Visual De-Animation for Spheres on Planar Surfaces

Jonathan Gustafsson Frennert

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2023

Abstract

We present a new visual de-animation model for spheres on planar surfaces, demonstrating high performance in object detection, object parameter estimation, scene prediction, and reconstruction. Our model improves upon the original visual de-animation model by using circularity-based object proposal generation and optical flow for extrinsic parameter estimation, as opposed to colour filters and a neural network for both intrinsic and extrinsic parameters. This approach results in increased robustness and precision, as circularity is a more direct detection method for spheres and extracting velocity directly from optical flow reduces the risk of estimation corruption during CNN feed-forward processing.

The model exhibits impressive results for object proposal generation, as well as strong performance in scene reconstruction and future prediction. The blob detection method used for object proposal generation proves effective for identifying spheres regardless of size and colour. Further, our results suggest that a neural optical flow network is not necessary for high model performance.

We provide open-source utilities for training, evaluation, and testing of our model, as well as a new data set generator for spheres on planar surfaces using PyBullet, allowing for easy experimentation with a structured generation process.

Research Ethics Approval

Instructions: This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jonathan Gustafsson Frennert*)

Acknowledgements

Firstly, I would like to thank my supervisor, Professor Subramanian Ramamoorthy for his support and guidance throughout this project. Without his goal-oriented perspective and wisdom, much more of my time would have been wasted on fruitless rabbit holes.

Secondly, I would like to thank all the friends I have made during university, who have given me memories I will treasure for the rest of my life, and useful input on my project.

Finally, I would like to thank my parents, for their unconditional support, and always being there for me when I need it most.

Table of Contents

Chapter 1

Introduction

Humans have an intuitive understanding of their physical environment and interactions. This allows us to quickly adapt to scenarios where we only have partial information and leverage it to achieve our desired outcomes. Machines struggle when it comes to these scenarios, generally requiring full information to achieve high performance. Inspired by this, we seek to imbue robots with an intuitive understanding of physical scenes. In a practical sense, human scene understanding is

1. **Predictive.** Our understanding allows us to make causal connections and infer how a scene is expected to evolve into the future. This suggest that an intuitive model should estimate the physical properties of individual and grouped objects in the scene.
2. **Fast.** Little thought is required for us to generate scene representations and a plan for manipulating them to achieve short-step goals, e.g., pick-and-place, or rigid object reorientation. This suggest that an intuitive model should be single-pass feed-forward.

A promising approach, satisfying both these properties, was found using an inverse graphics model to construct the scene representation and physics engine to imagine how the scene would move forward in time [49]. The inverse graphics engine relies on convolutional neural networks for efficient object detection, structure and property estimation to do scene reconstruction. Remaining scene properties are then inferred by forward simulation of the physics engine, which can be non-differentiable [9] or neural differentiable [8].

We propose an extension of this model, which relies more on optical flow for efficient property estimation. We train and demonstrate our system on a synthetic data set, which we generate, involving spheres of varying physical properties. The system accurately predicts the motion of the spheres by estimating their properties from a small number of interactions, capturing the predictive and fast nature of human scene understanding. Our results show the promise of this approach in advancing the field of intuitive physics towards more human-like scene understanding.

1.1 Dissertation Structure

In Chapter 2, we state the problem for spheres on planar surfaces, the theory behind visual de-animation, other approaches to intuitive physics, and the data sets used for training and benchmarking intuitive physics models. In Chapter 3 we present our visual de-animation model for spheres on planar surfaces. In Chapter 4 we introduce the synthetic data set, evaluation metrics and model implementation details. In Chapter 5, we discuss metric and qualitative results on the data set, including an ablation study. This is followed by a summary and final discussion of results in Chapter 6, after which potential future work and a plan for MInf Part 2 is proposed.

1.2 Contributions

1. Visual de-animation has been introduced and discussed in the context of spheres on planar surfaces.
2. New data set and simulation environment for spheres on planar surfaces created using PyBullet.
3. Visual de-animation model has been proposed which achieves high performance for spheres on planar surfaces with varying physical properties.
4. Extensive ablation study has confirmed the usefulness of visual de-animation model.
5. A new repository for visual de-animation has been created. As far as we know, the only open source implementation based on [49]. Allowing for data generation, multi-GPU training, training and test visualisations. It can be found at `https://github.com/J0HNN7G/intuitive_physics`.

Chapter 2

Background

2.1 Problem Definition

Let there be n uniformly rigid matte spheres intrinsically parameterised:

- Radius, $r \in \mathbb{R}^+$, in metres.
- Surface colour, $\vec{c} = \begin{bmatrix} c_R & c_G & c_B \end{bmatrix}^T, c_R, c_G, c_B \in [0,1]$. Represented in RGB colour space.
- Mass, $m \in \mathbb{R}^+$, in kilograms.
- Rolling friction coefficient, $\mu_r \in \mathbb{R}^+$, affects the resistance to rolling motion of an object as it moves along a surface.
- Lateral friction coefficient, $\mu_l \in \mathbb{R}^+$, affects the resistance to lateral motion of an object as it moves along a surface.
- Spinning friction coefficient, $\mu_s \in \mathbb{R}^+$, affects the resistance to rotation of an object that is in contact with another surface.
- Restitution coefficient, $b \in [0,1]$. Determines the ratio of the relative velocities of two colliding objects after the collision to their relative velocities before the collision, ranging from 0 (objects stick together) to 1 (objects bounce perfectly without losing energy).

and extrinsically parameterized in Cartesian coordinates at time step t by

- World frame pose $\vec{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ where $p_x, p_y, p_z \in \mathbb{R}^3$ are in metres. We will let origin O be the center of the plane on which the motion of spheres occurs.
- Linear velocity $\vec{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$ where $v_x, v_y, v_z \in \mathbb{R}^3$ are in metres per second.

We will order the spheres, so that $\vec{p}(i,t)$ refers to the world pose vector for the i-th sphere at time step t for a given simulation of the spheres. Further, when discussing estimates, we will differentiate between ground truth ($\texttt{var}^*$) and estimate ($\texttt{var}'$) using $*$ and $\prime$ in superscript.

These are the parameters used to define the dynamics of a sphere in PyBullet, the simulation software we are generating a synthetic data set to train and test visual de-animation [9]. For our experiments, we will assume that these parameters are sufficient to accurately capture the dynamics of a simulation involving several moving spheres. In the real-world, it is possible that further parameters are required.

The initial configuration for any i-th sphere in any simulation is

$$\vec{p}(i,1) = \begin{bmatrix} p_x(i,1) \\ p_y(i,1) \\ r(i) \end{bmatrix}, \tag{2.1}$$

where $p_x(i,1), p_y(i,1)$ are sampled uniformly such that the sphere does not overlap with any previously initialized spheres,

$$p_x(i,1), p_y(i,1) \sim U(-0.5 + r(i), 0.5 - r(i)), \tag{2.2}$$

$$\text{s.t. } (p_x(i,1) - p_x(j,1))^2 + (p_y(i,1) - p_y(j,1))^2 > (r(i) + r(j))^2 \tag{2.3}$$

$$\forall j = 1, 2, \ldots, i-1, \tag{2.4}$$

and

$$\vec{v}(i,1) = \begin{bmatrix} v_x(i,1) \\ v_y(i,1) \\ 0 \end{bmatrix}, \tag{2.5}$$

where $v_x(i,1), v_y(i,1)$ are sampled uniformly

$$v_x(i,1), v_y(i,1) \sim U(-2.5, 2.5) \tag{2.6}$$

The simulation environment is an open cube centered at origin with a floor plane at $z = 0$ and boundaries at $x = -0.5, x = 0.5, y = -0.5, y = 0.5$ (Figure 2.1). These conditions were chosen to be similar to how spheres would behave on a pool table, as it provides a straightforward source of real world data on which visual de-animation model can be evaluated on, although this is not done in the dissertation [1].

As the simulation evolves, the extrinsic properties of the spheres will be effected by the spheres interactions with their environment and each other based on their intrinsic properties, with all spheres having reached stationary positions by some time step t_f.

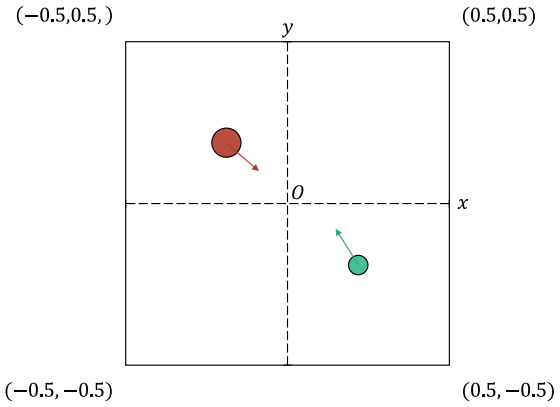

Figure 2.1: Simulation environment with two spheres from top-down view including velocity vectors arrows, axes (x, y), origin O, and boundary dimensions.

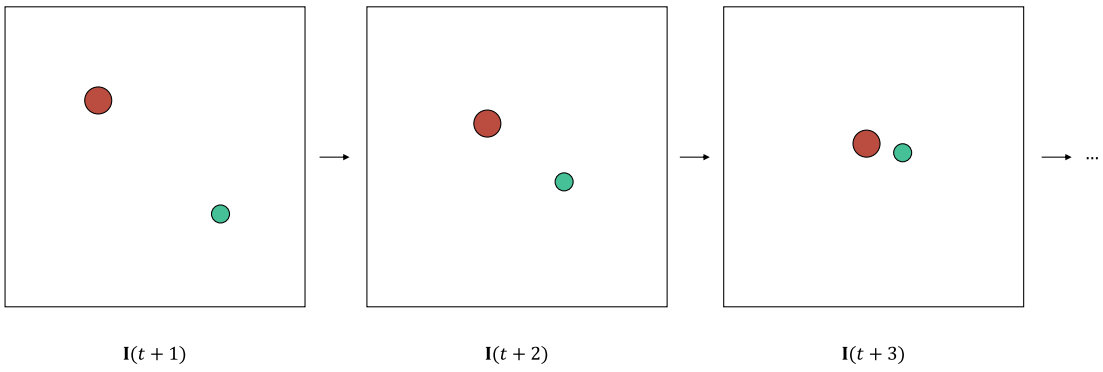

Figure 2.2: Image sequence from a simulation with two spheres.

As the spheres move on a planar surface, we can make the assumption that

$$p_z(i,t) = r(i), \quad v_z(i,t) = 0, \quad \forall t \in \mathbb{Z}^+ \tag{2.7}$$

As part of the problem, at each time step t, there is an RGB image $\mathbf{I}(t) \in [0,1]^{3\times h\times w}$ of the simulation from a top-down view, so that the boundary of the image corresponds to the boundaries of the box (Figure 2.2). Where h is the height of the image in pixels and w is the width of the image in pixels. Specifically for images, we will use $\mathbf{I}(i)[x,y]$ to denote pixel vector at row x and column y.

Our primary optimization problem (F_p), is as follows, given images from q consecutive time steps starting from time step $t+1$ such that $1 < q$ and $t+q < t_f$, estimate the intrinsic parameters θ_i and extrinsic parameters $\theta_e(t+q)$ at time step $t+q$ for all spheres so that

$$\theta_i = \begin{bmatrix} \vec{\theta}_i(1) \\ \vec{\theta}_i(2) \\ \vdots \\ \vec{\theta}_i(n) \end{bmatrix} = \begin{bmatrix} r(1) & \vec{c}(1)^T & m(1) & \mu_r(1) & \mu_l(1) & \mu_s(1) & b(1) \\ r(2) & \vec{c}(2)^T & m(2) & \mu_r(2) & \mu_l(2) & \mu_s(2) & b(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r(n) & \vec{c}(n)^T & m(n) & \mu_r(n) & \mu_l(n) & \mu_s(n) & b(n) \end{bmatrix} \tag{2.8}$$

$$\theta_e(t+q) = \begin{bmatrix} \vec{\theta}_e(1,t+q) \\ \vec{\theta}_e(2,t+q) \\ \vdots \\ \vec{\theta}_e(n,t+q) \end{bmatrix} = \begin{bmatrix} \vec{p}(1,t+q)^T & \vec{v}(1,t+q)^T \\ \vec{p}(2,t+q)^T & \vec{v}(2,t+q)^T \\ \vdots & \vdots \\ \vec{p}(n,t+q)^T & \vec{v}(n,t+q)^T \end{bmatrix} \tag{2.9}$$

$$J_p(\theta_i', \theta_e'(t+q)) = \frac{1}{n(t_f-(t+q))} \sum_{j=1}^{n} \sum_{k=t+q}^{t_f} \left\| \vec{p}^*(j,k) - \vec{p}'(j,k,\theta_i',\theta_e'(t+q)) \right\|^2 \tag{2.10}$$

$$(F_p) \quad \min_{\theta_i', \theta_e'(t+q)} J_p(\theta_i', \theta_e'(t+q)) \tag{2.11}$$

Where $\vec{p}'(j,k,\theta_i',\theta_e'(t+q))$ is the estimated world pose of the j-th sphere at time step k using parameters θ_i' and $\theta_e'(t+q)$ (Figure 2.3). For our case, $\vec{p}'(j,k,\theta_i',\theta_e'(t+q))$ is calculated by inputting θ_i' and $\theta_e'(t+q)$ into a physics engine at time step $t+q$ and recording pose at time step k. To make the problem more tractable given we only access

top-down images, we will focus on estimating only $r, \vec{c}, m, \mu_r$ for intrinsic properties, letting the rest be known constants.

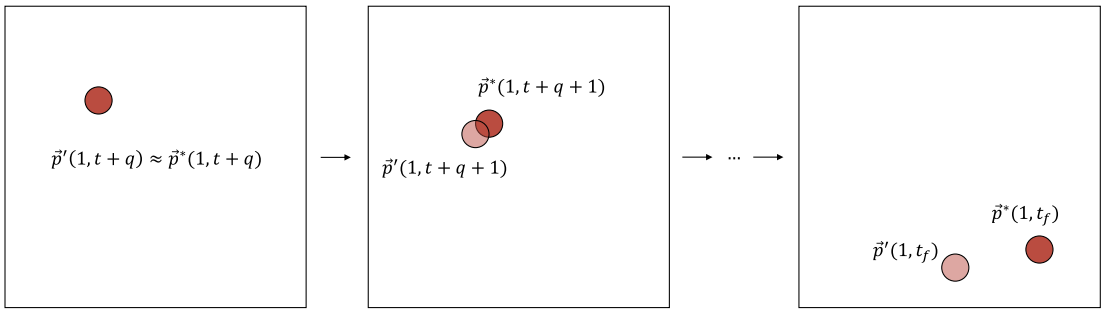

Figure 2.3: Simulation with single sphere from time step $t+q$ to t_f, where estimated world pose $p'(1)$ is visualized with actual world pose $p^*(1)$ as they diverge.

The feasible region is constrained by the range of values that each intrinsic and extrinsic parameter can take. The range of possible values for the intrinsic parameters are stated in the beginning of the section. For the extrinsic parameters, the pose is constrained by the environment:

$$-0.5 < p'_x(j,t+q), p'_y(j,t+q) < 0.5 \tag{2.12}$$
$$p'_z(j,t+q) = r'(j) \tag{2.13}$$
$$\forall j = 1,2,\ldots,n \tag{2.14}$$

For the velocity, a lower or upper bound constraint is less apparent, as the spheres can transfer energy in such a way that velocity for a given sphere can potentially increase beyond its initial value by time step $t+q$.

Trivially, the set of optimal solutions is,

$$S^* = \{(\theta_i^*, \theta_e^*(t+q))\} \tag{2.15}$$

such that

$$J_p(\theta_i^*, \theta_e^*(t+q)) \approx 0. \tag{2.16}$$

We can produce an upper bound on the objective function given our environmental constraints, as they constrain the maximum possible radius of a sphere ($r_{\max} = 0.5$), and the maximum displacement along x, y axis between two spheres. The maximum displacement between the actual pose of a sphere and its pose estimate is when the actual sphere is positioned in a corner and its pose estimate is positioned in the corner on the opposite side (Figure 2.4),

$$\left\|\vec{p}^*(j,k) - \vec{p}'(j,k,\theta'_i,\theta'_e(t+q))\right\| < \sqrt{1^2+1^2+0.5^2} = \frac{3}{2}, \quad \forall j = 1,2,\ldots,n \tag{2.17}$$
$$\implies J_p(\theta'_i,\theta'_e(t+q)) < \frac{1}{n(t_f-(t+q))} \sum_{j=1}^{n} \sum_{k=t+q}^{t_f} \frac{9}{4} \tag{2.18}$$
$$< \frac{9}{4} \tag{2.19}$$

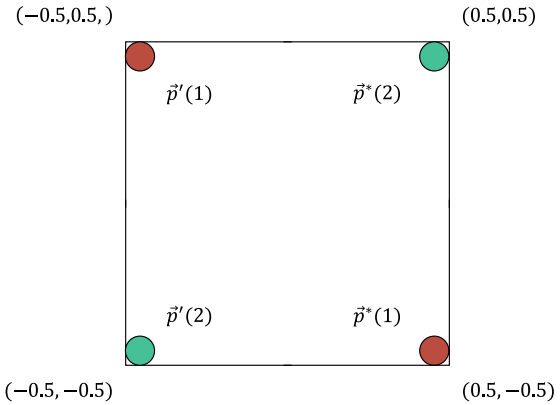

Figure 2.4: Configuration of two spheres, where the actual poses ($\vec{p}^*$) and estimated poses ($\vec{p}'$) are maximally displaced, with boundary dimensions included.

A secondary optimization problem, $(F_{\mathbf{I}})$, which is attempted by many end-to-end systems, which is not explicitly interested in solving for the underlying world representation, but solely future images of the simulation, solve for the following, given some model Θ:

$$\phi \in \Theta \tag{2.20}$$

$$\Theta = \underbrace{[0,1]^{3\times h\times w} \times [0,1]^{3\times h\times w} \times \cdots \times [0,1]^{3\times h\times w}}_{q \text{ images}} \times \mathbb{Z}^+ \to [0,1]^{3\times h\times w} \tag{2.21}$$

$$J_{\mathbf{I}}(\phi) = \frac{1}{n(t_f-(t+q))} \sum_{j=1}^{n} \sum_{k=t+q}^{t_f} \|\mathbf{I}^*(k) - \phi(\mathbf{I}^*(t+1), \mathbf{I}^*(t+2), \ldots, \mathbf{I}^*(t+q), k)\|^2 \quad \text{(MSE)} \tag{2.22}$$

$$(F_{\mathbf{I}}) \quad \min_{\phi} J_{\mathbf{I}}(\phi) \tag{2.23}$$

We will not discuss this further, given we focus on the primary problem, but it provides context to other physics models and a common evaluation metric.

2.2 Previous Work

To achieve human-like behaviour in machines we seek a method that provides an intuitive physical perspective. Such approaches are referred to as intuitive physics models within the research community.

2.2.1 Intuitive Physics

Several methods have been proposed to analyze and model the motion of objects in natural scenes by learning physical intuition [50, 13, 27, 3, 46, 49, 15, 34]. All of these methods have suggested that the most promising approach is through deep learning given several images from proceeding time steps for the physical system that is to be predicted. How this is done exactly differs from between the methods. Visual de-animation (VDA) [49], differs from its predecessors by incorporating a world-model and simulator for

training and prediction, and thus far, provides the highest performance when compared on the same tasks. This approach is supported by cognitive science, where a capacity for simulating scenarios is believed to be pivotal to for human physical intuition [4]. The preceding methods rely on an end-to-end learning approach, [13, 27, 15], where no world representation is implicitly given. Given the superior performance and support from human cognition, methods coming after VDA have adopted the principle of utilizing a world model and simulation rather than a purely end-to-end approach [34, 3]. In the rest of the related work, we will explain and discuss the components that make up these approaches to intuitive physics.

2.2.2 Inverse Graphics

The usage of a world model and simulator in VDA is inspired by inverse graphics, the process of recovering scene properties from images by finding the underlying 3D structure and other attributes of objects, such as shape, pose, and material, primarily through the use of neural networks [17]. Inverse graphics has been used to recover scene properties and object poses for various applications, including object recognition [51], human body pose estimation[20], 3D scene reconstruction [44] and intuitive physics [49].

2.2.3 Neural Networks & Deep Learning

Neural networks are a biologically inspired form of machine learning, originating from 1940, with the McCulloch-Pitts neuron, a simplified model of a biological neuron [29]. This was further developed in 1958 with introduction of the perceptron, allowing for simple feed-forward networks (multi-layer perceptron model, MLP) [39]. A method for effectively training these models was developed in 1986 with the introduction of the backpropagation algorithm [40]. The final concept that made the development of the intuitive physics models possible was the invention of convolutional neural networks (CNN) in 1989 [25]. The key innovation of CNNs was the use of convolutional layers, which exploit spatial invariance and local connectivity in images, making them more efficient for processing visual data. As hardware improved, the scale and size at which neural networks could be trained increased dramatically, leading to milestone performances of CNNs on tasks such as image classification, object detection and semantic segmentation, and the era of deep learning, which revolves around utilizing deeply layered neural networks trained on vast amounts of data [26, 23, 14, 42, 24, 16].

2.2.4 Residual Neural Networks (ResNet)

ResNets are a specific architecture of neural networks that employ residual connections to facilitate the training of deeper models [16]. This differs from previous methods in that certain layers feed not only to the the direct following layer, but layers further ahead, leading to improved performance as the model better leverages hierarchical features and avoids the vanishing gradient problem, a problem in which deep networks stop learning effectively as the backpropagated gradient becomes too small to effectively alter model weights. ResNets are core to many of the proposed intuitive physics models, including

VDA and PhysNet, an end-to-end model [49, 27]

2.2.5 Recurrent Neural Networks (RNN)

Although not used in VDA, RNNs are relevant for a number of existing intuitive physics models. RNNs were made to handle sequential data of variable lengths, capturing the temporal dependencies between data points. They do this through a looped feedback mechanism, which leverages a buffer of previous inputs to generate outputs. The basic component of an RNN is the recurrent neuron, which contains a state representation updated at each time step and encodes information about the inputs seen up to that point, passing it to the next time step. RNNs are typically trained with backpropagation through time, a variation of backpropagation designed for recurrent architectures [40].

2.2.6 Long Short-Term Memory Networks (LSTM)

The LSTM is a recurrent neural network architecture used by many intuitive physics models. It was made to capture long-range dependencies in sequential data, which matches well with many problems found in intuitive physics. It consists of interconnected memory cells that process input sequences step-by-step. Each memory cell contains an input gate, a forget gate, and an output gate. LSTMs utilize this gating mechanism to selectively remember and forget information. The input gate determines the importance of the current input, the forget gate decides which information from the previous input cell state should be retained and the output gate determines the information from the cell state that should be passed to the next layer or the next time step [18].

2.2.7 Encoder-Decoder Architecture

The encoder-decoder architecture is a two-component system comprised of an encoder and a decoder, both which can be composed of a feed-forward neural network, RNN, CNN or transformer. The encoder processes input data and produces a fixed-size vector representation that summarizes the input's relevant information. The decoder takes the context vector generated by the encoder and produces the desired output. Generally, the encoder down scales its input, and the decoder up scales its input. They are primarily trained by having the encoder downscale the input, and then have the decoder attempt to reconstruct the original input from the encoder output, using the difference between the original and reconstruction as training loss [43].

2.2.8 Optical Flow

Optical flow is a pivotal aspect of VDA. The aim of optical flow estimation is to compute a vector field that represents the motion of objects, surfaces, and edges in a scene over time by using their correspondence with the motion of brightness in an image sequence. The original Horn-Schunk method modeled optical flow as a global optimization problem with a smoothness constraint [19]. Various other optical flow methods have developed, relying on other underlying assumptions [28, 41, 12]. More

recently, deep learning-based methods have been adopted, exploiting the ability of convolutional networks to learn features at multiple levels of scale and abstraction [10]. As part of processing image frames, VDA applies a spatial pyramid network (SpyNet) to calculate the optical flow. SpyNet is a lightweight optical flow estimation network that employs a spatial pyramid structure to process multi-scale image features [36].

2.2.9 Self-Supervised Learning (SSL)

Self-supervised learning is a type of learning in which a model learns to predict certain aspects of the input data without explicit supervision. The learning signal comes from the data itself, rather than from any external labeling or annotation. This is in contrast to supervised learning, where a model learns the data by receiving labels associated with each sample. An important aspect of intuitive physics models are that they should not require human annotation, and thus should avoid being supervised, so that they can handle new scenarios independent from human supervision. In a strict sense, VDA is a supervised method, however, as these labels are generated without requiring human annotation, they satisfy some criteria of being self-supervised.

2.3 Related Work

2.3.1 Models

This section provides an overview of the key advancements in the development of intuitive physics models. The following list summarizes the significant contributions in the field:

- **Galileo** (Wu et al., 2015): a self-supervised LeNet model (CNN) that processes a single image containing two objects and estimates their mass and friction. The model generates self-supervised labels for object parameters from real-life training videos using maximum likelihood estimation (MLE) through the single-site Metropolis Hastings algorithm (SSMH). Mass and friction sampling is performed in simulation for a given scene [50, 26, 30].
- **Billiard Learning Network**[1] (Fragkiadaki et al., 2016): an encoder-decoder network that, given four frames of a simulated billiard sphere and forces applied by an agent on the ball, predicts the ball's velocities in the next h frames. The encoder follows an AlexNet architecture (CNN), and the decoder incorporates long short-term memory (LSTM) units to capture long-range temporal dynamics [13].
- **PhysNet** (Lerer et al., 2016): a ResNet34 architecture with upsampling that processes three frames of stacked simulated blocks and predicts segmentation masks for each block in the subsequent frame as they fall. This model was also applied to real-world blocks [27].

[1] We chose this name.

- **Visual De-Animation (VDA)** Wu et al. (2017): a ResNet18 model takes in three frames of a simulated billiard balls, calculates optical flow between frames using a SpyNet, and inputs the three frames and two optical flow frames masked for each object into a ResNet18 to predict the pose, velocity, friction and mass for each ball. This approach was demonstrated on real-world billiard balls and adapted for predicting how stacking blocks fall [49].
- **Visual Interaction Network (VIN)** (Watters et al., 2017): consists of three components: a visual encoder, a dynamic predictor, and a state decoder. The visual encoder, a CNN, takes in three frames and returns a state code, which is a list of vectors representing the position and velocity of each object in the scene. The dynamic predictor processes a sequence of state codes and predicts the state code for the next frame using interaction networks (IN), which model the state code as a graph structure in matrix form, with each interaction between objects represented by an edge. A multi-layer perceptron (MLP) model is applied to predict the future state code. The state decoder, a linear layer, then decodes the future state code into the position and velocity of each object in the subsequent frame. VIN was demonstrated on synthetic data of various physical systems [3, 46].
- **Plato** (Piloto et al., 2022): further extended the VIN model to incorporate an LSTM with the dynamic predictor for capturing long-term temporal dependencies [34].

There has been no comprehensive benchmark of all these models for the same task. In the visual de-animation paper, VDA is compared against PhysNet for a real-world stack stability task, where the goal is to predict how a stack of unstable blocks will fall. This was the original task for which PhysNet was developed. VDA outperforms PhysNet by a far margin [49]. We chose to use a VDA approach to our problem, because in the original paper, VDA achieved promising results for predicting dynamics of spheres on planar surfaces. Our approach however, relies less on the ResNet18 for state estimation, and more on optical flow. Further, we utilize a more sophisticated object proposal generation method. We believe this provides improved performance and robustness.

2.3.2 data sets

We present a list of prominent intuitive physics data sets developed to facilitate research and evaluation in the field of physical reasoning and visual understanding. They are as follows:

1. **CLEVR** (Johnson et al, 2017): A diagnostic tool for evaluating compositional language and elementary visual reasoning. It consists of a large number of rendered images and associated questions to assess models' understanding of various visual and linguistic concepts [21].
2. **AI2-THOR** (Kolve et al., 2017): an interactive 3D environment for visual AI research. It provides a rich set of tasks and scenes that require navigation, manipulation, and interaction with objects in a visually realistic setting [22].

3. **IntPhys** (Riochet et al., 2018): benchmark for visual intuitive physics reasoning. It includes a collection of video sequences depicting simulated objects interacting in physically plausible or implausible ways, testing a model's ability to reason about underlying physical principles [37].
4. **Roll4Real** (Erhardt et al., 2018): consists of videos of real objects rolling on complex terrains (pool table, elliptical bowl, and random height-field) [11].
5. **ShapeStacks** (Groth et al., 2018): consists of 3D-rendered scenes of stacks of objects with varying levels of complexity. It aims to test a model's ability to predict the stability of the stacks [15].
6. **PHYRE** (Bakhtin et al., 2019): benchmark for assessing physical reasoning abilities. Provides diverse and challenging tasks that require understanding of physical interactions and relationships [2].

None of the existing data sets perfectly address our specific problem. Roll4Real comes closest to meeting our requirements; however, it has several limitations. Primarily designed for object tracking, this data set does not offer ground truth for the intrinsic parameters of the balls in the scenes, and the balls are not rolled on planar surfaces. Additionally, the data set mostly contains recordings of a single ball, or at most two, which limits the potential for interaction between them. Consequently, our model has less opportunity to infer the underlying object states from these interactions.

Chapter 3

Visual De-Animation

3.1 Overview

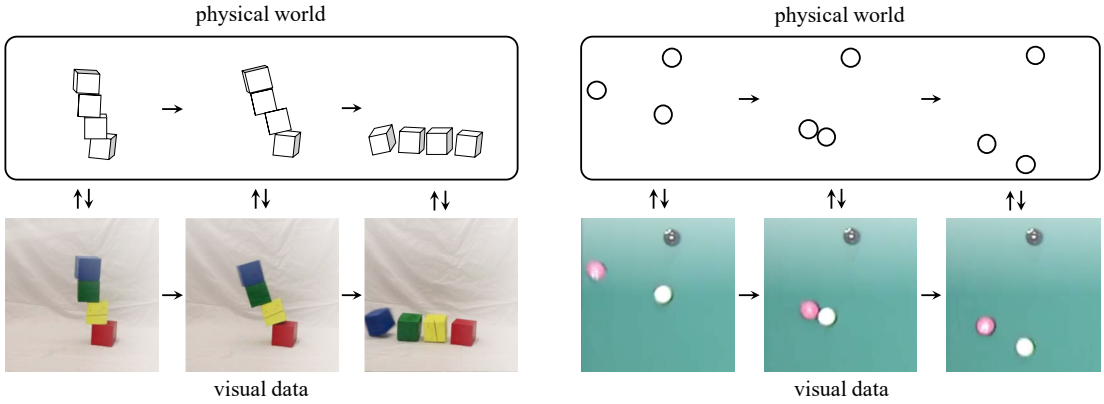

Figure 3.1: Visual de-animation — recover the world representation from a visual input and combine it with generative physics simulation and rendering engine [49].

In this section, we will describe the method of VDA for predicting the temporal evolution of visual scenes, and how it will used for modelling moving spheres on planar surfaces. VDA is a three-component system involving a perceptual module, physics engine, and graphics engine (Figure 3.1). The method works by extracting the world representation from the visual input using the perceptual module. This world representation is then fed into a physics engine to predict how the world will evolve over time. Finally, the evolved world representation is rendered using a graphics engine.

3.2 Perceptual Module

We will treat the perceptual module, ψ, as follow,

$$\psi(\mathbf{x}) = \mathbf{y}, \tag{3.1}$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{I}(t+1) & \mathbf{I}(t+2) & \dots & \mathbf{I}(t+q) \end{bmatrix}^T \in \mathbb{R}^{q\times 3\times h\times w}, \tag{3.2}$$

$$\mathbf{y} = \begin{bmatrix} \theta_i & \theta_e(t+q) \end{bmatrix} \in \mathbb{R}^{n'\times(\kappa+\tau)}, \tag{3.3}$$

where $\mathbf{x}$ is a tensor of consecutive frames from a given visual scene starting from some time step $t+1$ to time step $t+q$ for some buffer size q, and $\mathbf{y}$ is a tensor with estimated intrinsic and extrinsic parameter vectors for the estimated total number of objects in the scene at time step $t+q$. We let n' be the estimated total number of objects and κ, τ be the number of intrinsic parameters and extrinsic parameters respectively. In our case $\kappa = 7$ and $\tau = 6$, as determined in Section 2.1.

Object proposal generation From $\mathbf{x}$ we generate a set of object proposals $\mathbf{P}'(i)$ for each $\mathbf{I}(i)$,

$$\mathbf{P}'(i) = \left\{\vec{p}\,'(1,i), \vec{p}\,'(2,i), \dots, \vec{p}\,'(n'(i),i)\right\} \tag{3.4}$$

where $n'(i)$ is the number of object proposals made for $\mathbf{I}(i)$, and $\vec{p}\,'(j,i)$ is the j-th object proposal made for $\mathbf{I}(i)$.

When VDA was applied to predict motion of spheres in the original paper, this was done using colour filters, which assumes that we know the colour of each ball, or what the background colour of the scene is in each frame $\mathbf{I}(i)$ [49]. We make object proposals based on blob detection, filtering by circularity. Blob detection works by identifying regions of an image that have a similar appearance by applying filters to highlight areas of high contrast. Circularity of a region is determined by comparing the perimeter of a proposal to the perimeter of a circle with the same area, such that high circularity proposals have perimeters that closely match that of a circle:

$$\text{circularity} = \frac{4\pi \cdot \text{Area}}{(\text{Perimeter})^2} \tag{3.5}$$

OpenCV, an open source computer vision library, is used to do the blob detection [6]. Pseudo-code is provided for single image proposal generation in Algorithm 1. Blob detection specifically estimates the pixel-wise p_x, p_y, and r for each proposal. This is sufficient to generate $\mathbf{P}'(t+q)$. We make the assumption that $n' = n'(t+q)$, and estimate all $\vec{p}$ components of $\theta_e(t+q)$ and r components of θ_i using $\mathbf{P}'(t+q)$.

Algorithm 1 Object Proposal Generation ($\mathbf{I}(i)$)

```
Initialise Blob Detector D
Initialise Set P'(i) = {}

D.minCircularity = 0.1
D.filterByCircularity = True
P(i) = D.detect(I(i))
for all p(j,i) ∈ P(i) do
    P'(i).add(p(j,i))
```

For later stages of the model, we need to create object masks for each $\mathbf{I}(i)$ using $\mathbf{P}'(i)$. We generate $\mathbf{P}'(t+q-1), \mathbf{P}'(t+q-2), \ldots, \mathbf{P}'(t+1)$ by backtracking the object proposals from $\mathbf{P}'(t+q)$ using optical flow:

$$\Omega : \mathbb{R}^{h\times w} \times \mathbb{R}^{h\times w} \to \mathbb{R}^{2\times h\times w}, \quad \text{(optical flow)} \tag{3.6}$$

$$g : \mathbb{R}^{3\times h\times w} \to \mathbb{R}^{h\times w}, \quad \text{(grayscale)} \tag{3.7}$$

$$\mathbf{G}(i) = g(\mathbf{I}(i)) \in \mathbb{R}^{h\times w}, \quad (\mathbf{I}(i) \text{ grayscale}) \tag{3.8}$$

For optical flow to work, we must satisfy several conditions:

1. *Brightness Constancy*: The brightness of a moving object remains constant between consecutive frames:

$$\mathbf{G}(i)[x,y] = \mathbf{G}(i+1)[x+u, y+v] \tag{3.9}$$

$$\Omega(\mathbf{G}(i), \mathbf{G}(i+1))[x,y] = \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.10}$$

   where (x,y) are the coordinates of a pixel in $\mathbf{G}(i)$ and $(x+u, y+v)$ is the corresponding pixel in $\mathbf{G}(i+1)$.

2. *Small Motion*: The motion of objects between consecutive frames is small.

3. *Spatial Coherence*: The neighboring pixels have similar motion.

Given the problem definition in Section 2.1, we satisfy all these conditions. Now, given that we can use Ω and have access to $\mathbf{I}(i)$, $\mathbf{I}(i+1)$ and $\mathbf{P}'(i+1)$, we calculate $\mathbf{P}'(i)$ as follows

$$\Omega(\mathbf{G}(i+1), \mathbf{G}(i))[p'_x(j,i+1), p'_y(j,i+1)] = \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.11}$$

$$\vec{p}'(j,i) = \begin{bmatrix} p'_x(j,i)+u \\ p'_y(j,i+1)+v \\ p'_z(j,i+1) \end{bmatrix} \tag{3.12}$$

$$\forall \vec{p}'(j,i) \in \mathbf{P}'(i) \tag{3.13}$$

Thus we can backtrack the object proposals.

Physical object state estimation In object proposal generation, we found that $\mathbf{P}'(t+q)$ is sufficient to calculate all $\vec{p}'$ components of $\theta_e(t+q)$ and r' components of θ_i under the assumption that $n' = n'(t+q)$.

We use optical flow to calculate the $\vec{v}$ components of $\theta_e(t+q)$:

$$\Omega(\mathbf{G}(t+q), \mathbf{G}(t+q-1))[\vec{p}'_x(j,t+q), \vec{p}'_y(j,t+q)] = \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.14}$$

$$\vec{v}'(j,i) = \begin{bmatrix} -u \\ -v \\ 0 \end{bmatrix} \tag{3.15}$$

$$\forall j = 1, 2, \ldots, n' \tag{3.16}$$

This approach assumes that $\vec{v}'(j,i) \approx \vec{v}'(j,i-1)$, which only holds when the interval between time steps is short.

We calculate all $\vec{c}'$ components for θ_i as the mean $\mathbf{I}(t+q)$ pixel value inside the corresponding proposal boundary:

$$\varepsilon'(j,t+q) = \left\{ [x,y] \,\middle|\, (p'_x(j,t+q)-x)^2 + (p'_y(j,t+q)-y)^2 \leq r(j)^2 \right\} \tag{3.17}$$

$$c'(j) = \frac{1}{|\varepsilon'(j,t+q)|} \sum_{[x,y]\in\varepsilon'(j,t+q)} \mathbf{I}(t+q)[x,y], \quad \forall j = 1,2,\ldots,n' \tag{3.18}$$

We use a convolutional network to recognize $m'(j)$ and $\mu'_r(j)$ for each object proposal j. The input to the network is the masked image sequence of the proposal and the corresponding forward sequence of optical flow, with all images concatenated along the first dimension, and the output is a discrete label $l'(j)$ corresponding to a $\begin{bmatrix} m' & \mu'_r \end{bmatrix}^T$ vector for the given object proposal:

$$\mathbf{V}(i) = \Omega(\mathbf{G}(i) = \Omega(\mathbf{G}(i), \mathbf{G}(i+1)) \quad \text{(Forward optical flow image)}$$

$$\mathbf{M_I}(j,i)[x,y] = \begin{cases} \mathbf{I}(i)[x,y], & [x,y] \in \varepsilon'(j) \\ \vec{0}, & \text{otherwise} \end{cases} \quad \text{(masked } \mathbf{I}(i)\text{)}$$

$$\mathbf{M_V}(j,i)[x,y] = \begin{cases} \mathbf{V}(i)[x,y], & [x,y] \in \varepsilon'(j) \\ \vec{0}, & \text{otherwise} \end{cases} \quad \text{(masked } \mathbf{V}(i)\text{)}$$

$$\mathbf{B}(j) = \text{Concat}(\mathbf{M_I}(t+1),\ldots,\mathbf{M_I}(t+q),\mathbf{M_V}(t+1),\ldots,\mathbf{M_V}(t+q-1))$$

$$\alpha = 3q \quad \text{(RGB dim.)} \quad \beta = 2(q-1) \quad \text{(Optical flow dim.)}$$
$$\phi : \mathbb{R}^{(\alpha+\beta)\times h\times w} \to \mathbb{N}, \quad \text{(CNN)}$$
$$\phi(\mathbf{B}(j)) = l'(j)$$

$$f_m(l'(j)) = m'(j)$$
$$f_{\mu_r}(l'(j)) = \mu'_r(j)$$

CNN implementation and training is described Section 4.4. We now have an approach for estimating all components of $\theta_e(t+q)$ and θ_i.

3.3 Physics Engine

The physics engine, λ, allows us to simulate a given scenario w time steps into the future using the output from ψ:

$$\lambda : \mathbb{R}^{n'\times(\kappa+\tau)} \times \mathbb{Z}^+ \to \mathbb{R}^{n'\times\tau} \tag{3.19}$$

$$\psi(\mathbf{x}) = \mathbf{y} \tag{3.20}$$

$$\lambda(\mathbf{y}, w) = \theta'_e(t+q+w) \tag{3.21}$$

There exists non-differentiable and differentiable physics engines.

Non-differentiable

These represent the standard class of physics engines, which implement explicit mechanical equations to predict object dynamics. This provides them with a robust and mature foundation. We choose to utilize PyBullet, a standard rigid-body simulation engine, primarily intended for robotics research [9]. This was primarily due to it being open-source and used in the original VDA paper [49].

Differentiable

Another approach is with neural, differentiable physics engines. These are trained on a scenario-basis, fed with intrinsic and extrinsic parameters for all objects in a scene, then aim to predict the future extrinsic parameters for each object in the scene. This is similar to the dynamic predictor in a VIN model [46]. A neural physics engine was specifically developed for colliding balls by Chang et al. 2016, and used in the original VDA paper, simulating scene dynamics by taking ball mass, position, and velocity [8]. We did not have time to train this model for our data, so we focus on solely non-differentiable physics engines.

3.4 Graphics Engine

The graphics engine, γ, allows us to render an image $\mathbf{I}(i)$ of a scenario given intrinsic parameters and extrinsic parameters at a given time step i:

$$\gamma : \mathbb{R}^{n' \times (\kappa+\tau)} \rightarrow [0,1]^{3 \times h \times w} \tag{3.22}$$

$$\gamma(\theta_i, \theta_e(i)) = \mathbf{I}(i) \tag{3.23}$$

Given that we are using PyBullet as the physics engine, we use the in-built OpenGL graphics engine to render scenes [48]. This allows for quick performance with GPU integration.

A summary of the model is shown in Figure 3.2.

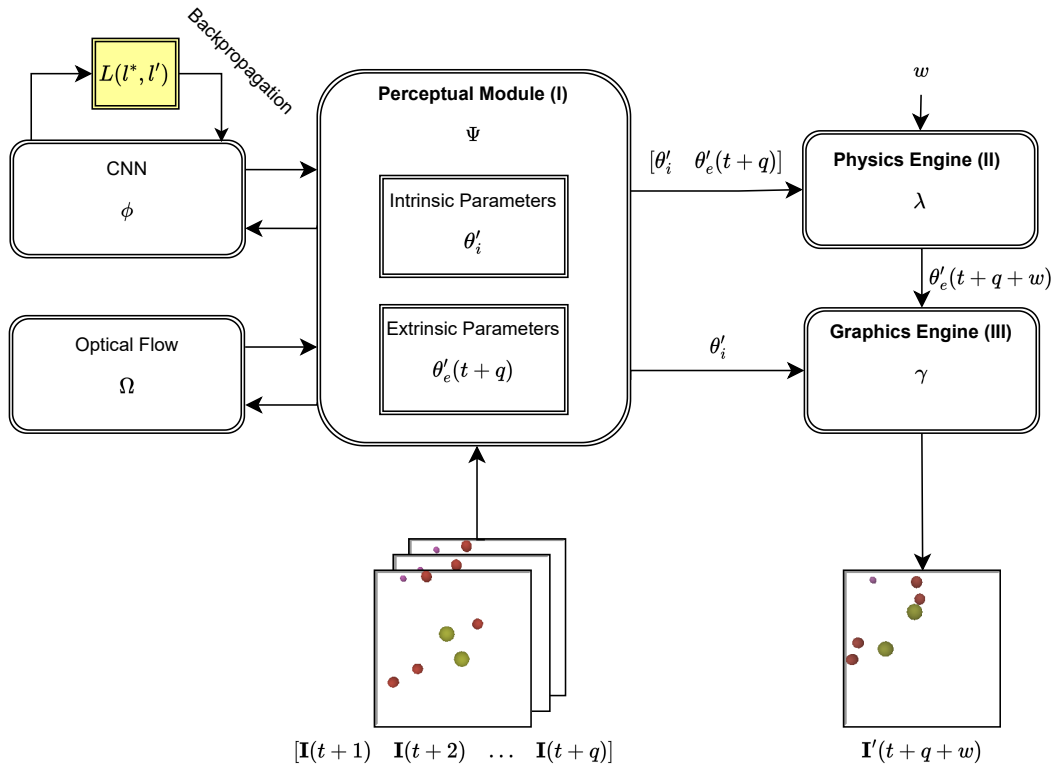

Figure 3.2: Our Visual De-Animation (VDA) model is composed of three key elements: a Perception module (I), a physics engine (II), and a graphics engine (III). The Perception module plays a crucial role in inverting the graphics engine, by inferring the physical object state for each segment proposal in the input. These inferred states are then combined to obtain a physical world representation, denoted as θ_i and $\theta_e(t+q)$. Subsequently, the generative physics and graphics engines are engaged to progress forward and reconstruct the visual data as $\mathbf{I}'(t+q+w)$.

Chapter 4

Methodology

4.1 Data set

To train and evaluate the VDA model, we created a data generation package using PyBullet. This is provided in our open-source repository with usage intstructions: `https://github.com/J0HNN7G/intuitive_physics`

4.1.1 Content

The package allows you to generate simulations of spheres as specified in Section 2.1. For any given simulation, you can randomize or specify the number of spheres and all intrinsic, extrinsic parameters that are to be inferred by the VDA model. Further, you can specify the resolution, duration and frame rate per second (FPS) of a simulation. For the intrinsic parameters, the following options are available (ordered sets):

$$r \in R = (0.04, 0.07, 0.09) \tag{4.1}$$

$$m \in M = (0.2, 0.4, 0.6) \tag{4.2}$$

$$\mu_r \in F = (10^{-4}, 10^{-2}) \tag{4.3}$$

These values were selected to be similar to balls on a pool table [1]. As well as a palette of colour-blind friendly RGB values for $\vec{c}$, as shown in Table 4.1 [35]. We denote the ordered set as C.

Table 4.1: Range of RGB values which $\vec{c}$ can take in data generation package.

RGB	Colour
0.729, 0.298, 0.251	
0.271, 0.753, 0.592	
0.341, 0.204, 0.522	
0.659, 0.682, 0.243	
0.533, 0.455, 0.851	
0.412, 0.627, 0.314	
0.745, 0.392, 0.698	
0.737, 0.490, 0.212	

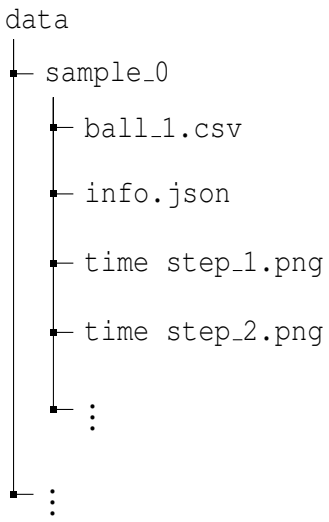

Figure 4.1: Directory structure for generated simulation data.

When generating a simulation, we store all data in a predefined structure (Figure 4.1). The actual intrinsic parameters for all spheres are stored in `info.json` (Figure 4.2), the actual extrinsic parameters for the i-th sphere are stored in `ball_<i>.csv` (Table 4.2), and the image associated with time step j is stored in `time step_<j>.png`.

```
{
    "fps": 30,
    "num_balls": 3,
    "0": {
            "label": 1,
            "color": [0.729, 0.298, 0.251],
            "radius": 0.07,
            "mass": 0.2,
            "friction": 0.01
    }
}
```

Figure 4.2: Contents of `info.csv` from a generated simulation.

Table 4.2: Contents of a ball CSV from a generated simulation (not all columns included).

time step	pose-x	pose-y	lin-vel-x	lin-vel-y
1	0.4705	0.7279	0.3474	0.3372
2	0.4821	0.7392	0.3474	0.3372
⋮	⋮	⋮	⋮	⋮
60	0.5183	0.7742	0.0000	0.0000

4.1.2 Collection

For all the simulations generated for our experiments, we have a resolution of 256×256 pixels, duration of 2 seconds and 30 FPS. We use $n = 3$ as this provided a significant number of interactions between spheres without cluttering environment. As mentioned in Section 2.1, μ_l, μ_s, b are fixed. For every experiment, we use a training set of 1000 simulations, validation set of 200 simulations, and a test set of 200 simulations. We deemed this a sufficiently large training set to avoid over-fitting for the CNN we employ.

4.2 Experiments

To evaluate our model, we apply it in four different settings:

1. *Predict Mass*, $r \to m$, (Figure 4.3a):

$$\forall j = 1, 2, \ldots, n, \text{ we sample index } i \sim U\{1, 2, 3\} \tag{4.4}$$
$$m(j) = M(i), \quad r(j) = R(i) \tag{4.5}$$

   Model should learn to associate higher mass m with higher radius r. This configuration tests the feasibility of training the model to differentiate objects of various sizes.

2. *Predict Friction*, $\vec{v} \to \mu_r$, (Figure 4.3b):

$$\forall j = 1, 2, \ldots, n, \text{ we sample index } i \sim U\{1, 2\}, \tag{4.6}$$
$$\mu_r(j) = F(i) \tag{4.7}$$

   Model should learn to associate change in position between images and magnitude of optical flow with μ_r. This configuration tests the feasibility of training the model to differentiate objects by motion.

3. *Predict Material*, $\vec{c} \to m, \mu_r$, (Figure 4.3c):

$$\forall j = 1, 2, \ldots, n, \text{ we sample index } i \sim U\{1, 2, 3\}, k \sim U\{1, 2\}, \tag{4.8}$$
$$m(j) = M(i), \quad \mu_r(j) = F(k), \tag{4.9}$$
$$\vec{c}(j) = C(3(k-1) + (i-1)) \tag{4.10}$$

   Model should learn to associate visual appearance with m and μ_r. This configuration tests the feasibility of training the model to differentiate objects by material (represented by colour).

4. *Predict Mass and Friction*, $(r, \vec{v}) \to (m, \mu_r)$, (Figure 4.3d):

$$\forall j = 1, 2, \ldots, n, \text{ we sample index } i \sim U\{1, 2, 3\}, k \sim U\{1, 2\}, \tag{4.11}$$
$$m(j) = M(i), \quad r(j) = R(i) \tag{4.12}$$
$$\mu_r(j) = F(k) \tag{4.13}$$

   Model should learn to associate both higher m with higher r and change in position between images and magnitude of optical flow with μ_r independently. This configuration tests the feasibility of the model to differentiate objects by several uncorrelated factors.

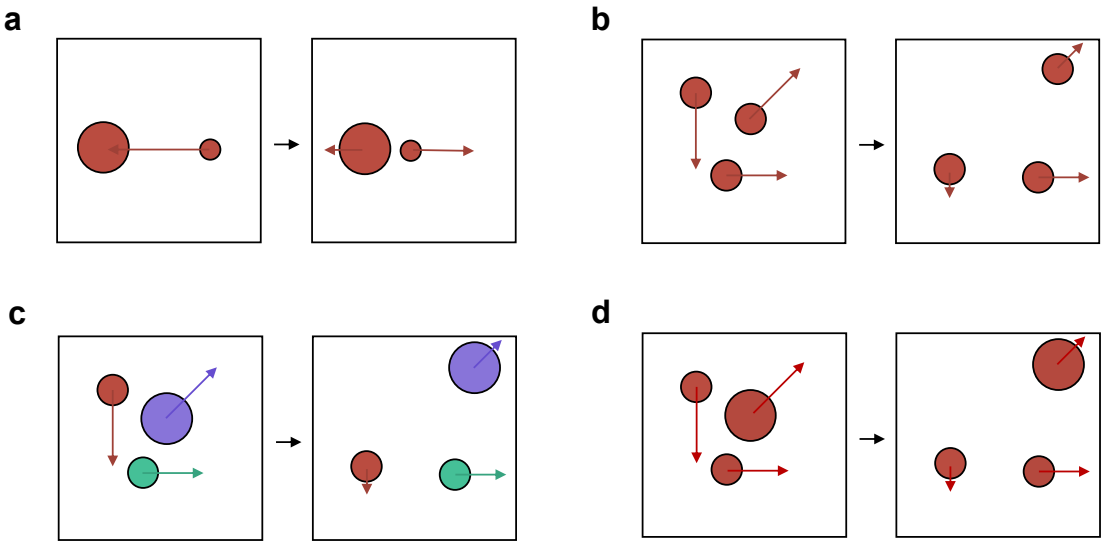

Figure 4.3: The four simulation settings used in our experiments: (a) the spheres vary in mass and radius, which are related to each other, and the system learns about their physics from their appearance; (b) the spheres differ in friction, and the system learns about their physics by observing their motion; (c) the spheres have different mass, radius, and friction correlated with colour, which are interrelated, and the system learns to associate appearance cues with underlying object states; (d) combine settings (a) and (b), where spheres vary in mass, radius, and friction, and the system learns about their physics by analyzing both their appearance and motion.

We use $q = 3$, as this is the minimum number of images required to see the velocity change for objects. Any less, and the model would not be able to detect physical properties from solely motion.

4.3 Evaluation Metrics

Our evaluation metrics are informed by our optimization problems F_p and $F_{\mathbf{I}}$.

We get a good approximation for how well the model will perform on F_p by using the Mean (x,y) Manhattan Distance between $\vec{p}^*$ and $\vec{p}'$ as time steps are increased. We exclude the z component as this remains the same for all time steps, and only reflects how well the model can estimate the r parameter. We choose to use L_1 norm (Manhattan distance) instead of L_2 norm (Euclidean distance), as is used in our objective function, as this is used in the original VDA paper, making it easier for us to compare results. We also do this pixel-wise instead of in metres, to make for easier comparison. We

calculate this for a given time step as follows:

$$p^* = \begin{bmatrix} \vec{p}^*(1) & \vec{p}^*(2) & \dots & \vec{p}^*(n) \end{bmatrix}^T \tag{4.14}$$

$$p' = \begin{bmatrix} \vec{p}'(1) & \vec{p}'(2) & \dots & \vec{p}'(n) \end{bmatrix}^T \tag{4.15}$$

$$\text{Mean Position Prediction Error}(p^*, p') = \frac{1}{n}\sum_{j=1}^{n}(|\vec{p}_x^*(j) - \vec{p}_x'(j)| + |\vec{p}_y^*(j) - \vec{p}_y'(j)|) \tag{4.16}$$

To better understand the model, we use this for the velocity as well,

$$v^* = \begin{bmatrix} \vec{v}^*(1) & \vec{v}^*(2) & \dots & \vec{v}^*(n) \end{bmatrix}^T \tag{4.17}$$

$$v' = \begin{bmatrix} \vec{v}'(1) & \vec{v}'(2) & \dots & \vec{v}'(n) \end{bmatrix}^T \tag{4.18}$$

$$\text{Mean Velocity Prediction Error}(v^*, v') = \frac{1}{n}\sum_{j=1}^{n}(|\vec{v}_x^*(j) - \vec{v}_x'(j)| + |\vec{v}_y^*(j) - \vec{v}_y'(j)|) \tag{4.19}$$

For both the position and velocity, we also compute the standard deviation, to understand how the model becomes less stable as it makes prediction farther into the future.

Although we are primarily interested in F_p, its in our interest to see how well our model can reconstruct the image corresponding to our scene, as is the objective of $F_{\mathbf{I}}$. To do this we look at the pixel mean squared error (MSE) between the original image and reconstructed image at time step $t+q$:

$$\text{Reconstruction Pixel MSE}(\mathbf{I}^*, \mathbf{I}') = \frac{1}{h \cdot w}\sum_{x=1}^{h}\sum_{y=1}^{w}\left\|\mathbf{I}^*[x,y] - \mathbf{I}'[x,y]\right\|^2 \tag{4.20}$$

The model performance is directly related to how well the CNN component of the perceptual model can predict the labels corresponding to the intrinsic parameters of the spheres. We quantify this with the label accuracy,

$$\vec{l}^* = \begin{bmatrix} l^*(1) & l^*(2) & \dots & l^*(n) \end{bmatrix}^T \tag{4.21}$$

$$\vec{l}' = \begin{bmatrix} l'(1) & l'(2) & \dots & l'(n) \end{bmatrix}^T \tag{4.22}$$

$$[l^*(i) = l'(i)] = \begin{cases} 1 & \text{if } l^*(i) = l'(i) \\ 0 & \text{otherwise} \end{cases} \tag{4.23}$$

$$\text{Label accuracy}(\vec{l}^*, \vec{l}') = \frac{\sum_{i=1}^{n}[l^*(i) = l'(i)]}{n} \tag{4.24}$$

Further, to understand how well object proposal generation component performs, we compute its accuracy, recall, precision and F1 score.

Intersection over union (IoU) is a common metric in object detection for determining

how well two object proposals match,

$$\text{Area of Intersection}(A,B) = |A \cap B| \tag{4.25}$$
$$\text{Area of Union}(A,B) = |A \cup B| \tag{4.26}$$
$$\text{IoU}(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.27}$$

Our object proposals are circular. Given two circles $C_1(x_1, y_1, r_1), C_2(x_2, y_2, r_2)$, the area of intersection and union are calculated as follows, ignoring the trivial cases were the circles do not overlap at all, or fully overlap [47]:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.28}$$
$$g = \sqrt{(-d + r_1 + r_2)(d + r_1 - r_2)(d - r_1 + r_2)(d + r_1 + r_2)} \tag{4.29}$$
$$|C_1 \cap C_2| = r_1^2 \arccos\left(\frac{d^2 + r_1^2 - r_2^2}{2dr_1}\right) + r_2^2 \arccos\left(\frac{d^2 + r_2^2 - r_1^2}{2dr_2}\right) - \frac{1}{2}g \tag{4.30}$$
$$|C_1 \cup C_2| = \pi(r_1^2 + r_2^2) - I \tag{4.31}$$

We will not discuss the proof for this here as it is relevant to the VDA model.

Accuracy, recall, precision, F1 score are calculated as follows:

$$\text{Accuracy} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives}} \tag{4.32}$$
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.33}$$
$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.34}$$
$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{4.35}$$

We determine a true positive as when the highest IoU for a ground truth proposal is above 0.8, where that highest IoU is from a proposal that has not already been selected as a true positive for a different ground truth proposal. We use this to match the index of object proposals with the ground truth for the other metrics mentioned above. Mean IoU is calculated for IoU values selected as true positives.

Finally, its is desirable for the model to be quick, hence we measure the mean inference time to make a prediction for a given future time step (without visualization).

4.4 Implementation Details

For our perceptual module,

$$\phi = \text{ResNet18} \tag{4.36}$$

We use a ResNet18 architecture as it is lightweight and used in the original VDA paper [49, 16]. The input is modified from the standard configuration for ImageNet to the number of input channels in our experiments and ablation study, as is the output shape.

A standard training regime for ϕ is employed with cross-entropy loss and stochastic gradient descent (SGD) optimizer using a learning rate of 0.1, and momentum of 0.9 [5, 38]. For all experiments, we train a newly-initialized model for 5 epochs, where each epoch is 1000 iterations, involving a batch size of 8, i.e., each epoch involves 8000 samples. 10^{-4} weight decay is employed for regularization such that after each iteration, learning rate is updated as follows:

$$\text{scale} = \left(1 - \frac{\text{current iteration}}{\text{max epochs} \times \text{iterations}}\right) \times 10^{-4} \tag{4.37}$$

$$\text{learning rate} = \text{learning rate} \times \text{scale} \tag{4.38}$$

For the optical flow,

$$\Omega = \{\text{Farnebäck}, \text{SpyNet}\} \tag{4.39}$$

Farnebäck method is a classical computer vision approach to dense optical flow, implemented in OpenCV [6, 12]. We use the standard parameters provided with the OpenCV implementation. SpyNet is a deep learning optical flow model used in the original VDA paper [36, 49]. We utilize a SpyNet model pre-trained on the MPI Sintel data set [7, 31].

All neural networks are implemented with PyTorch and benchmarking is done with a single Nvidia GTX 1060 (6GB GPU Memory) [33, 32].

Chapter 5

Results

5.1 Experimental Results

5.1.1 Object Proposal Generation

Table 5.1: Accuracy, Recall, Precision, F1, mean IoU (mIoU) for object proposal generation in each setting.

Setting	Accuracy	Recall	Precision	F1	mIoU
$r \to m$	0.8346	0.8450	0.9692	0.8877	**0.9947**
$\vec{v} \to \mu_r$	**0.9083**	**0.9083**	**0.9950**	**0.9360**	0.9942
$\vec{c} \to m, \mu_r$	0.8112	0.8217	0.9625	0.8705	0.9850
$(r, \vec{v}) \to (m, \mu_r)$	0.8542	0.8633	0.9633	0.8948	0.9846

In the *predict friction* setting, object proposal generation demonstrates the highest performance in accuracy, recall, precision, and F1 scores. This is likely due to the constant radius of all spheres in this setting, as opposed to the other settings where the radius may be smaller, making sphere detection more challenging. In contrast, the *predict mass* setting exhibits the highest mean intersection over union (mIoU), albeit only slightly (with a difference of 0.0005 from the next highest mIoU value). The negligible range of mIoU values (0.0101) suggests that the blob detection method can effectively match a sphere despite variations in radius and colour. This can be attributed to the fact that object proposal generation relies on circularity as the primary criterion for object detection, which remains unaffected by changes in these parameters.

Overall, the object proposal generation approach demonstrates high performance across all tasks, as indicated in Table 5.1.

5.1.2 Scene Reconstruction

Table 5.2: Pixel MSE between final buffer image and reconstruction by model (Pixel MSE) in each setting.

Setting	Pixel MSE
$r \rightarrow m$	0.0025
$\vec{v} \rightarrow \mu_r$	0.0015
$\vec{c} \rightarrow m, \mu_r$	0.0023
$(r, \vec{v}) \rightarrow (m, \mu_r)$	0.0015

Table 5.2 shows that the Pixel MSE between the final buffer image and reconstruction is consistently low across all settings. This indicates that the model successfully extracts the world representation and accurately reconstructs it in each scenario. Notably, the lowest error is observed in the *predict friction* and *predict mass and friction* settings. However, since the difference in Pixel MSE across all settings is negligible (0.01 range), we choose not to highlight any specific values as having superior performance.

5.1.3 Convolutional Neural Network (CNN)

Table 5.3: Inference time mean and standard deviation ($\mu \pm \sigma$) in seconds (Inf. Time) and mean label accuracy (Label Acc.) for each setting using $\Omega \in \{\text{Farnebäck}, \text{SpyNet}\}$.

Setting	Ω	Inf. Time (s)	Mean Label Acc.
$r \rightarrow m$	Farnebäck	$0.2430 \pm \mathbf{0.0742}$	0.9998
	SpyNet	$\mathbf{0.2295} \pm 0.0794$	**0.9999**
$\vec{v} \rightarrow \mu_r$	Farnebäck	$\mathbf{0.2199 \pm 0.0654}$	0.8656
	SpyNet	0.2347 ± 0.0695	**0.8869**
$\vec{c} \rightarrow m, \mu_r$	Farnebäck	$\mathbf{0.2033 \pm 0.0693}$	**0.9989**
	SpyNet	0.2161 ± 0.0751	0.9986
$(r, \vec{v}) \rightarrow (m, \mu_r)$	Farnebäck	$\mathbf{0.2095 \pm 0.0692}$	**0.8670**
	SpyNet	0.2239 ± 0.0742	0.8631

Table 5.3 displays the low and stable inference time for the perceptual module using either the Farnebäck method or SpyNet. Excluding the *predict mass* setting, the Farnebäck method appears to be marginally faster and more consistent than using SpyNet, with a mean inference time difference of approximately 0.01 seconds and a standard deviation difference of around 0.005 seconds. With regard to label accuracy, it is unclear which model, trained on either the Farnebäck method or SpyNet optical flow, achieves the highest performance, as this varies across settings. However, when excluding the *predict friction* setting, the perceptual module utilizing Farnebäck optical flow exhibits slightly better label accuracy. Notably, for the *predict friction* setting, we observe the most significant difference in label accuracies between the perceptual modules trained on Farnebäck and SpyNet optical flow, with the SpyNet model achieving a 0.0213 higher accuracy. In other settings, the difference in label accuracy is less pronounced.

5.1.4 Future Prediction

The performance of the model can be analyzed based on the mean prediction errors and standard deviations for position and velocity at different future time steps and settings, as shown in Figure 5.1 and Figure 5.2.

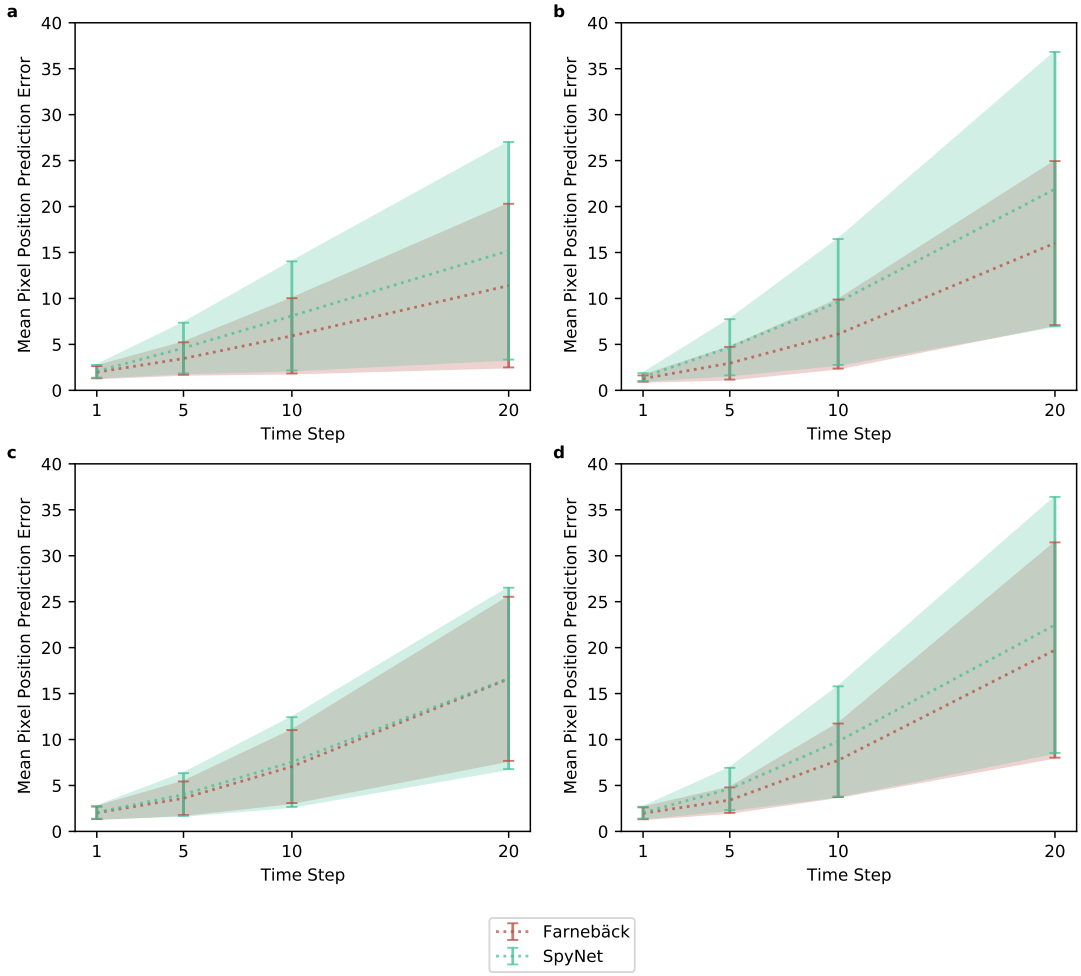

Figure 5.1: Mean pixel position prediction error over time with standard deviation bars for each setting using $\Omega \in \{\text{Farnebäck}, \text{SpyNet}\}$: (a) $r \rightarrow m$; (b) $\vec{v} \rightarrow \mu_r$; (c) $\vec{c} \rightarrow m, \mu_r$; (d) $(r, \vec{v}) \rightarrow (m, \mu_r)$.

For mean pixel position prediction error, the Farnebäck algorithm consistently outperforms the SpyNet algorithm, achieving lower values across all settings and time steps. This trend indicates that the Farnebäck algorithm is more accurate in predicting position than SpyNet. The model's performance in position prediction tends to degrade as the future time steps increase, which is expected due to the inherent uncertainty in predicting further into the future.

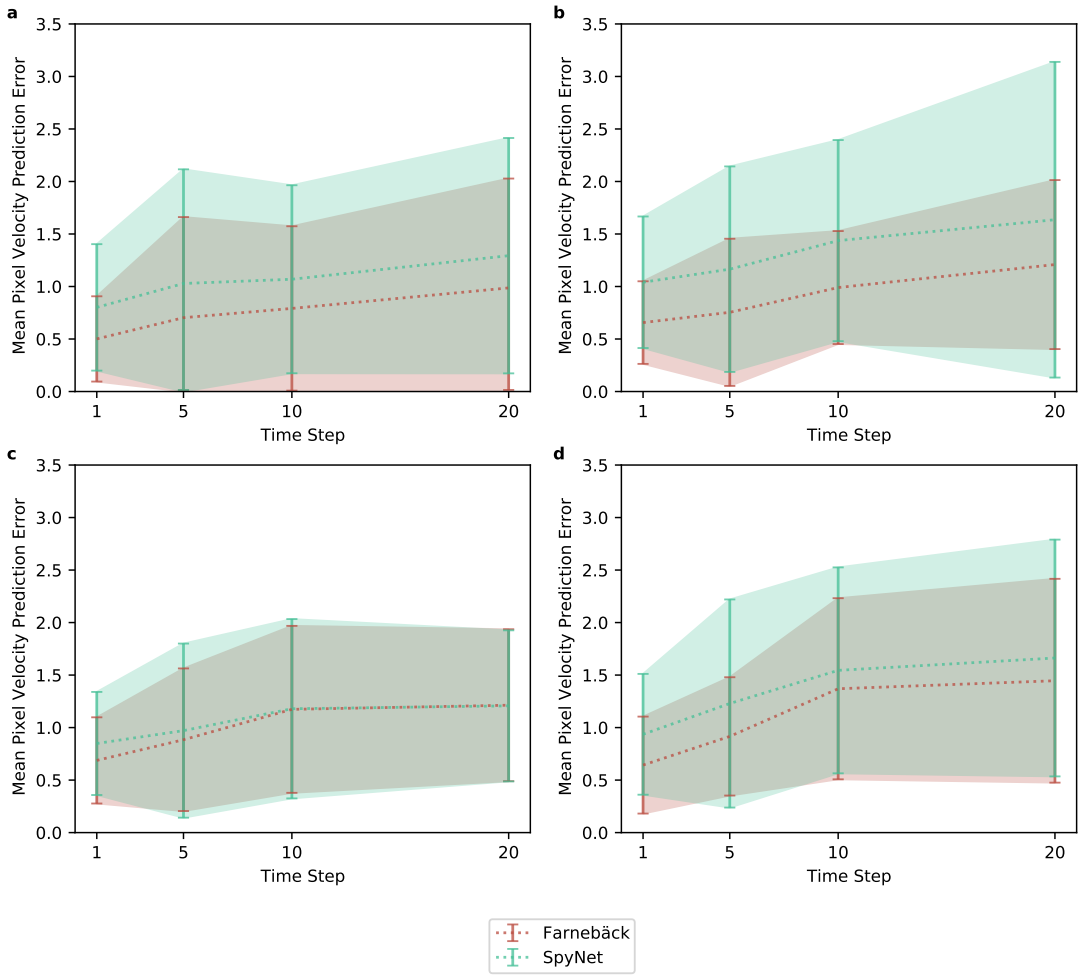

Figure 5.2: Mean pixel velocity prediction error over time with standard deviation bars for each setting using $\Omega \in \{\text{Farnebäck}, \text{SpyNet}\}$: (a) $r \to m$; (b) $\vec{v} \to \mu_r$; (c) $\vec{c} \to m, \mu_r$; (d) $(r, \vec{v}) \to (m, \mu_r)$.

When examining the mean pixel velocity prediction error, the Farnebäck algorithm still demonstrates superior performance compared to SpyNet. It consistently achieves lower mean velocity prediction errors across all settings and time steps. Similar to the position prediction, the model's performance in velocity prediction also declines as the future time steps increase, which is a natural consequence of predicting further into the future. Overall, these results suggest that the Farnebäck algorithm is a more reliable choice for both position and velocity prediction tasks compared to SpyNet. Given the lower range for velocity, we see that the standard deviation is much lower than for the position error.

We see that the model is able to effectively infer the position and velocity of all spheres in a given scene. Position and velocity prediction error results are also displayed in tables (see Table 5.4, Table 5.5)

Table 5.4: Mean pixel position prediction error (Pos. Pred. Error) and mean pixel velocity prediction error (Vel. Pred. Error) at each future time step $t \in [1, 5, 10, 20]$ for each setting using $\Omega \in \{\text{Farnebäck}, \text{SpyNet}\}$.

Setting	Ω	Mean Pos. Pred. Error				Mean Vel. Pred. Error			
		1st	5th	10th	20th	1st	5th	10th	20th
$r \to m$	Farnebäck	**1.97**	**3.46**	**5.93**	**11.40**	**0.50**	**0.70**	**0.79**	**0.99**
	SpyNet	2.07	4.61	8.10	15.19	0.80	1.03	1.07	1.29
$\vec{v} \to \mu_r$	Farnebäck	**1.28**	**2.95**	**6.12**	**16.02**	**0.66**	**0.75**	**0.99**	**1.21**
	SpyNet	1.45	4.69	9.61	21.90	1.04	1.16	1.44	1.64
$\vec{c} \to m, \mu_r$	Farnebäck	**2.02**	**3.61**	**7.06**	**16.61**	**0.69**	**0.88**	**1.17**	**1.21**
	SpyNet	2.07	4.01	7.55	16.65	0.85	0.97	1.18	1.21
$(r, \vec{v}) \to (m, \mu_r)$	Farnebäck	**1.97**	**3.40**	**7.74**	**19.74**	**0.64**	**0.92**	**1.37**	**1.45**
	SpyNet	2.04	4.63	9.79	22.47	0.94	1.23	1.54	1.66

Table 5.5: Standard deviation of pixel position prediction [Pos. Pred. (SD)] and standard deviation of pixel velocity prediction [Vel. Pred. (SD)] at each future time step $t \in [1, 5, 10, 20]$ for each setting using $\Omega \in \{\text{Farnebäck}, \text{SpyNet}\}$.

Setting	Ω	Pos. Pred. (SD)				Vel. Pred. (SD)			
		1st	5th	10th	20th	1st	5th	10th	20th
$r \to m$	Farnebäck	**0.66**	**1.77**	**4.10**	**8.91**	**0.41**	**0.96**	**0.78**	**1.04**
	SpyNet	0.69	2.75	5.94	11.84	0.60	1.09	0.90	1.12
$\vec{v} \to \mu_r$	Farnebäck	**0.35**	**1.78**	**3.76**	**8.92**	**0.39**	**0.7**	**0.54**	**0.80**
	SpyNet	0.42	3.06	6.85	14.93	0.63	0.98	0.96	1.50
$\vec{c} \to m, \mu_r$	Farnebäck	0.68	**1.83**	**3.97**	**8.93**	**0.41**	**0.68**	**0.79**	0.72
	SpyNet	0.68	2.32	4.88	9.87	0.49	0.83	0.85	0.72
$(r, \vec{v}) \to (m, \mu_r)$	Farnebäck	0.65	**1.38**	**4.00**	**11.71**	**0.46**	**0.56**	**0.86**	**0.97**
	SpyNet	**0.64**	2.29	5.99	13.93	0.58	0.99	0.98	1.13

5.2 Ablation Study

Table 5.6: Inference time mean and standard deviation ($\mu \pm \sigma$) in seconds (Inf. Time) and mean label accuracy (Label Acc.) for *predict friction*, $\vec{v} \to \mu_r$, setting for ϕ trained with different inputs.

Setting	ϕ Input	Inf. Time (s)	Mean Label Acc.
$\vec{v} \to \mu_r$	Buffer + Optical Flow (SpyNet)	0.2347 ± 0.0695	**0.8869**
	Optical Flow (SpyNet)	0.2288 ± 0.0687	0.8858
	Optical Flow (Farnebäck)	$\mathbf{0.2111 \pm 0.0630}$	0.8686
	Buffer + Optical Flow (Farnebäck)	0.2199 ± 0.0654	0.8656
	Buffer Only	0.2167 ± 0.0651	0.8560

The results presented in Table 5.6 shed light on how the choice of input affects both the inference time and label accuracy of the model, revealing the significance of optical

flow for training the CNN. We specifically examine the outcomes for the *predict friction* setting, which relies solely on the motion of the spheres to determine object state.

The highest label accuracy (0.8869) is achieved when the model is trained with buffer images combined with SpyNet optical flow. A close accuracy (0.8858) is obtained when using SpyNet optical flow alone, indicating that the model heavily relies on optical flow for inference in this setting. The lowest label accuracy (0.8560) corresponds to the model trained with buffer images only, further supporting this observation.

The quickest inference time is achieved using only Farnebäck optical flow as input. However, the improvement is marginal compared to other models and is largely negligible when considering the standard deviation.

Interestingly, the combination of buffer images and Farnebäck optical flow results in a lower label accuracy (0.8656) than using Farnebäck optical flow alone (0.8686). This implies that the additional information provided by the buffer may not significantly contribute to enhancing label accuracy in this specific setting.

The results suggest that optical flow is vital for attaining higher label accuracy, and its influence on inference time is minimal. Hence, it is clearly beneficial to the model architecture.

5.3 Qualitative Results

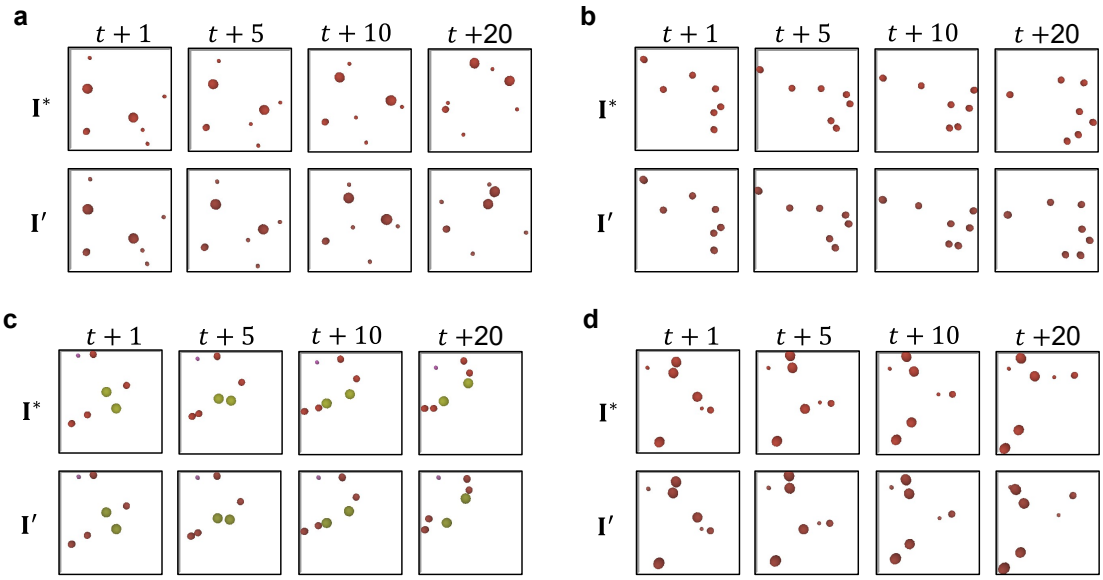

Figure 5.3: Comparison of actual Image and reconstruction at each future time steps $t \in [1,5,10,20]$ for each setting with model using Farnebäck Optical Flow and $n = 7$: (a) $r \to m$; (b) $\vec{v} \to \mu_r$; (c) $\vec{c} \to m, \mu_r$; (d) $(r, \vec{v}) \to (m, \mu_r)$.

In Figure 5.3, we display predictions for random samples (with $n = 7$) in each setting using model with Farnebäck optical flow. In all cases, the model's predictions closely mirror the ground truth images up to $t + 20$, which is consistent with our experimental

findings. These results show high object proposal generation accuracy and mIoU, along with low position prediction errors as time progresses in a cluttered environment.

Chapter 6

Conclusion

6.1 Summary

In this project, we introduce a new visual de-animation model for spheres on planar surfaces, which exhibits high performance in object detection, object parameter estimation, scene prediction, and reconstruction. Our model builds upon the original visual de-animation model by employing circularity for object proposal generation and optical flow for extrinsic parameter estimation, as opposed to colour filters and a neural network for both intrinsic and extrinsic parameters. This modification increases the robustness and precision of our model, since circularity is a more direct detection method for spheres compared to colour filters, and extracting velocity directly from the optical flow reduces the risk of velocity estimation corruption during feed-forward processing in the CNN.

We provide open-source utilities for easy training, evaluation, and testing of our model, available at `https://github.com/J0HNN7G/intuitive_physics`. This repository includes scripts for reproducing the training and evaluations described in this dissertation.

Additionally, we offer a new data set generator for spheres on planar surfaces using PyBullet, facilitating structured experimentation with an easy generation process.

6.2 Discussion

6.2.1 Performance

Our VDA model demonstrates high performance and speed (approximately 5 FPS) on all settings, including those with traditional and neural network optical flow. The model achieves impressive accuracy, precision, recall, F1 score, and mIoU for object proposal generation. It also performs well in scene reconstruction pixel MSE and future prediction position and velocity error. This suggests that our model is effective in handling scenes involving spheres on planar surfaces and can accurately extract the underlying object states from both appearance and motion.

6.2.2 Object Proposal Generation

The blob detection method used for object proposal generation has shown high performance across various settings with different appearances of spheres. It effectively identifies spheres regardless of their size and colour, indicating that circularity is a potent criterion for detecting objects like spheres on planar surfaces. This success highlights the generalizability of our model to different appearances and scenarios involving spherical objects.

6.2.3 Optical Flow Estimation

The optical flow estimation for initial position and velocity of spheres demonstrates low error and good stability in future predictions. While we do not have access to the original VDA model or their data set, a comparison with the results in their paper suggests that our model achieves lower or at least comparable error [49].

The choice between the Farnebäck method and SpyNet for dense optical flow estimation remains inconclusive based on our results. The Farnebäck method seems to be slightly faster and more consistent than SpyNet. Farnebäck and SpyNet yields similar label accuracy for all settings, except for the *predict friction* setting, where it achieves a more significant improvement. Overall, our results indicate that a neural optical flow network is not necessary for high model performance, a point not clarified in the original VDA paper.

6.3 Future Work

Below we provide a list of potential avenues that could be explored to validate and extend our VDA model:

- **Model comparison:** A direct comparison between our VDA model and the original VDA model, as well as other intuitive physics models, will provide more insight into the performance and significance of our model. Unfortunately, we did not have time for this during the project.
- **Real-life data set:** Evaluating our model on real-life data would reveal how well it generalizes to real-world scenarios. Unfortunately, we did not have time to pursue this during the project.
- **Extend VDA model:** Investigating the applicability of our VDA model to handle other tasks or scenarios, such as non-spherical objects, non-planar surfaces, or more complex environments, will help expand its potential use cases.
- **Experiment with physics engines:** Testing the VDA model with different physics engines can help determine the impact of various underlying physics simulations on the model's performance and predictions. More specifically, we did not have time to test our model with a differentiable physics engine during the project.
- **Experiment with graphics engines:** Exploring different graphics engines can reveal how different rendering methods and image representations affect the

model's performance in scene reconstruction. We did not have time to experiment with a differentiable neural renderer.

- **Experiment with convolutional neural networks:** Evaluating the VDA model with different convolutional neural networks can provide insights into the most effective network architectures for processing and extracting useful features from the input data. More specifically, it would be interesting to explore whether a more lightweight architecture, such as LeNet, would be sufficient for extracting the underlying state representation. This could potentially reduce memory usage and increase inference speed for the model [26].
- **Improve optical flow:** Our model largely relies on the high accuracy of the optical flow method for its performance. Developing and testing new optical flow methods or enhancing existing ones could lead to better performance in velocity estimation, allowing the model to handle more complex scenarios.

6.4 MInf Part 2

The primary goal of this report has been the comprehension and implementation of visual de-animation for spheres on planar surfaces. In the next phase, it would be desirable to compare the developed model with the original VDA model and the VIN model, using both synthetic and real-life data. Furthermore, extend the model's capabilities to tackle 3D intuitive physics problems, such as a block-stacking task where the model predicts how blocks will fall. The current model's high performance is largely attributed to its reliance on optical flow, so integrating 3D scene flow would be a promising adaptation of the model to handle scenarios in 3D [45].

Owing to the novelty of visual de-animation and intuitive physics models, there are numerous unexplored avenues for improvement that have not been addressed in this dissertation. Consequently, the ideas presented here should not be considered as absolute goals for Part 2 of the project.

Bibliography

[1] D.G Alciatore. *The illustrated principles of pool and billiards*. Sterling, 2004.

[2] Anton Bakhtin, Laurens , Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning. *arXiv Computing Research Repository (CoRR)*, 2019.

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[4] Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences (PNAS)*, 2013.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] G. Bradski and A. Kaehler. Opencv. *Dr. Dobb's Journal of Software Tools*, 25(11):120–126, 2000.

[7] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. *European Conference on Computer Vision (ECCV)*, 2012.

[8] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv Computing Research Repository (CoRR)*, 2016.

[9] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `https://pybullet.org/wordpress/`, 2016.

[10] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick , Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[11] Sébastien Ehrhardt, Aron Monszpart, Niloy J Mitra, and Andrea Vedaldi. Unsupervised intuitive physics from visual observations. *Asian Conference on Computer Vision (ACCV)*, 2018.

[12] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. *Image Analysis*, 2003.

[13] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *International Conference on Learning Representations (ICLR)*, 2016.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[15] Oliver Groth, Fabian B Fuchs, Ingmar Posner, and Andrea Vedaldi. Shapestacks: Learning vision-based physical intuition for generalised object stacking. *European Conference on Computer Vision (ECCV)*, 2018.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. *Artificial Neural Networks and Machine Learning (ICANN)*, 2011.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[19] Berthold K.P. Horn and Brian G Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981.

[20] Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *Computer Vision and Image Understanding*, 2015.

[21] Justin Johnson, Bharath Hariharan, Laurens , Maaten, Li Fei-Fei, Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv Computing Research Repository (CoRR)*, 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[25] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Howard R E, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[26] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv Computing Research Repository (CoRR)*, 2016.

[28] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of DARPA Image Understanding Workshop (IUW)*, 1981.

[29] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[30] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[31] Simon Niklaus. A reimplementation of SPyNet using PyTorch. `https://github.com/sniklaus/pytorch-spynet`, 2018.

[32] NVIDIA. GeForce GTX 1060. `https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1060/specifications/`, 2016.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary 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. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[34] Luis Piloto, Ari Weinstein, Peter Battaglia, and Matthew Botvinick. Intuitive physics learning in a deep-learning model inspired by developmental psychology. *Nature Human Behaviour*, 2022.

[35] Guillaume Plique. iwanthue, software to generate and refine palettes of optimally distinct colors. `https://medialab.github.io/iwanthue/`, accessed 2023.

[36] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[37] Ronan Riochet, Mario Castro, Mathieu Bernard, Adam Lerer, Rob Fergus, Véronique Izard, and Emmanuel Dupoux. Intphys: A framework and benchmark for visual intuitive physics reasoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018.

[38] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[39] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[41] Jianbo Shi and Carlo Tomasi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[43] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[44] Shubham Tulsiani, Saurabh Gupta, David Fouhey, Alexei A Efros, and Jitendra Malik. Factoring shape, pose, and layout from the 2d image of a 3d scene. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[45] Sundar Vedula, Simon Baker, Robert Collins, Takeo Kanade, and Peter Rander. Three-dimensional scene flow. *Proceedings of the International Conference on Computer Vision (ICCV)*, 2:722, 1999.

[46] Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. Visual interaction networks: Learning a physics simulator from video. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[47] Eric W. Weisstein. Circle-circle intersection. `https://mathworld.wolfram.com/Circle-CircleIntersection.html`, accessed 2023.

[48] Richard S Wright, Benjamin Lipchak, and Nicholas Haemel. *OpenGL(R) SuperBible: Comprehensive tutorial and reference (4th edition)*. Addison-Wesley Professional, 4 edition, 2007.

[49] Jiajun Wu, Erika Lu, Pushmeet Kohli, William T Freeman, and Joshua B Tenenbaum. Learning to see physics via visual de-animation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[50] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[51] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.