Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

ML-based Visual Analysis of Droplet Behaviour in Multiphase Flow Simulations

Moritz Heinemann

Course of Study:	Simulation Technology
Examiner:	Prof. Dr. Thomas Ertl
Supervisor:	Dr. Steffen Frey Alexander Straub, M.Sc. Gleb Tkachev, M.Sc. Dr. Sebastian Boblest
Commenced:	January 12, 2018
Completed:	July 12, 2018

Abstract

Modern multiphase flow solvers can simulate flows with increasing domain size and precision. This produces large simulation results which need to be analyzed, and to this end visualized. Because of the amount of data, classical visualization approaches become more and more unfitting. Therefore, it is hard to find interesting regions because of visual clutter which is produced by too much data. One solution could be semi automatic assistance systems to support the observer of the visualization.

Over the last years, machine learning has grown to a widely researched area. Development not only brought many use cases in research and industry, but highly sophisticated programming frameworks. This makes it much easier to use machine learning in a wide area of applications, such as visualization.

In this work we are interested in analyzing multiphase simulations with thousands of droplets. We use machine learning to train artificial neural networks with the droplet data gained from simulations. These trained models are used for finding interesting droplet behavior in the simulation, which is then visualized. Our trained models can predict the development of physical properties and quantities over time, and therefore errors in prediction can guide us to areas of interest which then can be investigated further.

The prediction error is visualized as colored dots directly within the 3D simulation dataset using ParaView. Additionally we can plot the properties and their predictions of single droplets over time and show the prediction error separated by property within a spider chart. Finally we show the results, which cover an evaluation of the learning process and an analysis of the used datasets with our method, as well as give an outlook on possible improvement in future work.

Contents

List of Figures

List of Algorithms

List of Abbreviations

CFD computational fluid dynamics. 15

DNS direct numerical simulation. 15

FS3D Free Surface 3D. 15

ML machine learning. 13

PLIC piecewise linear interface calculation. 15

QCQP quadratically constrained quadratic program. 28

RANS Reynolds-averaged Navier Stokes. 15

VOF Volume of Fluid. 15

1 Introduction

With increasing computational power of modern computers, simulations have grown larger, in both spatial and temporal resolution, as well as more detailed and accurate. Therefore, large amounts of data need to be analyzed. This is only possible with the help of visualization. But for growing data sizes and with time-dependency within the data, the analysis becomes more and more complex and tedious. Machine learning could be one solution to automate parts of this task.

This work concentrates on multiphase flow simulations. We analyze the behavior of individual droplets within its surrounding fluid. The data comes from two multiphase flow simulations with thousands of droplets. They are tracked and observed over a time series. We calculate physical quantities of each droplet, like oscillation and rotation as well as surface deformation for each individual time step. The development of these droplet properties is used to train an artificial neural network, in order to predict the future behavior of this droplet. We then compare this prediction to the ground truth and visualize the error of the prediction.

The idea behind this is that we want to find interesting droplets or regions within the flow simulation. By interesting droplets we are thinking of droplets that behave different than the average droplet. Our assumption is that the artificial neural network would learn how the droplets behave on average and therefore predicts this average behavior. The droplets where the prediction error is high seems to not behave like the average droplet and therefore are interesting. With this we can use the prediction error as scale of how interesting a droplet would be for further analysis. Additionally we can visualize the individual error for each of the droplet properties, to give the viewer further insight in how exactly the droplet does not behave like an average droplet.

1.1 Structure

This thesis is structured as follows: **Chapter 2 - Related Work** gives an overview over the related work. This includes work we directly set up on, as well as a short description of similar work. Next we introduce basic topics which are used in this work in **Chapter 3 - Basics and Theory**. We provide information about the datasets and where they come from, as well as a short introduction to machine learning (ML) and artificial neural networks. After this we present our methods in **Chapter 4 - Methods and Implementation**. Starting with droplet processing, to training of the artificial neural network. This is followed by a discussion and our results in **Chapter 5 - Results**. This includes an evaluation of the learning process, as well as providing visualizations. Finally this thesis is completed with **Chapter 6 - Conclusion and Outlook**.

2 Related Work

The computational fluid dynamics (CFD) solver Free Surface 3D (FS3D) [EEG+16] is used for multiphase flow simulations. Two of the simulations of this tool are used within this work. FS3D uses direct numerical simulation (DNS) to solve the incompressible Navier-Stokes equation. It is based on the Volume of Fluid (VOF) method, which was first described by Noh and Woodward [NW76], as well as Hirt and Nichols [HN81]. For the reconstruction of the phase interfaces within the VOF method piecewise linear interface calculation (PLIC) [You82] [You84] [RK98] is used. Additionally, many other complex phenomena, like freezing and evaporation can be simulated. The code base is optimized for massive parallel architectures. Further improvement was done by Liu and Bothe [LB16] to stabilize lamella structures which occur in simulations of binary water droplet collisions. This was needed for the second of the two datasets we use in this work.

For visualization of multiphase flow simulations, the basic idea is to show the interface between the different phases. This can be done with classical isosurface techniques, for example the marching cubes algorithm [LC87]. Karch et al. [KSM+13] present a visualization technique based on PLIC, which is the same method used by FS3D with the VOF method to calculate the interfaces during simulation. Furthermore Karch et al. describe a generalization of PLIC to higher-order approximations with their framework.

For machine learning and artificial neural networks in general, there is a good introduction to this topic available by Kriesel [Kri07]. A deeper and more general overview to ML and deep learning in particular is the book *Deep Learning* from Goodfellow et al. [GBC16]. Also we like to mention the frameworks Tensorflow [MAP+15] and Keras [Cho+15] here as related work, which where used for implementation.

Much work was done by using ML in context of fluid simulations. Ling and Templeton [LT15] analyze different machine learning algorithms for finding regions of high uncertainty within Reynolds-averaged Navier Stokes (RANS) simulations. Training data is gained by comparing RANS simulations to results from DNS or large eddy simulations. The analyzed algorithms are support vector machines, Adaboost decision trees and random forests. Artificial neural networks are even used within solvers of the Navier Stokes equation [TSSP16] for acceleration of computation time. Other work does use ML to reconstruct a full pressure field from sparse measurement points within simulated flow around a cylinder [BLK13]. Yetilmezsoy and Saral [YS07] train a neural network to determine the collection efficiency of single droplets in countercurrent spray towers. Oliveira and Sousa [OS01] use a neural network to predict heat flux within air/water sprays, based on different parameters of the spray and the droplets within the spray. But in difference to other work they use experimental data and not fluid simulations.

3 Basics and Theory

3.1 Multiphase flow simulations data

In this work we use data from incompressible multiphase flow simulations computed with the solver FS3D [EEG+16]. The simulation output is a VOF [NW76] [HN81] field, as well as a velocity field, both stored on a rectilinear grid. A rectilinear grid has axis-aligned rectangular cells, with possibly different cell sizes along each axis. An example is shown in Figure 3.1.

For each cell, a VOF value f_c between 0 and 1 is given. A value of 1 means that the cell is completely filled with a phase, while a value of 0 means that this cell does not contain the phase. Values between 0 and 1 mean that there is an interface. Here, the value indicates the ratio of the volume phase to the whole cell. An example with water and air is shown in Figure 3.2.

PLIC is used to determine the surface between the two phases in a VOF field. This is needed during the simulation, to avoid numerical diffusion. But the phase interface can also be used in visualization.

The idea behind PLIC is to model the surface as a linear plane in each VOF cell, where the value f_c is $0 < f_c < 1$. As normal of the plane, the inverse gradient of the VOF field is used:

$$\mathbf{n} = -\frac{\nabla f(\mathbf{x}_c)}{\|\nabla f(\mathbf{x}_c)\|} \tag{3.1}$$

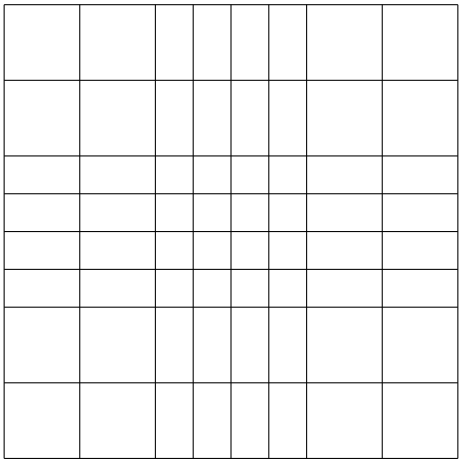

Figure 3.1: 2D example of a rectilinear grid with coarser cells in the outer regions and finer cells in the center.

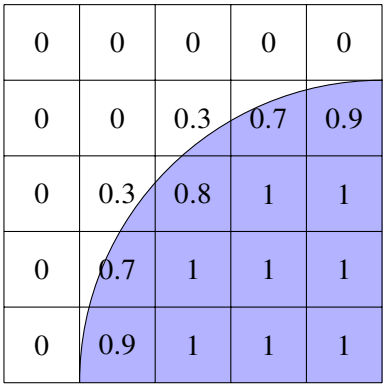

Figure 3.2: 2D Example of VOF data. Blue is water, white is air. Numbers show the approximate ratio of fluid to air.

Furthermore the plane is positioned, such that the volume enclosed between the cell boundaries and the plane has the same ratio to the total cell volume as the VOF value [KSM+13].

3.2 Used datasets

For machine learning, we need many input droplets, which we can use as training data. Therefore, we use two simulations, *jet* and *splash*, which both provide us with a lot of small droplets from atomization.

The *jet* dataset is a simulation of a jet of an aqueous solution with 0.3% Praestol 2500 within air. Temperature is at 20 °C and air pressure is 1 bar. The diameter of the jet nozzle is 0.25 cm und the average injection velocity is 5525 cm s^{-1} with a parabolic profile and maximum velocity of 8287.5 cm s^{-1}. Further characteristics are the Reynolds number of 3000 and the Ohnesorge number of 0.1. The simulation domain has a size of $10 \times 4 \times 4 cm^3$ with a grid of $1152 \times 384 \times 384$ cells. The cells are smaller at the center of the domain along the x-axis. We have 111 time steps of this dataset which are distributed over a duration of 2 ms. The size of this dataset is 562 GB. Renderings of selected time steps are shown in Figure 3.3.

The *splash* dataset is a simulation of a binary water droplet heads-on collision within air. The problem was symmetric, therefore only half of the domain was simulated. The initial velocity of the droplets is 1000 cm s^{-1}. The Weber number is 803. For this simulation the FS3D solver needed further improvements to stabilize the lamella structures of the fluid within the simulation [LB16]. The size of the half domain is $0.6 \times 0.6 \times 0.075 cm^3$ on grid of $512 \times 512 \times 64$ cells. The duration of this simulation covers 0.25 ms with 251 time steps. The size of this dataset is 126 GB. Renderings of selected time steps are shown in Figure 3.4.

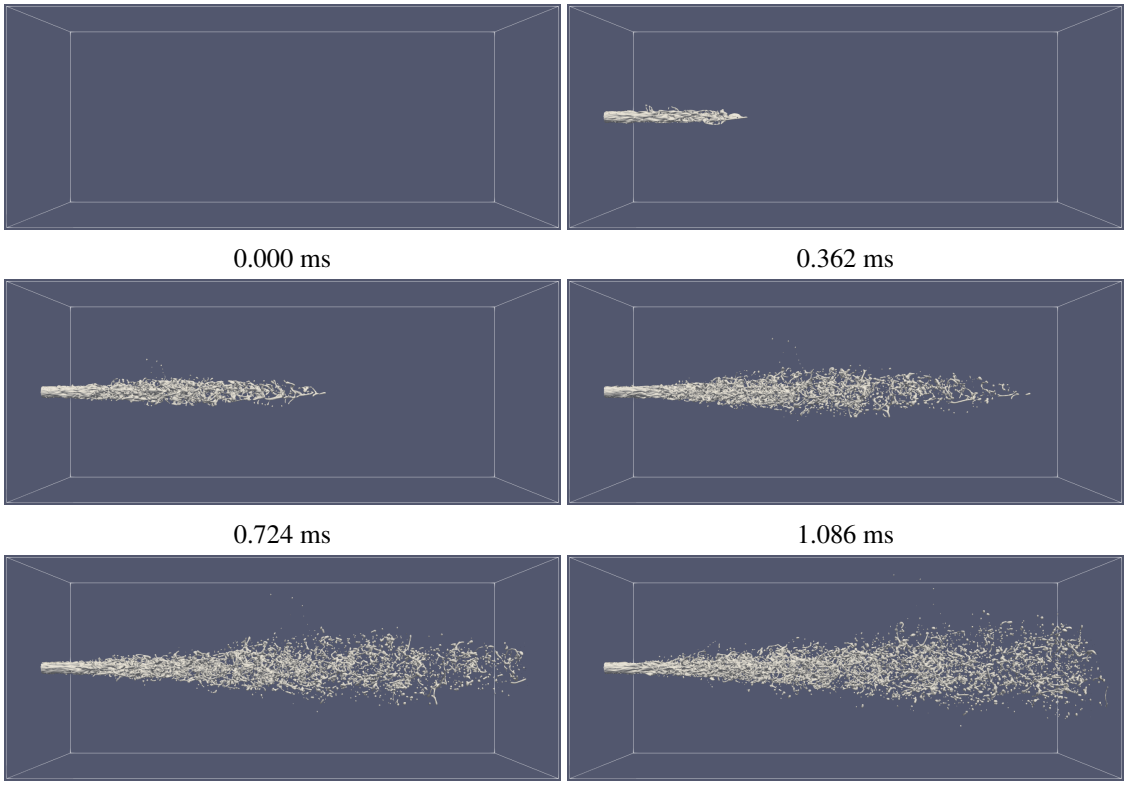

Figure 3.3: Overview of the jet dataset. Surface is calculated with the PLIC method. The time step is written below the frames.

3.3 Machine learning and artificial neural networks

An artificial neural network could be imagined as a mathematical function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ in a black box [Kri07, p. 7] [GBC16, p. 164]. This function has a given input vector $x \in \mathbb{R}^m$ and an output scalar or vector $y \in \mathbb{R}^n$. The input is the data the network should evaluate or analyze. This could be an image, measurement results or any other kind of data. The output is the result the network should give. If a classification problem is given, the output should return the corresponding class to the given input or usually the probability of each class. If a regression problem is given, the network should output a prediction of the data.

It is of course mostly impossible or at least very hard to quickly write down such a function f for the problem it should solve. That is where machine learning should start. The basic idea behind this is to use a lot of data samples, later called training data, show it to the black box function f, and within the black box the function should learn to interpret the data.

Before we take a look into the black box and what exactly happens during learning, we already can distinguish three different types of learning, by looking at what data is provided. *Unsupervised learning* does mean we only have sample data for the input vectors. The network should then find useful output by itself, in example classes of similar data. The second method is called *reinforcement*

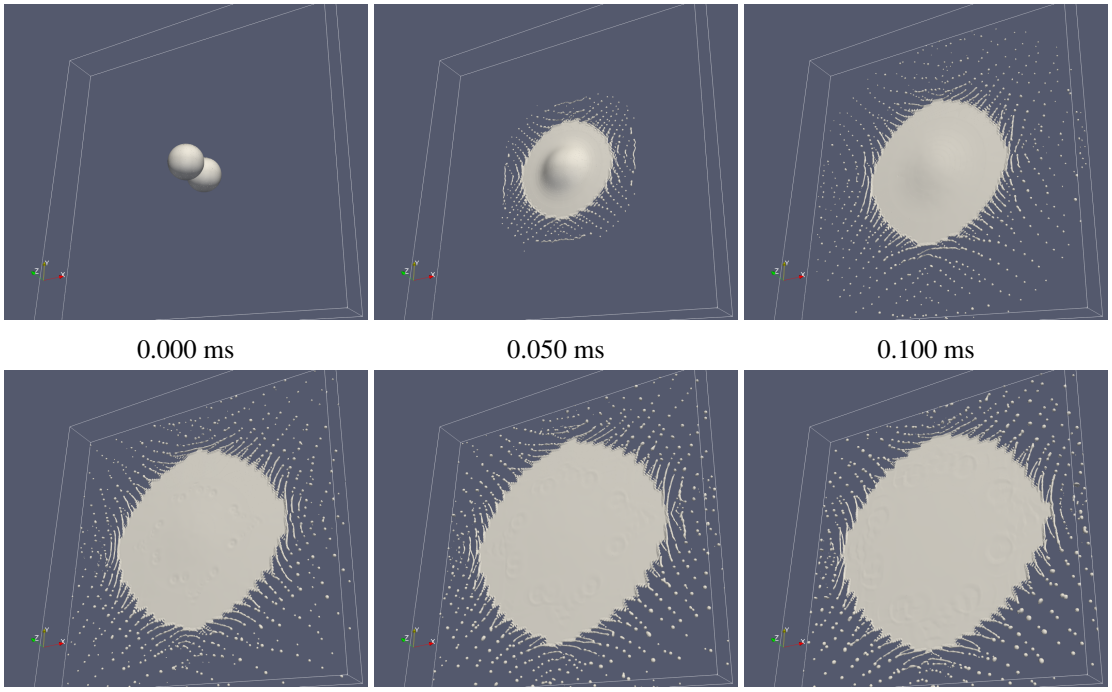

Figure 3.4: Overview of the slash dataset. Surface is calculated with the PLIC method. The time step is written below the frames.

learning. This means additionally to the input vector samples, a validation of the output is given. If the neural network calculates an output to the input, the validation can tell if this output is good or bad. The third method is *supervised learning*. There we have corresponding output sample data to all input samples. In this work we only use supervised learning. In the next sections we first describe how an artificial neural network looks and afterwards how it learns.

3.3.1 Artificial neural networks

As the name artificial neural network already suggests, this is inspired by nature from biological neural networks. An artificial neural network is a set of neurons and weighted connections between those neurons. Formal this could be written as triple (N, V, w), with the set of neurons N, the set of connections $V = \{(i, j)|i, j \in \mathbb{N}\}$ and weights of the connections $w : V \rightarrow \mathbb{R}$ [Kri07, p. 34]. An example is shown in Figure 3.6.

Next we describe the structure of a single neuron according to Kriesel [Kri07, p. 34 ff.]. A single neuron is build like in Figure 3.5. We see multiple input connections from other neurons, which are received with the so called propagation function. The propagation function takes the output values from the previous neurons and the connections weights and outputs a scalar value called network input. Very common is to use the weighed sums as propagation function [Kri07, p. 35]

$$net_j = \sum_{i \in I} (o_i w_{i,j}) \tag{3.2}$$

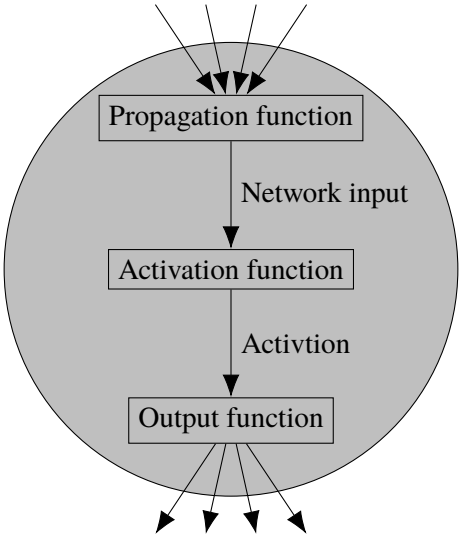

Figure 3.5: Structure of a single neuron according to Kriesel [Kri07, p. 35].

with current neuron j and I set of neurons which has a connection to neuron j with weight $w_{i,j}$ and output o_i.

The activation function models the reaction of the neuron. This is motivated by biological neurons, which gets activated when input signals reach a certain threshold. In general the activation function depends on the network input net_j, a threshold value Θ and the activation from the last time step, assuming discrete time. This could formal be written as [Kri07, p. 36]

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \Theta_j) \tag{3.3}$$

A very common choice [GBC16, p. 171] is to use the rectified linear unit (ReLU) [NH10] as activation function:

$$f_{ReLU}(x) = max\{0, x\}. \tag{3.4}$$

Note that is possible to model the threshold Θ as weight of an connection by inserting a so called bias neuron to the network which always outputs a value of one and has a connection to every other neuron. The weights of the connections from the bias neuron brings the threshold as input of the neuron and therefore the threshold is not explicitly included here.

After this the output function will determine the output value from the activation. It is very common to use the identity as output function [Kri07, p. 38], but sometimes it could be useful to use a different output function, for example to rescale the value range.

To build a complete artificial neural network, a set of neurons is used with connections in between them. You can think of an arbitrary graph. We can distinguish multiple networks by common topologies. One class are feed forward neural networks. They are organized in layers, with connections only going in one direction from one layer to the next layer. An example with three layers is shown in Figure 3.6. On the left, we see an input layer with three neurons, in the middle one hidden layer with four neurons and on the right an output layer with two neurons. In this example the layers are fully connected, which means, that every neuron from one layer is connected to every

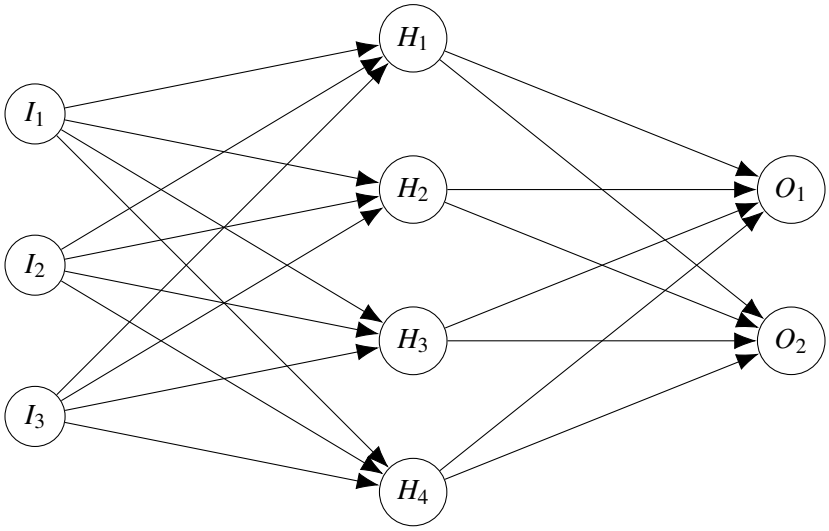

Figure 3.6: Example of an artificial neural network with an input layer, an output layer and one hidden layer in between. The layers are fully connected.

neuron on the next layer. There are more topologies, for example recurrent neural networks can have backwards connections, which leads to loops in the graph, but here we focus on feed forward networks.

As you see, we already gave the first and the last layer special names by naming them input and output layers. The input layer gets directly the input values. That means the input layer must have the same number of neurons as the data input vector is long. The input neurons directly use the input values as network input, without a propagation function. All output values from the output neurons, are the network output data vector.

3.3.2 Training

The learning process within ML is a very huge topic for itself. Therefore we only want to introduce very basic ideas and terms and refer the reader to further literature [GBC16] [Kri07].

Above the general structure of an artificial neural network was described. The knowledge of the neural network is within the weights of the connections between the neurons. Learning is the process in which we want to change the connection weights in order to get an optimal output from the network.

Now, we think of a neural network with random weights at the beginning. We see the network as a mathematical function, which maps one input vector to an output vector. As we are only looking at supervised learning, we know the correct output for each input vector. Therefore we can calculate an error of the network output to the correct output vector. When we change the weights of the connections the output of the network will probably change, and this will change the error of the

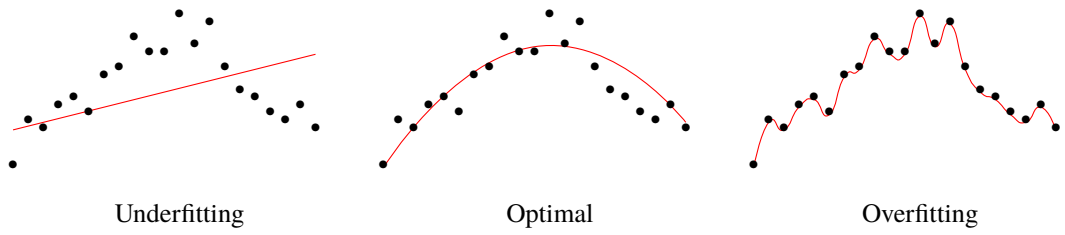

Figure 3.7: Example of underfitting and overfitting of datapoints (black) with a function (red).

network output. With this we can think of a mathematical error function with all weights of the network as parameter and the error value as output [Kri07]:

$$Err : W \mapsto \mathbb{R} \tag{3.5}$$

Learning will be the changing of weights to reduce the error. This could be done with gradient decent algorithms on this function, like the backpropagation of error. Gradient decent is an numerical optimization algorithm for finding the minimum of function f. The algorithm is an iterative process where we are going in each step from a point x_i in the direction of the negative gradient to find the next position

$$x_{i+1} = x_i - a\nabla f(x_i), \tag{3.6}$$

with step size a. For more details we refer to literature [Kri07] [GBC16].

We have multiple input data which can be used. Therefore within the learning process not only a single function like the gradient decent example above is optimized, but a set of functions. Think of one error function like in Equation (3.5) for each input data point. It is needed to iterate with all of this functions one after each other. Iterating one step with all of them is called an *epoch*. Usually the error of the network output, also called loss in this context, is evaluated after each epoch. We can plot the loss over the epochs to see how the network learns. An example will be seen later in Figure 5.1.

The step size within the gradient decent algorithm is also called *learning rate*, because it influences how fast the network weights change and therefore how fast the network learns. But a too big learning rate can lead to divergence. That is why this parameter must be selected carefully as well. This is strongly dependent on the concrete problem, a general best learning rate cannot be given.

The network should learn a general estimation of the data. It should not be a too coarse approximation, but also we do not want the network to simply memorize the input data. This effects are shown in Figure 3.7. Underfitting is simpler to handle, because a too coarse approximation will lead to a higher error in the output. More complicated is overfitting, because if the network memorizes the exact training data the error is zero or very small. To compensate this the input data is split into two sets, the training and the validation set. The training set is used for learning and the validation set is only used for evaluation of the loss. With this we can detect overfitting if the loss of the training data will be small, but the validation loss is much higher. If both, the training and the validation set, show a small loss, we know that the network has learned a general pattern within the data.

Further important concepts used in this work are *normalization* and *regularization*. One problem could be that different values within the input vector are from completely different value ranges. An example would be if one value stands for the mass of very small droplets and an other value stands for the velocity. This values could be on different magnitudes. For the network it would be hard to compensate this. Therefore normalization could be used. This means every value is scaled by the mean and variation of this value within all sets of the training data. The network then process the normalized data. It is possible to normalize only the input data points or also the output data.

Regularization does mean that the weights of the connections within the network should stay as small as possible. This is achieved by adding a penalty term to the loss calculation for bigger weights in the network. This could help against overfitting.

4 Methods and Implementation

Due to the availability of the machine learning frameworks Tensorflow [MAP+15] and Keras [Cho+15], we have chosen to implement our framework in Python. Because of the partially long computation time, we split our framework into multiple Python scripts, which are parts of a pipeline. Each script writes its output on disk, enabling the next scripts to use this data for further processing. This allows us to only run single stages of this pipeline and to keep data which does not change between multiple runs. We start with reading the raw data from multiphase flow simulations and extracting single droplets. For each extracted droplet, we calculate different physical properties and quantities. Next, we need to match droplets between time steps, to be able to track droplets over time. From this data, we generate training input for the machine learning models. We train the models and then use the output for various visualizations. These steps of the pipeline are described in more detail in the following sections.

4.1 Droplet separation

As written above, we are interested in a lot of single droplets to use them for machine learning. To get many droplets we use the data from simulations which feature secondary breakup and atomization. To this end, we have to separate the VOF field data. A droplet should be the region of neighboring cells, where the volume of fluid value is not zero. As neighboring cells we only see cells which share a surface, cells only sharing an edge or corner are not seen as neighbors. Our algorithm picks a random cell as start point for a droplet and iterates over all neighbors to grow the region of cells which belong to this droplet. Algorithm 4.1 shows this in detail.

4.2 Calculation of droplet properties

We want the machine learning algorithm to run on as many droplet configurations as possible. This includes variable simulation parameters, such as different fluids, different spatial and temporal resolution, and different boundary conditions. To handle all these differences, we need do calculate more general droplet parameters, which are independent of the actual simulation output data and the used grid. As general parameters we choose: center of mass, mass, velocity, inertia, total energy, translational energy, rotational energy, oscillation energy, angular velocity, surface, surface to volume ratio, momentum and angular momentum.

The following sections describe how we calculate these properties from the simulation output.

Here, i is the index of a cell in the rectilinear grid. In the context of droplet properties, we write $\sum_i$ for the sum over all cells of this droplet.

Algorithm 4.1 Droplet separation algorithm

```
procedure SeparateDroplets(vof)
    cellIds ← getCellIdsWhereValueIsNotZero(vof)
    while size(cellIds) > 0 do
        cellId = cellIds.pop()
        cellGroup.append(cellId)
        cellGroupCheck ← 0
        while cellGroupCheck < size(cellGroup) do
            neighbors ← getNeighbors(cellGroup[cellGroupCheck])
                                                  // Front, Back, Top, Bottom, Left, Right
            for all n ∈ neighbors do
                if n ∈ cellIds then
                    cellGroup.append(n)
                    cellIds.remove(n)
                end if
            end for
            cellGroupCheck ← cellGroupCheck + 1
        end while
        makeDroplet(cellGroup)
    end while
end procedure
```

The **volume** of a single droplet is defined as

$$V_{droplet} = \sum_i f_i V_i, \tag{4.1}$$

with VOF f_i of cell i and volume V_i. As we have no density given in the data, we assume a density of 1 and use the volume as synonym for mass.

The **center of mass** is defined as

$$\mathbf{r}_{droplet} = \frac{\sum_i \mathbf{r}_i m_i}{\sum_i m_i}, \tag{4.2}$$

with cell center $\mathbf{r}_i$ and mass m_i of cell i.

We get the **velocity of the center of mass** by

$$\mathbf{v}_{droplet} = \frac{\sum_i \mathbf{v}_i m_i}{\sum_i m_i}, \tag{4.3}$$

with velocity $\mathbf{v}_i$ and mass m_i of cell i.

The **inertia tensor** $\Theta_{droplet}$ is used as a parameter for the droplet deformation.

$$\Theta_{droplet} = \begin{pmatrix} \sum_i m_i(y_i^2 + z_i^2) & -\sum_i m_i x_i y_i & -\sum_i m_i x_i z_i \\ -\sum_i m_i x_i y_i & \sum_i m_i(x_i^2 + z_i^2) & -\sum_i m_i y_i z_i \\ -\sum_i m_i x_i z_i & -\sum_i m_i y_i z_i & \sum_i m_i(x_i^2 + y_i^2) \end{pmatrix}, \tag{4.4}$$

with $(x_i, y_i, z_i)^T = \mathbf{r}_i - \mathbf{r}_{droplet}$.

To calculate the total energy of a droplet, we assume a particle system where each cell is a particle of mass m_i. The energy of one cell is

$$E_i = \frac{1}{2} m_i \mathbf{v}_i^2. \tag{4.5}$$

The **total energy** of the droplet is the sum of all cell energies

$$E_{droplet} = \sum_i E_i. \tag{4.6}$$

In the next sections, we want to split the total energy into a part for translational energy E_T, rotation energy E_R and oscillation energy E_O. Assuming the conservation of energy, the total energy is

$$E_{droplet} = E_T + E_R + E_O, \tag{4.7}$$

with $E_T \geq 0, E_R \geq 0, E_O \geq 0$.

The **translational energy** is

$$E_T = \frac{1}{2} m_{droplet} \mathbf{v}_{droplet}^2. \tag{4.8}$$

Rotation energy is defined as

$$E_R = \frac{1}{2} \omega_{droplet}^T \Theta_{droplet} \omega_{droplet}, \tag{4.9}$$

with the angular velocity $\omega_{droplet}$, whose calculation is given below.

The **oscillation energy** can be written as the rest term of the total droplet energy

$$E_O = E_{droplet} - E_T - E_R. \tag{4.10}$$

Given the **angular velocity**, we know the rotation velocity of each cell is

$$\mathbf{v}_{i,R} = \omega_{droplet} \times \mathbf{r}_i', \tag{4.11}$$

where $\mathbf{r}_i'$ is the cell center position relative to the center of mass of the droplet $\mathbf{r_i} - \mathbf{r}_{droplet}$. Even if we would know $\mathbf{v}_{i,rot}$ we cannot easily calculate $\omega_{droplet}$ because the cross product has no inverse function.

We can invert this function if we knew at least the direction of omega. Therefore, we assume that the droplet rotates around the principal axis of inertia, which should be physically plausible. ω_{dir} is a unit vector in direction of the principal axis of inertia. This leads to the equation

$$\omega_{droplet} = \hat{\omega}_{droplet} \omega_{dir}, \tag{4.12}$$

where $\hat{\omega}_{droplet}$ is only a scalar. Furthermore, we now assume a moving coordinate system aligned with the center of mass and the center of mass velocity of the droplet, hence $\mathbf{v}'_{droplet} = 0$. Inserting this into Equation (4.11) gives us

$$\hat{\omega}_i = \frac{|\mathbf{v}'_{i,R}|}{|\omega_{dir} \times \mathbf{r}'_i|}, \tag{4.13}$$

with $\mathbf{v}'_{i,R}$ as projection of $\mathbf{v}'_i$ into the direction of $\omega_{dir} \times \mathbf{r}'_i$.

The individual $\hat{\omega}_i$ of all cells are averaged by cell weight

$$\hat{\omega}_{droplet} = \frac{\sum_i \hat{\omega}_i m_i}{\sum_i m_i}. \tag{4.14}$$

We can extend this method to use three axes (i.e. all three principal axis of inertia) and combine these three resulting rotations to a total rotation. The problem with this is the limited numerical precision. We can get cases where $E_T + E_R > E_{droplet}$ and with this $E_O < 0$, which is physically incorrect. Therefore we use a second method for physical correctness.

The idea is to formulate this as an optimization problem with constraints.

The velocity of each cell could be split in three parts: translational velocity, rotation velocity and oscillation velocity:

$$\mathbf{v}_i = \mathbf{v}_{droplet} + \mathbf{v}_{i,R} + \mathbf{v}_{i,O}, \tag{4.15}$$

where $\mathbf{v}_i$ and $\mathbf{v}_{droplet}$ are known and $\mathbf{v}_{i,R} = \omega_{droplet} \times \mathbf{r}'_i$.

Now, we want to minimize the remaining oscillation velocity $\sum_i \|\mathbf{v}_{i,O}\|_2$, finding an optimal $\omega_{droplet}$.

$$\min_{\omega_{droplet}} \left\{ \sum_i \|\mathbf{v}_{i,O}\|_2 \right\}. \tag{4.16}$$

The constraint for this optimization problem is the conservation of energy in Equation (4.7)

We can bring this optimization problem to the form of a quadratically constrained quadratic program (QCQP), which could be solved with optimization toolboxes. In our implementation we use the IBM CPLEX Optimizer [IBM]. With this method we get physically correct energies in all cases.

Next we need the **surface** of a single droplet. To calculate the surface of a single droplet we used an implementation of the PLIC algorithm [KSM+13]. We extract all planes of the droplet and sum the area of them together to the total surface area of the droplet. By dividing the surface by the volume, we can get the **surface to volume ration**.

Finally the **momentum** of the droplet is defined as

$$\mathbf{p}_{droplet} = m_{droplet} \mathbf{v}_{droplet}, \tag{4.17}$$

and the **angular momentum** as

$$\mathbf{L}_{droplet} = \Theta_{droplet} \omega_{droplet}. \tag{4.18}$$

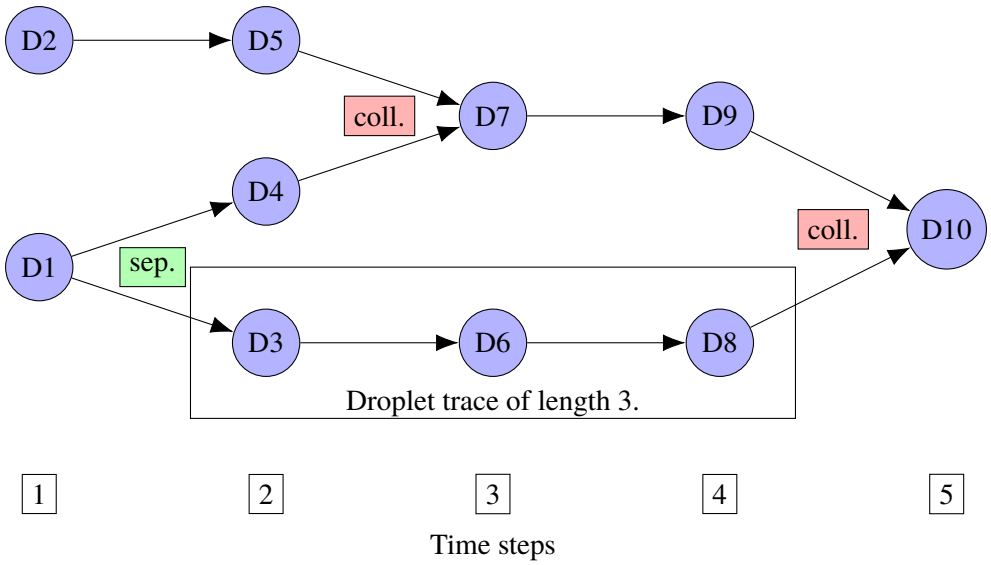

Figure 4.1: Example of the droplet graph showing separations, collisions and a droplet trace.

4.3 Matching between time steps

Above, we have extracted single droplets and calculated droplet properties. This was done for each time step individually. Now, we want to observe droplets over time and therefore we need to track the droplets between time steps. To this end, we want to build a graph for all droplets in time. For each droplet in each time step there is a node in the graph. We want to know which of this nodes belong to the same droplet at different time steps. Therefore we want connections in the graph between nodes which stands for the same droplet at different time steps. And of course we only want the connection between neighboring time steps.

We use the method of forward and backwards advection. This is a simple version of the method from Karch et al. [KSB+17]. For each droplet in one time step, we look at each cell. For each cell center we have a velocity vector. With the cell center position, the velocity and the time difference to the next step, we can calculate one step with the Euler method to approximate the position of the cell center in the next time step. We look up to which cell of the next time step this point in space belongs and then if this cell is part of a droplet. If we find a droplet in the next time step, we can add a connection to our droplet graph between the droplet in the current time step and in the next time step. An example of such a graph is given in Figure 4.1 The same thing is also done in backwards direction by calculating the inverse movement from the current time step to the previous time step. This increases precision as the method is only an approximation.

Within this graph, we now can also see separations and collisions of droplets. If we have edges from one droplet in time step t to two or more droplets in time step $t + 1$, this droplet has separated. If we have edges from two or more droplets in time step t to one droplet in time step $t + 1$, droplets have collided. We want to ignore separations and collisions and therefore are interested in the droplets with exactly one incoming and one outgoing edge. Of course, more combinations are possible. For example, a droplet which does not have a connection to the previous, or next time step, could happen, if this droplet enters, or leaves, the simulation domain in the current time step.

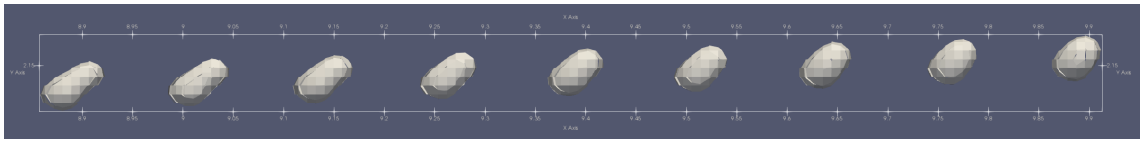

Figure 4.2: Rendering of a droplet within one trace from the jet dataset. The surface is rendered with the PLIC method. The single droplets are rendered at their actual position in the simulation domain. The offset between them is the actual droplet movement, from left to right.

As we are interested in single droplets over time, we define the term droplet trace. Hereby we mean all paths where we can follow one single droplet as long as possible over time. A separation, a collision, or if the droplet just appears, means the start of a new trace. The trace goes as long as we find exactly one connection in the graph to the next time step and then ends also with a collision, a separation, or just by leaving the domain. In Figure 4.2, we show renderings of the single droplets within one trace. The trace is nine time steps long. Here, the beginning of the trace was a separation from a bigger droplet and the trace ends because the droplet collides with another droplet in the next time step.

In the section above, we have described how to calculate the properties and physical quantities of this droplets. Figure 4.3 shows all of these values plotted over time within this example trace. These values will be used for further processing.

4.4 Generation of ML input and model training

We want to analyze two different problem cases. The first would be to generate a model which could predict the actual droplet values in the next time step. Therefore we need traces of constant length l. We then use $l - 1$ time steps as input to predict the model in the l-th time step of the trace. To get this constant length traces and preferably many data for training we use overlapping subtraces. For example if l is 5 and we have a trace of length 8, we can get 4 subtraces.

The second case is to predict whether a droplet separates or not. To predict this we get a similar set of subtraces of constant length. The difference is that we use the complete subtrace of length l as input and additionally we get the information from the graph if the subtrace ends within the next n time steps after the subtrace ends.

The actual input vector is then generated by using a vector of the properties for each droplet of a subtrace. To generate the actual input data we replace the subtraces by a vector of the properties of each droplet. The single vectors for each droplet of one time step are then sticked together.

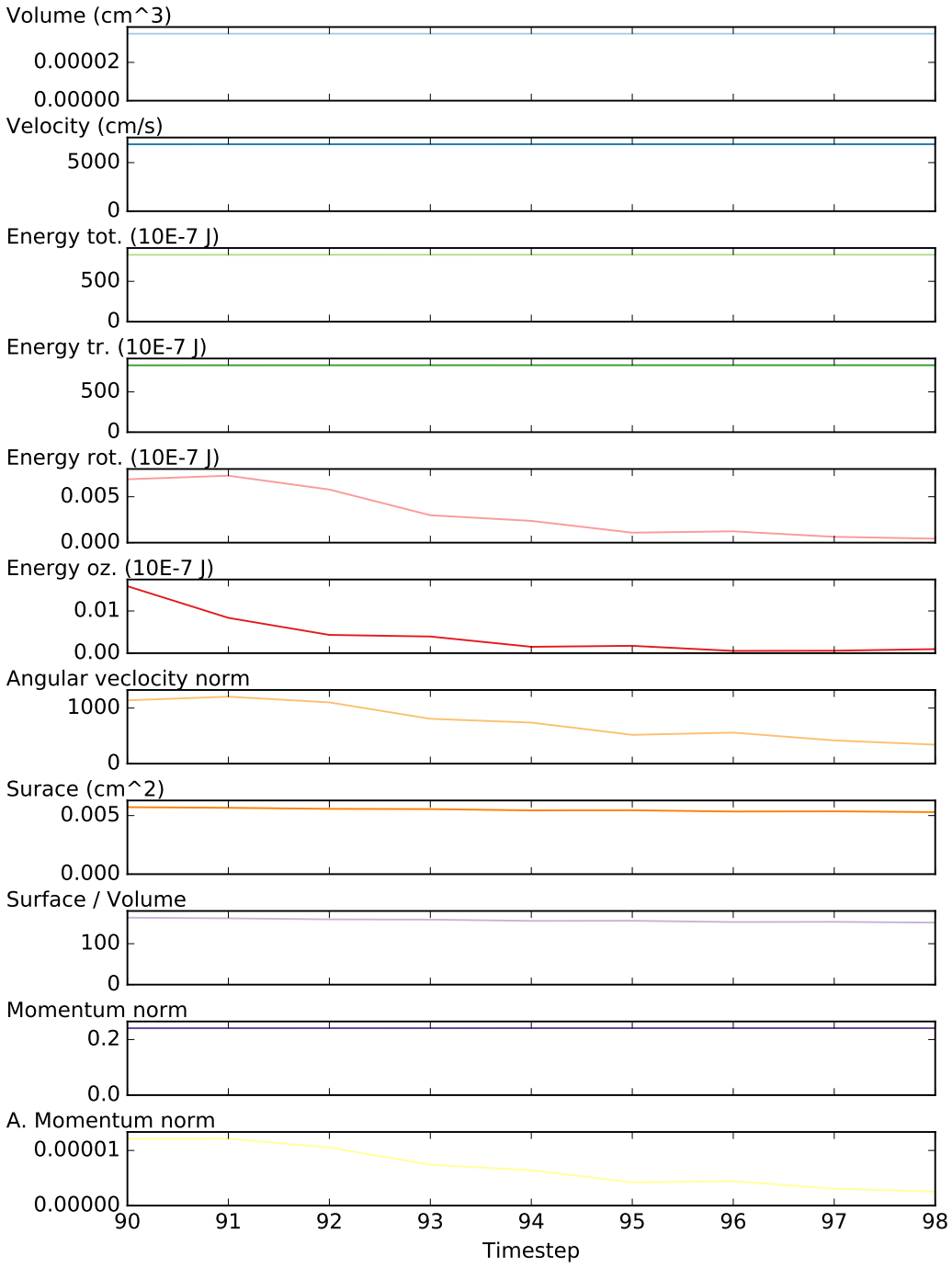

Figure 4.3: Development of the physical properties of single droplet over time.

5 Results

Our final neural network consists of an input layer with the length of the input data vector, followed by two hidden layers with 16 fully connected neurons each, followed by an output layer with a single neuron for one value. The hidden layers use the ReLU activation function, while the output layer uses the linear activation function. The input of our network is the full vector of a droplet trace as described in the last section. We only use a single output neuron to predict exactly one droplet property. To predict multiple droplet properties, we train different networks with the same input vectors, but a different output property on each. We did this, because not all properties show the same kind of predictability, at least with our used datasets. By splitting the property prediction to different networks, the influence of more badly predictable properties does not influence the learning of the other properties. We also think this does increase the understanding of the learning results. The network layout of two hidden layers with 16 neurons each has shown good results during this work. We have tried different network layouts to find the best by looking at overfitting and underfitting, but this could of course be analyzed more deeply in possible future work. Furthermore adding regularization to the network has shown better results of learning. Because the input data values varies over multiple magnitudes between different properties, we used normalization of the input. Each variable was scaled to have a standard deviation of 1 around its mean.

Loss curves of the training process are shown in Figure 5.1. This example is for the jet dataset with a trace length of seven, which means that the input vector covers six time steps for the eleven properties each. From the jet dataset we could extract 62 560 overlapping traces of length seven. We split this in training and validation data by 70% to 30% after random mixing. As you notice, there is an offset in the loss between validation and test data even at the beginning before the first learning step. Further investigation has shown that this is probably some bias within the data. Repeating the random mixing and splitting of the training and validation data changes this offset. In average with multiple repeats this offset is near zero, but with high variation for single evaluations. Therefore we ignore this offset for now and just evaluate the shape of the loss over time. Also surprisingly seems the fact that the surface to volume ratio shows a bad learning performance. This is surprising because the mass and surface show good results. As we wrote above, mass is directly bound to the volume and the surface to volume ratio is simply the division of surface by volume.

With the trained models, we predict future droplet properties with one trace and compare this to the ground truth. The difference is shown as error. We have error values for each properties individually and a total error as euclidean norm of all error values. A visualization of the total error is shown in Figure 5.2. The error values are rendered as colored dot at the center of mass of each droplet. Additionally, a transparent rendering of the total jet with PLIC surface is shown in the image for context. We used the tool ParaView [Aya15] for this visualizations.

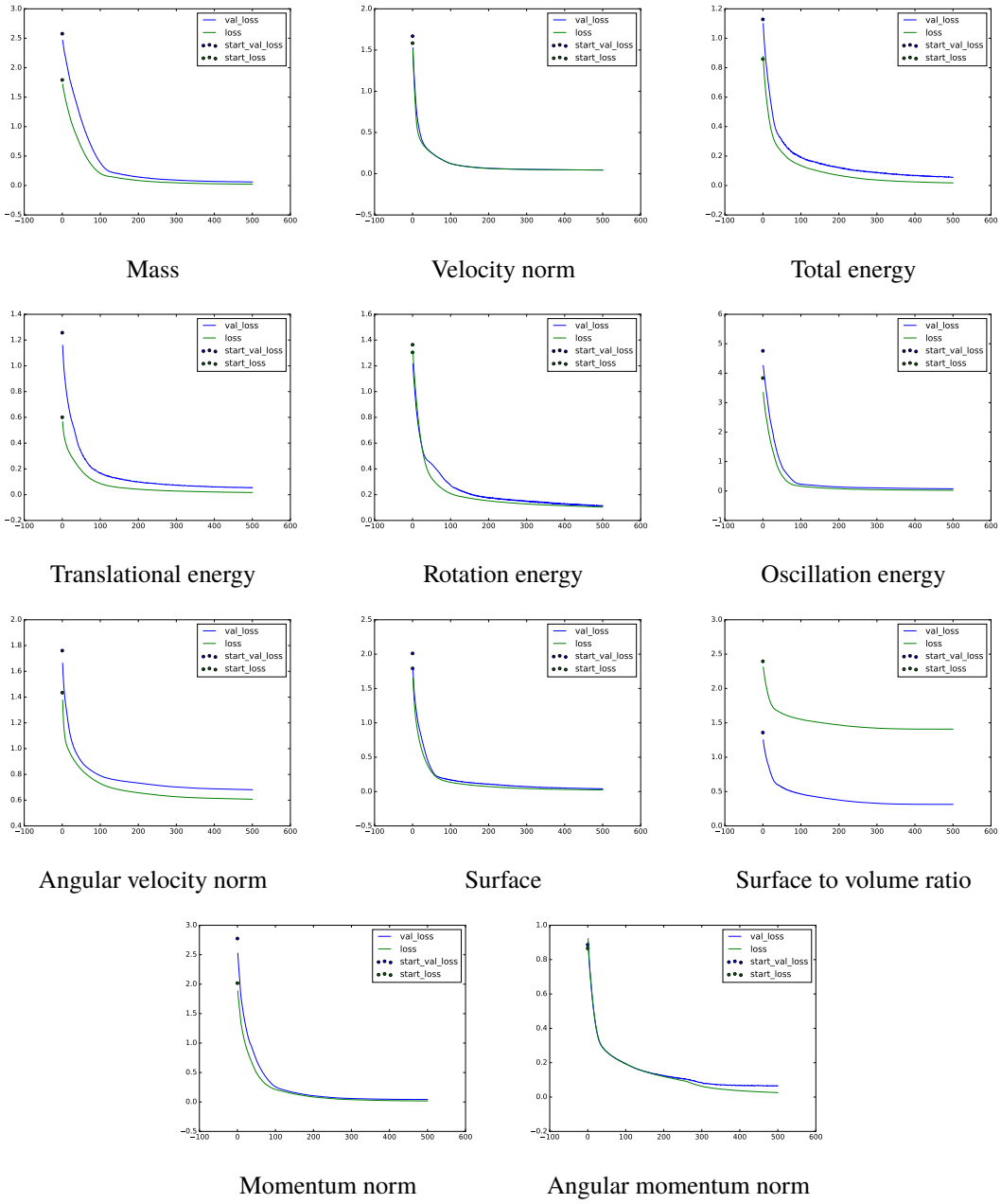

Figure 5.1: Loss curves of the learning process with the jet dataset and an input trace length of 6 time steps.

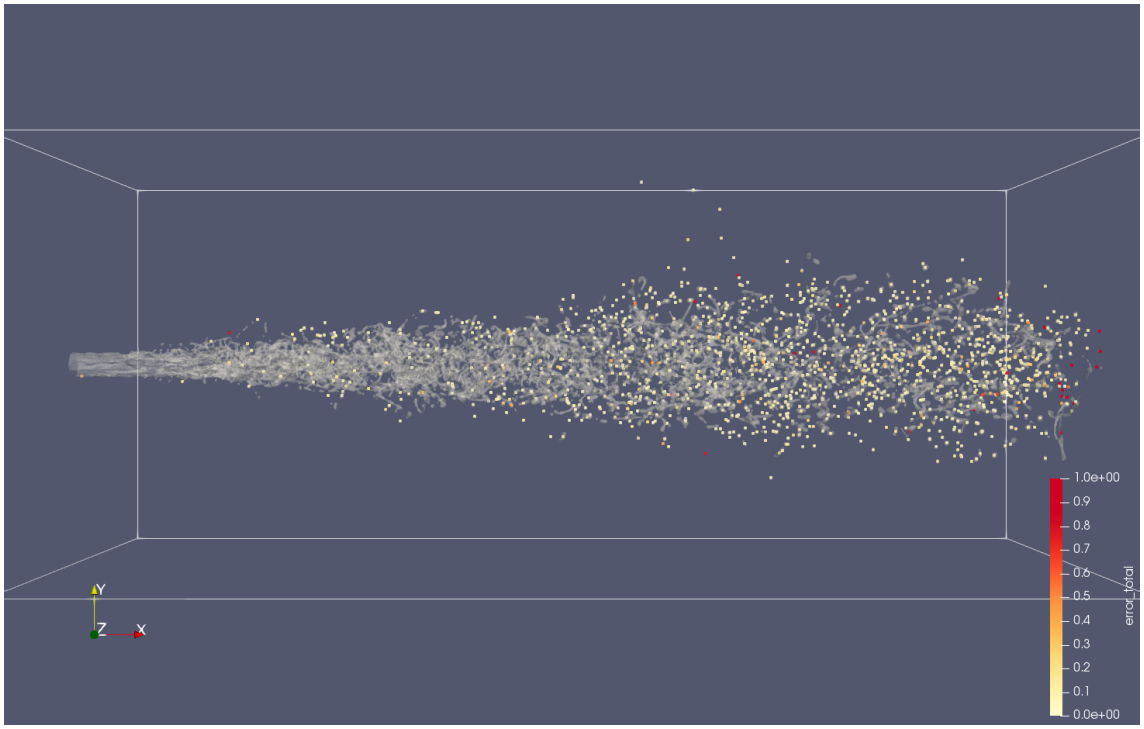

Figure 5.2: Droplet prediction total error visualized as colored dots for time step 95 of the jet dataset.

You can see a few red dots distributed over the domain. However, there is a concentration of red dots at the right of the jet. This is where the droplets leave the domain. The calculation of the properties there could be wrong, for droplets which have already partially left the domain. It is plausible that we can see a larger error there.

Furthermore, we picked some of the dots with large error from the center and have subjected them to a deeper investigation. In Figure 5.3 we look at the development of the properties over time of such a droplet. Additionally, we plotted the predicted values as blue dots for each property. Because we used six time steps as input vector and the trace is only eight time steps long, we only have predictions for the last two time steps in this trace. We notice that some values have a massive increase in the last time step. Looking at the mass, we see also a small increase in the last time step. This is physically incorrect as the mass should stay constant. A deeper look at the data shows, that here the matching between the time steps is incorrect and a collision was overlooked. Therefore, the droplet properties change, as we have a new droplet after the collision. However, we think it is a good validation that this is still found as an interesting droplet with higher prediction error. The error of the last time step is also visualized as spider chart in Figure 5.4. In contrast to Figure 5.3 where the physical values are shown, we calculate the error within the normalized values which are used for training of the neural network. We think this allows for a better comparison of the error values.

We have picked a second random example with high error within the 3D view. Also for this droplet, we show the same diagrams as before in Figure 5.5 and Figure 5.6. First of all, we see much more variation within the properties. There are changes in rotational energy, oscillation energy, angular

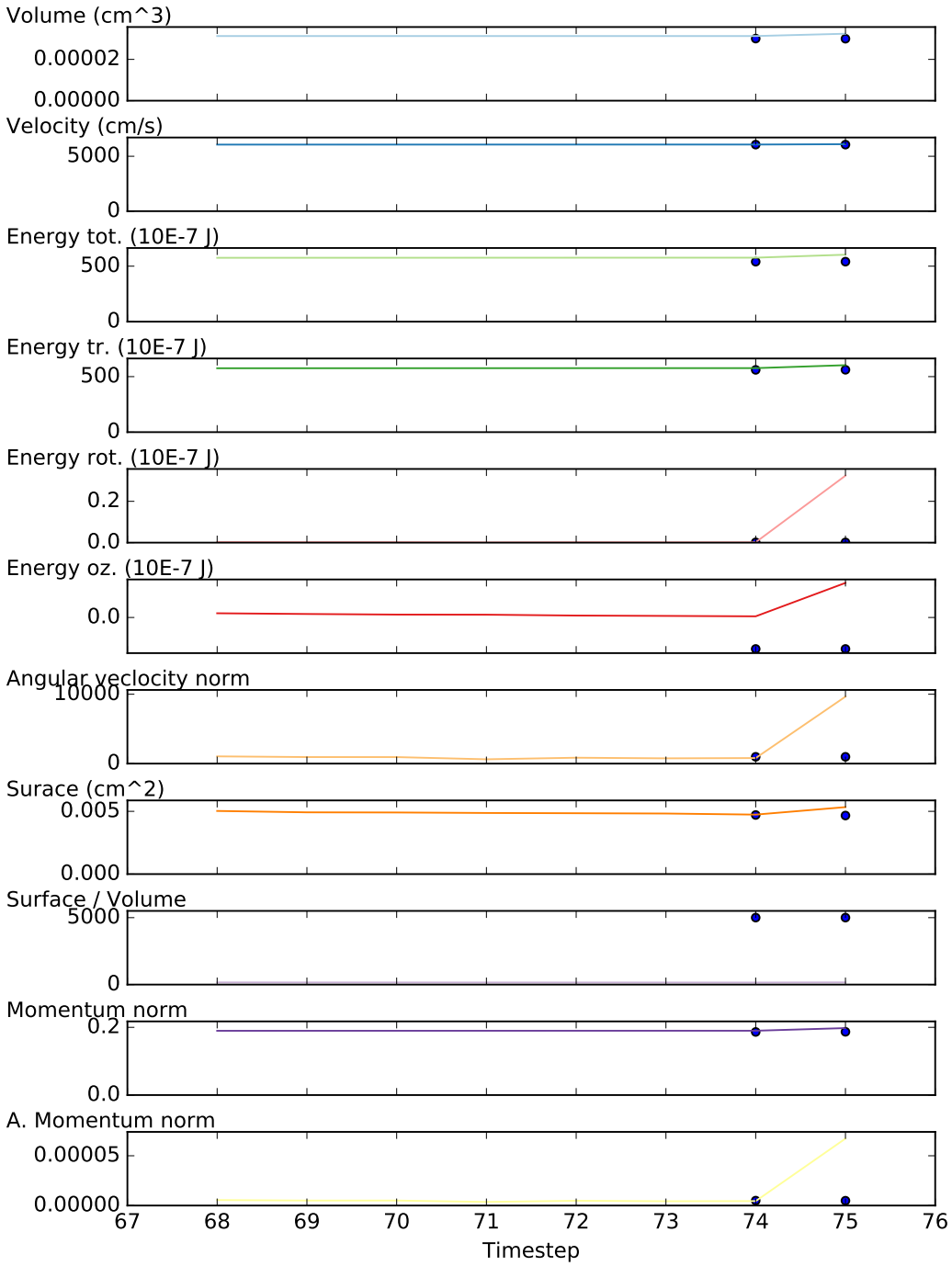

Figure 5.3: Development of the properties over time for the first example droplet including the predicted values (blue dots). Some of the values increase massively in the last time step, because of an error in the droplet matching. We see the prediction does not expect this.

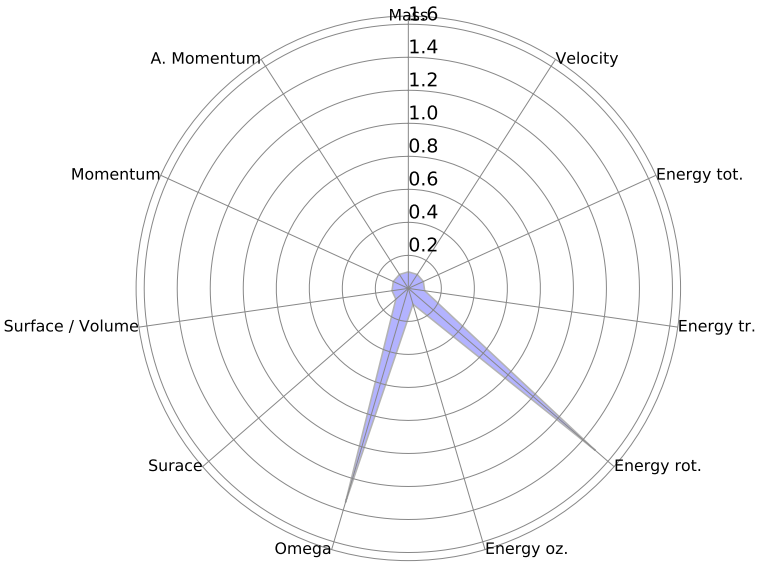

Figure 5.4: Spider chart of the error by property for the last time step of the first example droplet. We see a large error for rotational energy and angular velocity within the normalized value range. Error values smaller than 0.1 was set to the value of 0.1 for better visibility of the glyph.

velocity and the surface area. In Figure 5.7 we show renderings if this droplet over time. The droplet has a very bumpy shape and the shape changes a lot over time. Therefore, we think we have found a droplet which could be interesting.

For comparison we look now at a droplet with a very small error value in the 3D view. Again we look at the development of the properties over time and the predicted values in Figure 5.8. First of all, we see that the trace of this droplet is a lot longer. Therefore we can see much more predicted values than in the last examples. A rendering of this droplet is shown in Figure 5.9. This droplet looks a lot more uniform and constant over time. But we see that the rotational energy, the angular velocity and the angular momentum have variation over time. They all share a similar pattern within the variation, which is not at all surprising, as angular momentum and rotational energy are directly related to the angular velocity and all other values are more or less constant. For angular velocity and rotational energy, we can see that the prediction can follow this trend. Unfortunately, the prediction of the oscillation energy and the angular momentum seem to be a lot worse.

The same kind of analysis was done with the splash dataset. Figure 5.10 shows the total error within the 3D view of the data for one time step. Overall we got very similar results, and therefore we want to only look at one example.

In contrast to the jet, the splash dataset contains very long droplet structures, called ligatures. You can see an example in Figure 5.12. The ligatures do not behave like the the round droplets. As expected, the network output has a high error. Again, we show the plot of the properties over time

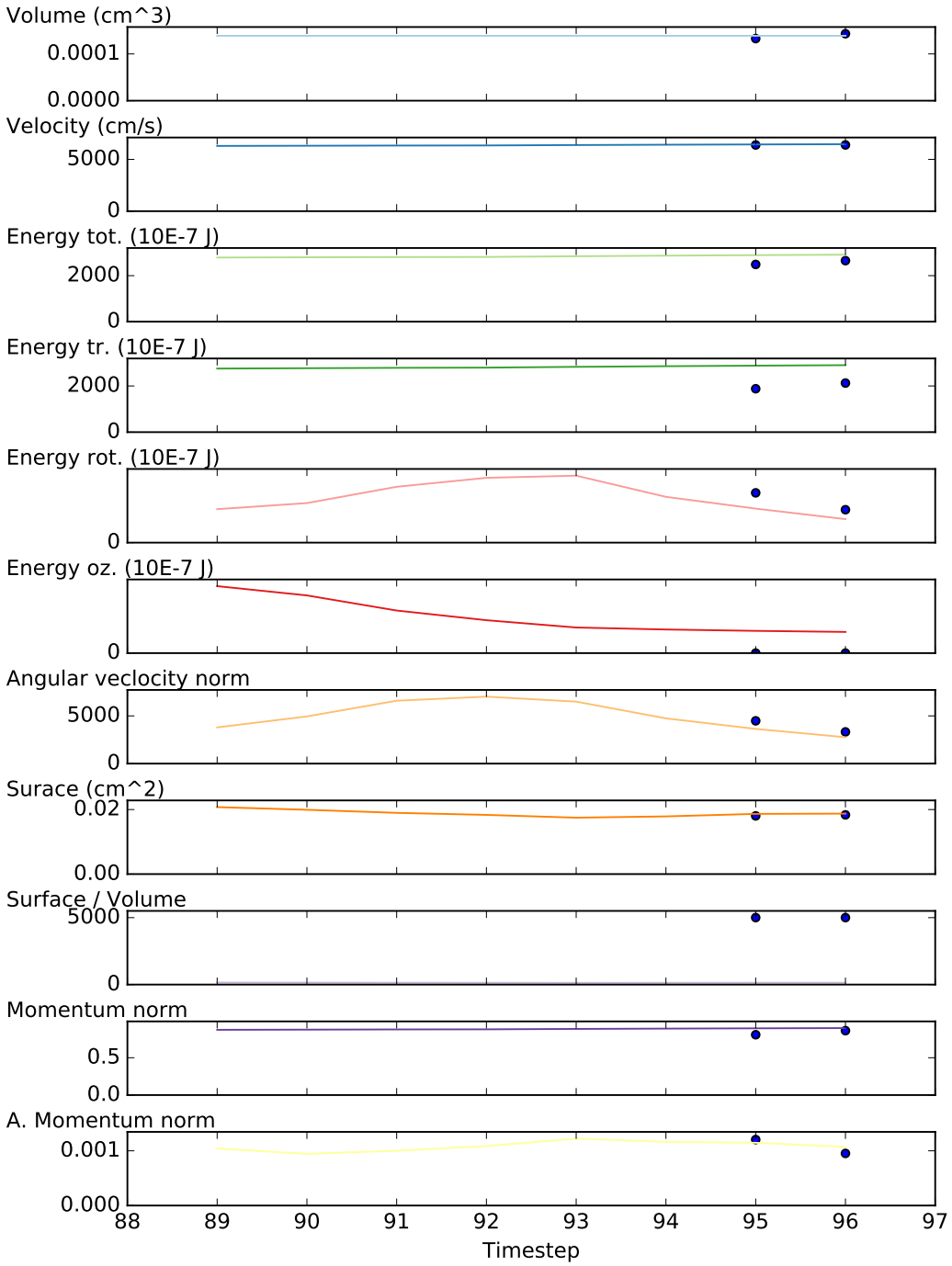

Figure 5.5: Development of the properties over time for the second example droplet including the predicted values (blue dots). We see variation the rotational and oscillation energy, as well as the angular velocity. Some of the predictions are clearly different to the ground truth.

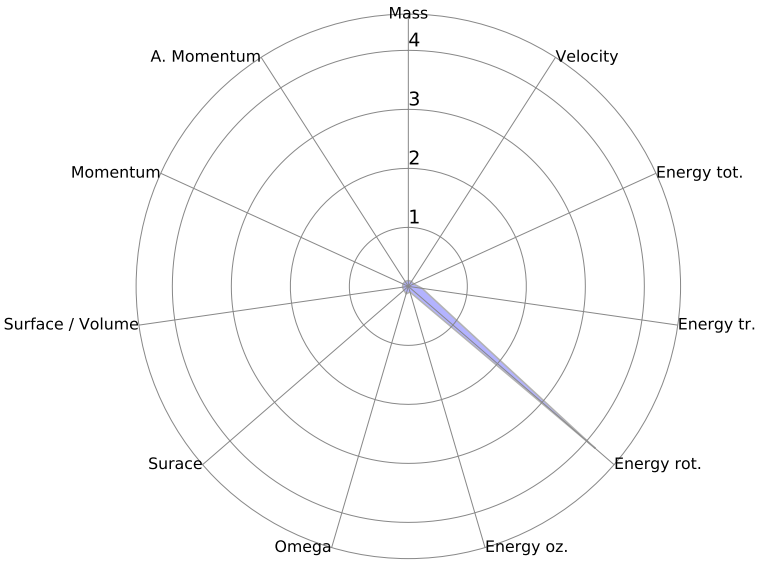

Figure 5.6: Spider chart of the error by property for the second to last time step of the second example droplet. We see a large error for rotational energy within the normalized value range. Error values smaller than 0.1 was set to the value of 0.1 for better visibility of the glyph.

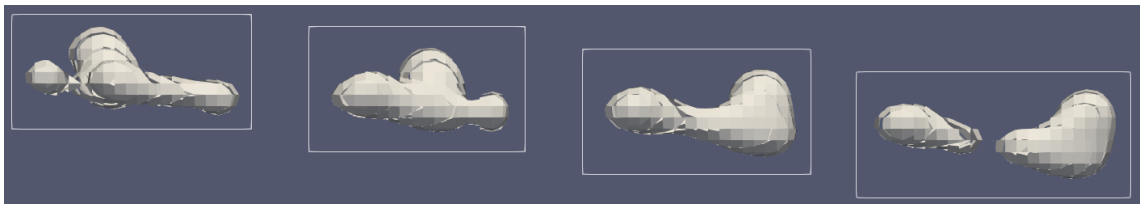

Figure 5.7: Rendering of the second example droplet over time. Only every second time step is shown. The surface was reconstructed using PLIC. The right droplet looks already separated, but this is still one droplet, as defined in Section 4.1. The surface approximation of the PLIC method is inaccurate at this point.

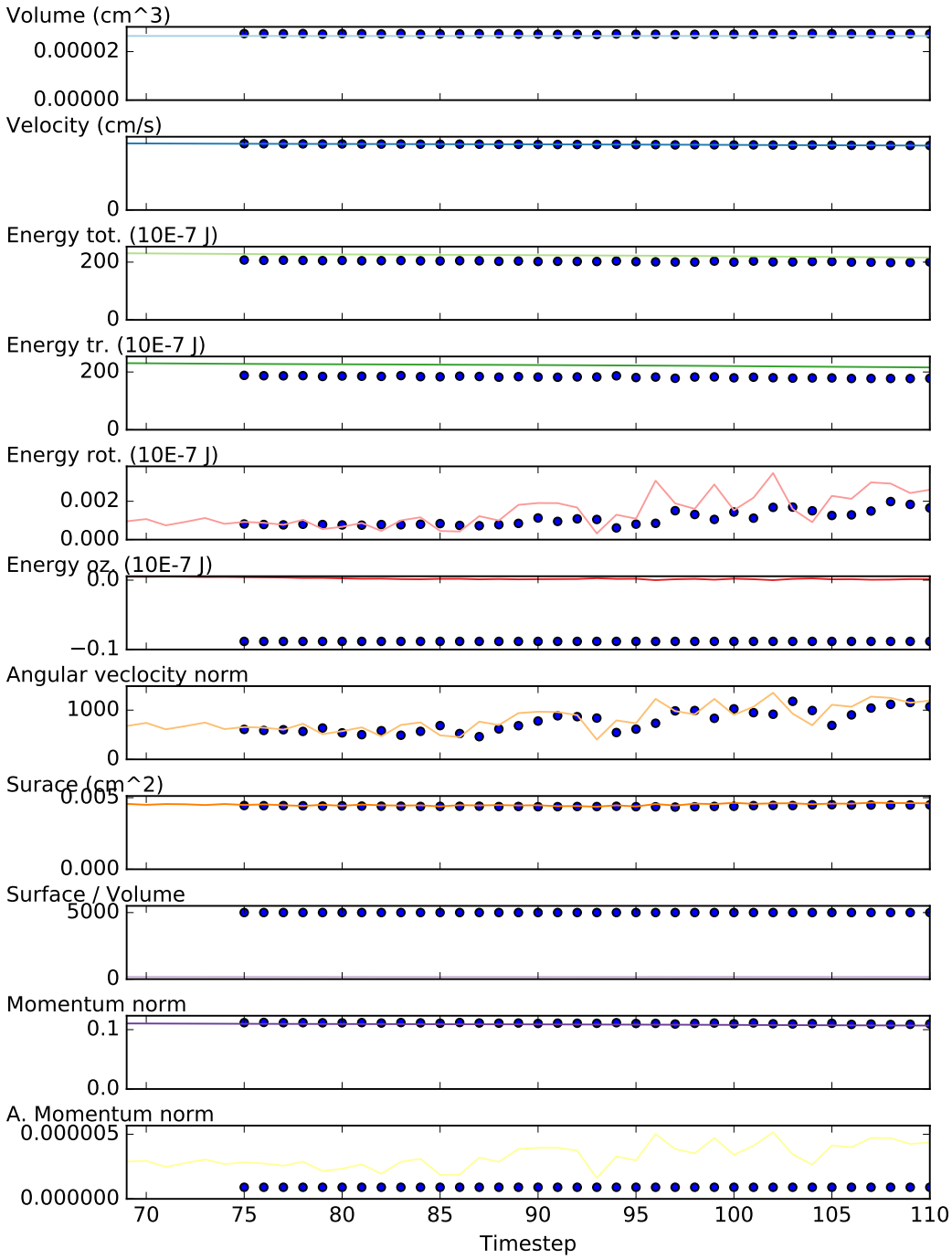

Figure 5.8: Development of the properties over time for the example droplet with low error including the predicted values (blue dots). We see a longer trace with many predictions. The prediction for angular velocity and rotational energy does follow the ground truth.

Figure 5.9: Rendering of the example droplet with low error over time. The surface was reconstructed using PLIC. Only a part of the trace for this droplet is shown.

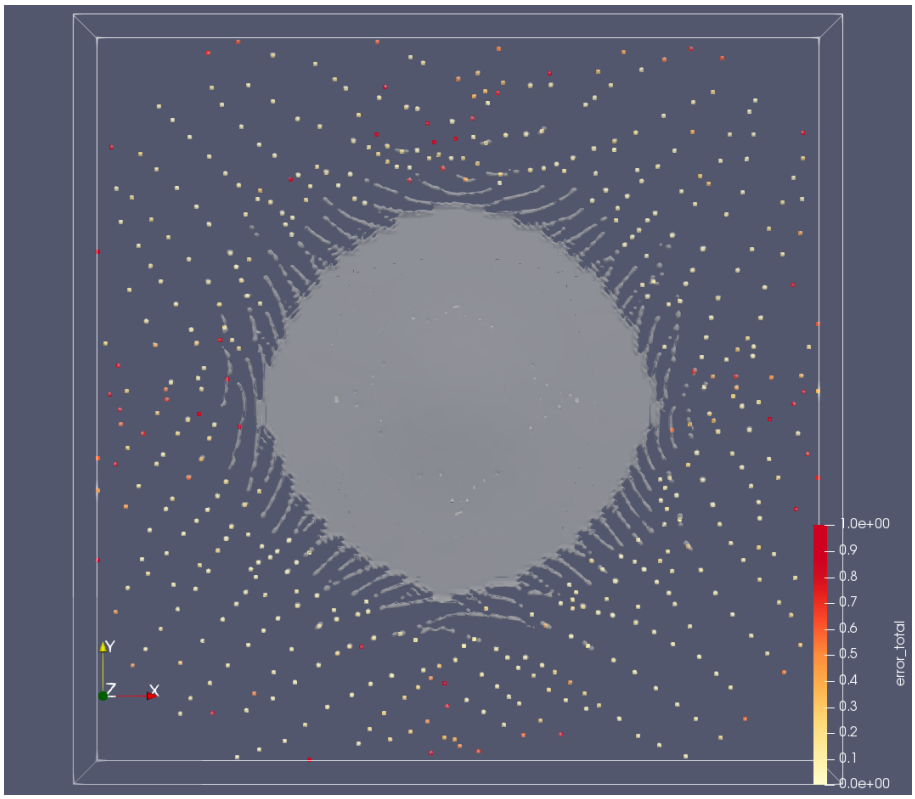

Figure 5.10: Droplet prediction total error visualized as colored dots for time step 111 of the splash dataset.

with the predicted values in Figure 5.11. For this example, we used a trace length of four time steps. So far this was expectable, but we see a different problem with the ligatures. In Figure 5.10, we see that only some of the ligatures have a point with an error value at all. The problem here is that ligatures are not very stable and therefore have a lot of separations. We cannot find a trace which is long enough to use for prediction, even if we only used a shorter trace length here. This is a general limitation of our method, as we explicitly restricted our model to traces without collision and separation.

Additionally to the development of the droplet parameters, we want to predict droplet separations. We divided the droplet traces in two classes. One class for the droplets which separate within the next few time steps after the trace, and the other class for droplets which stay stable for the same amount of time steps after the trace. On this data, we did not find a network which was able to

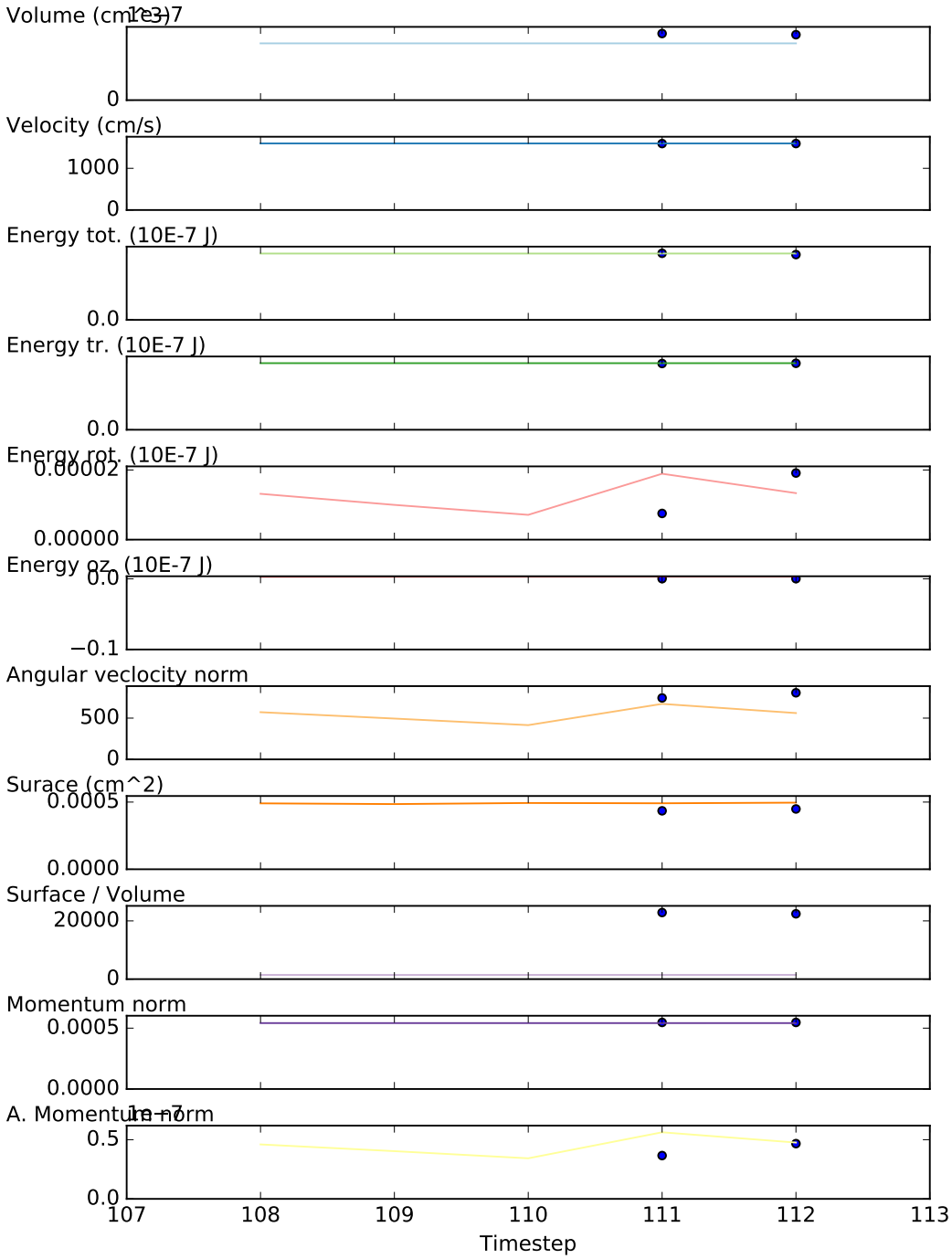

Figure 5.11: Development of the properties over time for a droplet from the splash dataset including the predicted values (blue dots). Some of the predictions show a difference to the ground truth. The trace of this ligature is only five time steps long and therefore very short.

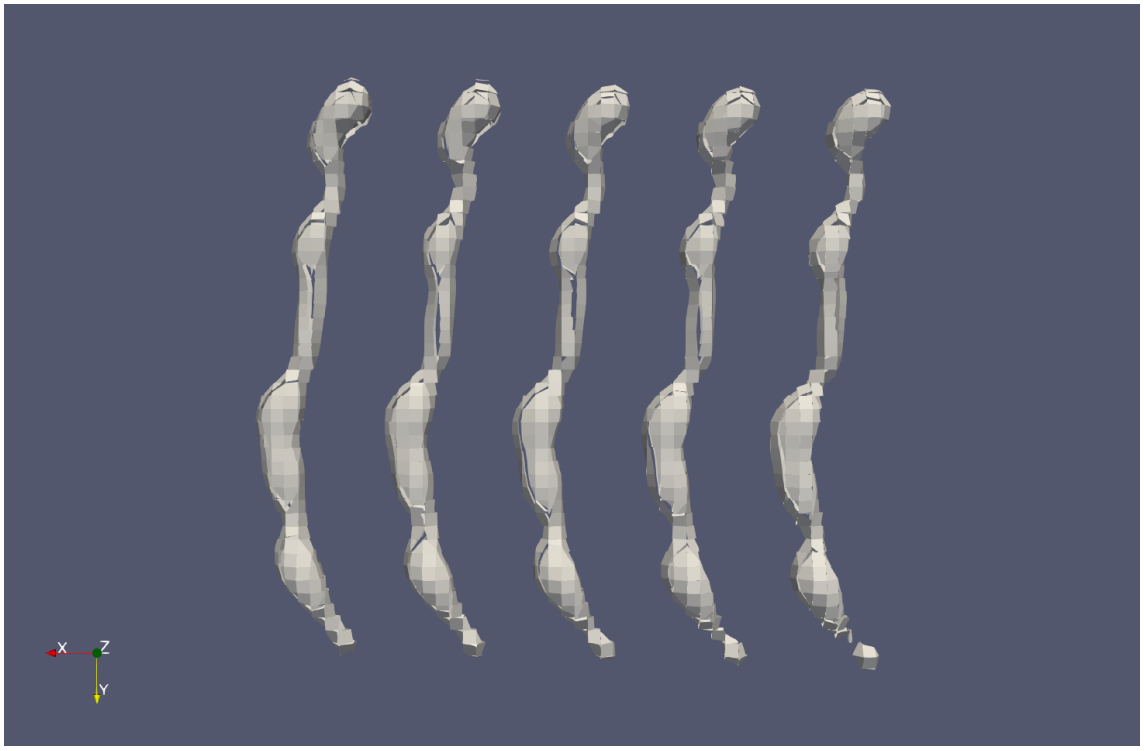

Figure 5.12: Rendering of the example droplet from the splash dataset. The surface was reconstructed using PLIC. Droplets with such long structures are also called ligatures.

learn a separation prediction with satisfying results. We think there are multiple reasons for this. One is that the datasets only contains very few separations. For example, within the splash dataset for a trace length of five and a search for separation within the next 3 time steps, we get 72 667 training sets for no separation, but only 630 sets for a separation. This is similar for the jet dataset: 58 238 sets for no separation and 1 970 sets for separation. Also, it could be possible that there is no particular pattern within the data, where a separation could be predicted from. And finally, more time is needed for a deeper investigation.

6 Conclusion and Outlook

6.1 Conclusion

In this thesis, we have presented our framework for analyzing droplets from multiphase flow simulations with the help of ML. This is used to produce a visualization where we can find interesting droplets within the data.

We started with extracting droplets from simulation data from the solver FS3D. These droplets where tracked over time and we calculated physical properties and quantities of them. With this, we trained artificial neural networks which could predict the future development of the droplet properties. We compared the prediction to the ground truth and used the resulting error as rate of how interesting a droplet is. Our theory is that the neural network has learned the average behavior of the droplets and our definition of an interesting droplet is that it behaves different to the average droplet.

We used these results to analyze the two given datasets and have shown the usability of our work on examples. For the prediction of the droplet properties, we have seen good results within the learning process at least for most of the properties. Unfortunately, the results for the droplet separation where not usable to generate meaningful visualizations. We need to investigate this further.

6.2 Future work

In future, our work could be extended in multiple directions. It would be interesting to apply our method to more datasets and include further parameters, such as the type of fluid or further boundary conditions from the simulation. We could also try to use more or different properties from the droplets, for example a more explicit measurement for the droplet shape. Furthermore the learning process of the droplet property prediction could be analyzed more systematic for optimization of the network layout and of course an improvement of the separation prediction is needed.

Finally, the visualization should be improved. The property prediction error values are shown as point data wihtin the 3D view. The visualization could be changed such that the PLIC surfaces show the color and we do not need the points anymore. Also, the implementation could me more interactive. At the moment, our framework writes vtk-files, which we than can view within ParaView. The other charts were generated by scripts. The analysis process would be easier for the observer, if we can automate this by offering the possibility to select single droplets within the 3D view and automatically generating the charts for this droplet.

Bibliography

[Aya15] U. Ayachit. *The ParaView Guide: A Parallel Visualization Application*. USA: Kitware, Inc., 2015. ISBN: 1930934300, 9781930934306 (cit. on p. 33).

[BLK13] I. Bright, G. Lin, J. N. Kutz. "Compressive sensing based machine learning strategy for characterizing the flow around a cylinder with limited pressure measurements". In: *Physics of Fluids* 25.12 (2013), p. 127102. DOI: 10.1063/1.4836815. eprint: https://doi.org/10.1063/1.4836815. URL: https://doi.org/10.1063/1.4836815 (cit. on p. 15).

[Cho+15] F. Chollet et al. *Keras*. https://keras.io. 2015 (cit. on pp. 15, 25).

[EEG+16] K. Eisenschmidt, M. Ertl, H. Gomaa, C. Kieffer-Roth, C. Meister, P. Rauschenberger, M. Reitzle, K. Schlottke, B. Weigand. "Direct numerical simulations for multiphase flows: An overview of the multiphase code FS3D". In: *Applied Mathematics and Computation* 272 (2016). Recent Advances in Numerical Methods for Hyperbolic Partial Differential Equations, pp. 508–517. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2015.05.095. URL: http://www.sciencedirect.com/science/article/pii/S0096300315007195 (cit. on pp. 15, 17).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 15, 19, 21–23).

[HN81] C. Hirt, B. Nichols. "Volume of fluid (VOF) method for the dynamics of free boundaries". In: *Journal of Computational Physics* 39.1 (1981), pp. 201–225. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(81)90145-5. URL: http://www.sciencedirect.com/science/article/pii/0021999181901455 (cit. on pp. 15, 17).

[IBM] IBM. *IBM ILOG CPLEX Optimization Studio*. https://www.ibm.com/analytics/cplex-optimizer (cit. on p. 28).

[Kri07] D. Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: http://www.dkriesel.com (cit. on pp. 15, 19–23).

[KSB+17] G. K. Karch, F. Sadlo, S. Boblest, M. Ertl, B. Weigand, K. Gaither, T. Ertl. "Visualization of Feature Separation in Advected Scalar Fields". In: *arXiv preprint arXiv:1705.05138* (May 2017). arXiv: 1705.05138. URL: http://arxiv.org/abs/1705.05138 (cit. on p. 29).

[KSM+13] G. K. Karch, F. Sadlo, C. Meister, P. Rauschenberger, K. Eisenschmidt, B. Weigand, T. Ertl. "Visualization of piecewise linear interface calculation". In: *2013 IEEE Pacific Visualization Symposium (PacificVis)*. Feb. 2013, pp. 121–128. DOI: 10.1109/PacificVis.2013.6596136 (cit. on pp. 15, 18, 28).

[LB16] M. Liu, D. Bothe. "Numerical study of head-on droplet collisions at high Weber numbers". In: *Journal of Fluid Mechanics* 789 (2016), pp. 785–805. ISSN: 14697645. DOI: 10.1017/jfm.2015.725 (cit. on pp. 15, 18).

[LC87] W. E. Lorensen, H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169. ISBN: 0-89791-227-6. DOI: 10.1145/37401.37422. URL: http://doi.acm.org/10.1145/37401.37422 (cit. on p. 15).

[LT15] J. Ling, J. Templeton. "Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty". In: *Physics of Fluids* 27.8 (2015), p. 085103. DOI: 10.1063/1.4927765. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.4927765. URL: https://aip.scitation.org/doi/abs/10.1063/1.4927765 (cit. on p. 15).

[MAP+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on pp. 15, 25).

[NH10] V. Nair, G. E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 21).

[NW76] W. F. Noh, P. Woodward. "SLIC (Simple Line Interface Calculation)". In: *Proceedings of the Fifth International Conference on Numerical Methods in Fluid Dynamics June 28 – July 2, 1976 Twente University, Enschede*. Ed. by A. I. van de Vooren, P. J. Zandbergen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1976, pp. 330–340. ISBN: 978-3-540-37548-7 (cit. on pp. 15, 17).

[OS01] M. Oliveira, A. Sousa. "Neural network analysis of experimental data for air/water spray cooling". In: *Journal of Materials Processing Technology* 113.1 (2001). 5th Asia Pacific conference on Materials processing, pp. 439–445. ISSN: 0924-0136. DOI: https://doi.org/10.1016/S0924-0136(01)00646-X. URL: http://www.sciencedirect.com/science/article/pii/S092401360100646X (cit. on p. 15).

[RK98] W. J. Rider, D. B. Kothe. "Reconstructing Volume Tracking". In: *Journal of Computational Physics* 141.2 (1998), pp. 112–152. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1998.5906. URL: http://www.sciencedirect.com/science/article/pii/S002199919895906X (cit. on p. 15).

[TSSP16] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin. "Accelerating Eulerian Fluid Simulation With Convolutional Networks". In: *CoRR* abs/1607.03597 (2016). arXiv: 1607.03597. URL: http://arxiv.org/abs/1607.03597 (cit. on p. 15).

[You82] D. L. Youngs. "Time-dependent multi-material flow with large fluid distortion". In: *Numerical methods for fluid dynamics* 24 (1982), pp. 273–285 (cit. on p. 15).

[You84] D. L. Youngs. “An interface tracking method for a 3D Eulerian hydrodynamics code”. In: *Atomic Weapons Research Establishment (AWRE) Technical Report* 44/92 (1984), p. 35 (cit. on p. 15).

[YS07] K. Yetilmezsoy, A. Saral. “Stochastic modeling approaches based on neural network and linear–nonlinear regression techniques for the determination of single droplet collection efficiency of countercurrent spray towers”. In: *Environmental Modeling & Assessment* 12.1 (Feb. 2007), pp. 13–26. ISSN: 1573-2967. DOI: 10.1007/s10666-006-9048-4. URL: https://doi.org/10.1007/s10666-006-9048-4 (cit. on p. 15).

All links were last followed on July 11, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature