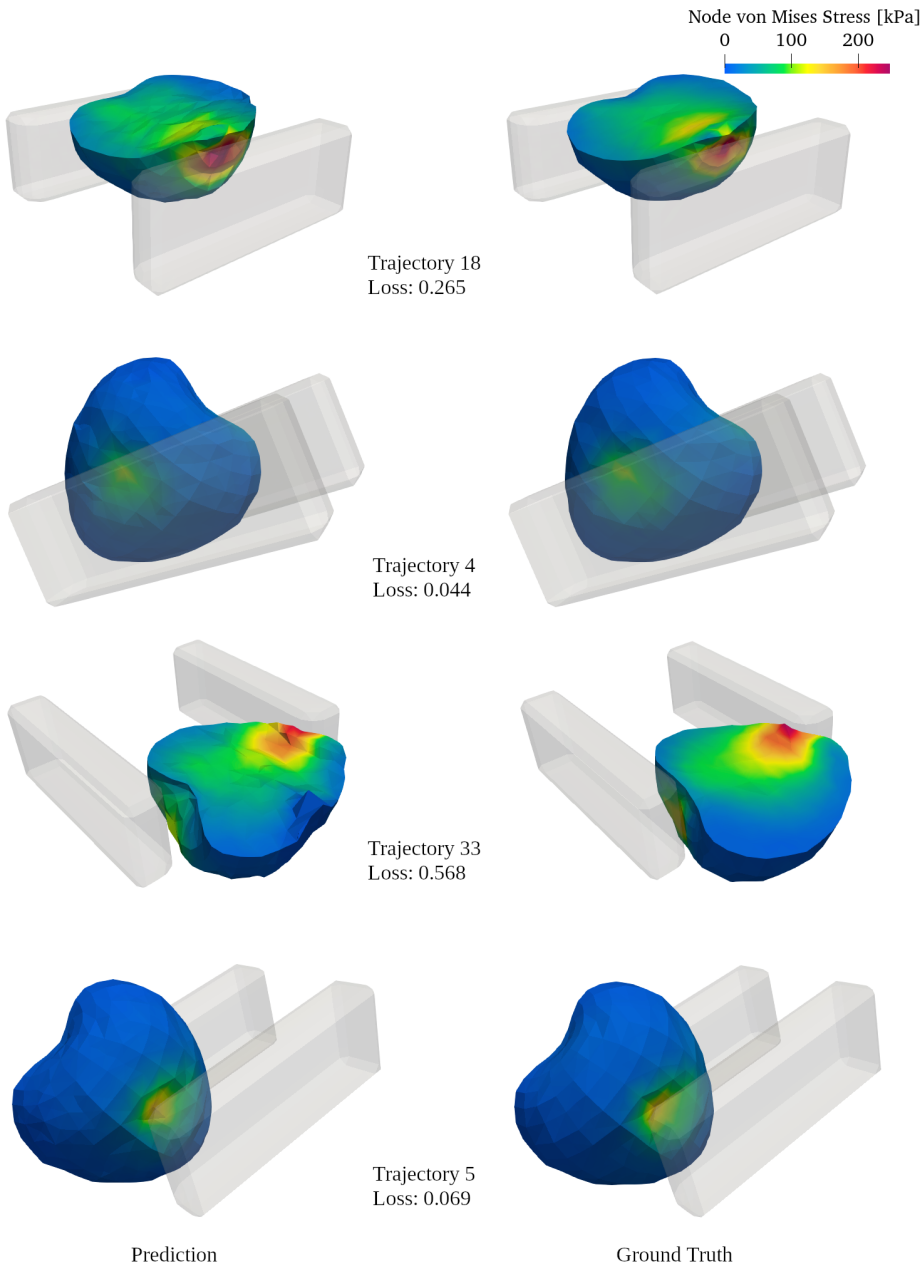

Figure A.7.: Predicted and ground truth deformation and stress of selected test trajectories for the baseline model trained on `strawberry01`.

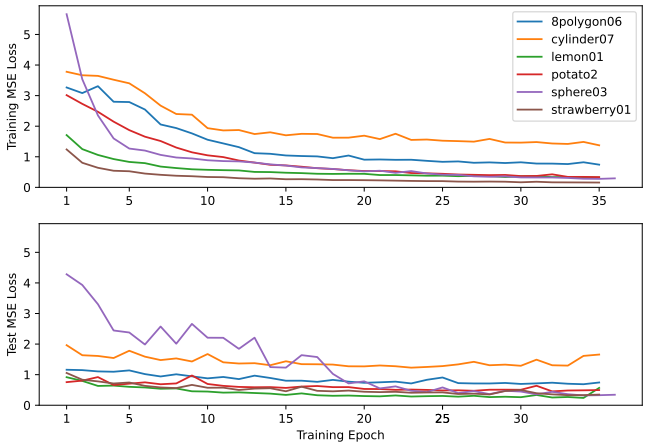

Figure A.8.: Mean loss values for training and test set over training epochs for models with undeformed input trained on a single object each.

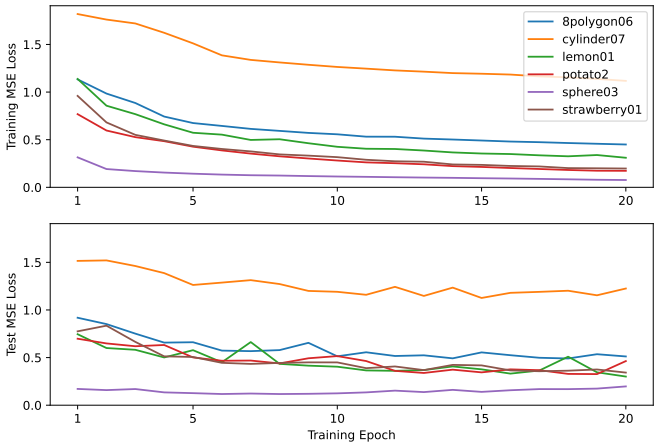

Figure A.9.: Mean loss values for training and test set over training epochs for models learning tetrahedral stress trained on a single object each.