Towards learning monocular 3D object localization from 2D labels using the physical laws of motion

Daniel Kienzle, Katja Ludwig, Julian Lorenz, Rainer Lienhart

Angaben zur Veröffentlichung / Publication details:

Kienzle, Daniel, Katja Ludwig, Julian Lorenz, and Rainer Lienhart. 2024. "Towards learning monocular 3D object localization from 2D labels using the physical laws of motion." In *International Conference on 3D Vision 2024 (3DV), March 18-21, 2024, Davos, Switzerland*, edited by Theodora Kontogianni, Akihiro Sugimoto, and Gopal Sharma, 1564–73. Piscataway, NJ: IEEE. https://doi.org/10.1109/3DV62453.2024.00155.

Towards Learning Monocular 3D Object Localization From 2D Labels Using the Physical Laws of Motion

Daniel Kienzle Katja Ludwig Julian Lorenz Rainer Lienhart
University of Augsburg
86159 Augsburg, Germany
{firstname.lastname}@uni-a.de

Abstract

We present a novel method for precise 3D object localization in single images from a single calibrated camera using only 2D labels. No expensive 3D labels are needed. Thus, instead of using 3D labels, our model is trained with easy-to-annotate 2D labels along with the physical knowledge of the object's motion. Given this information, the model can infer the latent third dimension, even though it has never seen this information during training. Our method is evaluated on both synthetic and real-world datasets, and we are able to achieve a mean distance error of just 6 cm *in our experiments on real data. The results indicate the method's potential as a step towards learning 3D object location estimation, where collecting 3D data for training is not feasible.*

1. Introduction

The 3D location of objects is crucial in many application domains, such as in robotics or in the performance analysis of athletes in sports tournaments. Especially sports broadcasting companies are greatly interested in obtaining 3D information about an object's location in a fixed environment [22], for example the position of the ball in soccer, basketball, tennis, or squash. Traditional technologies like Hawk-Eye [13] and View 4D [32] often rely on triangulation techniques [21, 23, 26] for the calculation of the ball's position. A major limitation of these methods is the need for expensive hardware (e.g. multiple synchronized cameras), consequently preventing their application in low budget and amateur sports where only a single camera is used to record the game. In contrast to these methods, a neural network can be trained to predict the 3D position of the ball even if the game is recorded with just a single camera. However, training such a network typically requires 3D ground truth data. Although ground truth data may be readily available for some disciplines such as soccer, it is often lacking in lower-budget sports like squash. Consequently, there is a need for a method that can perform ball localization without relying on 3D ground truth data as supervision.

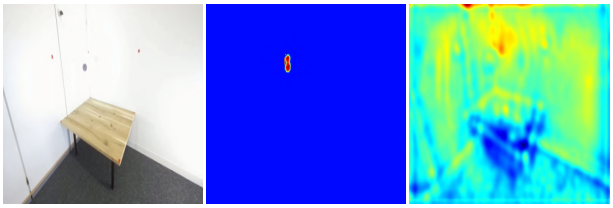
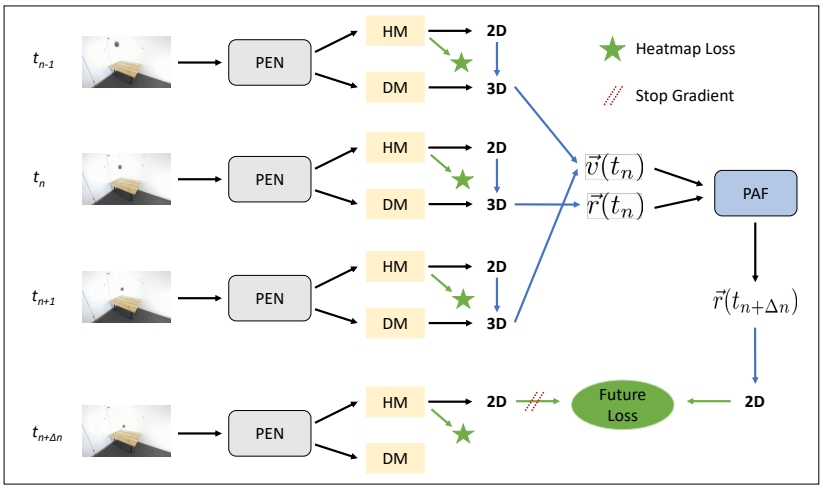


Figure 1. Predictions of the *Position Estimation Network*. The left image is the input to the network, the middle image shows the predicted heatmap, and the right image shows the predicted depthmap. By combining the heatmap and depthmap, the 3D position of the ball can be calculated.

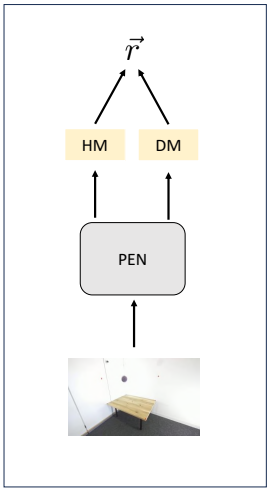
In this paper, we present a novel method for training a neural network to perform 3D localization that does not need 3D ground truth labels as supervision at all. Instead, only 2D image labels of the position of a desired object are needed, which can be easily obtained through methods such as object detection or manual annotation by clicking on the object's center in an image. Our model is trained using video clips, leveraging the physical laws of motion to enable the network to infer the latent third dimension.

A key advantage of our method is its flexibility in inference. While videos are needed during training, our trained model can also be applied to single images during inference. This ensures that our network can even be applied in situations where the object's motion cannot be precisely described by physics (e.g. due to human interaction) or where the physical parameters are not known completely (e.g. if a sudden wind is affecting the ball). While methods processing multiple frame will likely fail in those situations, our method is robust to such changes.

The primary focus of this paper is on training our neural network on video sequences of the ball's movement between contacts with players. During this period, the ball remains unaffected by non-deterministic forces, enabling us to ef-



(a) Training with videos using explicit physical knowledge.



(b) Inference with single images. No explicit physical knowledge necessary.

Figure 2. Training and inference pipeline. The *Position Estimation Network (PEN)* estimates the heatmap (HM) and depthmap (DM) for each input image, from which the 2D image coordinates and camera depth are extracted. Using the intrinsic camera matrix, we compute the 3D camera coordinates from the image coordinates and camera depth. With the extrinsic camera matrix, the world coordinates $\vec{r}$ are calculated. During training, we apply the *PEN* to multiple coherent frames for the estimation of the velocity $\vec{v}$. The *Physics Aware Forecast Module (PAF)* utilizes these initial conditions to calculate the world coordinates at a later time $t_{n+\Delta n}$ by solving the differential equations of motion. These coordinates are then projected back to image coordinates and the *future loss* is calculated. The *heatmap loss* is calculated by comparing the predicted and ground truth heatmaps at all time steps. During inference, the PEN is applied to single images. Thus, it is robust to incalculable situations like sudden wind or human interventions. Blue arrows indicate the use of camera matrices, and dotted red lines indicate detached gradients.

fectively utilize the physical laws of motion to describe its trajectory in the training videos. Importantly, we are able to describe the ball bouncing off walls and the floor, which is e.g. important in various disciplines like tennis or squash.

Since our method is not only limited to sport's applications, we study videos of a moving ball in a more general context. The main contributions of this paper are:

- We propose a general method to train a neural network for 3D object localization in single images without the need of 3D labels.
- We create a synthetic and a real-world dataset with 3D ground truths for the evaluation of our method. The datasets will be published in order to ensure reproducibility and to encourage further research in this field.
- We prove experimentally that our model is able to accurately predict the 3D position of the ball and discuss the experimental results thoroughly.

2. Related Work

Pose estimation is widely used in many sport disciplines, and it is of great importance in athletics [20], table tennis [16], swimming [9], and soccer [2]. In this paper we estimate the position of an object instead of a human pose. Nevertheless, our architecture is inspired by methods commonly used in human pose estimation, since we use 2D heatmaps to encode the object's position.

In addition to human pose estimation, accurate ball localization is crucial in the analysis of many sport disciplines. Several methods have been developed to calculate physical reasonable 3D positions of the ball from the sequence of related 2D positions. For instance, [18] localize a badminton ball using videos, [5] reconstruct the 3D trajectory of a basketball from a sequence of 2D positions and [6] calculate the trajectory of a volleyball. However, these methods rely on fitting the ball's trajectory to physically calculated paths and, thus, are only applicable to video sequences exhibiting perfect ballistic trajectories. They cannot handle situations where the ball is subject to non-deterministic forces (e.g., human actions) or when the physical parameters change (e.g., the introduction of strong winds or changes in drag coefficients due to wear and tear). Moreover, these methods do not account for ball bouncing. In contrast to these methods, we are also able to describe more intricate physical situations like a ball bouncing off the floor. Furthermore, since our method can be applied to single images during inference, it remains effective even when the ball's behavior is non-deterministic or when precise knowledge of physical parameters is lacking. Thus, our method overcomes many limitations of previous approaches.

Other studies like [30, 31] estimate the 3D position of a bas-

ketball in single images by measuring the diameter of the ball in the images. Similarly, [4] utilizes the ball diameter to predict the 3D position of a table tennis ball. Nevertheless, for these methods additional relatively costly segmentation labels of the ball are needed. In contrast, our method only needs less expensive 2D image-coordinate annotations of the ball either from manual or automatic annotations.
To the best of our knowledge, we are the first to develop a method for localizing the ball's 3D center coordinates in single images using only 2D labels together with knowledge of the physical motion. Even though we focus on the 3D localization of a ball, it is worth noting that our method can be applied to other objects as well.

As we utilize the physical knowledge of the moving ball for training our neural network, its physical motion has to be described in a differentiable manner. While the motion can be described by an analytic function for many toy problems, the physical equations of motion have to be solved numerically for most real life applications. We implement the differential equations using the framework of Neural Ordinary Differential Equations (NDEs) [7] in order to be able to calculate the gradient of the numerical solution. Our method furthermore resembles Hamiltonian Neural Networks (HNNs) and Lagrangian Neural Networks (LNNs) [11]. However, in contrast to HNNs and LNNs, we learn the ball's 3D coordinates given a fixed Hamilton function instead of learning the Hamilton or Lagrange function given fixed data.

3. Method

3.1. Overview

Our method consists of two main modules: A Position Estimation Network (PEN) and a Physics Aware Forecast Module (PAF). The PEN is a neural network that takes a single image as input and generates a heatmap as well as a depthmap as output. By extracting the 2D image coordinates from the heatmap and using the depth information from the depthmap along with the intrinsic and extrinsic camera calibration matrix, we calculate the camera and world coordinates of the ball. The PEN is applied to three successive images at time t_{n-1}, t_n, t_{n+1} to obtain the ball's world-coordinates for each image. Using the coordinates at time t_{n-1} and t_{n+1}, we estimate the ball's velocity at time t_n. Given the ball's world coordinates and velocity at time t_n, the PAF calculates the world coordinates at a later time $t_{n+\Delta n}$ by solving the differential equations of motion.
We implement two losses for the training of the PEN: The *heatmap loss* and the *future loss*. The heatmap loss compares the predicted and ground-truth heatmaps to teach the network the ball's 2D image coordinates. The *future loss* ensures that the PEN learns the correct camera depth by projecting the PAF's predicted coordinates at time $t_{n+\Delta n}$ to the 2D image coordinates for the frame at $t_{n+\Delta n}$.
An overview of our method is depicted in Figure 2. We note that only the PEN is used during inference.

3.2. Position Estimation Network (PEN)

The PEN takes an image $I \in \mathbb{R}^{C\times H\times W}$ as input with C, H and W being the number of channels, height and width of the image. Using a backbone architecture like ResNet [14] or ConvNeXt [19, 34], the features after each stage are bilinearly upsampled to the shape of the first stage and, afterwards, they are concatenated. A *heatmap head* consisting of one 1×1 convolution, two 7×7 convolutions and a final 1×1 convolution is applied to obtain a heatmap $\tilde{H} \in \mathbb{R}^{H\times W}$ from the concatenated features. Additionally, a separate *depthmap head* with the same architecture is applied to the concatenated features to calculate the depthmap $D \in \mathbb{R}^{H\times W}$. We intentionally choose a simple architecture for the heads to demonstrate the effectiveness of our method without requiring extensive architecture modifications. The architecture of the PEN is depicted in Figure 3.

The 2D image coordinates $\vec{\mathbf{r}}^{(\mathrm{I})} = \begin{pmatrix} x^{(\mathrm{I})} & y^{(\mathrm{I})} \end{pmatrix}^{\mathrm{T}}$ are extracted from the heatmap with a 2D soft-argmax by weighting each pixel-position with its corresponding heatmap value. Furthermore, the camera depth $z^{(\mathrm{C})}$ is calculated as the sum over all values of the depthmap weighted with the corresponding values of the 2D softmax of the heatmap. Consequently, the coordinates are calculated from the heatmap $\tilde{H}$ and the depthmap $\tilde{D}$ as

$$\begin{aligned} x^{(\mathrm{I})} &= \sum_{h=0}^{H-1}\sum_{w=0}^{W-1} w \cdot \left(\mathrm{softmax}\left(\beta \cdot \tilde{H}\right)\right)_{h,w} \\ y^{(\mathrm{I})} &= \sum_{h=0}^{H-1}\sum_{w=0}^{W-1} h \cdot \left(\mathrm{softmax}\left(\beta \cdot \tilde{H}\right)\right)_{h,w} \\ z^{(\mathrm{C})} &= \sum_{h=0}^{H-1}\sum_{w=0}^{W-1} \tilde{D}_{h,w} \cdot \left(\mathrm{softmax}\left(\beta \cdot \tilde{H}\right)\right)_{h,w} \end{aligned} \tag{1}$$

with the constant factor $\beta = 10$ to obtain sharp softmax probabilities and the pixel indices h and w.

3.3. Coordinate Transformations

We assume that the intrinsic camera matrix $\mathbf{M}_{\text{int}}$ and the extrinsic camera matrix $\mathbf{M}_{\text{ext}}$ are known, and we discuss in the supplementary material in more detail how the camera matrices can be obtained.
We use homogeneous coordinates and transform world coordinates into camera coordinates with the extrinsic matrix $\mathbf{M}_{\text{ext}} \in \mathbb{R}^{4\times 4}$ and camera coordinates into image coordinates with the intrinsic matrix $\mathbf{M}_{\text{int}} \in \mathbb{R}^{3\times 3}$. Since these transformations are invertible we calculate the camera-coordinates $\vec{\mathbf{r}}^{(\mathrm{C})} = \begin{pmatrix} x^{(\mathrm{C})} & y^{(\mathrm{C})} & z^{(\mathrm{C})} \end{pmatrix}^{\mathrm{T}}$ and the world-coordinates $\vec{\mathbf{r}}^{(\mathrm{W})} = \begin{pmatrix} x^{(\mathrm{W})} & y^{(\mathrm{W})} & z^{(\mathrm{W})} \end{pmatrix}^{\mathrm{T}}$ given the im-

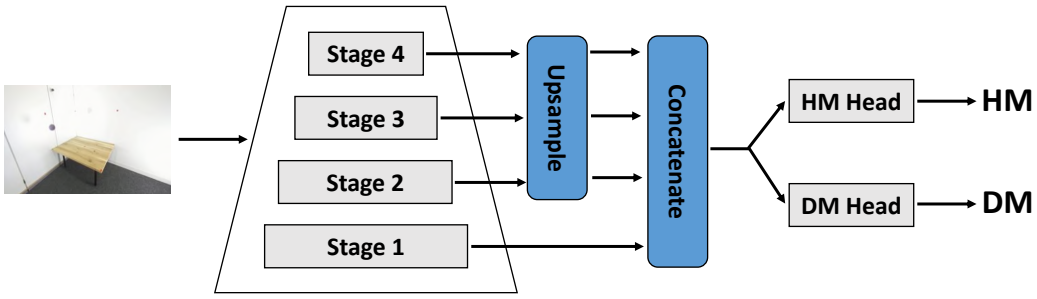


Figure 3. *Position Estimation Network (PEN)* architecture for generating both heatmap (HM) and depthmap (DM) from an input image. Features from the backbone network are concatenated and passed through the two heads. One generates a heatmap and the other produces a depthmap.

age coordinates $\mathbf{r}^{(\mathrm{I})}$ as well as the camera depth $z^{(\mathrm{C})}$ using the following formulas:

$$\begin{pmatrix} x^{(\mathrm{C})} \\ y^{(\mathrm{C})} \\ z^{(\mathrm{C})} \end{pmatrix} = \mathbf{M}_{\text{int}}^{-1} \cdot \begin{pmatrix} x^{(\mathrm{I})} z^{(\mathrm{C})} \\ y^{(\mathrm{I})} z^{(\mathrm{C})} \\ z^{(\mathrm{C})} \end{pmatrix}, \quad \begin{pmatrix} x^{(\mathrm{W})} \\ y^{(\mathrm{W})} \\ z^{(\mathrm{W})} \\ 1 \end{pmatrix} = \mathbf{M}_{\text{ext}}^{-1} \cdot \begin{pmatrix} x^{(\mathrm{C})} \\ y^{(\mathrm{C})} \\ z^{(\mathrm{C})} \\ 1 \end{pmatrix}. \quad (2)$$

Furthermore, the velocity $\vec{\mathbf{v}}(t_n)$ at time t_n is calculated from the ball's world coordinates at time t_{n-1} and t_{n+1} using the symmetric finite difference quotient as

$$\vec{\mathbf{v}}(t_n) = \frac{\vec{\mathbf{r}}^{(\mathrm{W})}(t_{n+1}) - \vec{\mathbf{r}}^{(\mathrm{W})}(t_{n-1})}{t_{n+1} - t_{n-1}}. \quad (3)$$

3.4. Physics Aware Forecasting Module (PAF)

Given the ball's world coordinates $\vec{\mathbf{r}}^{(\mathrm{W})}(t_n)$ and velocity $\vec{\mathbf{v}}^{(\mathrm{W})}(t_n)$ at time t_n as initial conditions, the PAF calculates the ball's coordinates at a later time $t_{n+\Delta n}$ by solving the differential equations of motion. While the ball's motion can be expressed as an analytical function for many simple toy problems, it is not always possible to find an analytical solution for more intricate situations. Since we want to describe a bouncing ball in an arbitrary environment, we solve the differential equations of motion numerically. By using the framework of Neural Differential Equations [7] we are able to calculate the gradient of the numerical solution that is needed for the backpropagation algorithm.

We adopt the Hamilton formalism to describe the motion of the ball. A physical system can be described according to [12] using the Hamilton function

$$\mathcal{H} = \frac{|\vec{\mathbf{p}}|^2}{2m} + V(\vec{\mathbf{r}}^{(\mathrm{W})}) \quad (4)$$

with the ball's mass m, the momentum $\vec{\mathbf{p}} = m\vec{\mathbf{v}}$ (for euclidean coordinates) and the potential $V(\vec{\mathbf{r}}^{(\mathrm{W})})$, which describes the physical behavior of the ball. According to the Hamilton formalism [12], the equations of motion are derived from the Hamilton function as

$$\frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{r}}^{(\mathrm{W})} = \frac{1}{m}\frac{\mathrm{d}}{\mathrm{d}\vec{\mathbf{v}}}\mathcal{H}, \qquad m\frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{v}} = -\frac{\mathrm{d}}{\mathrm{d}\vec{\mathbf{r}}^{(\mathrm{W})}}\mathcal{H} \quad (5)$$

and by solving these differential equations given the initial ball position and velocity we get the ball's position $\vec{\mathbf{r}}^{(\mathrm{W})}(t)$ at time t. Consequently, the equations 5 are solved in the PAF to obtain the ball's position at a later time.

The potential $V(\vec{\mathbf{r}}^{(\mathrm{W})})$ defines the physical behavior of the ball and, thus, we have to model it accordingly to ensure the correct prediction of the ball's movement. The potential of a ball in free fall above the floor at $z^{(\mathrm{W})} = 0$ is described as

$$V_{\mathrm{G}}(\vec{\mathbf{r}}^{(\mathrm{W})}) = m \cdot g \cdot \mathrm{ReLU}(z^{(\mathrm{W})}) \quad (6)$$

with g being the gravitational constant. Additionally, we also want to describe the bouncing off the floor. Ideally, the potential at the floor position is described as an infinite potential barrier. However, it is sufficient to approximate the infinite barrier as a fast increasing potential. Therefore, we model the potential at the floor position as

$$V_{\mathrm{F}}(\vec{\mathbf{r}}^{(\mathrm{W})}) = m \cdot c \cdot \mathrm{ReLU}(-z^{(\mathrm{W})}) \quad (7)$$

with $c = 1000\,\frac{\mathrm{J}}{\mathrm{m \cdot kg}}$ being a large constant. We note that the exact value of c is not important as long as it is significantly larger than g.

In some environments, we also have to include the bouncing off the walls or various obstacles. These can be defined similarly and are explained in the supplementary material for our specific datasets in more detail. As a result, the overall potential of a moving ball is described as

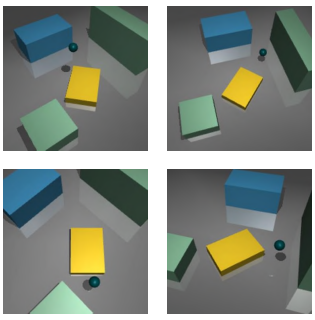
$$V(\vec{\mathbf{r}}^{(\mathrm{W})}) = V_{\mathrm{G}}(\vec{\mathbf{r}}^{(\mathrm{W})}) + V_{\mathrm{F}}(\vec{\mathbf{r}}^{(\mathrm{W})}) + V_{\mathrm{W}}(\vec{\mathbf{r}}^{(\mathrm{W})}) + V_{\mathrm{O}}(\vec{\mathbf{r}}^{(\mathrm{W})}) \quad (8)$$

with $V_{\mathrm{W}}(\vec{\mathbf{r}}^{(\mathrm{W})})$ describing the bouncing off the walls and $V_{\mathrm{O}}(\vec{\mathbf{r}}^{(\mathrm{W})})$ describing the bouncing off obstacles.

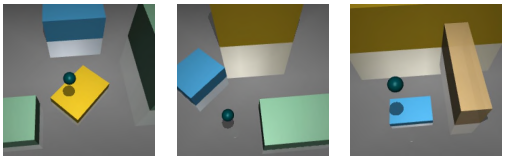
After defining the potential for a specific physical situation, we implement the differential equation 5 in the PAF. In our experiments, we use the 5$^{\text{th}}$ order Dormand-Prince method [8] to solve the differential equations numerically, but we note that other solvers can be used as well.

3.5. Loss

Since we do not use 3D ground truth labels, we utilize the *heatmap loss* and the *future loss* to train the PEN. As it is



(a) *SD*: images captured from different camera locations in the 1st environment.



(b) *SD*: images captured in different environments.

(c) *RD*: images captured from different camera locations.

Figure 4. Example images from the *synthetic dataset* (*SD*) and *real dataset* (*RD*).

common in human pose estimation (e.g. [17]), we also use heatmaps to teach the model to predict the 2D image coordinates. We calculate the L_2 loss between the heatmap output $\tilde{H}(t_n)$ of the PEN for the frame at time t_n and the ground truth heatmap $\tilde{H}_{\text{gt}}(t_n)$. This ground truth heatmap is a two-dimensional Gaussian centered at the ground truth image coordinates at time t_n.

The *future loss* utilizes the prediction of the PAF to teach the model the correct prediction of the camera depth. First, we transform the prediction of the PAF at time $t_{n+\Delta n}$ to image coordinates using the camera matrices and obtain the image coordinates $\vec{\mathbf{r}}^{(\text{I})}_{\text{PAF}}$. Then, we calculate the L_1 loss between these coordinates and the image coordinates calculated by the PEN for the frame at time $t_{n+\Delta n}$.

Consequently, the total loss is calculated as

$$\begin{aligned} \text{L} = &\left\|\vec{\mathbf{r}}^{(\text{I})}_{\text{PEN}}(t_{n+\Delta n}) - \vec{\mathbf{r}}^{(\text{I})}_{\text{PAF}}(t_{n+\Delta n})\right\|_{\text{L1}} \\ &+ \frac{1}{|\mathcal{T}|}\sum_{t_i\in\mathcal{T}} \left\|\tilde{H}(t_i) - \tilde{H}_{\text{gt}}(t_i)\right\|_{\text{L2}} \end{aligned} \tag{9}$$

with $\mathcal{T} = \{t_{n-1}, t_n, t_{n+1}, t_{n+\Delta n}\}$.

3.6. Evaluation Metrics

In order to test the quality of the PEN's 3D predictions we introduce the *distance to groundtruth (DtG)* metric. It measures the euclidian distance between the prediction and the ground truth 3D position. Accordingly, the metric is calculated as

$$DtG = \frac{1}{|\mathcal{V}|}\sum_{t_i\in\mathcal{V}} \left\|\vec{\mathbf{r}}^{(\text{C})}_{\text{gt}}(t_i) - \vec{\mathbf{r}}^{(\text{C})}_{\text{PEN}}(t_i)\right\|_{\text{L2}} \tag{10}$$

over all images in the test set $\mathcal{V}$. We note that the metric remains the same regardless of whether world coordinates or camera coordinates are used.

Since we expect the estimation of the ball's 3D position to be less accurate if the ball is further away from the camera, we define the *binned distance to groundtruth (DtG$_b$)* metric. Depending on the ball's groundtruth camera depth $z^{(\text{C})}_{\text{gt}}$, the predictions are grouped into b bins and the *DtG* is calculated for each bin separately. Thus, it is calculated as

$$DtG_b = \begin{pmatrix} \frac{1}{|\mathcal{V}_0|}\sum\limits_{t_i\in\mathcal{V}_0} \left\|\vec{\mathbf{r}}^{(\text{C})}_{\text{gt}}(t_i) - \vec{\mathbf{r}}^{(\text{C})}_{\text{PEN}}(t_i)\right\|_{\text{L2}} \\ \vdots \\ \frac{1}{|\mathcal{V}_{b-1}|}\sum\limits_{t_i\in\mathcal{V}_{b-1}} \left\|\vec{\mathbf{r}}^{(\text{C})}_{\text{gt}}(t_i) - \vec{\mathbf{r}}^{(\text{C})}_{\text{PEN}}(t_i)\right\|_{\text{L2}} \end{pmatrix} \tag{11}$$

with $\mathcal{V}_j = \{t_i \in \mathcal{V} \mid z^{(\text{C})}_{\text{gt}}(t_i) \in [j\cdot\frac{z_{\max}-z_{\min}}{b} + z_{\min}, (j+1)\cdot\frac{z_{\max}-z_{\min}}{b} + z_{\min}]\}$ and $j \in \{0, ..., b-1\}$. We select suitable values for the number of bins b, the minimal distance $z_{\min}$ and the maximal distance $z_{\max}$ depending on each dataset.

4. Dataset

To test our method we create two datasets consisting of monocular videos of a moving ball: the *synthetic dataset (SD)* and the *real dataset (RD)*. For the training we provide the ball's 2D positions for every frame and for the evaluation we also provide the ball's 3D position. Some example images are depicted in Figure 4 and more images are visualized in the supplementary material.

4.1. Synthetic Dataset

We create the *SD* using the general purpose physics engine MuJoCo [29] and, thus, we are able to create synthetic data with realistic physical behavior. We simulate a moving ball under the influence of gravity. The ball bounces off an infinite floor and additional finite obstacles placed in the environment. We generate 3 different environments and for each environment we capture the scenes from 9 different camera location. However, each distinct scene is only captured from a single camera in contrast to traditional triangulation and binocular vision settings. The cameras are placed such that the distance to the origin of the scene is between $8\,\text{m}$ and $10\,\text{m}$, and the exact camera locations are given in the supplementary material. The videos are recorded with $30\,\text{FPS}$. We create 3 versions of the dataset:

Table 1. *DtG* scores for the *synthetic dataset*.

(a) *DtG* scores per camera location evaluated on the 1st environment.

	DtG $\pm$ Δ*DtG* (cm) $\downarrow$			
training set	camera 1	camera 7	camera 8	camera 9
SD-S	22 ± 19	-	-	-
SD-M	19 ± 10	27 ± 23	23 ± 9	21 ± 10
SD-L	11 ± 6	28 ± 25	15 ± 8	16 ± 7

(b) *DtG* scores per environment.

	DtG $\pm$ Δ*DtG* (cm) $\downarrow$		
training set	env 1	env 2	env 3
SD-S	-	-	-
SD-M	22 ± 16	-	-
SD-L	18 ± 16	17 ± 16	18 ± 13

- *SD-S*: We use a single environment and capture the scene from a single fixed camera location. We create 100 videos for the training set, 100 for the validation set and 100 for the test set.
- *SD-M*: We use a single environment and capture the scene from 9 different camera locations. The training set consists of 100 videos per camera location recorded from camera locations 1-6, while the validation and test set each contain 100 videos per camera location recorded from all 9 camera locations.
- *SD-L*: We include videos recorded in 3 different environments and for each environment the scene can be captured from 9 different camera locations. For each environment, the training set contains 100 videos per camera location recorded from camera locations 1-6, while the validation and test set contain 100 videos per camera location recorded from all 9 camera locations.

The versions are designed such that *SD-M* is a subset of *SD-L* and *SD-S* is a subset of *SD-M*. Each image has a resolution of 224×224 pixel and each clip consists of 30 consecutive frames during training. Example frames for different camera locations in the 1st environment are shown in Figure 4a, and for the three different environments in Figure 4b. Random initial positions and velocities of the ball are selected for each video to ensure diversity among the samples.

By applying our method to *SD-S* we are able to show that our model is capable of learning the ball's 3D position in general. However, real use cases like sport broadcasting might not use a fixed camera, instead the camera sometimes moves and changes its location between the scenes. Therefore, we use the *SD-M* to show that our model is also able to learn the ball's 3D position if the camera changes its location. Finally, it is beneficial for practical applications to train a model that is working on different environments like different known courts in sport broadcasting. Using the *SD-L*, we show that our model is also able to learn the ball's 3D position even if the environment changes. By applying our method to these three synthetic datasets we test some characteristics needed for practical application.

4.2. Real Dataset

Since the synthetic images are generated by a physics engine, they are visually not perfectly realistic. Therefore, we create the *RD* to prove that our method is also able to cope with noisy input data. We use a ZED 2i stereo camera [28] to record a rubber ball bouncing off the floor, the walls and an additional obstacle. We record each video from one of two distinct camera locations and use a frame rate of 60 FPS. The distance of the camera to the origin of the scene is between $1.3\,\mathrm{m}$ and $1.4\,\mathrm{m}$. Only the data of the left camera is used as input to the PEN, and we manually labelled the 2D image coordinates for each frame. We depict example frames from the two different camera locations in Figure 4c. Each frame is resized to 224×384 pixel. We split the videos into clips consisting of 16 frames each, with 251 clips in the train set, 34 clips in the validation set, and 81 clips in the test set. In order to evaluate our method we use the data of the stereo camera to calculate a ground truth depthmap of the scene. The ground truth camera depth is obtained from the depthmap value at the 2D image coordinates of the ball. By applying our method to this dataset we are able to show that it also works with simple yet realistic input data.

5. Experiments

In this section we perform multiple experiments and apply our method to the datasets described in section 4. We use the first 4 stages of a ResNet34 [14] pretrained on ImageNet [27] as the backbone of the PEN and provide additional experiments with different backbones in the supplementary material. To ensure performance stability, we maintain a second copy of our model whose weights are an exponential moving average (EMA) of the trained model weights, and we update the EMA model after each iteration. At the end of each epoch the EMA model is evaluated on the validation set, and we only keep the model with the best *DtG* score. We calculate the *future loss* at multiple time steps. For the synthetic data we use $\Delta n = 4$, $\Delta n = 8$ and $\Delta n = 15$ and for the real data we use $\Delta n = 2$, $\Delta n = 4$ and $\Delta n = 6$, because the average speed of the real ball is larger than the speed of the synthetic ball. Our code and datasets will be published at www.example.com where we will also provide additional visual evaluations of our experiments.

5.1. Synthetic Experiments

In this section we evaluate our method on the synthetic datasets described in section 4. We train three models, one for each subset of the synthetic datasets:

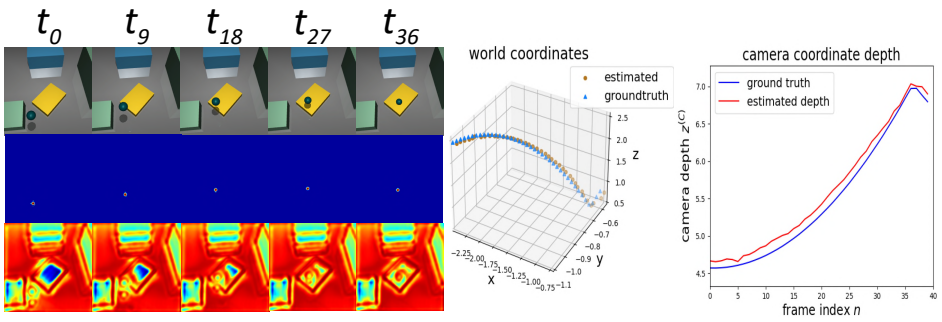


Figure 5. Predictions of the *SD*-L model on a sample video from the 1st camera location in the 1st environment. The first row shows the input images, the second row shows the predicted heatmaps, and the third row shows the predicted depthmaps. On the right, a 3D plot of the predicted ball trajectory $\vec{\mathbf{r}}^{(\mathrm{W})}$ and a 2D plot of the camera depth $z^{(\mathrm{C})}$ is shown.

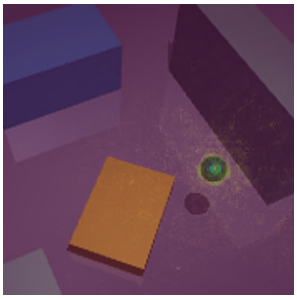


Figure 6. Saliency Map for a prediction of the SD-L model.

- *SD-S*: The first model is trained solely on the *SD-S* training set. Therefore, the model is only exposed to images from a single camera location and a single environment during both training and testing.
- *SD-M*: The second model is trained on the *SD-M* training set, which includes images from the first 6 camera locations of the 1st environment. However, during evaluation, it is tested on videos recorded from all camera locations of the 1st environment. By comparing its results on camera 1 to the results of the first model, we test if training on multiple camera locations improves the model's performance. By looking at the results on the last three camera locations we test if the model is able to generalize to previously unseen camera locations.
- *SD-L*: The third model is trained on the *SD-L* training set, which includes images from the first 6 camera locations of each environment. During evaluation, it is tested on all camera locations of each environment in the test set. With this model we are able to test whether the model is able to learn the ball's position for different known environments. Moreover, we test if the model benefits from being trained on multiple environments by comparing the results of this model to the results of the second model on the 1st environment.

The results on the test set are given in Table 1. For the three models, we evaluate the average *DtG* sores on the videos from exemplary camera locations in the 1st environment in Table 1a. Since the performance of the *SD-M* model is significantly better than the performance of the *SD-S* model, we conclude that the PEN is not just able to deal with input from multiple camera perspectives, but it even benefits from being trained with additional data from multiple camera location. Moreover, we see that the model is able to generalize to previously unseen camera locations since the scores for camera 7 to 9 are also good. By comparing the *SD-L* model to the *SD-M* model, we see that the *SD-L* model achieves even slightly better scores and, consequently, we conclude that it is possible to apply the PEN to multiple known environments. In Table 1b we see the average *DtG* scores on the three environments. The *SD-L* model performs again better than the *SD-M* model, thus, reinforcing the impression that training the model on multiple environments is beneficial. The results of the *SD-L* model are visualized in Figure 5, and it is visible that the predictions match the 3D ground truth very accurately.

To conclude, we see that it is not only possible to train our model with distinct videos from multiple camera locations, but even recognize the fact that using distinct videos from multiple camera locations is beneficial. This is important for the application to realistic data (e.g. sports broadcasting) as in many cases the camera is changing its position instead of being stationary. Moreover, we also show that our method is capable of dealing with multiple known environments, which is also an important property for practical applications (e.g. different stadiums or courts in sport broadcasting).

5.2. Real Experiments

Even though we were able to experimentally prove some properties needed for practical applications on the *synthetic dataset*, we still need to show that our model is able to work with real data. Important difficulties in working with real videos are that the model has to be able to filter out pixel noise in the image data and that it needs to be capable of working with noisy labels. Because the 2D image labels are manually annotated and do not always perfectly match the center of the ball, additional noise is also added to the calculation of the future loss. Furthermore, the physical description of the ball might not perfectly match the ball's real behavior (e.g. due to deformations of the ball or spin), resulting in reduced precision of the forecasting. To experimentally demonstrate that our model is able to cope with these difficulties, we evaluate our method on the *real dataset*. The results on the test set of the real dataset are shown in Table 2. We achieve a *DtG* score of 7 cm for the images recorded from the 1st camera location and a *DtG*

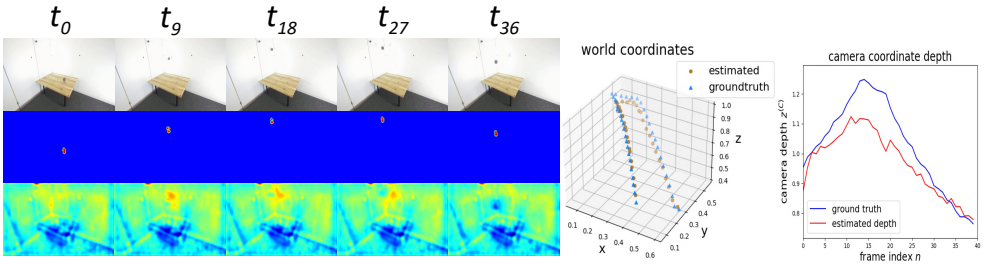


Figure 7. PEN's predictions on a sequence from the *real dataset*. The first row shows the input images, the second row shows the predicted heatmaps, and the third row shows the predicted depth map. On the right, a 3D plot of the predicted ball trajectory $\vec{\mathbf{r}}^{(\mathrm{W})}$ and the camera depth $z^{(\mathrm{C})}$ is shown.

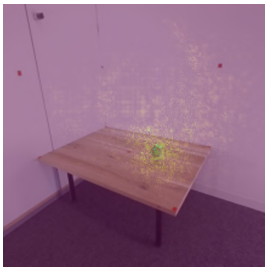


Figure 8. Saliency map for a prediction of the RD model.

Table 2. Scores per camera location on the *real dataset*.

	$DtG \pm \Delta DtG$ (cm) ↓		$DtG_3 \pm \Delta DtG_3$ (cm) ↓	
training set	camera 1	camera 2	camera 1	camera 2
RD	7 ± 4	6 ± 4	$\begin{pmatrix} 8 \pm 4 \\ 6 \pm 2 \\ 11 \pm 4 \end{pmatrix}$	$\begin{pmatrix} 4 \pm 2 \\ 6 \pm 3 \\ 11 \pm 4 \end{pmatrix}$

score of 6 cm for the images from the 2nd camera location, thus achieving very precise predictions. Consequently, our method is able to deal with the difficulties of real data.
Furthermore, we calculate the DtG_3 scores by dividing the data into 3 bins according to equation 11 with $z_{\min} = 5\,\mathrm{cm}$ and $z_{\max} = 2\,\mathrm{m}$. By analyzing the individual bins, we see that the scores in the 3rd bin are noticeably worse. This can be explained by the fact that it is harder for the model to predict the ball's location when it is farther away from the camera. However, the predictions are still very accurate in the third bin. Nevertheless, further research on this scaling behavior is needed for the next step towards future applications. In figure 7 we show the predictions of the model for an example video sequence. Looking at the 3D plot, it is also clear that the predicted trajectory of the ball matches the ground truth trajectory very closely.

5.3. Emergence of Depthmaps

Figure 5 and 7 clearly reveal the emergence of the scene's geometric structure in the depthmaps, despite the fact that precise depth prediction is only necessary in the vicinity of the ball to produce accurate outputs. Thus, it appears that the model is able to learn a full depthmap of the environment. This observation highlights an intriguing characteristic of our method that raises the possibility of indirect training for monocular depth estimation. However, a comprehensive analysis of this phenomenon is beyond the scope of this paper, and future research is necessary to evaluate this aspect in more detail.

5.4. Interpretation of the learning process

In this section, we delve into the question of what information the PEN utilizes in the images to calculate the depth. It is possible that the network simply measures the diameter of the ball in the image for the calculation of the depth (similar to [30]). Another possibility is that the network compares the ball to its surrounding environment.
To gain a rough understanding of the regions in the image that are important for the calculation of the depth, we calculate a saliency map S for the input images as

$$S_{i,j} = \sum_{c=0}^{3} \frac{\partial z^{(\mathrm{C})}}{\partial I_{c,i,j}} \tag{12}$$

where I is the input image, $z^{(\mathrm{C})}$ is the predicted depth of the ball, i and j index the pixels of the image, and c denotes the channel index. In Figure 6 and 8 we depict the saliency map for a real as well as a synthetic image. We observe the highest values around the ball, indicating that the PEN focuses mainly on the size of the ball for the estimation of the depth. However, we also see that the PEN considers the local surroundings of the ball, suggesting that this additional information enhances the depth estimation beyond a simple diameter measurement. Furthermore, we conjecture that the learned depthmap of the scene is a result of the PEN attending to the local context around the ball.

6. Conclusion

This paper has introduced a novel approach for monocular 3D object localization and demonstrated its effectiveness through experiments on both synthetic and real datasets. Our proposed method eliminates the need for 3D labels as supervision by leveraging the physical equations of motion, allowing the model to infer the latent third dimension. We discussed various properties relevant to real-world scenarios and demonstrated the robustness of our method in handling them. Consequently, our method is a promising step towards 3D object localization applications, although further research is needed to enhance its practical usability. Therefore, we intend to conduct further research to refine and expand upon our method, focusing on areas like scalability, interpretability, and generalization ability.

References

[1] Y.I Abdel-Aziz, H.M. Karara, and Michael Hauck. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. *Photogrammetric Engineering and Remote Sensing*, 81(2):103–107, 2015. 2

[2] Lewis Bridgeman, Marco Volino, Jean-Yves Guillemaut, and Adrian Hilton. Multi-person 3d pose estimation and tracking in sports. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[3] C. G. BROYDEN. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970. 2

[4] Jordan Calandre, Renaud Peteri, Laurent Mascarilla, and Benoit Tremblais. Extraction and analysis of 3d kinematic parameters of table tennis ball from a single camera. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 9468–9475, 2021. 3

[5] Hua-Tsung Chen, Ming-Chun Tien, Yi-Wen Chen, Wen-Jiin Tsai, and Suh-Yin Lee. Physics-based ball tracking and 3d trajectory reconstruction with applications to shooting location estimation in basketball video. *Journal of Visual Communication and Image Representation*, 20(3):204–216, 2009. 2

[6] Hua-Tsung Chen, Wen-Jiin Tsai, Suh-Yin Lee, and Jen-Yu Yu. Ball tracking and 3d trajectory approximation with applications to tactics analysis from single-camera volleyball sequences. *Multimed Tools and Applications*, 60:641–667, 2012. 2

[7] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018. 3, 4

[8] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. 4, 3

[9] Moritz Einfalt, Dan Zecha, and Rainer Lienhart. Activity-conditioned continuous human pose estimation for performance analysis of athletes using the example of swimming. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 446–455, 2018. 2

[10] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. http://github.com/google/brax, 2021. 2

[11] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. 3

[12] William R. Hamilton. On a general method of expressing the paths of light, and of the planets, by the coefficients of a characteristic function. *Dublin University Review and Quarterly Magazine*, 1(1088):795–826, 1833. 4

[13] Hawk-Eye Innovations. Hawk-eye. https://www.hawkeyeinnovations.com. Accessed: 2023-05-16. 1

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 3, 6

[15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014. 3

[16] Kaustubh Kulkarni and Sucheth Shenoy. Table tennis stroke recognition using two-dimensional human pose estimation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4571–4579, 2021. 2

[17] Yanjie Li, Shoukui Zhang, Zhicheng Wang, Sen Yang, Wankou Yang, Shutao Xia, and Erjin Zhou. Tokenpose: Learning keypoint tokens for human pose estimation. *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11293–11302, 2021. 5

[18] Paul Liu and Jui-Hsien Wang. Monotrack: Shuttle trajectory reconstruction from monocular badminton video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 3513–3522, 2022. 2

[19] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[20] Katja Ludwig, Daniel Kienzle, and Rainer Lienhart. Recognition of freely selected keypoints on human limbs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022. 2

[21] Andrii Maksai, Xinchao Wang, and Pascal Fua. What players do with the ball: A physically constrained interaction modeling. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 972–981, 2016. 1

[22] Igor Olaizola, Mikel Labayen, Julián Esnal, and Naiara Aginako. Accurate ball trajectory tracking and 3d visualization for computer-assisted sports broadcast. *Multimedia Tools and Applications*, 73:1–24, 2013. 1

[23] Pascaline Parisot and Christophe Vleeschouwer. Consensus-based trajectory estimation for ball detection in calibrated cameras systems. *Journal of Real-Time Image Processing*, 16, 2019. 1

[24] 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. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019. 3

[25] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, Jinkyoo Park, and Stefano Ermon. Torchdyn: Implicit models and neural numerical methods in pytorch. 3

[26] Jinchang Ren, James Orwell, Graeme A. Jones, and Ming Xu. Real-time modeling of 3-d soccer ball trajectories from multiple fixed cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(3):350–362, 2008. 1

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy,

Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. 6

[28] Stereolabs. Zed 2i. https://www.stereolabs.com/zed-2i/. Accessed: 2023-05-16. 6

[29] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. 5

[30] Gabriel Van Zandycke and Christophe De Vleeschouwer. 3d ball localization from a single calibrated image. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3471–3479, 2022. 2, 8

[31] Gabriel Van Zandycke, Vladimir Somers, Maxime Istasse, Carlo Del Don, and Davide Zambrano. Deepsportradar-v1: Computer vision dataset for sports understanding with high quality annotations. In *Proceedings of the 5th International ACM Workshop on Multimedia Content Analysis in Sports*, page 1–8, 2022. 2

[32] Vieww GmbH. View 4d 2.0. https://vieww.com. Accessed: 2023-05-16. 1

[33] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 3

[34] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3

Towards Learning Monocular 3D Object Localization From 2D Labels Using the Physical Laws of Motion

Supplementary Material

Abstract

In this supplementary material, we provide additional details on the methodology, offer further descriptions and visualizations of the datasets, and present additional results. The code and dataset is published at https://kiedani.github.io/3DV2024/.

7. Methods

7.1. Derivation of the Differential Equations of Motion

In order to define the differential equations of motion 5, we first need to model the full potential 8. While the gravitational potential is already defined in equation 6 and the bouncing off the floor is modeled through equation 7, we still need to define V_W and V_O. The potential V_W models the interaction of the ball with infinite walls of the room and, thus, it is very similar to V_f. Since there are no walls in the synthetic dataset, we set $V_\mathrm{W} = 0$. In the real dataset are two walls, one in the y-z plane at $x = 0\,\mathrm{m}$ and one in the x-z plane at $y = 0.6\,\mathrm{m}$. Thus, the ball is constrained by the walls and the floor to the area $x \geq 0\,\mathrm{m}$, $y \leq 0.6\,\mathrm{m}$ and $z \geq 0\,\mathrm{m}$. Therefore, we model the potential V_W as

$$V_\mathrm{W}(\vec{\mathbf{r}}^{(\mathrm{W})}) = mc \cdot \mathrm{ReLU}(-x^{(\mathrm{W})}) + mc \cdot \mathrm{ReLU}(y^{(\mathrm{W})} - 0.6\,\mathrm{m}) \tag{13}$$

with $c = 1000$ being a large constant.

The collisions with finite obstacles are described by the potential V_O, and there are multiple obstacles in the three different environments of the synthetic dataset and one obstacle in the real dataset. To model the potential, we consider a cubic object lying on the floor at the origin of the so-called *object coordinate system* with the length $2l$, width $2w$ and height h such that the *object coordinates* $\vec{\mathbf{r}}^{(\mathrm{o})} = \begin{pmatrix} x^{(\mathrm{o})} & y^{(\mathrm{o})} & z^{(\mathrm{o})} \end{pmatrix}^\mathrm{T}$ with $x^{(\mathrm{o})} \in [-l, l] \wedge y^{(\mathrm{o})} \in [-w, w] \wedge z^{(\mathrm{o})} \in [0, h]$ describe points inside the cube. To accurately model collisions with this object, the potential has to be large around the boundaries of the cube and zero everywhere outside the cube. In this situation, the potential V_o describing collisions with this single obstacle at the origin can be described as

$$\begin{aligned} V_o(\vec{\mathbf{r}}^{(\mathrm{o})}) = m\frac{c}{\beta_\mathrm{o}} &\cdot \left(\sigma\left(\beta_\mathrm{o}(x^{(\mathrm{o})} + l)\right) - \sigma\left(\beta_\mathrm{o}(x^{(\mathrm{o})} - l)\right)\right) \\ &\cdot \left(\sigma\left(\beta_\mathrm{o}(y^{(\mathrm{o})} + w)\right) - \sigma\left(\beta_\mathrm{o} \cdot (y^{(\mathrm{o})} - w)\right)\right) \\ &\cdot \left(\sigma\left(\beta_\mathrm{o} \cdot (z^{(\mathrm{o})})\right) - \sigma\left(\beta_\mathrm{o} \cdot (z^{(\mathrm{o})} - h)\right)\right) \end{aligned} \tag{14}$$

using the sigmoid function $\sigma(\cdot)$ and the large constant $\beta_\mathrm{o} = 66$ to sharpen the softmax function.

In order to describe multiple obstacles at different known positions, we have to transform the ball's world coordinates $\vec{\mathbf{r}}^{(\mathrm{W})}$ into the object coordinates $\vec{\mathbf{r}}^{(\mathrm{o})}$ for each obstacle separately such that the obstacle is centered at the origin of the object coordinate system. This transformation can be realized using rotation and translation and is described by the transformation $\vec{\mathbf{r}}^{(o)} = \mathcal{T}_o(\vec{\mathbf{r}}^{(\mathrm{W})})$ for each obstacle o. As a result, the potential V_O describing collisions with all obstacles in the specific environment is described as

$$V_\mathrm{O}(\vec{\mathbf{r}}^{(\mathrm{W})}) = \sum_{o \in \mathcal{O}} V_o\left(\mathcal{T}_o(\vec{\mathbf{r}}^{(\mathrm{W})})\right) \tag{15}$$

with $\mathcal{O}$ being the set of all obstacles placed in the environment.

With the potentials V_G, V_F, V_W and V_O defined, the differential equations of motion 5 can now be derived. This can be achieved using automatic differentiation, but we decided to calculate the derivatives of the Hamilton function analytically for the sake of a small speed up. We approximate the derivative of the ReLU function with a sigmoid function as $\frac{\partial}{\partial x}\mathrm{ReLU}(x) \approx \sigma(\beta_{\mathrm{g/w/f}} \cdot x)$ with $\beta_{\mathrm{g/w/f}} = 200$ being a large constant. There are no walls in the *synthetic dataset* and, thus, we obtain the differential equations of motion as

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{r}}^{(\mathrm{W})} &= \vec{\mathbf{v}}\,, \\ \frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{v}} &= -\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \sigma\left(\beta_{\mathrm{g/w/f}}\vec{\mathbf{r}}^{(\mathrm{W})}\right) + \begin{pmatrix} 0 \\ 0 \\ c \end{pmatrix} \sigma\left(-\beta_{\mathrm{g/w/f}}\vec{\mathbf{r}}^{(\mathrm{W})}\right) \\ &\quad + \sum_{o \in \mathcal{O}} \beta_o V_o(\vec{\mathbf{r}}^{(o)}) \mathcal{T}_o \\ &\quad \cdot \left(1 - \sigma\left(\beta_o\left(\vec{\mathbf{r}}^{(o)} + \begin{pmatrix} l \\ w \\ 0 \end{pmatrix}\right)\right) - \sigma\left(\beta_o\left(\vec{\mathbf{r}}^{(o)} - \begin{pmatrix} l \\ w \\ h \end{pmatrix}\right)\right)\right). \end{aligned} \tag{16}$$

For the *real dataset* we additionally consider the two walls and obtain the equations of motion as

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{r}}^{(\mathrm{W})} &= \vec{\mathbf{v}}\,, \\
\frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{v}} &= -\begin{pmatrix}0\\0\\g\end{pmatrix}\sigma\left(\beta_{\mathrm{g/w/f}}\vec{\mathbf{r}}^{(\mathrm{W})}\right) + \begin{pmatrix}0\\0\\c\end{pmatrix}\sigma\left(-\beta_{\mathrm{g/w/f}}\vec{\mathbf{r}}^{(\mathrm{W})}\right) \\
&\quad + \begin{pmatrix}c\\0\\0\end{pmatrix}\sigma\left(-\beta_{\mathrm{g/w/f}}\vec{\mathbf{r}}^{(\mathrm{W})}\right) + \begin{pmatrix}0\\c\\0\end{pmatrix}\sigma\left(\beta_{\mathrm{g/w/f}}\left(\vec{\mathbf{r}}^{(\mathrm{W})} - 0.6\,\mathrm{m}\right)\right) \\
&\quad + \sum_{o\in\mathcal{O}} \beta_o V_o(\vec{\mathbf{r}}^{(o)})\mathcal{T}_o \\
&\quad \cdot \left(1-\sigma\left(\beta_o\left(\vec{\mathbf{r}}^{(o)} + \begin{pmatrix}l\\w\\0\end{pmatrix}\right)\right) - \sigma\left(\beta_o\left(\vec{\mathbf{r}}^{(o)} - \begin{pmatrix}l\\w\\h\end{pmatrix}\right)\right)\right).
\end{aligned} \tag{17}$$

In this paper we model the equations of motion directly. However, it is also possible to use a differential physics engine like BRAX [10] in the PAF. This would also allow to describe some physical phenomenons like spin even more easily.

7.2. Obtaining the Camera Matrices

We need the intrinsic as well as extrinsic camera matrix for our method. The intrinsic camera matrix is defined as

$$\mathbf{M}_{int} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{18}$$

with $f_{x/y}$ being the focal length and $c_{x/y}$ being the principal point. With this matrix the *euclidean camera coordinates* can be converted into *homogeneous image coordinates*. The extrinsic camera matrix is defined as

$$\mathbf{M}_{ext} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \tag{19}$$

with $\mathbf{R} \in \mathbb{R}^{3\times 3}$ describing the rotation and $\mathbf{t} \in \mathbb{R}^{3\times 1}$ describing the translation of the world coordinate system. With this matrix the *homogeneous world coordinates* are converted into *homogeneous camera coordinates*.

While the camera matrices are commonly available in modern broadcasting cameras [31], they can also be calculated, for example, by using a Direct Linear Transform [1]. This is done particularly straightforward in the context of sports, because characteristic points on the court can be easily annotated automatically.

In the *synthetic dataset* the camera matrices are provided by the physics engine MuJoCo, thus, the exact matrices are already available. In the *real dataset* the camera is calibrated such that intrinsic camera matrix is already known. Therefore, we only have to calculate the extrinsic camera matrix. We label 11 characteristic points of the scene for each camera location and calculate an initial guess of the extrinsic camera matrix using the Direct Linear Transform. We then further optimize the extrinsic camera matrix using the BFGS optimization algorithm [3]. This way we obtain an accurate estimation of the extrinsic camera matrix. As this estimation is accurate but not exact, we conclude that our model is able to cope with approximate camera locations which is an important property for practical applications.

8. Dataset

In this section, we present additional details on the datasets introduced in the main paper. In Figure 9, we visualize some exemplary videos from different camera positions and in different environments to provide a better overview of the datasets. In the *synthetic dataset*, we record the videos from 9 possible camera locations that are shown in Table 3. All camera locations are chosen such that the origin of the world coordinate system corresponds to the center of the image. We set the gravitational constant to $g = 1\,\frac{\mathrm{m}}{\mathrm{s}^2}$. This reduces the speed of the ball, making it easier to automatically generate the synthetic dataset, because we have to ensure that the ball does not leave the field of view.

Since we record all videos in the *real dataset* with a ZED 2i Stereo Camera, we are able to calculate a ground truth depthmap for each image. For this purpose, we select the *Neural Depth Mode* of the camera to obtain the best possible results. Nevertheless, the results are still noisy and, thus, we extract the ground truth depth at the annotated ball's position as average over the neighboring values of the depth map D as

$$z^{(\mathrm{C})} = \frac{1}{9}\cdot\sum_{i=-1}^{1}\sum_{j=-1}^{1} D_{x^{(\mathrm{I})}+i,\,y^{(\mathrm{I})}+j} \tag{20}$$

with $\begin{pmatrix} x^{(\mathrm{I})} & y^{(\mathrm{I})}\end{pmatrix}^{\mathrm{T}}$ being the ground truth ball's position in image coordinates. Based on this ground truth depth and the intrinsic camera matrix we calculate the ground truth camera coordinates. We define a fixed origin of the world coordinate system such that the physical potentials can be described easily, and we visualize the origin in Figure 10. The 2D image coordinates are annotated manually by simply clicking on the ball's position in the image. To expedite this process, we use a simple automatic ball detection algorithm based on the Hough Transformation and only correct the ball position manually if the automatic detection fails.

9. Experiments

In this section we give further details on the training process, provide additional results from the main paper and discuss new experiments.

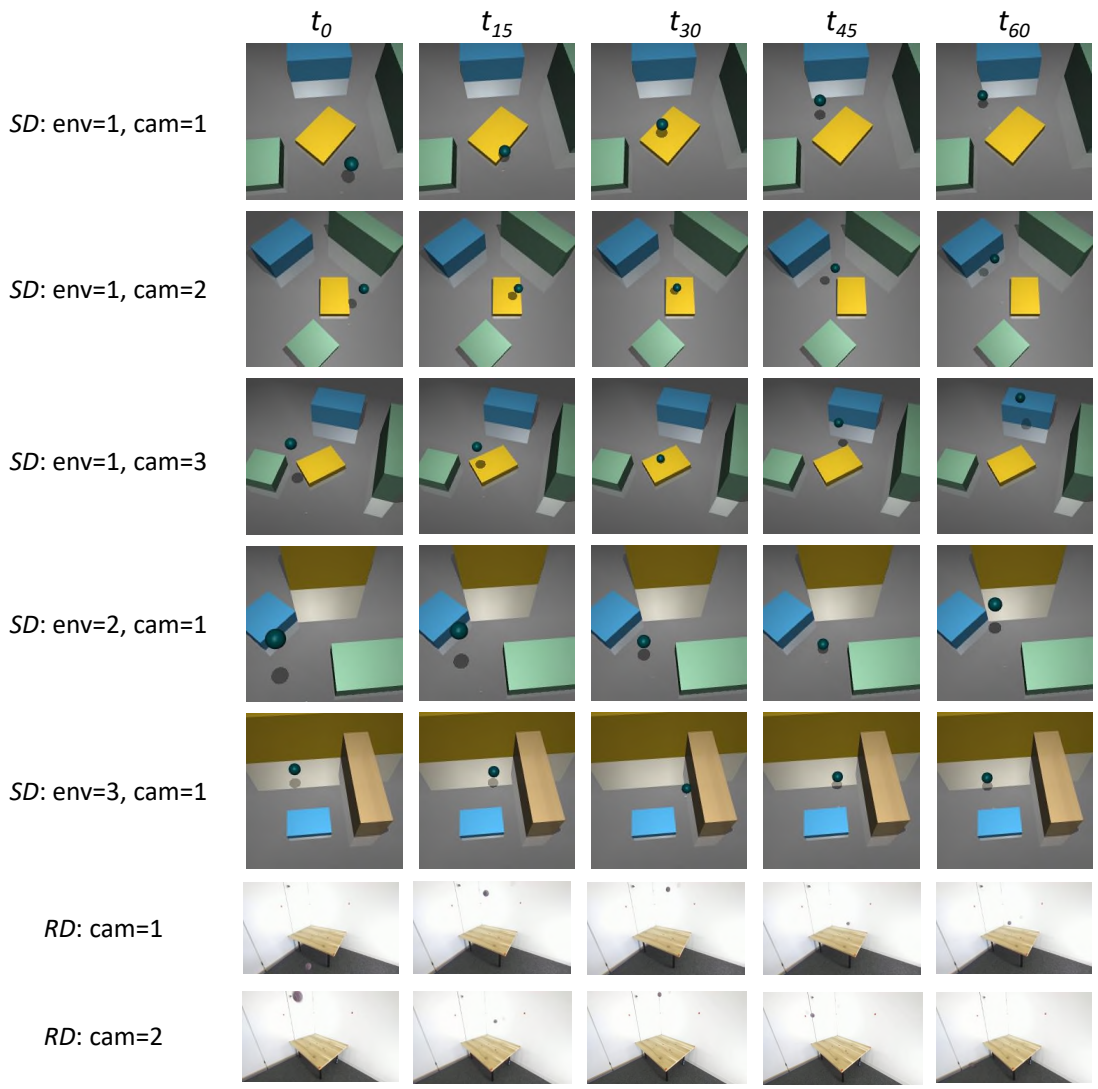


Figure 9. Frames from example sequences of multiple camera locations and environments are shown.

Discussion of training process

Our code is implemented in PyTorch [24], and we utilize models and architectures from the timm library [33]. We optimize our model with the ADAM [15] optimizer and use a batch size of 8 together with a learning rate of $2 \cdot 10^{-5}$. The Dormand-Prince method [8] is applied for the numerical solution of the differential equations 5 and its implementation utilizes the torchdyn [25] library.

We train the *RD* and *SD-S* models for 400 epochs, the *SD-M* model for 200 epochs and the *SD-L* model for 100 epochs. Although the models show slight improvements with longer training, the performance gains are small. We evaluate the model after each epoch on the validation set and select the model with the best performance for the final evaluation on the test set. We multiply the *future loss* (first term in equation 9) with a factor, which linearly increases from 0 to 1 during the first 300 epochs of the training. This way we ensure that the model focuses on learning to recognize the ball in the images first, and thus, prevents a model collapse.

Our experiments were conducted on a combination of Nvidia RTX 3090, V100, A100, and H100 GPUs. Therefore, we do not provide the actual runtime of our experiments as a comparison is not meaningful. However, to provide a general idea of the runtime, training our model for 400 epochs on the *real dataset* in Table 2 takes approximately 12 hours on a single V100 GPU. We note that a considerable good performance is already reached after only 100 epochs (3 hours on a single V100 GPU). Furthermore,

Table 3. Camera locations in the synthetic dataset. Each camera looks at the origin of the world-coordinate system and is positioned at a radial distance d, polar angle θ and azimuthal angle ϕ with respect to the origin.

camera	d (m)	θ (°)	ϕ (°)
1	8	−60	40
2	10	−65	0
3	10	−55	60
4	9	−60	20
5	7	−70	50
6	9	−50	10
7	10	−60	30
8	8	−70	0
9	8	−50	60

Figure 10. Origin of the *real dataset*. Blue represents the z-axis, green the y-axis and red the x-axis.

Table 4. *DtG* scores per camera location evaluated on the 1st environment of the *synthetic dataset*. An extension of Table 1a from the main paper.

	DtG ± Δ*DtG* (cm) ↓								
training set	camera 1	camera 2	camera 3	camera 4	camera 5	camera 6	camera 7	camera 8	camera 9
SD-S	22 ± 19	-	-	-	-	-	-	-	-
SD-M	19 ± 10	19 ± 18	28 ± 24	13 ± 9	26 ± 12	19 ± 14	27 ± 23	23 ± 9	21 ± 10
SD-L	11 ± 6	20 ± 19	28 ± 25	11 ± 8	10 ± 6	19 ± 13	28 ± 25	15 ± 8	16 ± 7

we anticipate that further code optimizations, such as improved data loading techniques or model compilation, could lead to significant increases in training speed. For all results in this paper we report the mean and standard deviation (represented by the symbol Δ) of the metric calculated across all images in the test set.

While we provide only results for camera locations 1, 7, 8, and 9 in Table 1a, the results for all camera locations are given in Table 4. We see that the *SD-M* and *SD-L* models are able to estimate the ball's 3D position for all camera locations very precisely. Since the *SD-L* model achieves better scores for most camera locations, this reinforces our assumption that training with additional data recorded in multiple environments is actually beneficial.

Comparison of different backbones

We perform an additional experiment to compare the performance of different backbones in the PEN. For each backbone, we use the implementation provided in the timm library. In the other experiments, we always use the first 4 out of 5 stages of a ResNet34. We now compare this backbone with 4 out of 4 stages of the ConvNeXt-nano architecture and 4 out of 4 stages of the ConvNeXtv2-nano architecture. All models are pretrained on ImageNet-1K, and we train and test them on the *SD-M* dataset. The results are shown in table 5.

Interestingly, the ConvNeXtv2 backbone performs notably worse than the other two architectures, despite its seemingly advanced design. In summary, the ResNet performs slightly better than the ConvNeXt, despite the ConvNeXt having twice as many parameters. One reason for this behavior might be that our input image resolution is small, and we do not benefit from a very large receptive field. Since the ConvNeXt architecture uses 7×7 convolutions, the receptive field grows faster than in the ResNet. Once the receptive field size becomes as large as the image resolution, the additional stages may not provide significant benefits and could potentially hinder the model's ability to learn an effective representation. Since the ResNet only uses 3×3 convolutions and the features are extracted at an earlier stage compared to the other architectures, the receptive field is smaller and the model might be able to learn a better representation. However, additional experiments using different backbones and higher input image resolutions are required to validate this hypothesis. We note that our code can be readily extended with other backbones, provided that they support the extraction of 4 feature stages, which is a requirement in our implementation. If a different number of features is used, the implementation of the *depthmap head* and *heatmap head* needs to be adjusted. In conclusion, even though the ResNet is a relatively old architecture, it remains a highly suitable choice for our task.

Table 5. Results for different backbones. Each model is trained on the *SD-M* dataset. *DtG* scores per camera location evaluated on the 1st environment of the *synthetic dataset*.

		DtG $\pm$ Δ*DtG* (cm) $\downarrow$								
backbone	# params	camera 1	camera 2	camera 3	camera 4	camera 5	camera 6	camera 7	camera 8	camera 9
resnet	$0.8 \cdot 10^7$	19 ± 10	19 ± 18	28 ± 24	13 ± 9	26 ± 12	19 ± 14	27 ± 23	23 ± 9	21 ± 10
convnext	$1.5 \cdot 10^7$	14 ± 12	26 ± 21	32 ± 27	15 ± 13	15 ± 11	24 ± 21	37 ± 31	22 ± 11	20 ± 12
convnextv2	$1.5 \cdot 10^7$	23 ± 14	33 ± 28	45 ± 32	25 ± 20	18 ± 11	31 ± 27	49 ± 36	53 ± 22	35 ± 20

Analysis of future loss

In this section, we delve deeper into the concept of the *future loss*. As can be seen in figure 2, we compare the image coordinates $\vec{\mathbf{r}}^{(\mathrm{I})}_{\mathrm{PEN}}(t_{n+\Delta n})$ predicted by the PEN with the coordinates $\vec{\mathbf{r}}^{(\mathrm{I})}_{\mathrm{PAF}}(t_{n+\Delta n})$ forecasted by the PAF. This way of computing the *future loss* is explicitly written in the first term of equation 9, and we refer to this method as *2D-predict* in this section. Because the ground truth image coordinates are available for every frame, we can also compute the *future loss* by comparing the PEN's predictions with the ground truth coordinates $\vec{\mathbf{r}}^{(\mathrm{I})}_{\mathrm{GT}}(t_{n+\Delta n})$. Consequently, we simply replace the coordinates $\vec{\mathbf{r}}^{(\mathrm{I})}_{\mathrm{PAF}}(t_{n+\Delta n})$ with $\vec{\mathbf{r}}^{(\mathrm{I})}_{\mathrm{GT}}(t_{n+\Delta n})$ in equation 9. This way of computing the loss is denoted as *2D-gt*. Moreover, we also add a third method of computing the *future loss* by comparing the PEN's predictions with the ground truth camera coordinates $\vec{\mathbf{r}}^{(\mathrm{C})}_{\mathrm{GT}}(t_{n+\Delta n})$, which are usually not used for the training. Consequently, the *future loss* is computed in this case as

$$\textit{future loss} = \left\| \vec{\mathbf{r}}^{(\mathrm{C})}_{\mathrm{PEN}}(t_{n+\Delta n}) - \vec{\mathbf{r}}^{(\mathrm{C})}_{\mathrm{GT}}(t_{n+\Delta n}) \right\|_{\mathrm{L1}} . \quad (21)$$

We refer to this way of computing the loss as *3D-gt* in this section, and we use this method as baseline. We note that for this way of computing the loss, we utilize the 3D ground truth, which is usually not available in real world scenarios. Nonetheless, we believe that examining this baseline can provide valuable insights into our analysis.

We train three models with the different future losses and report the results in table 6. By comparing the *2D-predict* model with the *2D-gt* model, we can see that the *2D-predict* model performs better than the *2D-gt* model. This is due to the ground truth image coordinates being noisy, since they are annotated manually. Obviously, the PEN learns to predict more accurate image coordinates and, consequently, the calculation of the *future loss* is more accurate.
As expected, the *3D-gt* model performs best, since it is trained with 3D ground truth information. However, despite the *2D-predict* model being worse, it is still able to predict the 3D position of the ball surprisingly well, since the metric is in the same order of magnitude as with the *3D-gt* model. Consequently, we conclude that the method presented in this paper provides a viable alternative to fully supervised training. Additionally, we highlight the potential for combining our method with supervised training, particularly in scenarios where 3D ground truth data is only accessible for a subset of the data. Such integration could lead to further enhancements in existing applications.

Table 6. *DtG* scores and DtG_3 scores per camera location for the *real dataset*. Models trained with different *future losses* are compared.

	DtG $\pm$ Δ*DtG* (cm) $\downarrow$		*DtG*$_3$ $\pm$ Δ*DtG*$_3$ (cm) $\downarrow$	
future loss	camera 1	camera 2	camera 1	camera 2
2D-predict	7 ± 4	6 ± 4	$\begin{pmatrix} 8 \pm 4 \\ 6 \pm 2 \\ 11 \pm 4 \end{pmatrix}$	$\begin{pmatrix} 4 \pm 2 \\ 6 \pm 3 \\ 11 \pm 4 \end{pmatrix}$
2D-gt	18 ± 10	17 ± 10	$\begin{pmatrix} 6 \pm 5 \\ 14 \pm 7 \\ 33 \pm 7 \end{pmatrix}$	$\begin{pmatrix} 5 \pm 3 \\ 15 \pm 6 \\ 32 \pm 7 \end{pmatrix}$
3D-gt	3 ± 3	3 ± 4	$\begin{pmatrix} 3 \pm 2 \\ 2 \pm 2 \\ 6 \pm 5 \end{pmatrix}$	$\begin{pmatrix} 3 \pm 2 \\ 2 \pm 3 \\ 6 \pm 5 \end{pmatrix}$

Learning arbitrary physics

An important advantage of our method is its ability to describe arbitrary physical systems by utilizing numerical solutions for the equations of motion, rather than relying on analytical solutions. In this section we show two things:

- We demonstrate that our method can also be used for physical systems that behave differently from the previous described bouncing balls.
- We calculate an analytic solution of the equations of motion and compare the numerical forecasting in the PAF with the analytic forecasting.

Therefore, we introduce a new synthetic dataset denoted as *spring dataset*. It consists of video of a moving ball connected with a spring to a fixed ball in the center. The moving ball of mass $m = 1\,\mathrm{kg}$ is attracted to the fixed center ball by the spring with spring constant $k = 3\,\frac{\mathrm{N}}{\mathrm{m}}$. We record 100 videos from a single fixed camera location for training, 100 videos for validation and 100 videos for testing. The resolution of the frames is 224×224 and the videos are recorded with 30 FPS. We visualize frames from an exam-

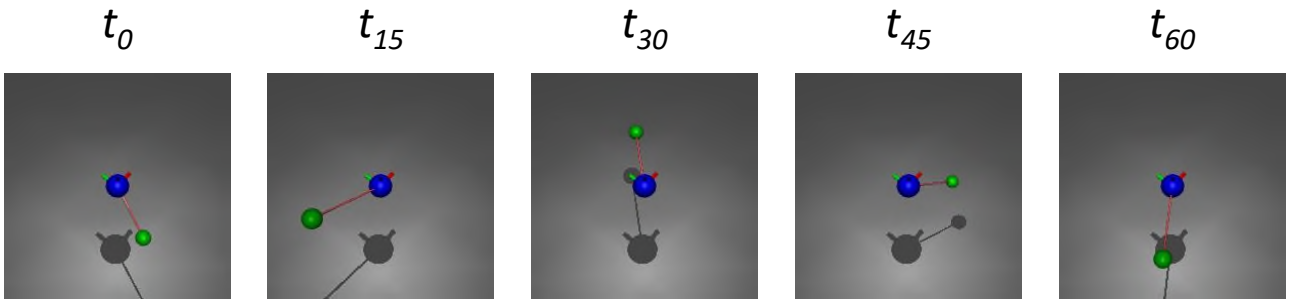


Figure 11. Frames from an example video of the *spring dataset*.

ple video depicting the motion of the ball connected to the spring in Figure 11.

According to Hooke's law, the potential of the moving ball is given by $V(\vec{\mathbf{r}}^{(\mathrm{W})}) = \frac{1}{2}k\left|\vec{\mathbf{r}}^{(\mathrm{W})}\right|^2$ and by applying the Hamilton formalism (see equation 5), we obtain the equations of motion as

$$\frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{r}}^{(\mathrm{W})} = \vec{\mathbf{v}} \quad , \quad \frac{\mathrm{d}}{\mathrm{dt}}\vec{\mathbf{v}} = -\frac{k}{m}\cdot\vec{\mathbf{r}}^{(\mathrm{W})} \quad . \tag{22}$$

The analytic solution to these differential equations can be experessed as

$$\vec{\mathbf{r}}^{(\mathrm{W})}(t) = \frac{1}{k}\cdot\vec{\mathbf{v}}_0\cdot\sin\left(k\cdot t\right) + \vec{\mathbf{r}}_0^{(\mathrm{W})} \tag{23}$$

where $\vec{\mathbf{r}}_0^{(\mathrm{W})}$ and $\vec{\mathbf{v}}_0$ are the initial position and velocity of the moving ball.

We implement the numeric solver of equation 22 similar to the previous experiments, and compare its performance with the utilization of the analytic function 23 in the PAF. The results are given in Table 7. The model trained with the numeric PAF and the model trained with the analytical PAF perform very similarly, with the numeric version achieving slightly better scores. This shows that using numeric solvers in the PAF is a valid approach that is able to describe arbitrary physical systems.

Table 7. *DtG* and DtG_3 scores on the *spring dataset*. The analytic solution of the equations of motion is compared to a numeric solution.

PAF mode	$DtG \pm \Delta DtG$ (cm) ↓	$DtG_3 \pm \Delta DtG_3$ (cm) ↓
numeric	9 ± 15	$\begin{pmatrix} 6 \pm 6 \\ 5 \pm 6 \\ 16 \pm 22 \end{pmatrix}$
analytic	10 ± 16	$\begin{pmatrix} 8 \pm 8 \\ 4 \pm 5 \\ 21 \pm 21 \end{pmatrix}$

In general, the numeric model achieves a *DtG* of just 9 cm, demonstrating the applicability of our method to physical systems beyond bouncing balls.

While the physics of all datasets presented in this paper is determined by the laws of classical mechanics, it is worth considering the potential extension of our method to other domains such as quantum mechanics or electrodynamics, where analytic solutions are not available in most cases. However, exploring these possibilities lies beyond the scope of this work.

Discussion of stability

To ensure that our results are not just random fluctuations, we test our model with different random seeds and compare the results. For this experiment we choose the *SD-S* scenario from the main paper and set 8 different random seeds at the beginning of the training. This results in a different initialization of the *heatmap head* and *depthmap head*, as well as in a different ordering of the training data in each epoch. We present the results of the models in Table 8.

The results obtained for all seed values show a high degree of similarity. The mean value across multiple runs is $E_{DtG} = 24.4\,\mathrm{cm}$, with a small standard deviation of $\sigma_{DtG} = 1.6\,\mathrm{cm}$. The low standard deviation among individual runs suggests that the outcomes are not mere random fluctuations. This observation is further supported by

Table 8. *DtG* scores for the 1st camera location in the 1st environment. The results of the *SD-S* model are compared for different initial seeds.

seed	$DtG \pm \Delta DtG$ (cm) ↓
1	24 ± 20
2	27 ± 21
3	26 ± 21
4	23 ± 19
5	23 ± 21
6	25 ± 21
7	25 ± 19
8	22 ± 21

the fact that the standard deviation over the multiple runs is an order of magnitude smaller than the standard deviations among the individual images ΔDtG. Hence, the effect of the random seed is negligible.

While the results are relatively stable over different seeds, we observe that other factors have a much larger impact on the results. One such factor is the selection of the number of forecast steps. As described in equation 9, we compare the coordinates at time $t_{n+\Delta n}$ with the forecasted coordinates at this time. Depending on the speed of the ball, a good value for Δn has to be chosen carefully. If Δn is too small, the ball may not have moved a significant distance, resulting in a very low loss. Consequently, the PEN may converge to a trivial solution. Conversely, if Δn is too large, the distance might be too large, and the PEN may struggle to learn the correct solution. Therefore, for each scenario an appropriate value for Δn has to be chosen.
In this paper, we do not use a single value for Δn, instead we calculate the *future loss* for multiple values of Δn and then compute the average. This way, the small Δn values help to stabilize the training, while larger ones ensure that the PEN does not collapse to a trivial solution. Although this approach aids in training, it is still necessary to choose appropriate values for Δn.

One way to significantly improve the stability of the training is to use a better pretraining of the PEN. For example a segmentation or even a depth estimation task could be chosen for pretraining the model. Since this teaches the model 3D knowledge, a model collapse is less likely to happen. However, in this paper we want to show that our method is able to learn 3D dynamics without any 3D supervision. Thus, we do not further explore more advanced pretraining strategies, and instead only initialize the backbone of the PEN with ImageNet weights in our experiments.

Limitations and improvements

We view this paper as a step towards 3D object location estimation without requiring 3D supervision. Our findings demonstrate the feasibility of training a model without depth information and provide a comprehensive analysis of our approach. However, there are still several limitations to address. Specifically, we only use low resolution images to ensure an efficient training of our models. It is possible to use such low resolution images in our experiments, because the ball's relative size in the images is large enough to be clearly identified. In contrast, sports videos typically involve high-resolution frames where the ball appears much smaller. Consequently, simply downscaling the entire image would not suffice to maintain clear ball identification. Hence, the development of more sophisticated methods is necessary to optimize computation time. One approach is to leverage an object detection pipeline to extract the relevant region around the ball, which can then be utilized for 3D location estimation. We anticipate that the insights presented in this paper will inspire the advancement of more sophisticated techniques in this field.