Black-Box vs. Gray-Box: A Case Study on Learning Table Tennis Ball Trajectory Prediction with Spin and Impacts

Jan Achterhold [1]   JAN.ACHTERHOLD@TUE.MPG.DE
Philip Tobuschat [2]   PHILIP.TOBUSCHAT@TUE.MPG.DE
Hao Ma [2]   HAO.MA@TUE.MPG.DE
Dieter Buechler [3]   DIETER.BUECHLER@TUE.MPG.DE
Michael Muehlebach [2]   MICHAEL.MUEHLEBACH@TUE.MPG.DE
Joerg Stueckler [1]   JOERG.STUECKLER@TUE.MPG.DE

[1] *Embodied Vision Group*

[2] *Learning and Dynamical Systems Group*

[3] *Empirical Inference Department*

Max Planck Institute for Intelligent Systems, Tübingen, Germany

Editors: N. Matni, M. Morari, G. J. Pappas

Abstract

In this paper, we present a method for table tennis ball trajectory filtering and prediction. Our gray-box approach builds on a physical model. At the same time, we use data to learn parameters of the dynamics model, of an extended Kalman filter, and of a neural model that infers the ball's initial condition. We demonstrate superior prediction performance of our approach over two black-box approaches, which are not supplied with physical prior knowledge. We demonstrate that initializing the spin from parameters of the ball launcher using a neural network drastically improves long-time prediction performance over estimating the spin purely from measured ball positions. An accurate prediction of the ball trajectory is crucial for successful returns. We therefore evaluate the return performance with a pneumatic artificial muscular robot and achieve a return rate of 29/30 (97.7 %).

Keywords: robotic table-tennis, trajectory prediction, gray-box model learning

1. Introduction

Playing table tennis with a robot is a long-standing challenge in robotics research ([Andersson, 1989](#)). An important difficulty resides in tracking and predicting the ball's trajectory, which is strongly influenced by nonlinear effects, arising from drag, spin, and impacts with the table.

In this paper, we focus on designing an accurate model for estimating the ball's state and predicting its future trajectory. Predicting the future trajectory is crucial for computing a hitting point for the robot to return the ball. We build our model on existing knowledge about the physical dynamics of a flying ball, including drag and spin effects. Our method is based on the extended Kalman filter (EKF) and includes various parameters which are trained from offline data.

This gray-box approach demonstrates superior prediction performance compared to two deep-learning based (black-box) baselines in our experiments. In addition, we demonstrate that estimating the ball's initial spin from parameters of the ball launcher with a neural network drastically improves prediction performance over an uninformed initialization of the initial spin. We also highlight the performance of our model by successfully returning balls with a pneumatic artificial muscular robot.

2. Related Work

Models for time series can be categorized into white-box, gray-box, and black-box models. Black-box models follow a purely statistical, data-driven approach without incorporation of prior physical knowledge on the system to model. In contrast, white-box models are purely based on a-priori system knowledge, without incorporation of data. In gray-box models, both physical a-priori knowledge and data are used for model design and the identification of its parameters. For many dynamical systems, it is difficult or impossible to achieve accurate white-box modeling due to unmodelled effects or variable parameters. Incorporation of data enables black- and gray-box models to adapt to the specifics of the system at hand, which is why we will focus on these two model classes in the following. Inferring dynamics models and their parameters from data is classically referred to as *system identification*, see (Ljung, 1986) for an overview.

Black-box models Black-box time series models often leverage a latent space formulation. They comprise an encoder-decoder pair mapping to and from the latent space, and a forward model in the latent space. Recurrent cells such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) or gated recurrent units (GRU) (Cho et al., 2014) are commonly used as latent forward models. Such purely deterministic models can be extended by stochastic nodes to handle noisy observations and transitions, as in variational recurrent neural networks (VRNN) (Chung et al., 2015) or stochastic RNNs (SRNN) (Fraccaro et al., 2016). Several latent sequence models have been proposed which allow for state estimation through filtering, such as the Kalman-VAE (Fraccaro et al., 2017), Recurrent Kalman Networks (Becker et al., 2019), and Deep Variational Bayes Filters (Karl et al., 2017). Hafner et al. (2019) present a recurrent (variational) state-space model (RSSM) which they use for model-based reinforcement learning. We use RSSM as a black-box baseline in our paper. Girin et al. (2021) present a comprehensive overview of latent sequence models.

Gray-box models In our work, we follow a gray-box approach, yielding lower prediction error than a black-box baseline and allowing for physical interpretation of the estimated quantities. In contrast to the black-box models discussed above, gray-box models incorporate prior knowledge about the system to model, such as the laws of physics. One line of work in this direction are *differentiable physics engines* (de Avila Belbute-Peres et al., 2018). These engines enable gradient-based system identification from data for physical systems with a given structure. Our approach combines ideas from system identification with machine learning. It is therefore capable of exploiting prior knowledge from physics, while also learning parameters of the filter, dynamics, and a state initialization neural network from data.

2.1. Table tennis ball trajectory modeling

Approaches for table tennis ball trajectory modelling and prediction can also be categorized as white-box, black-box, and gray-box models. A common **white-box** approach is to use an aerodynamic model of the ball respecting gravity, drag, and Magnus forces (Andersson, 1989) and a physics-grounded rebound / impact model (Nakashima et al., 2010) in an extended / unscented Kalman filter (Zhang et al., 2010; Mülling et al., 2010; Wang et al., 2014; Zhang et al., 2015; Koç et al., 2018; Tebbe et al., 2018). Common **black-box** models approximate the ball trajectories with polynomial curves which are fitted to recorded data (Matsushima et al., 2005; Li et al., 2012; Tebbe et al., 2018; Lin et al., 2020). We compare our approach to the deep-learning based black-box approach by Gómez-González et al. (2020) which leverages a variational auto-encoder architecture for

table tennis ball trajectory prediction. The ball's spin constitutes a particular challenge for table tennis ball trajectory prediction, since it is hard to infer from position measurements of the trajectory. As a result, prior works have resorted to detecting the ball's spin by following the brand logo on the ball (Zhang et al., 2015) or by equipping the racket with an inertial sensor (Blank et al., 2017). In our **gray-box** approach, we use information from the ball's launch process to initialize the spin. More precisely, we learn the parameters of a neural network that relates the ball launcher settings to the initial spin of the ball. This is shown to drastically improve the quality and accuracy of the predicted ball trajectories.

3. Method

We present a gray-box method based on the extended Kalman filter (EKF), which includes parameters that are learned from data. In our notation, scalars are denoted by lowercase letters (σ), vectors are denoted by bold lowercase letters ($\boldsymbol{\sigma}$), and matrices are denoted by bold uppercase letters ($\boldsymbol{\Sigma}$). The operator $\text{diag}(\boldsymbol{x})$ forms a square matrix with $\boldsymbol{x}$ on its diagonal. The operator $[x]^+$ applies the softplus function and adds a constant: $[x]^+ = \log(1 + e^x) + 10^{-6}$ (elementwise for vectors).

3.1. Physical model

We assume the ball's dynamics in free-flight (not impacting with the table) to follow the ordinary differential equation (ODE)

$$\dot{\boldsymbol{v}}(t) = -k_\text{d}\left\|\boldsymbol{v}(t)\right\|_2 \boldsymbol{v}(t) + k_\text{m}(\boldsymbol{\omega}(t) \times \boldsymbol{v}(t)) + \mathbf{g} \tag{1}$$

with linear velocity $\boldsymbol{v}^\top(t) = (v_\text{x}(t), v_\text{y}(t), v_\text{z}(t))$ and its Euclidean norm $\left\|\boldsymbol{v}(t)\right\|_2$, angular velocity (spin) $\boldsymbol{\omega}^\top(t) = (\omega_\text{x}(t), \omega_\text{y}(t), \omega_\text{z}(t))$, drag coefficient k_d, Magnus effect coefficient k_m and gravitational acceleration $\mathbf{g}^\top = (0, 0, -9.802\,\text{m s}^{-2})$. We model the table impact by a linear map that relates the pre- and post-impact velocity $\boldsymbol{v}$ and spin $\boldsymbol{\omega}$ as

$$((\boldsymbol{v}^+)^\top, (\boldsymbol{\omega}^+)^\top)^\top = \mathbf{C}((\boldsymbol{v}^-)^\top, (\boldsymbol{\omega}^-)^\top)^\top, \quad \mathbf{C} \in \mathbb{R}^{6 \times 6} \tag{2}$$

where the superscripts $^+(^-)$ indicate the linear/angular velocities directly after (before) table impact.

3.2. Discrete-time state space model

We formulate a discrete-time state-space model for the ball trajectory for filtering and prediction.

Free flight   We introduce a state-space model for the ball dynamics with state

$$\boldsymbol{z}(t) = (\boldsymbol{p}(t)^\top, \boldsymbol{v}(t)^\top, \boldsymbol{\omega}(t)^\top, a_\text{d}(t), a_\text{m}(t))^\top \in \mathbb{R}^{11} \tag{3}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ is the position of the ball's center in Cartesian coordinates. The variables a_d, a_m parameterize the drag and Magnus coefficients $k_\text{d}(t) = a_\text{d}^2(t) + \epsilon, k_\text{m}(t) = a_\text{m}^2(t) + \epsilon$ to avoid explicit non-negativity constraints and stabilize training. We choose $\epsilon = 0.05$ in our experiments. As in Eq. (1), $\boldsymbol{v} \in \mathbb{R}^3$ and $\boldsymbol{\omega} \in \mathbb{R}^3$ relate to linear and angular velocity, respectively. To model free-flight phases, we time-discretize the ODE in Eq. (1) by Euler's method

$$\begin{aligned}
\boldsymbol{p}(t + \Delta_t) &= \boldsymbol{p}(t) + \Delta_t \boldsymbol{v}(t) \\
\boldsymbol{v}(t + \Delta_t) &= \boldsymbol{v}(t) + \Delta_t\left(-k_\text{d}(t)\|\boldsymbol{v}(t)\|\boldsymbol{v}(t) + k_\text{m}(t)(\boldsymbol{\omega}(t) \times \boldsymbol{v}(t)) + \mathbf{g}\right) \\
\boldsymbol{\omega}(t + \Delta_t) &= \boldsymbol{\omega}(t), \quad a_\text{d}(t + \Delta_t) = a_\text{d}(t), \quad a_\text{m}(t + \Delta_t) = a_\text{m}(t)
\end{aligned} \tag{4}$$

The function $z(t + \Delta_t) = g_{\text{free}}(z(t), \Delta_t)$ abbreviates Eq. (4). To refer to states at discrete time indices $n \in \{1, ..., N\}$ we use the notation $z_{n+1} = g_{\text{free}}(z_n, \Delta_T)$. In our setting, $\Delta_T = \frac{1}{180\,\text{s}^{-1}} \approx 5.56\,\text{ms}$ as the cameras of the video tracking system are triggered with a fixed frequency of $180\,\text{s}^{-1}$.

Impact model An impact of the ball with the table occurs within a discrete-time increment from n to $n + 1$ if the lower edge of the ball (with radius r) at $p_{\text{z},n+1} - r$ penetrates the table, i.e., $p_{\text{z},n+1} - r < \text{z}_{\text{table}}$. We approximate the time of impact $\Delta_{\text{imp}} \in [0, \Delta_T]$ with a simplified model to avoid numerical instabilities. It incorporates the velocity of the ball at the last discrete timestep before the impact $v_{\text{z},n}$, the gravitational acceleration g_{z}, and the height difference to the table $h = -((p_{\text{z},n} - r) - \text{z}_{\text{table}})$ such that $\Delta_{\text{imp}} = -(v_{\text{z},n} + \sqrt{v_{\text{z},n} \cdot v_{\text{z},n} + 2g_{\text{z}}h})/g_{\text{z}}$. The state of the ball just before the impact is given by $z^- = g_{\text{free}}(z_n, \Delta_{\text{imp}})$. At the time of impact, the velocity and spin are updated according to Eq. (2), yielding the state z^+ after impact. We denote this by $z^+ = C'z^-$ with $C' = \text{blockdiag}(I_3, C, 1, 1)$. After the impact a free-flight phase follows, such that at the next discrete timestep, the state is $z_{n+1} = g_{\text{free}}(z^+, \Delta_T - \Delta_{\text{imp}})$.

Joint model We denote our discrete-time forward step, incorporating free flight and impacts, as

$$z_{n+1} = g(z_n) = \begin{cases} g_{\text{free}}(z_n, \Delta_T) & [g_{\text{free}}(z_n, \Delta_T)]_{\text{z}} - r \geq \text{z}_{\text{table}} \\ g_{\text{free}}(C'g_{\text{free}}(z_n, \Delta_{\text{imp}}), \Delta_T - \Delta_{\text{imp}}) & \text{otherwise.} \end{cases} \quad (5)$$

where $[z]_{\text{z}}$ extracts the z-coordinate of the position in state z.

3.3. Extended Kalman Filter (EKF)

For filtering and prediction, we assume the ball dynamics as in Eq. (5) with additive Gaussian noise

$$\hat{z}_{n+1} = g(\hat{z}_n) + \zeta, \quad \zeta \sim \mathcal{N}(0, \text{diag}([\sigma_q]^+)), \quad \sigma_q \in \mathbb{R}^{11}. \quad (6)$$

We obtain measurements of the ball's center $m_n \in \mathbb{R}^3$ through a vision tracking system, which we assume to be perturbed by additive Gaussian measurement noise, i.e. $m_n = \hat{p}_n + \epsilon$, $\epsilon \sim \mathcal{N}(0, \text{diag}([\sigma_r]^+))$ with $\sigma_r \in \mathbb{R}^3$. We estimate a belief of the state $p(\hat{z}_n \mid m_{1:n})$, given past position measurements $m_{1:n}$, with an extended Kalman filter. Occasionally, the vision tracking system is unable to compute a position estimate (e.g. due to occlusions), which leads to missing measurements. To this end, we introduce the operator $\tau(n)$, which maps to the index of the nth *available* measurement.

State initialization We are interested in initializing the state belief at the time when the second measurement is available, i.e. $p(\hat{z}_{\tau(2)} \mid m_{\tau(1)}, m_{\tau(2)})$. The expected ball's position is estimated by the second measurement, $p = m_{\tau(2)}$, and the expected velocity v by a finite difference approximation $v = (m_{\tau(2)} - m_{\tau(1)})/(\Delta_T(\tau(2) - \tau(1)))$. The initial values for the position and velocity covariance are learned and denoted by $\Sigma_p = \text{diag}([\sigma_p]^+)$, $\Sigma_v = \text{diag}([\sigma_v]^+)$. Before a table impact has happened, we can relate the ball's spin to the launcher parameters (as we assume the spin to be constant within the free-flight phase). After an impact has happened, this is no longer possible, as the impact changes the initial spin of the ball. We indicate by $\mathbf{1}_{\text{ai}}$ whether $m_{\tau(2)}$ is taken after an impact. To compute a belief for the initial spin, we first assume the launcher's head to be oriented horizontally along the x-axis. For this launch orientation, we compute a "canonical spin" and its covariance depending on the motor parameters s_{m} (see Sec. 4.1.1), which we denote by $\omega_{\rightarrow\text{x}} = f^\omega(s_{\text{m}}, \psi_f)$, $\Sigma_{\omega_{\rightarrow\text{x}}} = \text{diag}([f^{\Sigma_\omega}(s_{\text{m}}, \psi_f)]^+)$. The functions $f^\omega, f^{\Sigma_\omega}$ are implemented by a neural network with
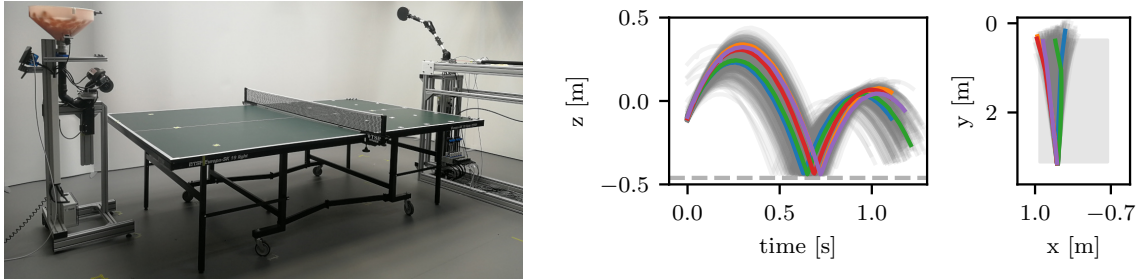
two heads with parameters $\boldsymbol{\psi}_f$. The azimuthal launch orientation can be changed by rotating the whole launcher frame by ϕ_f and by rotating the launcher's head by ϕ_l (see Fig. 3(c)). The elevational launch orientation can be changed by rotating the launcher's head by θ_l. To obtain the initial spin in the world coordinate system, we rotate the "canonical" spin $\boldsymbol{\omega}_{\rightarrow\mathrm{x}}$ accordingly. We absorb all rotations in a rotation matrix $\mathbf{R}_{\mathrm{rot}}(\phi_f + \phi_l, \theta_l)$, such that $\boldsymbol{\omega} = \mathbf{R}_{\mathrm{rot}}\boldsymbol{\omega}_{\rightarrow\mathrm{x}}$, $\boldsymbol{\Sigma}_\omega = \mathbf{R}_{\mathrm{rot}}\boldsymbol{\Sigma}_{\omega_{\rightarrow\mathrm{x}}}\mathbf{R}_{\mathrm{rot}}^\top$. These considerations only hold true for the free-flight phase after ball launch. After a table impact has happened, we cannot directly relate the ball spin to the launcher parameters. Therefore, in this case, we set the moments of the initial spin $\boldsymbol{\omega} = \mathbf{0}$, $\boldsymbol{\Sigma}_\omega = \mathrm{diag}([\boldsymbol{\sigma}_{\omega,\mathrm{ai}}]^+)$. During training, we obtain the angles ϕ_l, θ_l from piecewise linear regression models which map from launcher parameters $\boldsymbol{s}_\phi$, $\boldsymbol{s}_\theta$ to ϕ_l, θ_l. We obtained these models on the training split of trajectories recorded from the *default* launcher orientation. For the default orientation, we assume the launcher to shoot balls in the $-\mathrm{y}$ direction, i.e. $\phi_f = -90°$. When evaluating the filter in simulation or on the real robot, we infer the total azimuthal launch angle $\phi_f + \phi_l$ from the first two measurements of the trajectory, to avoid measuring the orientation of the launcher frame ϕ_f. In summary, we provide the information (ϕ_f, ϕ_l, θ_l, $\boldsymbol{s}_\mathrm{m}$, $\mathbf{1}_\mathrm{ai}$) of the ball launch to the model. As an ablation, we initialize $\boldsymbol{\omega} = \mathbf{0}$, $\boldsymbol{\Sigma}_\omega = \mathrm{diag}([\boldsymbol{\sigma}_\omega]^+)$, not depending on this information. For the drag and Magnus effect coefficient, we learn the mean and covariance of the initial state. The full initial state belief incorporating the first two available measurements is thus given by $\boldsymbol{\mu}_{\tau(2)|\tau(1:2)} = (\boldsymbol{p}^\top, \boldsymbol{v}^\top, \boldsymbol{\omega}^\top, a_\mathrm{d}, a_\mathrm{m})^\top$, $\boldsymbol{\Sigma}_{\tau(2)|\tau(1:2)} = \mathrm{blockdiag}(\boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_v, \boldsymbol{\Sigma}_\omega, [\sigma_{a_\mathrm{d}}]^+, [\sigma_{a_\mathrm{m}}]^+)$. In summary, the parameters of our gray-box model are $\boldsymbol{\Psi} = \{\mathbf{C}, \boldsymbol{\sigma}_q, \boldsymbol{\sigma}_r, \boldsymbol{\sigma}_p, \boldsymbol{\sigma}_v, \boldsymbol{\psi}_f, \boldsymbol{\sigma}_\omega, \boldsymbol{\sigma}_{\omega,\mathrm{ai}}, a_\mathrm{d}, a_\mathrm{m}, \sigma_{a_\mathrm{d}}, \sigma_{a_\mathrm{m}}\}$.

Prediction step   We follow the standard extended Kalman filter prediction step, yielding the prediction mean and covariance matrix $\boldsymbol{\mu}_{n+1|1:n} = g(\boldsymbol{\mu}_{n|1:n})$, $\boldsymbol{\Sigma}_{n+1|1:n} = \mathbf{J}\boldsymbol{\Sigma}_{n|1:n}\mathbf{J}^\top + \mathbf{Q}$, where $\mathbf{J}$ is the Jacobian matrix of the transition model, $\mathbf{J} = \frac{\partial}{\partial\boldsymbol{z}_n}g(\boldsymbol{z}_n)|_{\boldsymbol{z}_n=\boldsymbol{\mu}_{n|1:n}}$.

Correction step   In the case of missing observations, we perform multiple prediction steps before correcting the state belief with the next available measurement. We now assume that at timestep $n+1$ a measurement is available. For the correction step, we first compute the Kalman gain $\mathbf{K}$ with the observation matrix $\mathbf{H} = [\mathbf{I}_3, \mathbf{0}^{3\times8}]$ as $\mathbf{K} = \boldsymbol{\Sigma}_{n+1|1:n}\mathbf{H}^\top(\mathbf{H}\boldsymbol{\Sigma}_{n+1|1:n}\mathbf{H}^\top + \mathbf{R})^{-1}$. From this, we obtain the corrected moments as $\boldsymbol{\mu}_{n+1|1:n+1} = \boldsymbol{\mu}_{n+1|1:n} + \mathbf{K}(\mathbf{m}_{n+1} - \mathbf{H}\boldsymbol{\mu}_{n+1|1:n})$, $\boldsymbol{\Sigma}_{n+1|1:n+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\boldsymbol{\Sigma}_{n+1|1:n}$.

Learning   For notational simplicity, we again assume that there are no missing observations. Let $\mathcal{P} = \{(\hat{\mathbf{m}}_1^k, ..., \hat{\mathbf{m}}_{L_k}^k)\}_{k=1}^K$ denote the set of training trajectories, consisting of K sequences of ball position measurements, each sequence being of length L_k. The chunk operator expands a single trajectory into $L + 1 - N$ chunks of length $N = 50$: $\mathrm{chunk}(\hat{\mathbf{m}}_1, ..., \hat{\mathbf{m}}_L) = \{(\mathbf{m}_i, ..., \mathbf{m}_{i+N-1})\}_{i=1}^{L+1-N}$. By $\mathcal{P}_c = \bigcup_{k=1}^K \mathrm{chunk}(\hat{\mathbf{m}}_1^k, ..., \hat{\mathbf{m}}_{L_k}^k)$ we denote the set of training chunks and $\Delta(\mathcal{P}_c)$ the distribution over training chunks with uniform probability. For learning the filter parameters $\boldsymbol{\Psi}$, we maximize their expected marginal log-likelihood under the training chunk distribution, that is, $\max_{\boldsymbol{\Psi}} \mathbb{E}_{\mathbf{m}_{1:N} \sim \Delta(\mathcal{P}_c)} \log p(\mathbf{m}_{3:N} \mid \mathbf{m}_1, \mathbf{m}_2, \boldsymbol{\Psi})$. As we do not aim to learn a generative model of chunks but are only interested in applying the learned model for filtering and prediction, we additionally condition the marginal log-likelihood on the first two measurements, since we use these for initializing the filter. The marginal log-likelihood $\log p(\mathbf{m}_{3:N} \mid \mathbf{m}_1, \mathbf{m}_2, \boldsymbol{\Psi})$ can be decomposed (Särkkä, 2013) as follows

$$\log p(\mathbf{m}_{3:N}|\mathbf{m}_1, \mathbf{m}_2, \boldsymbol{\Psi}) = \sum_{n=3}^N \log p(\mathbf{m}_n|\mathbf{m}_{1:n-1}, \boldsymbol{\Psi}) = \sum_{n=3}^N \log \mathcal{N}(\mathbf{m}_n; \boldsymbol{\mu}_{n|1:n-1}, \boldsymbol{\Sigma}_{n|1:n-1}) \quad (7)$$

(*a*) Setup with the ball launcher (left) and the robot arm (right).

(*b*) Dataset of recorded table tennis trajectories, launched from the "default" orientation. Five randomly selected trajectories are colored.

Figure 2: Experimental setup (a) and visualization of recorded trajectories (b).

which allows for an iterative computation with complexity $\mathcal{O}(N)$ when filtering the chunk $(\mathbf{m}_1, ..., \mathbf{m}_N)$. We maximize the expected marginal log-likelihood on batches of chunks with batchsize 64 using the Adam optimizer (Kingma and Ba, 2015) with learning rate $5 \cdot 10^{-3}$.

4. Experiments

We conduct several experiments to answer the following research questions: **Q1:** How large is the prediction error of the EKF model, and how does it compare to black box baselines? **Q2:** Does supplying the launch parameters (launch direction, launcher motor speeds) to the predictive model improve prediction performance for the EKF and the RSSM baseline? We use a neural network to infer the initial ball spin from motor parameters, leading us to **Q3:** Does the spin inferred from the launcher parameters relate to a simple spin model derived from physical principles? Finally, we are interested in the ratio of balls returned by a robot arm (Büchler et al., 2016) when using the proposed model for trajectory prediction (**Q4**).

4.1. Setup

In Fig. 2 we show our experimental setup. A ball launcher shoots table tennis balls towards a robot arm that is actuated with pneumatic artificial muscles (Büchler et al., 2016). The position of the ball is measured using four RGB cameras as described in Gomez-Gonzalez et al. (2019). Details of the ball launcher are described in Sec. 4.1.1. The robot is only used for the return experiment in Sec. 4.4. For training the predictive models, we use recorded trajectories (Sec. 4.1.2).
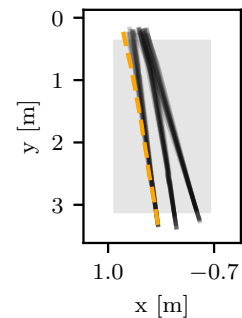

Figure 1: Trajectories of balls launched towards the robot arm (at $y \approx 0$), with unreturned trajectories colored orange. 29/30 launches are returned; details: Sec. 4.4.

4.1.1. LAUNCHER DETAILS

Our ball launcher follows the design of Dittrich et al. (2022). It accelerates the table tennis ball using three rubber wheels which are actuated by brushless motors. The azimuthal and elevational angle of the launcher's head can be adjusted with servo motors to change the direction of the launch.

(*a*) Ball launcher   (*b*) Schematic drawing (rear view)   (*c*) Schematic drawing (top/side view)   (*d*) Launch directions
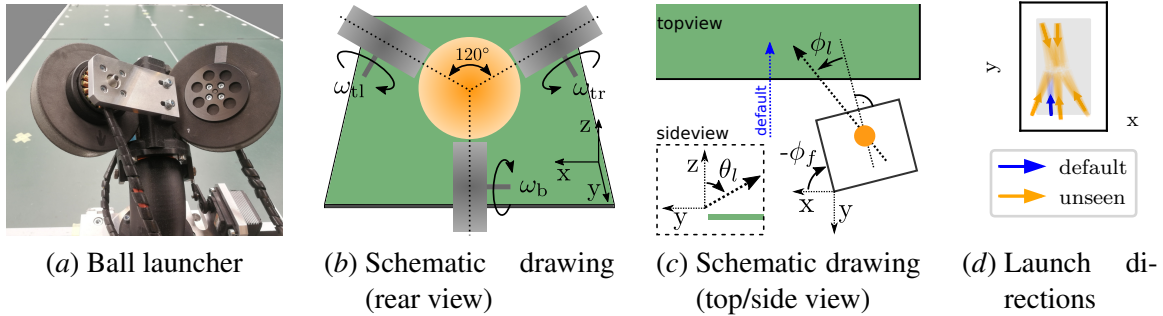
Figure 3: Experimental setup of the ball launcher. (a) Photo of the launcher (taken in the direction of ball launch, with table in background), (b) a schematic drawing of the launcher with rotating wheels (gray) and ball (orange), (c) frame (ϕ_f) and launch angles (ϕ_l, θ_l), (d) shoot directions for the "default" and "unseen" configurations.
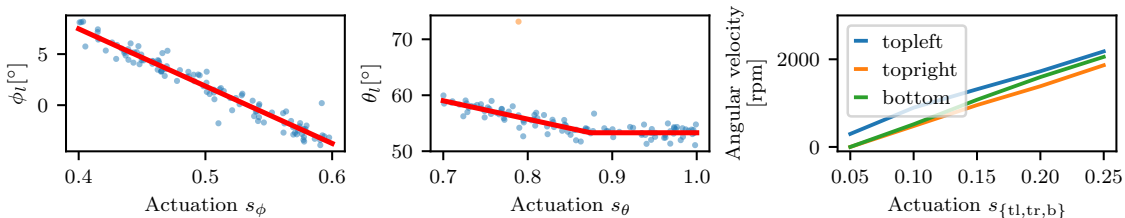
Figure 4: Relation of launcher parameters $s \in [0,1]^5$ to launch angles ϕ_l, θ_l and angular velocity of launcher wheels. The left two panels show the result of piecewise linear regression (red) to initial trajectory angles (blue); the right panel shows angular velocity of wheels depending on the actuation. The outlier (orange, center panel) is excluded.

The launcher frame can be freely positioned and rotated about the z-axis, as parameterized by the angle ϕ_f. We refer to Fig. 3 for more details on the launcher, including its geometry and the launch angles. The angular velocity of the top-left, top-right, and bottom motor are controlled through three actuation parameters $s_{\mathrm{m}} = (s_{\mathrm{m,tl}}, s_{\mathrm{m,tr}}, s_{\mathrm{m,b}})^{\top} \in [0,1]^3$. The mapping from actuation parameters to angular motor velocity is nonlinear, see Fig. 4. The azimuthal (ϕ_l) and elevational (θ_l) launch angles can be controlled through two parameters $s_\phi \in [0,1], s_\theta \in [0,1]$. We fit piecewise linear functions to the launch angles of the recorded trajectories to find a mapping from the actuation parameters $s_\phi \in [0,1], s_\theta \in [0,1]$ to launch angles ϕ_l, θ_l (see Fig. 4). The azimuthal launch direction can further be changed by rotating the launcher's frame about the z-axis by the angle ϕ_f. For the *default* orientation, we oriented the launcher such that it shoots along the negative y-axis for $\phi_l = 0$, i.e., $\phi_f = -90°$.

4.1.2. DATA RECORDING

For collecting trajectories for training, validation, and testing, we position the launcher at six different positions and orientations (see Fig. 3(d)). We term one particular position/orientation *default*, which we use both for training and testing, and the other five *unseen*, which we use for testing only. On the *default* orientation we collect 334 trajectories, which we split in 108 for training, 63 for

validation, and 163 for testing. For each of the five *unseen* configurations we collect 30 trajectories which are used for testing only. For each trajectory we randomly sample the launcher parameters uniformly from $s_\phi \in [0.4, 0.6]$, $s_\theta \in [0.7, 1.0]$, $s_{\text{tl}} \in [0.095, 0.155]$ (default), $s_{\text{tl}} \in [0.105, 0.165]$ (unseen), $s_{\{\text{tr},\text{b}\}} \in [0.135, 0.195]$ (default), $s_{\{\text{tr},\text{b}\}} \in [0.145, 0.205]$ (unseen). To simulate different launcher orientations, we optionally augment the training data by rotating each trajectory by a random angle $\phi_f \in [0, 2\pi]$ about the z-axis at the point with minimal z coordinate. We add 19 rotated trajectories for each existing trajectory to the training set, which forms the augmented dataset.

4.2. Prediction performance

Evaluation protocol The predictive performance of the investigated models is quantified by measuring the prediction error when filtering until one second before the trajectory ends, and predicting the remaining part of the trajectory. For shorter trajectories, we filter at least ten measurements. The prediction error for each sequence is given by the maximum Euclidean distance between the last five prediction-measurement pairs.

Baselines As a first baseline, we train a recurrent state space model, taken from a re-implementation[1] of (Hafner et al., 2019). We inherit the standard parameters except for the "free nats" parameter, which we determined empirically as 0.3 for minimal average prediction error on the validation split of the *default* dataset. Optionally, we pass the same ball launch information used in the EKF model for state initialization as action to the RSSM model. We train the model for 100,000 steps. As a second baseline, we train a trajectory variational auto-encoder (TVAE) from Gómez-González et al. (2020), using the provided implementation[2]. We use a model length of 250 as our longest trajectory is 235 steps. We train the model until the validation loss increases.

Results We refer to Fig. 5 for a visualization of the results. We observe that augmenting the training data is important for the EKF approach presented herein to generalize to the *unseen* launcher positions (Fig. 5(a)). In all settings, the EKF approach shows superior performance compared to the RSSM and TVAE baselines. Fig. 5(b) shows that initializing the spin using ball launch information reduced the prediction error drastically, both on the *default* and *unseen* launcher configurations. We show representative filtering and prediction results on three trajectories in Fig. 6.

4.3. Spin evaluation

With this experiment, we aim to verify the plausibility of spins which are estimated by the learned neural network $f^\omega(s_\text{m}, \psi_f)$ given launcher parameters s_m (see Sec. 3.3). For this, we formulate a simple model for the ball spin, which is derived from the geometry of the launcher (see Fig. 3(b)). We model the ball's spin for a launcher which is oriented to shoot balls in the $-y$ direction as

$$\left(\omega_{\text{x},\overrightarrow{-y}}, \omega_{\text{y},\overrightarrow{-y}}, \omega_{\text{z},\overrightarrow{-y}}\right)^\top = \omega_{\text{tl}}\alpha \left(\frac{1}{2}, 0, -\frac{\sqrt{3}}{2}\right)^\top + \omega_{\text{tr}}\beta \left(\frac{1}{2}, 0, \frac{\sqrt{3}}{2}\right)^\top + \omega_{\text{b}}\gamma \left(-1, 0, 0\right)^\top . \quad (8)$$

The reasoning behind the model is that every motor adds a spin component to the ball, which is the motor speed $(\omega_{\text{tl}}, \omega_{\text{tr}}, \omega_{\text{b}})$ scaled by a constant (α, β, γ). We note that the bottom motor causes a spin in negative y direction. The direction of the spin of the top motors is obtained by

---

1. https://github.com/Kaixhin/PlaNet
2. https://github.com/sebasutp/trajectory_forcasting

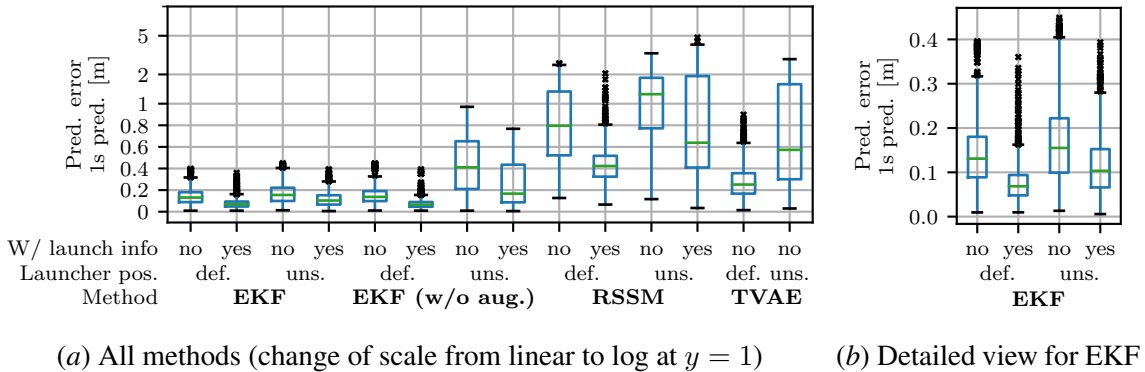(*a*) All methods (change of scale from linear to log at $y = 1$) (*b*) Detailed view for EKF

Figure 5: Prediction error for various methods on the test set of default (def.) and unseen (uns.) launcher positions, with and without ball launch information (launch info), for a prediction horizon of one second (see Sec. 4.2). Depicted statistics are over the prediction errors for ten independently trained models. All models are trained on the augmented dataset, except "EKF w/o aug.". The EKF model outperforms the RSSM (Hafner et al., 2019) and TVAE (Gómez-González et al., 2020) models (a). The EKF's prediction error can further be reduced by providing ball launch information (b).
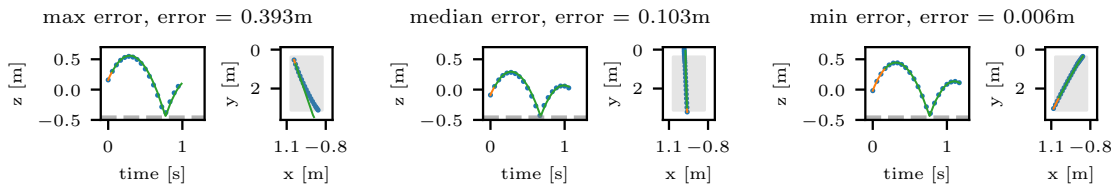
Figure 6: EKF filtering / prediction results on the *unseen* launcher orientations (with ball launch information). We filter until one second before the end of the trajectory (orange) and predict the remaining one second (green). Measurements are colored blue. We show every tenth measurement for visual clarity. Shown are the trajectories with max. / median / min. prediction error at the end of the trajectory from all *unseen* launcher configurations, over ten independently trained models. The median prediction error is $10.3\,\text{cm}$.

rotating the bottom-motor spin unit vector $(-1, 0, 0)^\top$ by $120°$ $(240°)$ about the y-axis. For this experiment, we obtain the values for $\omega_{\text{tl,tr,b}}$ from measurements for the angular launcher wheel velocity given the actuation parameters (see Fig. 4(c)). For all *test* trajectories from the *default* dataset, we first compute the spin for a launch in x direction with $f^\omega(\boldsymbol{s}_\text{m}, \boldsymbol{\psi}_f)$. We rotate this spin by $-90°$ about the z-axis to obtain the spin in $-y$ direction, as in Eq. (8). Finally, we obtain the parameters α, β, γ by minimizing a squared error between the rotated spins from f^ω and the spins estimated by Eq. (8). In Fig. 7 we show that the two spin estimates highly correlate, indicating that the values $f^\omega(\boldsymbol{s}_\text{m}, \boldsymbol{\psi}_f)$ indeed relate to the actual spin of the ball. It is important to note that without additional physical information, we can determine the spin only up to a scaling factor. This is because we both learn $k_\text{m} = a_\text{m}^2$ and $\boldsymbol{\omega}$ which appear as a product in Eq. (1). The actual spin could be inferred as $\boldsymbol{\omega}^* = k_\text{m} \boldsymbol{\omega}/k_\text{m}^*$ with $k_\text{m}^* = C_\text{m}\rho Ar/(2m)$ for known values of the ball's mass m, Magnus lift coefficient C_m, ball radius r, air density ρ and cross-sectional area $A = \pi r^2$.
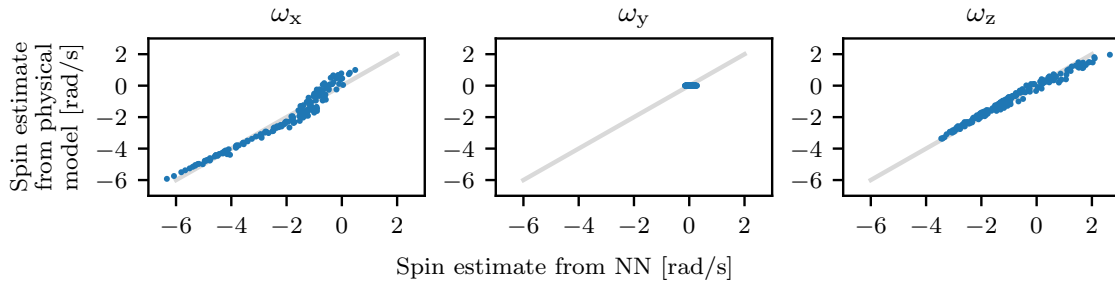
Figure 7: Correlation of spin values estimated by the neural network (NN) $f^\omega(s_\mathrm{m}, \psi_f)$ from motor actuations s_m and spin values computed with Eq. (8). Both estimated spin values highly correlate, indicating physical plausibility of spins estimated by $f^\omega(s_\mathrm{m}, \psi_f)$, see Sec. 4.3 for details.

4.4. Return performance

We evaluate the performance of our ball motion predictions by intercepting and returning balls with a four-degrees-of-freedom robot arm, where each degree of freedom is controlled by a pair of pneumatic artificial muscles (PAMs) (Büchler et al., 2016; Büchler et al., 2023). The robot arm is controlled by a learning-based iterative control framework for trajectory tracking (Ma et al., 2022). A table tennis racket is attached to the robot arm in order to return balls. As the ping-pong ball flies through the air, its position, velocity, and spin are continuously estimated with the EKF presented herein. The future evolution of the ball's states is then simulated in a receding horizon scheme using the learned model of the ball dynamics, with the latest state estimate as the initial condition. This predicted ball trajectory is used to determine the interception of the ball with the racket, which we represent as a pair of position (the interception position of the ball and the racket) and time (the time of interception). The predicted trajectory, and therefore the interception point, is repeatedly recalculated, as the state estimate is updated and improved with new measurements of the ball. The robot arm successfully returns 29 of 30 (97.7 %) launched balls, leveraging the EKF for filtering and prediction presented above. We refer to Fig. 1 for a visualization of returned and unreturned trajectories.

5. Conclusion

Based on a physically grounded model for the aerodynamic behavior of a flying ball respecting Magnus and drag effects and the extended Kalman filter, we have designed a filter and predictive model for table tennis ball trajectories. As we fit the parameters of the filter on offline data, no tedious tuning of initial, transition, and observation covariances is required. Our formulation allows for learning a neural model which estimates the ball's initial spin from ball launch information. This drastically improves the performance of long-term predictions compared to an uninformed initialization. Our results also support the findings of other works, which state that the ball's spin can only insufficiently be estimated from ball position measurements alone (Zhang et al., 2015; Blank et al., 2017). Our method could constitute groundwork for future research which, e.g., incorporates information on the racket movement to estimate the ball's spin and predict its future trajectory with high accuracy.

6. Acknowledgements

References

Russell L. Andersson. Understanding and applying a robot ping-pong player's expert controller. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1989.

Philipp Becker, Harit Pandya, Gregor H. W. Gebhardt, Cheng Zhao, C. James Taylor, and Gerhard Neumann. Recurrent Kalman Networks: Factorized inference in high-dimensional deep feature spaces. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Peter Blank, Benjamin H. Groh, and Björn M. Eskofier. Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor. In *Proceedings of the ACM International Symposium on Wearable Computers (ISWC)*, 2017.

Dieter Büchler, Heiko Ott, and Jan Peters. A lightweight robotic arm with pneumatic muscles for robot learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

Dieter Büchler, Roberto Calandra, and Jan Peters. Learning to control highly accelerated ballistic movements on muscular robots. *Robotics and Autonomous Systems (RAS)*, 159, 2023.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP, Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Filipe de Avila Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Alexander Dittrich, Jan Schneider, Simon Guist, Bernhard Schölkopf, and Dieter Büchler. AIMY: an open-source table tennis ball launcher for versatile and high-fidelity trajectory generation. *CoRR*, abs/2210.06048, 2022.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Marco Fraccaro, Simon Kamronn, Ulrich Paquet, and Ole Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Laurent Girin, Simon Leglaive, Xiaoyu Bie, Julien Diard, Thomas Hueber, and Xavier Alameda-Pineda. Dynamical variational autoencoders: A comprehensive review. *Foundations and Trends in Machine Learning*, 15(1-2):1–175, 2021.

Sebastian Gomez-Gonzalez, Yassine Nemmour, Bernhard Schölkopf, and Jan Peters. Reliable real-time ball tracking for robot table tennis. *Robotics*, 8(4), 2019.

Sebastián Gómez-González, Sergey Prokudin, Bernhard Schölkopf, and Jan Peters. Real time trajectory prediction using deep conditional generative models. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):970–976, 2020.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

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

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Okan Koç, Guilherme Maeda, and Jan Peters. Online optimal trajectory generation for robot table tennis. *Robotics and Autonomous Systems (RAS)*, 105:121–137, 2018.

Hailing Li, Haiyan Wu, Lei Lou, Kolja Kühnlenz, and Ole Ravn. Ping-pong robotics with high-speed vision system. In *Proceedings of the IEEE International Conference on Control, Automation, Robotics & Vision (ICARCV)*, 2012.

Hsien-I Lin, Zhangguo Yu, and Yi-Chen Huang. Ball tracking and trajectory prediction for table-tennis robots. *Sensors*, 20(2), 2020.

Lennart Ljung. *System Identification: Theory for the User*. Prentice-Hall, Inc., USA, 1986.

Hao Ma, Dieter Büchler, Bernhard Schölkopf, and Michael Muehlebach. A Learning-based Iterative Control Framework for Controlling a Robot Arm with Pneumatic Artificial Muscles. In *Robotics: Science and Systems (RSS)*, 2022.

Michiya Matsushima, Takaaki Hashimoto, Masahiro Takeuchi, and Fumio Miyazaki. A learning approach to robotic table tennis. *IEEE Transactions on Robotics (T-RO)*, 21(4):767–771, 2005.

Katharina Mülling, Jens Kober, and Jan Peters. Simulating human table tennis with a biomimetic robot setup. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)*, 2010.

Akira Nakashima, Yuki Ogawa, Yosuke Kobayashi, and Yoshikazu Hayakawa. Modeling of rebound phenomenon of a rigid ball with friction and elastic effects. In *Proceedings of the American Control Conference (ACC)*, 2010.

Simo Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

Jonas Tebbe, Yapeng Gao, Marc Sastre-Rienietz, and Andreas Zell. A table tennis robot system using an industrial KUKA robot arm. In *Proceedings of the German Conference on Pattern Recognition (GCPR)*, 2018.

Qizhi Wang, KangJie Zhang, and Dengdian Wang. The trajectory prediction and analysis of spinning ball for a table tennis robot application. In *Proceedings of the IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems (CYBER)*, 2014.

Yifeng Zhang, Rong Xiong, Yongsheng Zhao, and Jianguo Jack Wang. Real-time spin estimation of ping-pong ball using its natural brand. *IEEE Transactions on Instrumentation and Measurement (TIM)*, 64(8):2280–2290, 2015.

Zhengtao Zhang, De Xu, and Min Tan. Visual measurement and prediction of ball trajectory for table tennis robot. *IEEE Transactions on Instrumentation and Measurement (TIM)*, 59(12):3195–3205, 2010.