Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische

Friedrich-Wilhelms-Universität Bonn

Bachelor Thesis

Estimating Physical Properties from Video

Author:
Martin Link

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Dr. Hassan Errami

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Place, Date

Signature

Contents

1 Introduction

Bin picking is a core problem in computer science and robotics that consists of planning and performing a directed picking process of a single object from a box of different objects with a robot end-effector. The scenario is highly relevant, with applications ranging from warehouse control to robot assistants. Our group participated in the Amazon Picking Challenges 2015 and 2016, as well as the Amazon Robotics Challenge 2017 which had the goal to advance the state-of-the-art in bin picking [1, 2]. The process of bin picking itself combines several different disciplines, ranging from computer vision to grasp planning, motion planning, and execution control [3]. The disciplines of grasp and motion planning require a good model of the objects, agent, and world in order to allow for successful manipulation of the scene. Notably, this includes a model of the physical properties of the objects: Mass, surface friction, elasticity, moment-of-inertia, and density distribution/center-of-mass can all play a crucial role and - when guessed incorrectly - lead to failure cases. The goal of this thesis is to explore ways to improve the knowledge of physical parameters of objects, to be used in bin picking scenarios. One straightforward way to achieve this that we investigated was the use of a physics engine that in itself is differentiable, which allows for backpropagation *through* the engine, in order to update the physical parameters of a simulation. In the end, we settled for a more flexible and efficient approach, by developing a framework that uses a prediction module - e.g. a neural network - to *guess* the physical parameters in a simulation iteratively by comparing an observation to the simulation. Since this does not require backpropagation through the engine itself, it lifts the need for differentiability of the physics engine and instead relies on standard neural network and machine learning techniques. This simplification allows for the use of commercially available physics engines that are already optimized for speed and efficiency but not necessarily differentiable.

This thesis is structured as follows: In chapter 2, we describe the related work of this project in more detail. Chapter 3 concerns itself with the *simulation* or physics engine part of our new approach. We describe the used physics engine and the scene that was used for subsequent testing. We also estimate the achievable resolution. Chapter 4 is focussed on the *prediction module* of our approach. We compare different techniques, namely hyperparameter optimization and a neural network, and evaluate their performance when used to iteratively guess physics parameters. We furthermore investigate the role of second order processes, multiple objects, multiple parameters and different object shapes. In chapter 5, we provide a summary of the work and in chapter 6, we give a brief outlook on future investigations and the path to three dimensional scenes.

2 Related Work

Control system optimization plays a big role in bin picking scenarios, where the success of a planned operation is critically dependent on how well a robotic agent can be controlled. Oftentimes, controllers are optimised in simulated environments with reinforcement learning, particle swarms or genetic algorithms, completely free of any derivatives [4, 5, 6]. As a consequence, the robot is often treated as black box, which prevents efficient gradient-based deep learning methods. Physics engines like MuJoCo [7] allow for the evaluation of gradients between states and actions through a robot, however, they do not allow back-propagation to the initial model parameters. More recent work investigated special physics engines that are end-to-end differentiable [8, 9], however, when initially testing the use of such engines, we found that they were much slower and less adaptable to new scenarios than commercial engines like NVidia PhysX [10]. In our group, we developed a PhysX-based framework for the creation of simulated data to train agents in bin picking scenarios called Stillleben [11]. The goal of this thesis is to test approaches that will allow us in the future to use Stillleben for improving scene understanding by updating the knowledge of physical properties of objects. This includes keeping our framework efficient in order to be able to actively improve the scene-knowledge at runtime. To achieve this goal, we build on an iterative approach that is used to estimate the 6D pose of objects, see ref [12]. In this work by Li et al., a simulated pose is compared to an observed object in order to determine changes in the pose. This is done iteratively, in order to refine the predictions and allow for incremental improvements. In our case, we *observe* time series of frames to allow for efficient interpretation of the shown dynamics. Our *guess* comes from a physics engine that is used to simulate the scene. The comparison of both then allows to determine corrections to parameters of the simulation, that correspond to physical properties of the investigated objects. The network that we use throughout this thesis is based on the ResNet architecture [13], but modified for the use of 3D convolutions to account for the time-series dimension. The task of predicting physical parameters of objects has been investigated before, for example unsupervisedly from short video sequences [14]. Oftentimes, properties are also predicted from single pictures, for example via micro-CT pictures for porous media [15], pictures of stacked crops [16] or pictures of liquid crystals [17]. In a more robotics-centric context, neural networks have been used to predict the hardness of objects with a GelSight sensor [18]. In context to these earlier works, our dynamic approach utilises iteration over observed scenes that allow for constant refinement of the parameters by taking into account new information.

3 Simulating Physics for Bin Picking

Commercial physics engines are often developed with efficiency in mind and not to be differentiable, which prevents backpropagation to the initial input parameters. In our case, we decided to circumvent this problem by not doing a backward pass through the physics engine itself, but rather have a prediction module compare the ground truth observation to a simulation and then correct the corresponding input parameters that represent the physical objects' properties. This will allow for the use of an arbitrary physics engine, that takes the parameters that are subject to optimization as an input.

3.1 Physics engine

In principle, with our approach, the choice of the physics engine is arbitrary. For testing in a two-dimensional environment, we chose the Python-based physics library Pymunk [19], which is built on the physics engine Chipmunk [20]. The library allows for efficient simulation of two-dimensional rigid-body physics. Objects can be defined with arbitrary (two-dimensional) shapes, represented through polygons. Other physical properties like the mass, center-of-mass, moment of inertia, elasticity and friction can be set as well, which allows for several different parameters to be predicted by our framework. Forces can be added to objects, which allows for dynamical behaviour. To design scenes, fixed (immovable) objects can be added to the scene that can act as barriers. Pymunk includes helper functions for visualisation with pygame [21], which allows for prototyping and visualisation of our test scenes. Pymunk is independent of units, as the calculated physics only depends on the ratio between them. For this reason, throughout this thesis, no physical units will be given.

3.2 Test scene

The ultimate goal of the presented approach is to be used in a bin picking scenario in three dimensions. For testing purpose, we restrict ourselves to understanding the two-dimensional case in this thesis, in order to test different network architectures and prediction parameters. The scene we choose for testing consists of three objects in a schematic box, shown in Figure 1. The resolution of each image array is 400×400 pixels. The box is formed by three fixed lines, which provide confinement for the objects. Two objects are placed at a fixed position next to each other in the box. The third object is placed above the first two and accelerated by a force impulse. All object are subject to a gravitational force pointing down. We usually observe the scene for 30 frames, with the collision between the objects happening at frame 15. Each circle is marked by a line to provide an angle and measure the orientation of the circle.This simple scene setup allows us to investigate simple seperated collisions, which are the elementary processes used to determine physical properties.

3.3 Achievable resolution

The ability of the network to correct the input parameters is limited by the resolution of the observed scene: If a small parameter change causes a change in

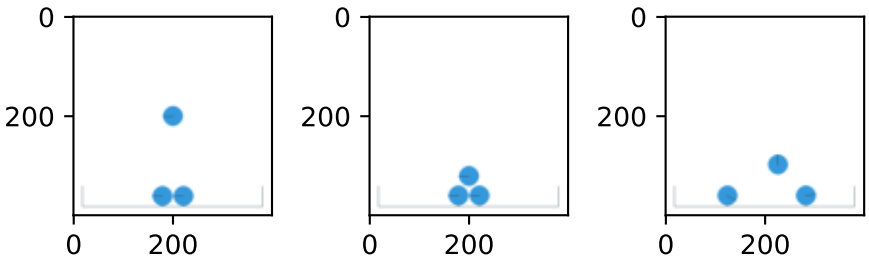

Figure 1: **Test scene.** Left to right: Frame 1, frame 15 and frame 30 of the test scene that we use throughout this thesis to estimate the performance of different approaches.

the object position that is smaller than one pixel, the network will – without additional processing, e.g. through antialiasing – perceive the scene as unchanged. To get an estimate of the achievable resolution, we investigated how large the observed changes are for a given change in input parameters for our scene. One parameter that is predicted by the network is the mass. With an initial mass of 5 for both objects in the bin, we create a reference time series of 30 frames in grayscale. We then change the mass by an amount Δm, re-simulate, and calculate the mean squared distance to the reference time series. The result can be seen in Figure 2.

Other parameters that can be tuned are elasticity and friction. In Pymunk, both are given by coefficients ranging from 0 to 1. Starting from 0.1, we again compare time series for several differences in elasticity and frictions, see Figures 3 and 4.

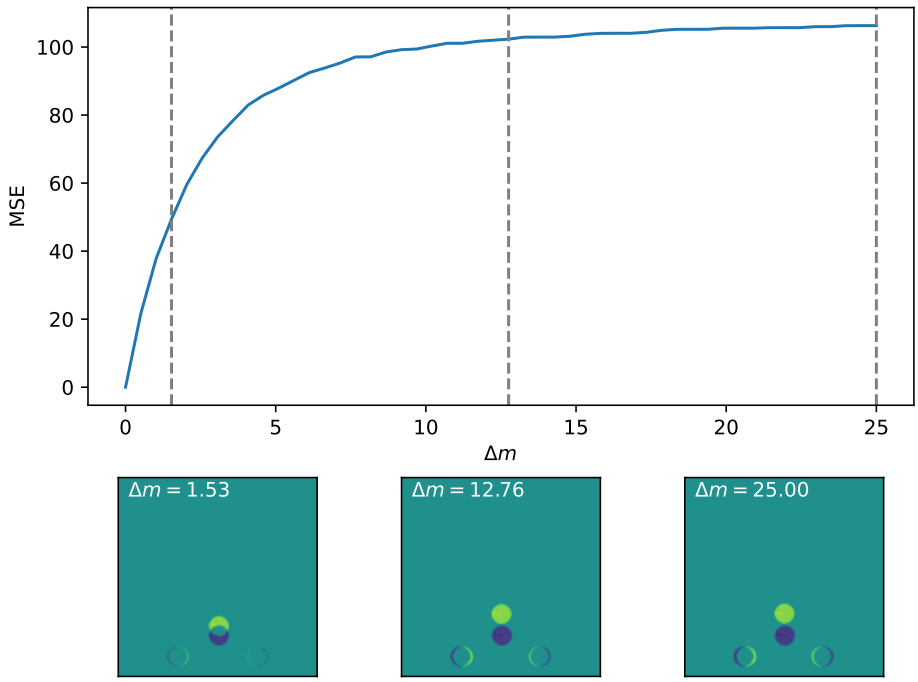

Figure 2: **Influence of the mass parameter on the mean squared error of the time series.** Top panel shows the mean squared error (MSE) for different mass differences Δm. The reference mass is $m = 5$. The panels in the bottom show the difference between the last picture of the reference and changed time series for different Δm, indicated by the grey dashed lines in the top panel.

In conclusion, all three parameters show a strong change towards small parameter changes, which is important towards the end of the iterative process, where small increments in parameter space are expected. For large paramter changes the curves for mass and especially friction flattens. It remains to be seen, wether this poses a problem for the neural network. In any case, good initial guesses help alleviate this situation.

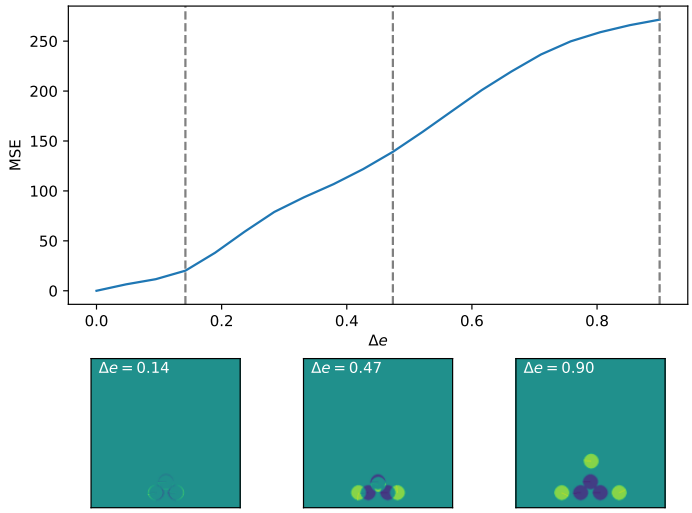

Figure 3: **Influence of the elasticity parameter on the mean squared error of the time series.** Top panel shows the mean squared error (MSE) for different elasticity differences Δe. The reference elasticity is $e = 0.1$. The panels in the bottom show the difference between the last picture of the reference and changed time series for different Δe, indicated by the grey dashed lines in the top panel.

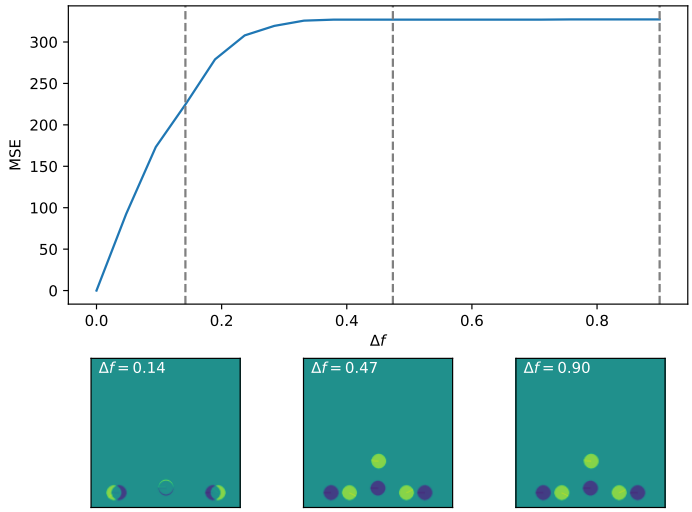

Figure 4: **Influence of the friction parameter on the mean squared error of the time series.** Top panel shows the mean squared error (MSE) for different friction differences Δf. The reference friction is $f = 0.1$. The panels in the bottom show the difference between the last picture of the reference and changed time series for different Δf, indicated by the grey dashed lines in the top panel.

3.4 Using a differentiable physics engine

As mentioned at the beginning of this chapter, our approach aims to make use of a non-differentiable physics engine, by predicting the parameters of objects that act as input to the engine. The already mentioned alternative is to use a physics engine, that in itself is differentiable. To be able to compare both approaches, we adapted the code of de Avila Belbute-Peres et al. [9] for testing with a similar scene. The results are shown in Figure 5. While the differentiable engine is able to reliably find the ground truth values of the two masses (given by the dashed lines), it takes several hundreds of iterations to do so. While this approach is very dynamic and can in principle determine the parameters of any scene setup, we chose an approach that aims to be more efficient. However, this comes at the cost of being very specialised on scenes that are captured by the training data.

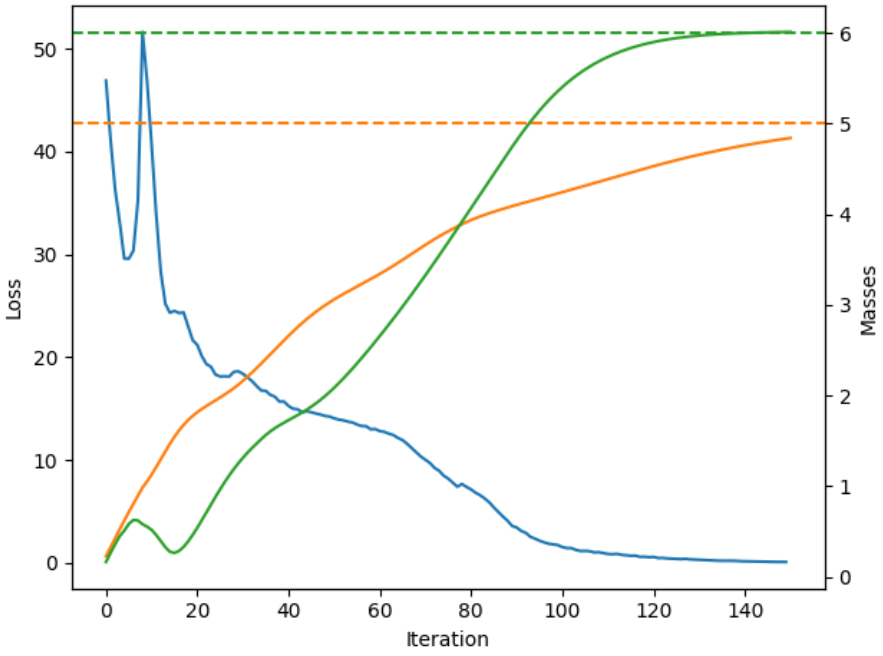

Figure 5: **Loss and predicted masses during inference with a differentiable physics engine.** Code adapted from [9]. The blue curve shows the loss value (mean squared error between predicted and ground truth mass) and the green and orange lines show the actual mass values of the objects. Dashed lines indicate the expected ground truth mass.

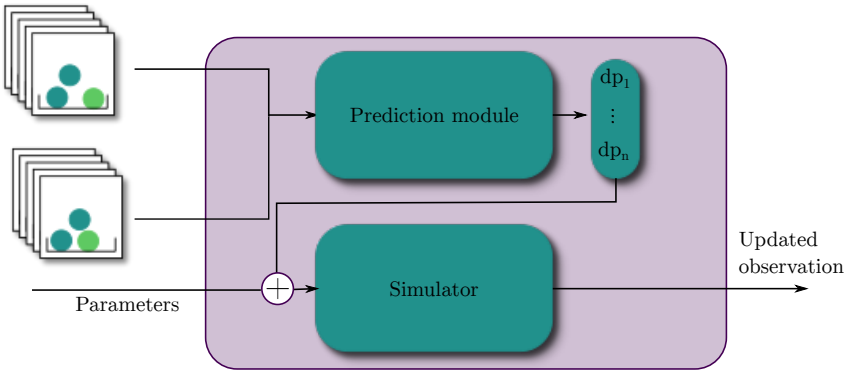

Figure 6: **Schematic of our approach.** Our approach consists of two independent modules in order to improve our understanding of the physical scene. We use a simulated and an observed time series and feed them into a *prediction module*, which then predicts possible changes to the input parameters of the simulation. We then update our initial input parameters and re-simulate the scene. This procedure is iteratively repeated to refine the parameter guesses and take into account new observations.

4 Predicting Physical Properties

In this chapter, we investigate the optimization of physical properties with different approaches. In the first section, we will present our general approach for parameter optimization, which is both independent of the physics engine and the optimizer that is used. In the later section we will investigate hyperparameter optimization and deep neural networks as a possible prediction modules.

4.1 Iterative approach for parameter optimization

Our general approach to predicting the physical parameters of objects in a bin picking scenario is inspired by earlier work on iterative predictions of pose estimation, see ref [12]. In summary, we take an observation and try to estimate the physical properties by matching a simulated scene with guessed parameters. Importantly, this approach seperates the simulation aspect from the task of predicting the input parameters. In comparison to approaches where the simulation itself has to be differentiable, we can independently choose which physics engine we use for simulation and which module to optimize the parameters.

The principle is sketched in Figure 6. We start with two time series of pictures, the ground truth observation and a simulation with guessed parameters. Both sets of pictures are passed to the prediction module, which outputs a set of parameter changes Δp_i for all n parameters entering the simulator. These parameter changes are then added to the initial parameters to yield the updated

simulation parameters. Afterwards, a re-simulation with updated parameters is performed. The new simulation results can then be used with the (possibly updated) observed ground truth, to re-iterate the physical parameters. In summary, each iteration takes one forward pass through the prediction module (potentially for each object) and one pass through the simulator.

4.1.1 Multiple objects

One important question is how our approach should treat different numbers of objects in the scene. When predicting physical parameter changes, the naive approach would be to predict all parameter updates at the same time. However, since the shape of the output is fixed at runtime, this would mean the network is only usable for a fixed number of objects. In our case, we decided to gain flexibility by predicting parameter changes for objects one at a time. We select the object on the input by marking it with a different color, see Figure 7. In the bin picking use case, this would amount to an initial segmentation task that can either be performed seperately, or by the prediction module itself. In the following, we will compare the performance of the single-object approach with the multi-object approach. An alternative approach to the color-marking would be to place the objects themselves in separate channels.

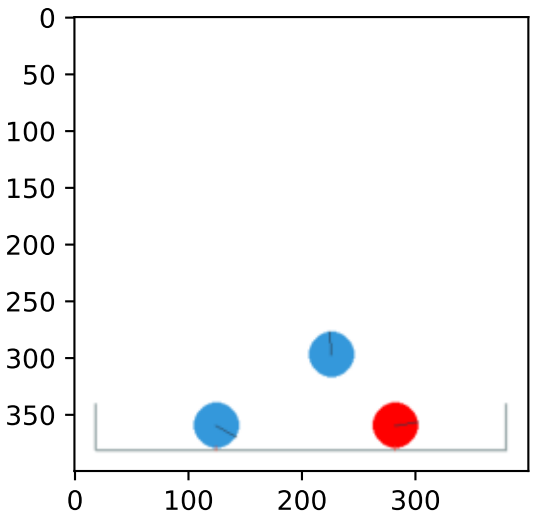

Figure 7: **Example frame with marked object.** One object is being marked by a different color. This allows the prediction module to identify the object under investigation.

4.2 Hyperparameter optimization

One possible choice for the prediction module is to use generic gradient-free optimization techniques. In our case, we test the hyperopt library [22] for Python, which implements a Tree of Parzen Estimators (TPE) approach to find the optimal set of parameters for a given objective function [23]. The objective function

has to take the parameters as input and provide a measure for the accuracy of the output. As a criterion for accuracy, we choose the mean squared error between the ground truth and simulated time series. Hyperopt itself provides the framework to log arbitrary measures during the optimization cycle, and provides the best set of parameters after optimization. In the following subsection, we will investigate the performance of hyperopt as prediction module.

4.2.1 Optimization and performance

We start by analysing the performance of the hyperopt approach, when the masses of the two unknown objects are the only free parameters. We repeat the optimization procedures several times by randomising the ground truth mass, and starting with a random initial guess. Hyperopt will then optimize the mass parameters to achieve the lowest mean squared error between the time series. The search space for the hyperopt algorithm is restricted to the space of possible masses for the ground truth. The performance of the hyperopt approach is shown in Figure 8.

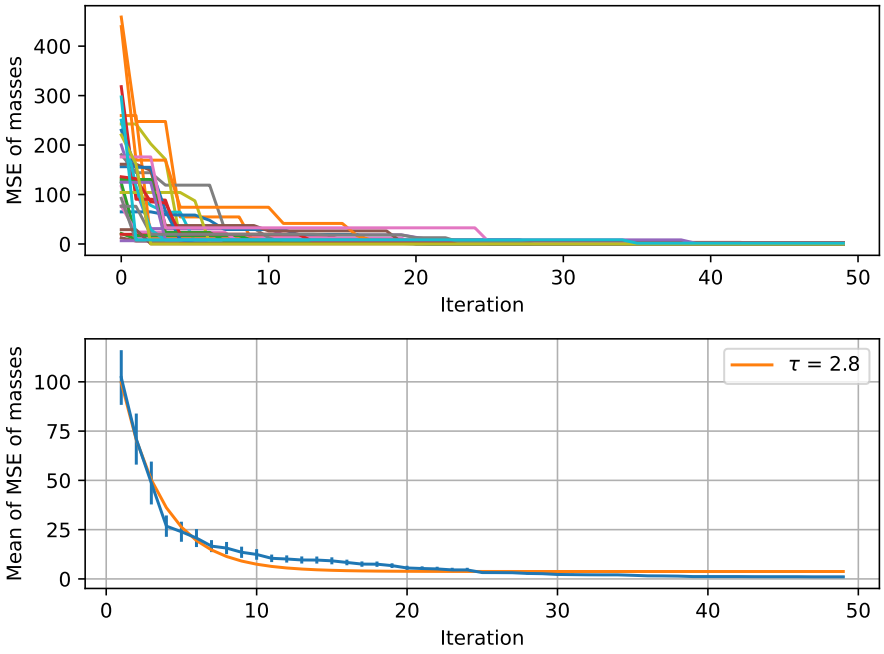

Figure 8: **Performance of our approach when using the hyperopt library as prediction module.** Top: Mean squared error of the estimated mass for 20 different scenes (different colors). Note that we show the mass error corresponding to the minimum visual error up until the current iteration, for easier analysis of convergence. Bottom: MSE averaged over over the 20 scenes with standard error. The orange curve is an exponential fit with a decay constant of $\tau = 2.8$.

Note that for the analysis of the performance, we look at the mean squared errors between the ground truth and guessed masses, while the hyperopt algo-

rithm itself only operates on the mean squared error between the time series pictures. The upper panel in Figure 8 shows the error curves for twenty different ground truth / initial guess pairs and for each curve the minimum mean squared error between the masses at a given iteration. The behaviour of the mean squared error between masses and time series pictures is qualitatively similar (a minimum in the latter always corresponds to a minimum in the former), however, we chose this visualisation for its straightforward interpretation. The lower panel in Figure 8 shows the mean error at a given iteration, excluding the randomized initial guess with errorbars denoting the error of the mean. The orange line represents an exponential fit to the curve with a decay parameter of $\tau \approx 2.8$. While the exact shape is not necessarily expected to behave exponentially, we found the fit to be reasonably accurate in the predicted range and the decay parameter allows for a quick estimation of the expected errors after a certain number of iterations. Figure 9 shows the trajectory of guesses through the parameter space for three exemplary curves (color coding similar to Figure 8). Note that this trajectory includes guesses that perform worse than the best guess at a given point. It can be seen that while the hyperoptimization uses the TPE algorithm to improve its guesses, it still samples a large part of the parameter space.

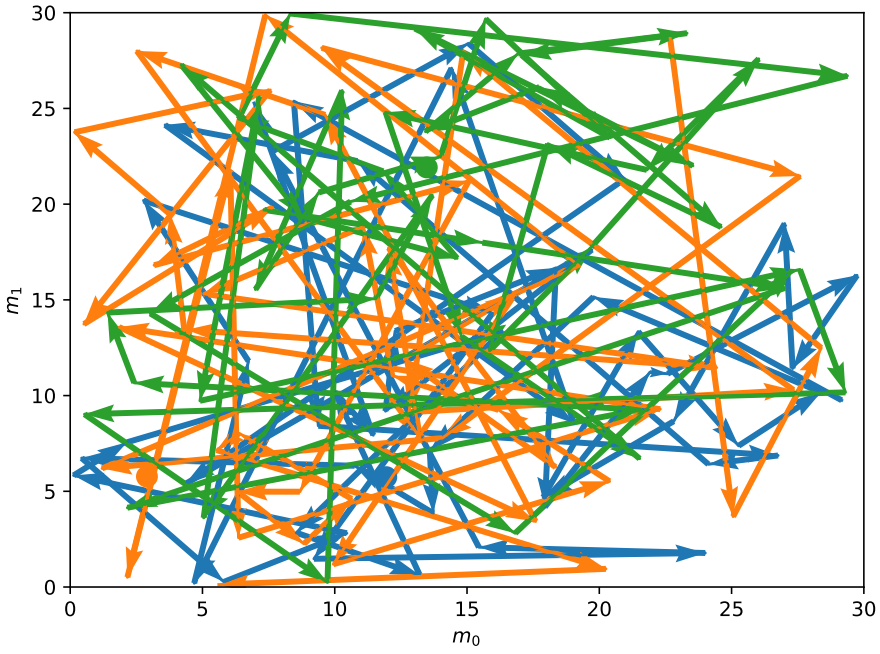

Figure 9: **Parameter space trajectories for three exemplary curves of the hyperopt optimization.** The curves correspond to the ones shown in fig 8 with similar color coding. Arrows denote the guess trajectories, points are the ground truth values. It is apparent that the hyperopt approach explores a large area in parameter space.

In the following section, we will investigate a more informed method for

updating physical parameters, namely neural networks.

4.3 Neural network

In this section, we investigate the performance of our approach when using a neural network as a prediction module. To keep the computational efficiency high, our goal was to use the simplest possible network architecture, as this will cut the time required for a forward pass. We found, however, that the most simple architecture given by several convolutional and pooling layers, was insufficient to achieve training progress. As a consequence, we changed our network architecture to closely resemble a ResNet architecture [13], however, we use 3D convolutional layers to accomodate for the time series collection of pictures. The input to our network has the shape $(B \times T \times C \times Y \times X)$, where B is the batch size, T the length of the concatenated time series (usually $2 \times 30 = 60$), C the number of color / object channels and $Y \times X$ the resolution of the frames. We furthermore found that in our case, dropout regularisation leads to more stable results as compared to the usually employed batch normalisation. In comparison to the hyperopt approach above, the network predicts a correction of the parameters that go into the simulator Δp_i (as compared to the absolute value p_i). The updated value can then be calculated by summing the old value with the parameter update $p_i^{t+1} = p_i^t + \Delta p_i^t$.

4.3.1 Training

To estimate the performance of the network, we train it on simulated data. For this, we use randomised input parameters for guess and ground truth simulations, and use the calculated differences between the input parameters as labels. We implemented the network with the PyTorch library [24]. We minimise the mean squared error between ground truth and perform backpropagation with the ADAM optimizer [25]. For predicting only the masses of the objects as free parameters, we found good convergence when using a batchsize of 20 for 6500 iterations, see Figure 10.

4.3.2 Performance

To evaluate the performance of the neural network, we perform iterative prediction of the parameter changes. For this, we randomly select ground truth and guess masses and predict the parameter changes several times. The resulting minimum mean squared error between the predicted and (updated) guess masses can be seen in the upper panel of Figure 11. The lower panel in the same Figure shows the mean minimum mean squared error between the masses after each iteration, excluding the randomised initial guess. Error bars denote the error of the mean. We again chose the minimum mean squared error of the masses as a measure of accuracy, as it is directly connected to the readily accessible mean squared error between the time series frames.

In comparison to the hyperopt results shown in Figure 8, the neural network approaches much faster to its final value. Already after two iterations, the network converges towards a constant value within error bars. This fast conversion is also highlighted in the parameter-space trajectory shown in Figure 12. Here, it can be seen that the first step traverses the majority of the distance between

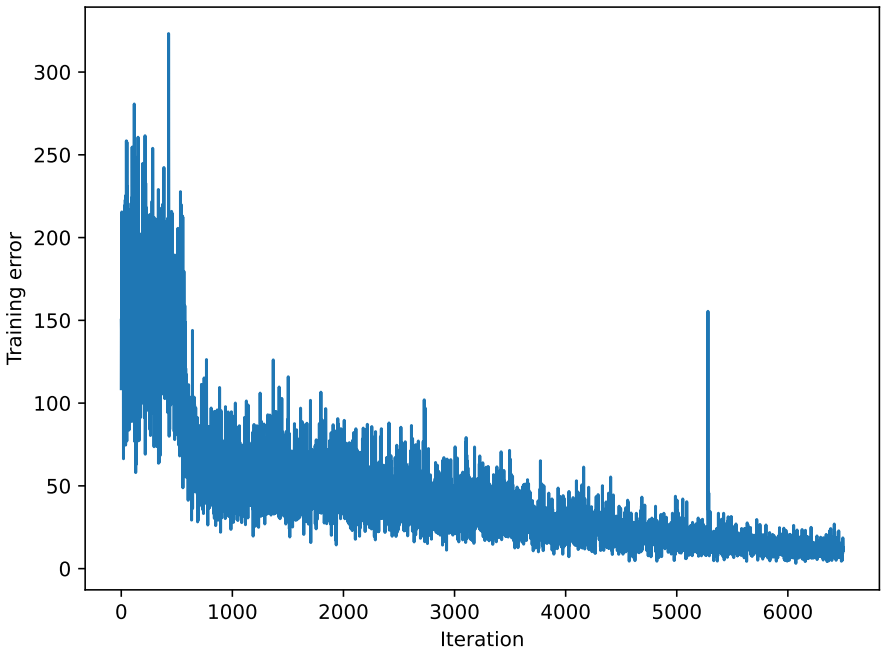

Figure 10: **Example loss trajectory during training.** We use the ADAM optimizer with a learning rate of 5×10^{-6} for 6500 iterations with a batch size of 20.

the start point and target point. In some cases,the final point lies at an offset, which may be caused by finite resolution of the frames.

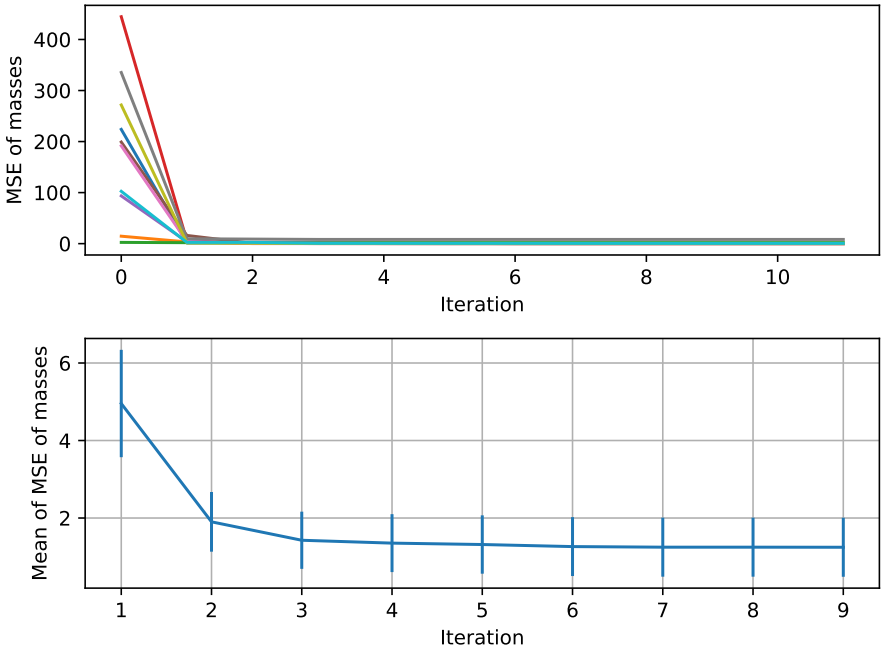

Figure 11: **Performance of a neural network as prediction module.** Top: Mean squared error of the estimated masses for 10 different scenes (different colors). We show the mass error that corresponds to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes, with standard error. Compared to the hyperopt approach shown in Figure 8, the neural network needs significantly less iterations to achieve lower error.

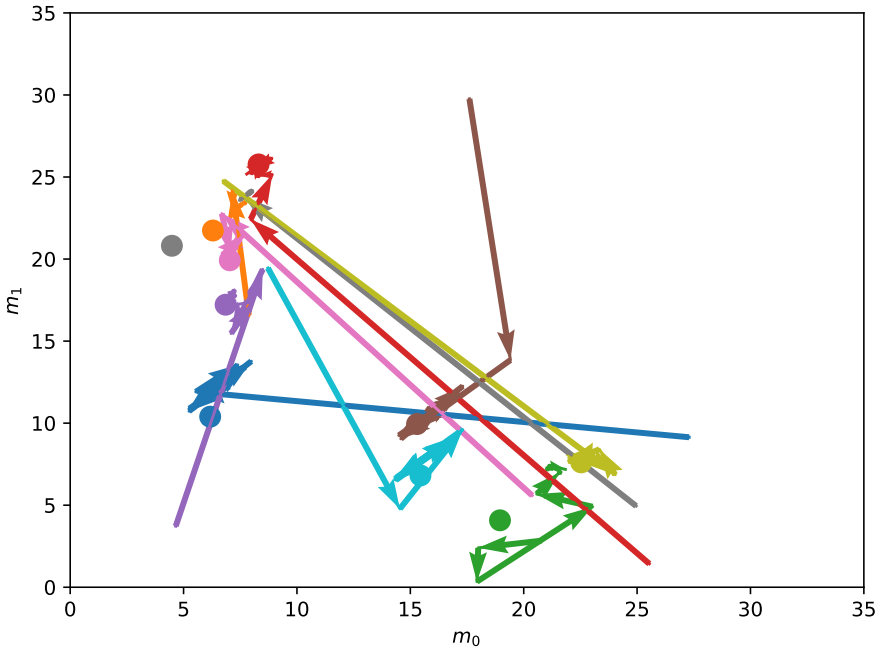

Figure 12: **Parameter space trajectories for the neural network.** Curves correspond to the ones shown in Figure 11, using the same color coding. Arrows show the trajectory of the guesses and points correspond to the ground truth values. In comparison to the hyperopt approach shown in Figure 9, the neural network does not need to explore the parameter space, but converges quickly to its final value, around which it oscillates afterwards. In some cases, the network has an offset to its final mass, most likely due to resolution effects discussed in section 3.3.

4.3.3 Second order processes

One possible complication compared to our baseline scene might be second order interactions between objects. This happens whenever an object does not directly interact with the test object, but with a different object that is also subject to optimization. In this situation the guess for the second object depends on the guess for the first object. While the guess for the second mass should improve once the guess for the first mass improves, it is worth to investigate how this affects the performance of the network. To investigate the effect of second order processes, we changed the scene such that the test objects collides with only one object, which is then accelerated towards a second object, see Figure 13. In this case it is guaranteed that the test mass does not have direct interaction with the second mass.

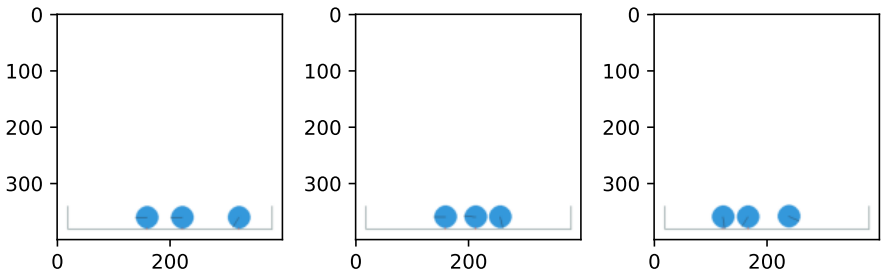

Figure 13: **Second order process experiment.** From left to right: frame 1, frame 15, frame 30 (last frame). The mass (circle) on the right is accelerated towards the two masses in the center. It collides with the middle mass, which then collides with the left mass. In this way, the left mass does not interact directly with the right mass.

We train the prediction module network network with the same hyperparameters and learning rate as before. We perform the same tests as compared to our initial test scene, see Figures 14 and 15. The performance of the network does not suffer from the second order process visibly.

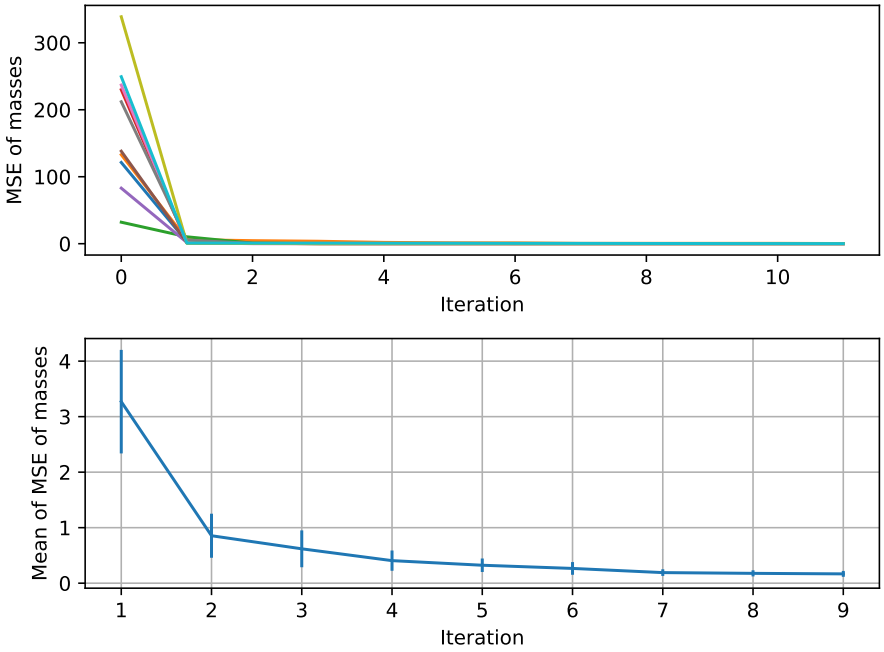

Figure 14: **Performance of the neural network with second order processes.** Top: Mean squared error of the estimated masses for 10 different scenes (different colors). Whe show the mass error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error. The performance is comparable to that of the test scene shown in Figure 11, which leads us to believe that second order processes do not play a large role.

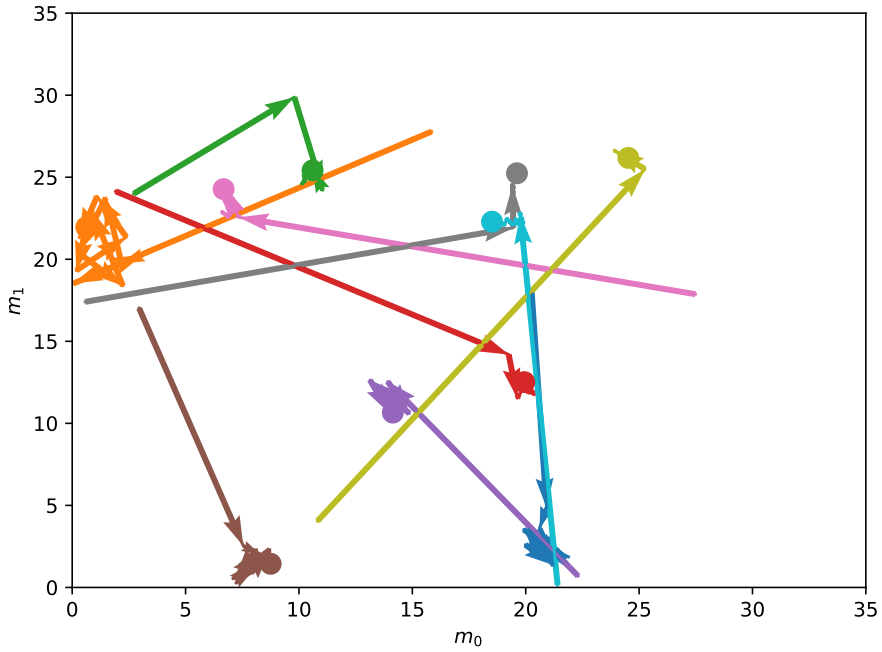

Figure 15: **Parameter space trajectories (arrows) for the second order scene.** The curves corresponds to the ones shown in Figure 14 with same color coding. Points show the ground truth values. Qualitatively this looks similar to the performance in the test scene shown in Figure 12.

4.3.4 Multiple objects

One modification to our presented approach would be the aforementioned parallel prediction of parameter corrections for multiple objects. While this takes away some of the flexibility of our single object approach, it requires $N-1$ less forward passes through the prediction module network, where N is the total number of objects. However, this approach only presents an alternative! when the exact number of objects is known. Alternatively, it might be feasible to train the network to a large number of objects, where all unused object-slots are masked out. In this section, we measure the performance of the multi-object approach and compare it to our single object approach presented above. The performance in this case is shown in Figures 16 and 17. It can be seen that the performance is comparable to the single object case. Predicting the parameters of several objects at the same time therefore might be feasible in situation, where the number of objects is known, in order to achieve less forward passes through the network. However, in our case, the physics simulation is the dominant time factor and increasing the number of forward passes through the neural network does not add much to the total runtime.

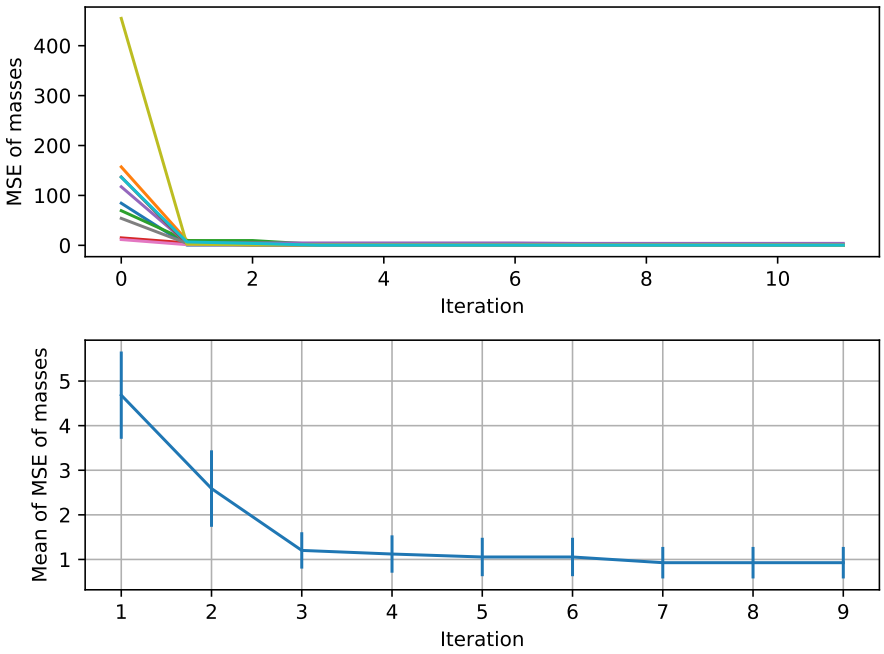

Figure 16: **Performance of the multi-object approach.** Top: Mean squared error of the estimated masses for 10 different scenes (different colors). We show the mass error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error. The performance is comparable to that of the test scene shown in Figure 11, choosing between multi-object detection and single-object detection therefore is tradeoff between speed and flexibility.

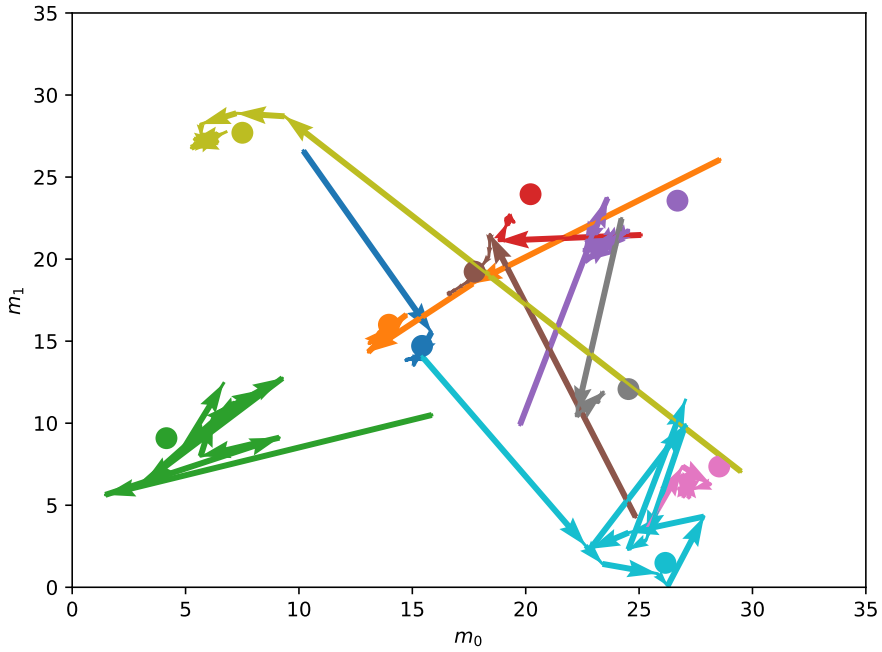

Figure 17: **Parameter space trajectories (arrows) for the multi-object approach.** The curves shown in Figure 16 have the same color coding. Points show the ground truth values. Qualitatively this looks similar to the performance in the dummy scene shown in Figure 12.

4.3.5 Imperfect observation of the scene

To simulate the effect of imperfect observation of the ground truth, we introduce some imperfection to the ground truth channel by applying a gaussian filter with $\sigma = 3$ to each picture of the time series. This blurs out the image and mimics the effect of segmentation errors or a misplaced focus. Example pictures from the imperfect ground truth and a simulated guess are shown in Figure 18. Performance and parameter space trajectories are shown in Figures 19 and 20. It can be seen that blurring the ground truth does not affect the performance of the network at all. This proves that the method is robust against some common error sources when observing the scene with a camera.

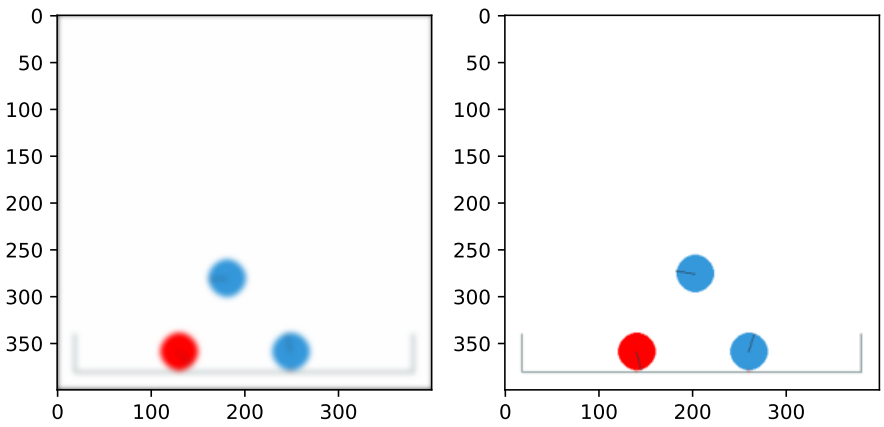

Figure 18: **Example frames from the imperfect ground truth.** A gaussian filter is applied to the ground truth frame on the left. A picture of a guess produced by the physics engine is shown on the right.

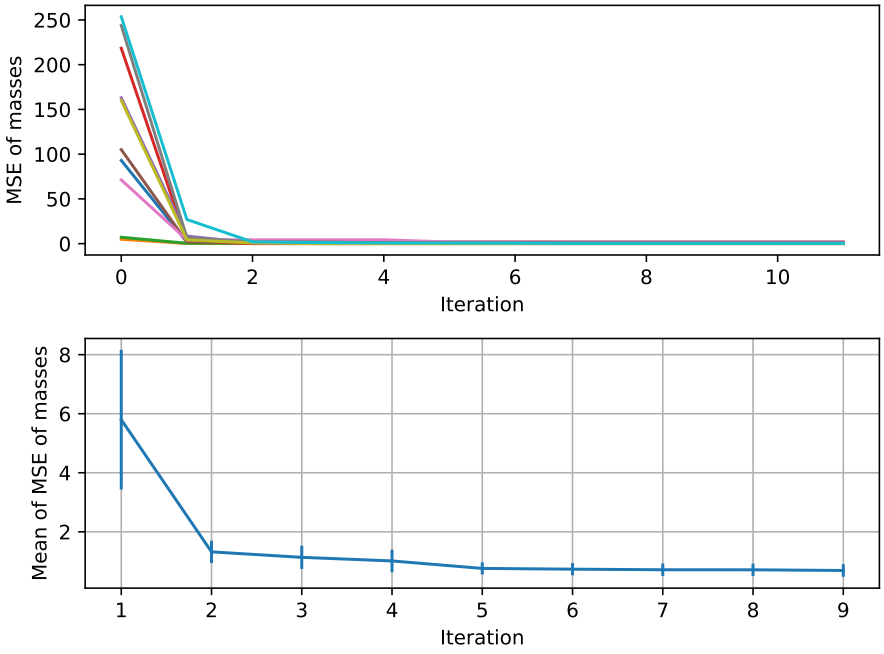

Figure 19: **Performance with imperfect observation.** Top: Mean squared error of the estimated masses for 10 different scenes (different colors). We show the mass error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error. The performance is comparable to that of the test scene shown in Figure 11, which demonstrate robustness agains certain kinds of imaging errors.

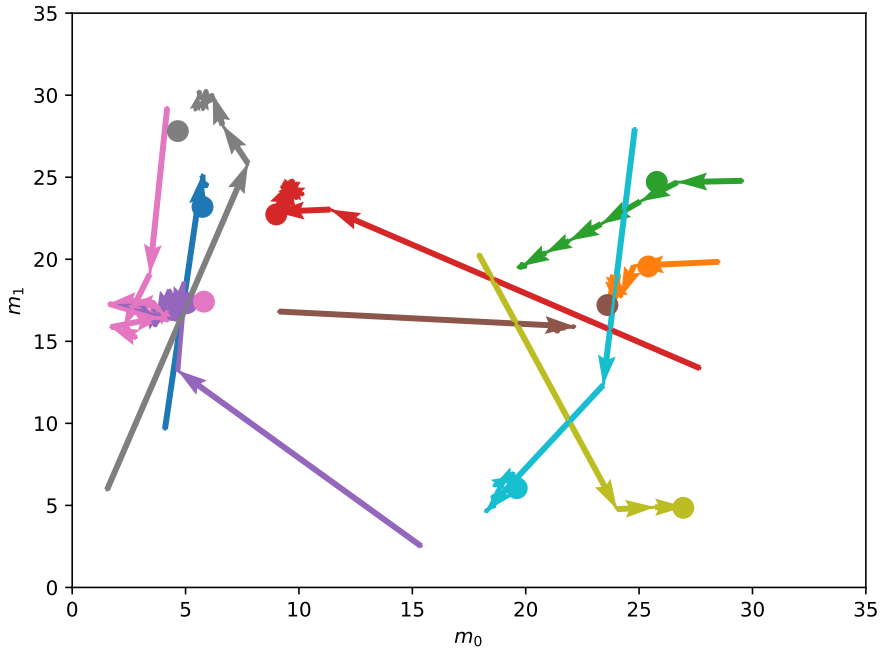

Figure 20: **Parameter space trajectories (arrows) for imperfect observation.** The Curves shown in Figure 19 have the same color coding. Points show the ground truth values. Qualitatively this looks similar to the performance in the dummy scene shown in Figure 12.

4.4 Increasing the complexity with friction and elasticity

In a realistic scenario, the dynamics of the physics is not only determined by a single parameter like the mass we investigated above. Rather, all kinds of different parameters play a role. In this section, we investigate wether our approach can correctly predict the role of elasticity and friction of the objects, while keeping the respective other parameters constant. In the end, we investigate if it is possible to predict all three relevant parameters - mass, elasticity and friction - at the same time.

4.4.1 Predicting other parameters separately

In this section, we investigate the performance of the network, when prediction either elasticity or friction updates. For this, we keep the masses of the test object fixed to $m_{0,1} = 15$. The performance when predicting the elasticity as free parameter is shown in Figures 21 and 22. The results show, that after only a few iterations the error drops significantly. We conclude, that the network is able to predict the elasticity parameter reliably.

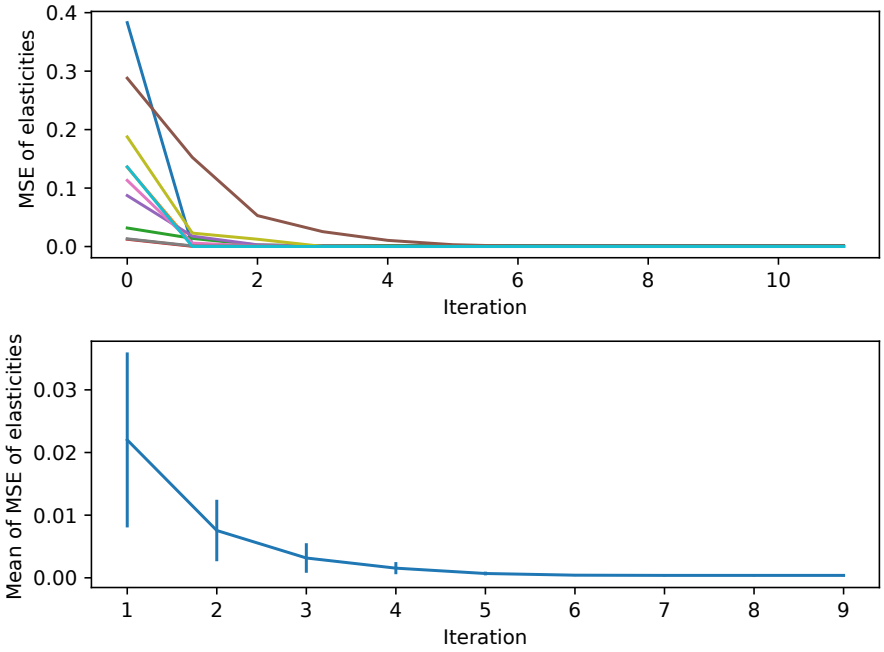

Figure 21: **Performance for elasticity prediction.** Top: Mean squared error of the elasticities for 10 different scenes (different colors). We show the elasticity error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error.

Things are more complicated when predicting the friction parameter, see Figures 23 and 24. In this case, the error drops less significantly, and after ≈ 5 iterations settles around a value of 0.01, which corresponds to an average error of $\delta f = 0.1$, corresponding to 10% of the total range. Interestingly, the param-

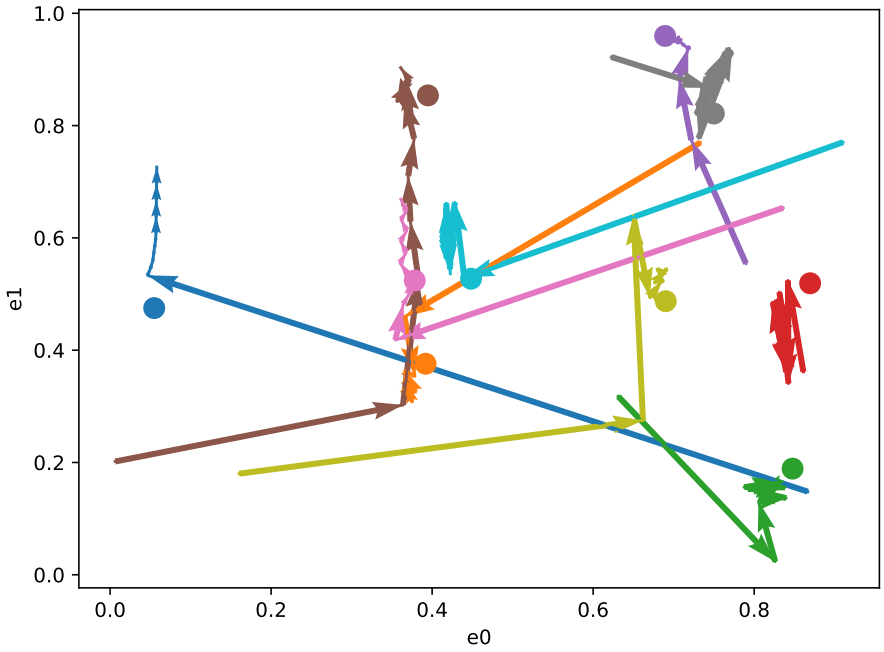

Figure 22: **Parameter space trajectories (arrows) for elasticity prediction.** The curves shown in Figure 21 have the same color coding. Points show the ground truth values.

eter space curves shown in Figure 24 show only small and directionally similar changes per iteration, especially when approaching high values for either f_1 or f_2. This could be a direct consequence of the flatness of the error curve shown in Figure 4. From a physical point of view, the reduced effectiveness in learning the friction parameter can be explained as well: The observed collision in our dummy scene happens between three circles. Each collision - especially those with high elasticity factor - happens on a single point on the circle surface, hence in good approximation the interaction can be modeled by elastic collisions. The following movement of the unknown circles is only weakly dependent on the friction parameter, as simple (ideal) rolling motion does not allow for quantitative analysis of the friction parameters. We expect therefore, that the motion is only weakly dependent on the friction parameter. This, however, could be vastly different when observing more complicated shapes, especially those with flat surfaces.

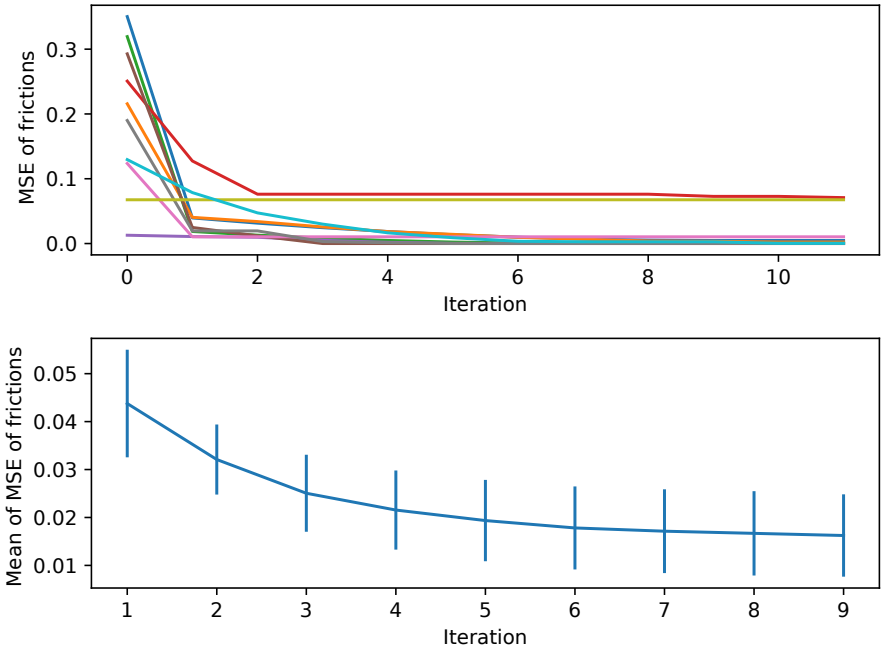

Figure 23: **Performance for friction prediction.** Top: Mean square error of the frictions for 10 different scenes (different colors). We show the friction error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error.

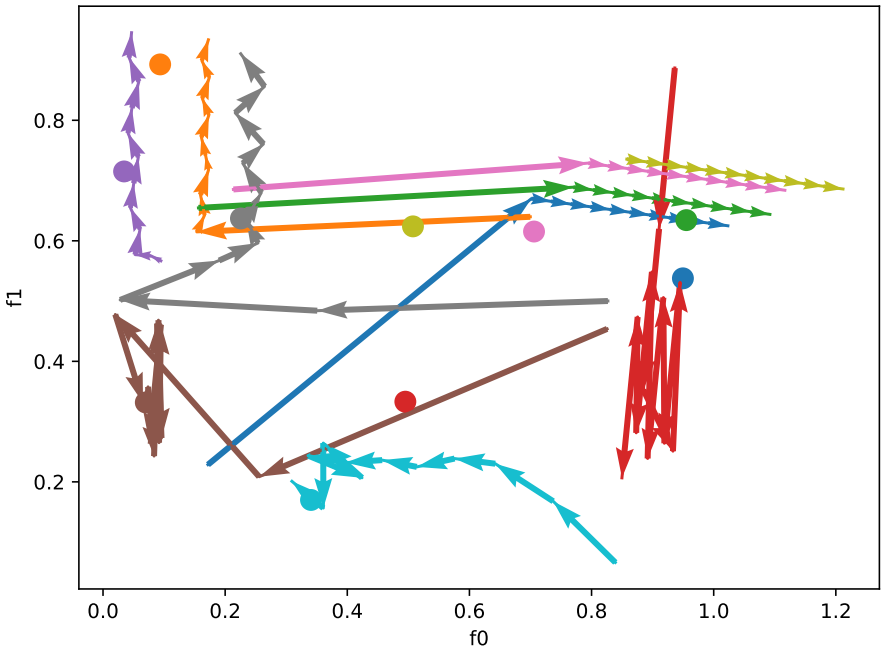

Figure 24: **Parameter space trajectories (arrows) for friction prediction.** The curves corresponds to the ones shown in Figure 23 with same color coding. Points show the ground truth values.

4.4.2 Predicting all parameters simultaneously

Predicting all parameters - mass, friction and elasticity - of the objects simultaneously turned out to be very complicated. We trained a neural network to predict updates to all three parameters. However, while training we found that the error - the mean squared error between masses, elasticities and frictions - did not converge sufficiently to produce usable results. One reason for this may be the difference in order of magnitude between for the parameter updates. While the mass difference is on the order of $\Delta m \approx 10$, the differences of friction and elasticity are on the order of $\Delta e, \Delta f \approx 0.1$. Therefore, using the mean squared error for the parameter updates does focus strongly on the mass difference, but excludes the other two parameters. To circumvent this behaviour, we normalise the mass difference that goes into the mean squared error to its maximum value, which is given by the maximum possible mass:

$$\Delta\tilde{m} = \Delta m / m_{\max}.$$

This restricts the mean squared error between the parameter update values to values between 0 and 1, with no dominating value. Unfortunately, retraining the network with the modified loss values did not lead to improvement. While the error naturally reduces (due to smaller values to begin with), we found that the final value levels at a value that does not allow for reliable prediction of parameter updates. We also found that the network does not predict parameter changes for the masses. The network does attempt to predict the elasticity and friction, but it is also limited by the achievable resolution of the friction parameter. While it cannot be excluded that for more elaborate network architectures it is possible to predict all three parameters simultaneously, we conclude that for our test scene the friction parameter prevents reliable updates. More expressive scenes (especially those that contain sliding motions) might improve this situation. To verify this conclusion, we train our network with both elasticity and mass and investigate whether a lower prediction error is achieved.

4.4.3 Predicting elasticity and mass

Suspecting the friction parameter to limit the achievable performance with the network, we retrained the network to just predict elasticity and mass. Using the normalised loss, however, we again found that the network is discouraged to predict the mass parameter and is limited in its accuracy on the elasticity parameter. We retrained the network with an unnormalised loss to encourage mass prediction. While the network seems to be able to predict the values for both mass and elasticity in some cases, it fails or is inaccurate in many more. We believe that one reason for this might be the predictive limitations of our scene. For more complex scenes, especially in three dimensions, small changes in parameters lead to larger changes in the final state, while in our case, small changes in mass can lead to situations that resemble small changes in elasticity - effectively limiting our prediction accuracy.

4.5 Changing the object shape

In a bin picking scenario our approach should be capable of handing different shapes of objects. So far, in our dummy scene, we tested circular objects. In

this section, we investigate, whether the network is able to determine physical parameter of box shaped objects as well. For this, we modified the scene, seen in Figure 25. The test object with fixed mass is a circle that is accelerated towards two stacked boxes with unknown mass. The goal of the network is to succesfully determine the mass of the boxes.

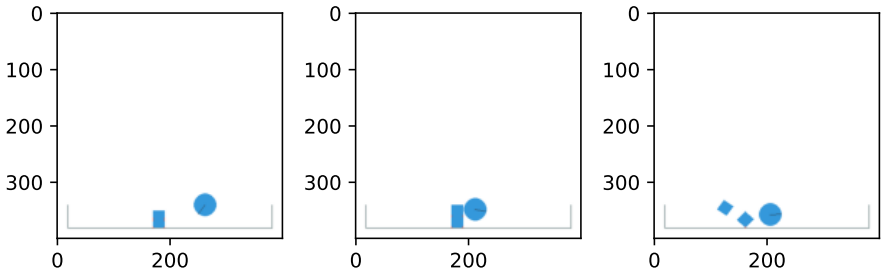

Figure 25: **Modified scene to evaluate the detection for box shaped objects.** Shown are (from left to right) frame 1, 15 and 30.

We train the network on this scene with the same parameters as our dummy scene. The results are shown in Figures 26 and 27. Again, the network is able to determine the mass accurately after very few iterations. However,the network seems to be less accurate for high ground truth masses. Most likely due to less dynamic behaviour of the boxes in this case.

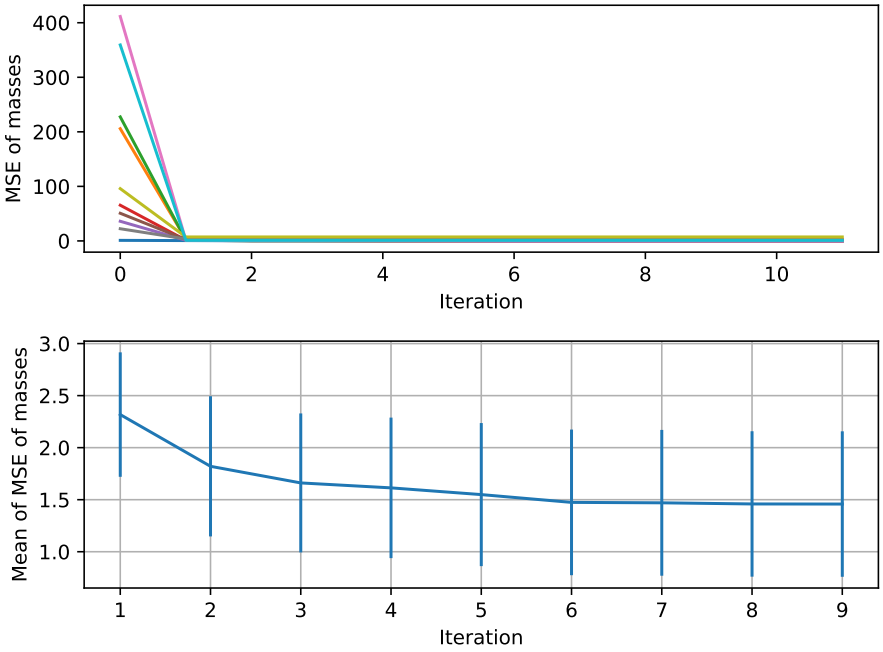

Figure 26: **Performance for mass prediction for box-shaped objects.** Top: Mean squared error of the masses for 10 different scenes (different colors). We show the mass error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error.

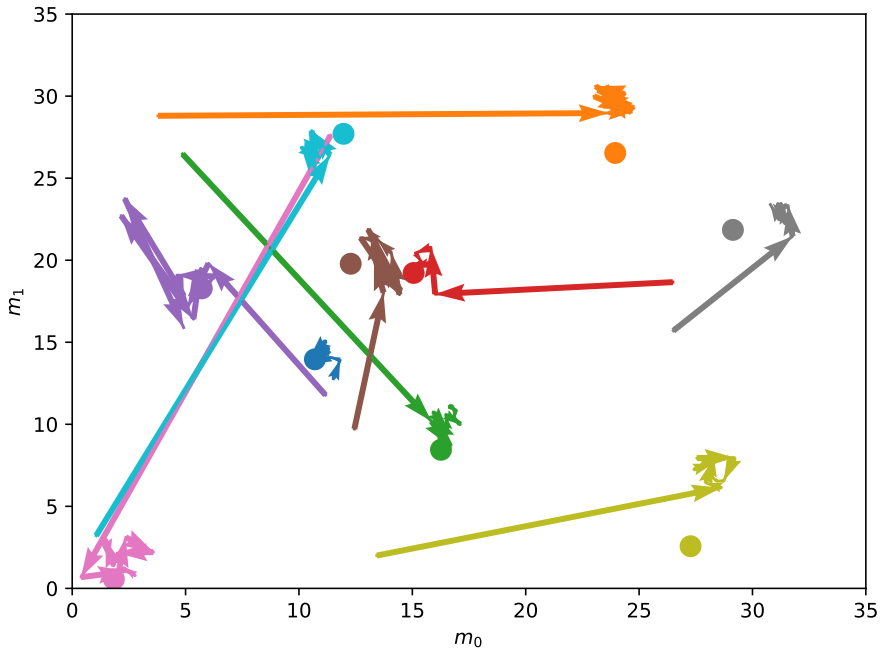

Figure 27: **Parameter space trajectories (arrows) for mass prediction of box shaped objects.** The curves shown in Figure 23 with same color coding. Points show the ground truth values.

4.5.1 Predicting friction for the box scene

In section 4.4.1, we explained that limited predictive capabilities for the friction parameter might be caused by the nature of the used scene. In more detail, we assumed that more or less elastic collision between circle-shaped objects do not allow for an accurate prediction of the friction parameter. To verify that it is indeed possible to predict the friction for more expressive scenes, we retrain our prediction module for the scene using the box-shaped objects. Here, the objects have more flat surfaces, which should lead to more dynamics involving frictional forces. Indeed, we see that the network is able to predict the friction with very high accuracy, see Figures 28 and 29.

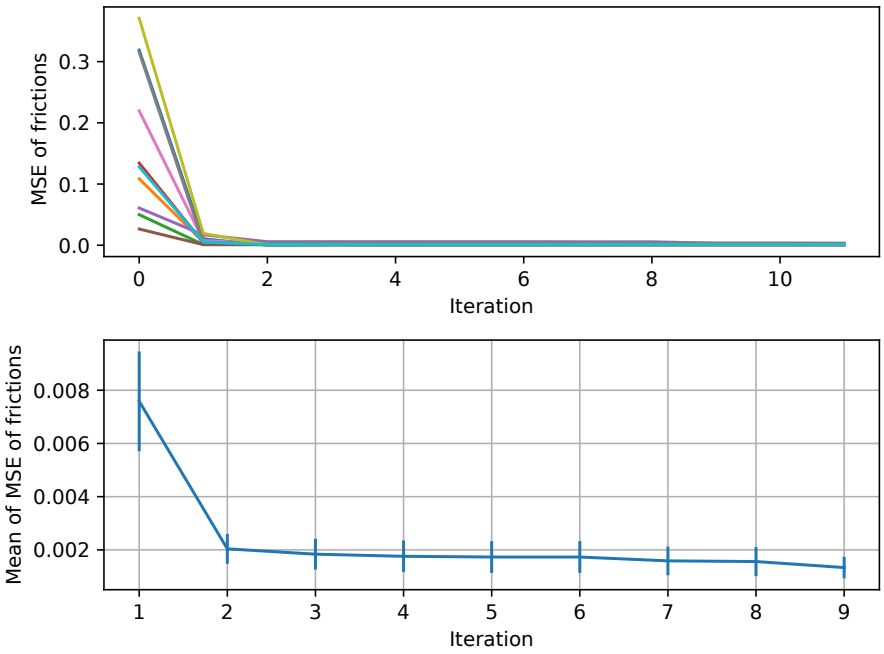

Figure 28: **Performance for friction prediction for box-shaped objects.** Top: Mean squared error of the estimated frictions for 10 different scenes (different colors). We show the friction error corresponding to the minimum visual error up until the current iteration. Bottom: MSE averaged over 10 scenes with standard error.

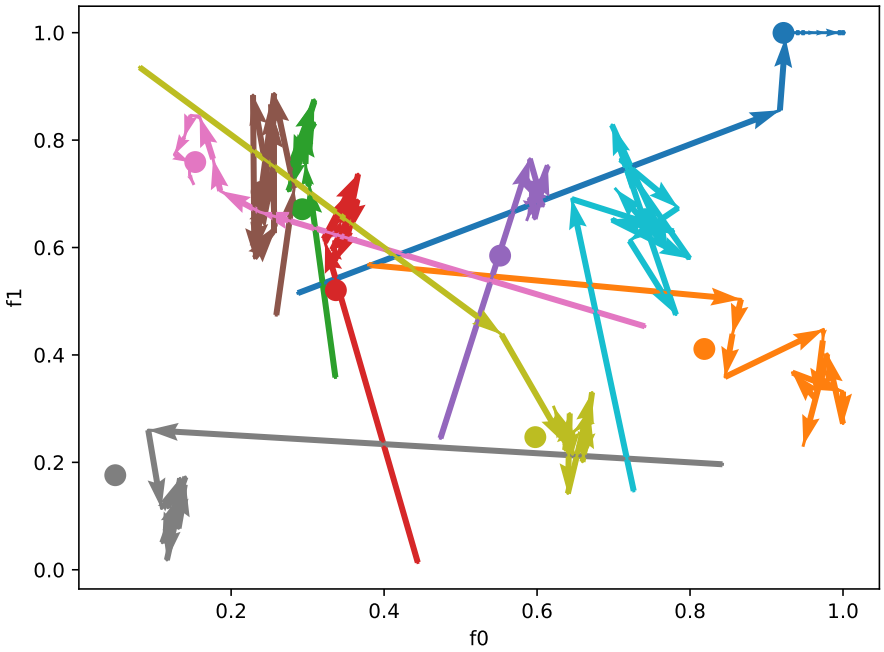

Figure 29: **Parameter space trajectories (arrows) for friction prediction of box shaped objects.** The curves correspond to the ones shown in Figure 23 with same color coding. Points show the ground truth values.

5 Summary

In this thesis we investigated how video observation of objects can be used to infer physical properties of said objects. We developed an iteration-based approach that compares an observed (yet simulated) video feed with a simulated *guessed* video feed. Our approach is strictly modular and decouples the physics simulation from the prediction module that derives the parameters for the guess. This allows flexibility in choosing both modules. To gain an understanding of the performance and limits of our approach, we tested it on a two-dimensional elementary test scene, simulated with the pymunk library. For the prediction module, we compared the hyperparameter-optimisation library *hyperopt* with a trained neural network in ResNet-architecture. We found that while the hyperopt-guesser works reliably, it requires tens of iterations to achieve low error between guess and ground truth. The neural network on the other hand only requires very few forward passes to reach low errors comparable to the observation threshold. In conclusion, the neural network presented itself as the superior approach.

To investigate the robustness of the neural network based prediction module, we investigated several test cases: For one, we investigated the performance for different approaches to the detection of multiple objects. Both simultaneous and seperated prediction for all objects perform similar, with the latter providing more flexibility regarding changing object numbers and the former needing less forward-passes. We also investigated the role of second order processes in which an object with unknown parameters does interact with a test object with known parameters, but only with another unknown objects. In this case we found the network still stable and performing on equal footing. To investigate the role of imperfect observation, we introduced a gaussian filter to the observation. In this case, the network still proved to be stable and performed well.

We also investigated how the network predicts other parameters, namely elasticity and friction. When only predicting one and keeping the other and mass constant, the network proves to be capable of predicting the elasticity. In case of the friction, the network is limited in its predictive power, most likely due to only small changes in the scene for high friction parameters. We assume our scene is not expressive enough in regards to movements that allow to determine the friction parameter.

Finally, we investigated whether our approach is able to determine physical properties of differently shaped objects, namely boxes. By creating a modified test scene, we observed that the performance when predicting the mass parameter is comparable to the other scene. In agreement with the assumption that scenes with flat surfaces will increase the performance for friction prediction, we found that this scene allows for an accurate determination of the friction parameter.

In conclusion, we found that our approach with seperated physical simulation and prediction module allows for an efficient prediction of physical parameters for two-dimensional scenes. However, the expressiveness of a scene with regards to certain parameters plays a big role. It is therefore imperative to provide our approach with as much meaningful data as possible - for example by extending and updating the time series on a rolling basis.

6 Outlook

While in this thesis we layed the foundation for predicting physical properties from video feeds, more tests and extensions have to be undertaken in order to apply our approach to a real world bin picking scenario. In this section, we give a brief outlook of the next steps towards applicability.

6.1 Increasing the scene complexity

While our test scenes capture elementary processes of a scene, the state-space is limited. One interesting future investigation may explore the performance of our approach in scenes with a (much) larger number of objects. Pymunk is easily capable of simulating hundreds of objects. This will increase the state space significantly and will allow us to judge whether the network is really able to capture the underlying physics, without overfitting. This test also allows us to evaluate how the duration learning process scales with complexity. Additionally, the duration of the scene can be extended and the iteration can be applied with a sliding window approach to allow for accumulation of additional information over time.

6.2 Generalisation to three dimensions

In this thesis, we investigated our approach using two dimensional scenes to understand the capabilities and limits, however, bin picking naturally happens in a three dimensional environment. Extending to three dimension comes with several caveats: For one, observing a scene with a video feed becomes more difficult as information is projected onto a two dimensional picture which necessarily leads to loss of information. This loss is not present in the two dimensional case, where the depth dimension does not exist. This also brings with it the possibility of overlapping objects, which makes segementation more difficult. However, segmentation of objects via pictures of three dimensional scenes has been demonstrated succesfully [1, 2, 3]. Secondly, the number of possible parameters becomes more complicated: Density, moment of inertia, center-of-mass all require more information as compared to the two-dimensional case. Lastly, the simulation-aspect becomes more involved, as a three dimensional physics engine is required. In our case, we are looking to use Stillleben to generate scenes, which has been developed in our group [11].

6.3 Simulating bin picking

While working on three-dimensional scene itself is interesting, the final use case explicitly needs to take into account a robotic arm that performs the picking process. While interacting with a *test object* provides us with information to determine the physical properties of other objects, in reality, this interaction will be caused by the robot arm and gripping process. It remains to be investigated how the gripping process allows for predictions and what role failure cases (failed gripping attempts) can play in this context. In our group, there is active work on creating a dataset that can be used for this purpose and includes these failure cases as well [26].

6.4 Real world performance

Finally, it remains to be seen how our approach generalises from simulated training data to real-world video feeds. While Stillleben is equipped to simulate realistic video feeds, evaluating the performance on real feeds is an important task and necessary to evaluate real-world applicability. While this was not planned in the scope of this bachelors thesis, future investigations with the developed framework will show how the performance scales to real use cases.

Acknowledgments

First and foremost I would like to thank Prof. Dr. Sven Behnke for allowing me to work on this project and for being so flexible regarding the conditions of my work in the research group. I really appreciate the scientific spirit in the group and learned many new things.

Many thanks also to Dr Hassan Errami, who acted as second examiner for this thesis.

A big thank you to Max Schwarz, who closely supervised me on this project. I learned a lot from you and it was always a pleasure talking to you.

Thank you to Jan Quenzel, who is working on a different project with me and is very patient with my questions.

Finally, I would like to thank the entire autonomous intelligent systems research group. While due to the pandemic I was not able to get to know many of you as good as I would've liked, I appreciated anyone who I got to interact with!

References

[1] Max Schwarz, Anton Milan, Christian Lenz, Aura Muñoz, Arul Selvam Periyasamy, Michael Schreiber, Sebastian Schüller, and Sven Behnke. Nimbro picking: Versatile part handling for warehouse automation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3032–3039, 2017.

[2] Max Schwarz, Christian Lenz, German Martin Garcia, Seongyong Koo, Arul Selvam Periyasamy, Michael Schreiber, and Sven Behnke. Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[3] Max Schwarz, Christian Lenz, German Martin Garcia, Seongyong Koo, Arul Selvam Periyasamy, Michael Schreiber, and Sven Behnke. Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[4] Carlos Aguilar-Ibañez. Stabilization of the pvtol aircraft based on a sliding mode and a saturation function. *International Journal of Robust and Nonlinear Control*, 27(5):843–859, 2017.

[5] José de Jesús Rubio, Enrique Garcia, Gustavo Aquino, Carlos Aguilar-Ibañez, Jaime Pacheco, and Alejandro Zacarias. Learning of operator hand movements via least angle regression to be teached in a manipulator. *Evolving Systems*, pages 317–332, June 2020.

[6] J. A. Meda. Estimation of complex systems with parametric uncertainties using a jssf heuristically adjusted. *IEEE Latin America Transactions*, 16(2):350–357, 2018.

[7] 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*, pages 5026–5033, 2012.

[8] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A Differentiable Physics Engine for Deep Learning in Robotics. *arXiv:1611.01652 [cs]*, November 2018. arXiv: 1611.01652.

[9] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-End Differentiable Physics for Learning and Control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 7178–7189. Curran Associates, Inc., 2018.

[10] NVIDIA PhysX. https://github.com/NVIDIAGameWorks/PhysX.

[11] Max Schwarz and Sven Behnke. Stillleben: Realistic Scene Synthesis for Deep Learning in Robotics. *arXiv:2005.05659 [cs]*, May 2020. arXiv: 2005.05659.

[12] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep Iterative Matching for 6D Pose Estimation. *International Journal of Computer Vision*, 128(3):657–678, March 2020. arXiv: 1804.00175.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[14] Jiajun Wu, Joseph Lim, Hongyi Zhang, Joshua Tenenbaum, and William Freeman. Physics 101: Learning physical object properties from unlabeled videos. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 39.1–39.12. BMVA Press, September 2016.

[15] *Deep Learning Convolutional Neural Networks to Predict Porous Media Properties*, volume Day 1 Tue, October 23, 2018 of *SPE Asia Pacific Oil and Gas Conference and Exhibition*, 10 2018. D012S032R010.

[16] Muhammad K. Hamdan, Diane T. Rover, Matthew J. Darr, and John Just. Mass estimation from images using deep neural network and sparse ground truth. In M. Arif Wani, Taghi M. Khoshgoftaar, Dingding Wang, Huanjing Wang, and Naeem Seliya, editors, *18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019, Boca Raton, FL, USA, December 16-19, 2019*, pages 1987–1992. IEEE, 2019.

[17] Higor Sigaki, Ervin Lenzi, Rafael Zola, Matjaz Perc, and Haroldo Valentin Ribeiro. Learning physical properties of liquid crystals with deep convolutional neural networks, 04 2020.

[18] Wenzhen Yuan, Chenzhuo Zhu, Andrew Owens, Mandayam A. Srinivasan, and Edward H. Adelson. Shape-independent hardness estimation using deep learning and a gelsight tactile sensor. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 951–958, 2017.

[19] Pymunk. http://www.pymunk.org/.

[20] Chipmunk. http://chipmunk-physics.net/.

[21] Pygame. https://www.pygame.org/.

[22] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proc. of the 30th International Conference on Machine Learning*, 2013.

[23] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang,

Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[26] Arul Selvam Periyasamy, Max Schwarz, and Sven Behnke. Synpick: A dataset for dynamic bin picking scene understanding. *17th IEEE International Conference on Automation Science and Engineering (CASE)*, 2021.