Learning Interpretable Reduced-order Models for Jumping Quadrupeds

Gioele Buriani, Jingyue Liu, Cosimo Della Santina

***Abstract*—This work introduces a novel methodology for the development of interpretable reduced-order dynamic models specifically tailored for jumping quadruped robots. Leveraging Symbolic Regression combined with autoencoder neural networks, the framework autonomously derives symbolic equations from data and fundamental physics principles capturing the complex dynamics of jumping actions with high fidelity. This approach significantly reduces model complexity while enhancing interpretability, facilitating deeper insights for legged robotic applications. The efficacy and accuracy of the proposed models are validated through comprehensive experimental studies, marking a substantial advancement in the design of agile and efficient legged robots. This research demonstrates the outperformance of a learned 2D model compared to existing template models such as the ASLIP. Also, an analysis of the dimensionality of the learned model is conducted showing the overarching tradeoff between accuracy and complexity. The method is validated on different simulated quadrupeds and an actual hardware robot.**

***Index Terms*—Quadruped Robots, Dynamic Modeling, Symbolic Regression, Autoencoders, Machine Learning, Interpretability, Reduced-order Models.**

I. Introduction

In the summer of 2022, at a live concert in Milan, the well-known Belgian artist Stromae was handed a cape by a Go1 quadruped robot right before he performed his hit song *Formidable*. This event is just one example of how legged robots, once confined to research labs, are now making their mark in popular culture. This shift is a direct result of the significant advancements in robotics technology in recent years, a trend that is expected to continue.

In the pursuit of developing state-of-the-art technologies for controlling legged systems, researchers are also endeavouring to accurately model these complex systems. Since most control strategies for these robots are model-based, having a precise model is crucial.

However, modelling such dynamic systems is not straightforward. Leading companies such as Boston Dynamics tend to keep their models proprietary. Even if these models were accessible, their utilization would remain beset with obstacles. The complete dynamic model, or the full-order model, of such systems, would be inherently complex, encompassing all dimensions of the robot. As a result, applying control algorithms directly using these full-order models is impractical and computationally burdensome. Consequently, many researchers now rely on simplified, two-dimensional reduced-order template models [1] that capture the essential dynamics of the robot, making it easier to develop control algorithms. However, this simplification reduces the accuracy of the model when describing the robot's actual behaviour, as only a few dimensions are actually tracked and others, such as the leg joint angles, are disregarded.

Current research also shows an increasing dependence on machine learning algorithms to directly learn dynamics models, often employing neural networks to map input-output relationships [2]. Despite their effectiveness in capturing complex relationships, these models often are not easily understandable or adjustable by humans, presenting challenges due to their opaque nature.

This work aims to bridge the gap between full-order and template models by learning dynamic models from data that allow for a balance between complexity and accuracy. Additionally, to address the challenge of interpretability, the study employs Symbolic Regression to generate models as symbolic equations, thereby enhancing understandability and adjustability.

The contribution of this work is developing an algorithm tailored to:

- Effectively reduce the dimensionality of dynamic systems, quadrupeds in this work, to a latent space that adequately represents the overall body dynamics for the jumping motions
- Express the dynamic model of such reduced systems in an interpretable way through symbolic equations within the latent space
- Validate the proposed methods through both simulation and hardware experiment

According to the author's knowledge, this research is one of the first to explore symbolic reduced-order dynamic models of legged systems. As such, it aims to represent an initial step in this emerging field, opening avenues for numerous advancements.

II. Related works

The landscape of dynamic system modelling and control for legged robots has witnessed advancements in recent years. This section reviews some key approaches in the existing literature, providing a comprehensive understanding of the state-of-the-art methodologies employed in similar domains.

A. Parameter identification

Zimmermann et al. [3] addressed the challenge of integrating a non-proprietary robotic arm with a Spot quadruped, where building a model of the robot was necessitated by the quadruped's opaque behaviour. Their focus was on parameter optimization rather than traditional machine learning approaches [4], aiming to refine a pre-established dynamic

model by optimizing its parameters using measured data. This method, while effective in subsequent control applications, relies on predefined template models for basic dynamics, limiting its adaptability to complex scenarios. The authors suggest that employing more advanced learning methods may yield a more accurate model, enhancing system performance in such contexts.

B. Gaussian Processes

Chatzilygeroudis et al. [5] model a hexapod robot employing Gaussian Processes (GPs) [6], known for their ability to quantify model uncertainty [7], [8]. Their approach integrates a parameterized simulator within the GP framework, utilizing the simulator's output as the mean function. Despite its strengths in handling uncertainties, this method faces challenges with discrepancies between simulator predictions and real-world data, particularly as system complexity grows. Exploring alternative approaches that learn reduced-order models without prior knowledge may address these issues.

III. Preliminary

This section presents some theoretical background relevant to the subjects explored in the later sections of this paper.

A. Quadruped Dynamics

In quadruped robot modelling, the floating base dynamics approach is widely used. This method, illustrated in Figure 1, conceptualizes the robot as a movable base (body), with kinematic chains (legs) attached. This model becomes intricate

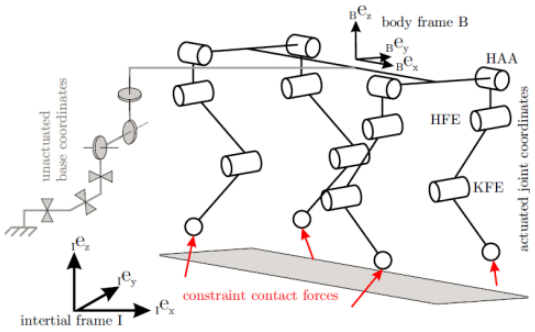

Fig. 1: Depiction of a quadruped robot using floating-base dynamics. Source: [9].

in quadrupeds due to their numerous leg joints, leading to a total of 12 dimensions for the legs alone. Additionally, floating base systems, being under-actuated, require an additional 6 degrees of freedom, without direct actuation, to describe the robot's position and orientation relative to an inertial frame [10].

The system's mathematical representation, adapted from [11], can be outlined as

$$q = \begin{bmatrix} q_j \\ q_b \end{bmatrix}, \tag{1}$$

where $q_j \in \mathbb{R}^n$ represents the actuated joint coordinates for the quadruped's n joints and $q_b \in \mathbb{R}^6$ denotes the un-actuated base coordinates relative to an inertial frame, including the body's Cartesian position $(x, y, z) \in \mathbb{R}^3$ and Euler angles $(\phi, \theta, \psi) \in \mathrm{SO}^3$.

The equations of motion for the robot interacting with its environment are expressed as:

$$M(q)\ddot{q} + h(q, \dot{q}) = S^T\tau + J_C^T(q)\lambda, \tag{2}$$

where

- $M(q) \in \mathbb{R}^{(n+6)\times(n+6)}$ is the mass matrix for the floating base
- $h(q, \dot{q}) \in \mathbb{R}^{(n+6)}$ includes centripetal, Coriolis, and gravitational forces in the floating base system
- $S = \begin{bmatrix} I_{n\times m} & 0_{n\times 6} \end{bmatrix}$ serves as the selection matrix for actuated joints, ensuring torques affect only the actuated joints and not the un-actuated base coordinates
- $\tau \in \mathbb{R}^n$ is the vector of applied torques
- $J_C \in \mathbb{R}^{k\times(n+6)}$ is the Jacobian matrix for k linearly independent constraints
- $\lambda \in \mathbb{R}^k$ represents the constraint forces for these k constraints

The complexity and high dimensionality of this system often pose challenges in developing a fully precise model with floating base dynamics. Therefore, reduced-order template models are frequently employed in practical applications to simplify computations and improve efficiency.

B. Learning method

The chosen machine learning method aims at learning a reduced-order model of the dynamic system to reduce complexity while ensuring the model's interpretability. The autoencoder, a self-supervised network, is employed to extract a lower-dimensional representation of the input data while minimizing information loss [12]. Then, for enhancing interpretability, symbolic regression is employed as the learning algorithm, particularly utilizing Sparse Identification of Nonlinear Dynamics (SYNDy) [13]. SINDy operates by identifying sparse, governing equations from high-dimensional data, efficiently distilling the dynamics of the system into concise, interpretable mathematical forms.

Champion et al. [14] devised a method, illustrated in Figure 2, that combines the principles of SINDy with an autoencoder to distil high-dimensional data into a lower-dimensional space, enhancing manageability and interpretability. This method

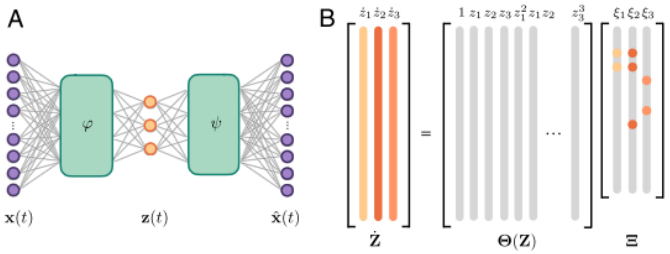

Fig. 2: Schematic of the SINDy autoencoder method. Source: [14].

trains both the autoencoder parameters and the SINDy coefficients jointly, encouraging the autoencoder to find a lower-dimensional representation of data that, at the same time,

presents certain characteristics, such as sparsity, that help the SINDy algorithm work efficiently.

The algorithm aims to identify reduced coordinates $z(t) = \varphi(x(t))$ in a lower-dimensional space $\mathbb{R}^d$ (where d is lower than the original dimension n), linked to a first-order dynamical model $\dot{z}(t) = g(z(t))$. It provides coordinate transformations φ and ψ, mapping the original data to these latent coordinates via the encoder $z = \varphi(x)$, and reconstructing the original measurements from the reduced coordinates through the decoder $x_{\text{dec}} = \psi(z)$.

The final symbolic equations are created by constructing a library of basis functions $\Theta(z) = [\theta_1(z), \theta_2(z), \ldots, \theta_p(z)]$ and learning a set of SINDy coefficients $\Xi = [\xi_1, \ldots, \xi_d]$ such that:

$$\dot{z}(t) = g(z(t)) = \Theta(z(t))\Xi \tag{3}$$

Throughout training, all trainable parameters in the autoencoder φ and ψ, and the SINDy coefficients Ξ, are optimized for the best possible result, based on the joint loss function. The function includes the standard autoencoder loss, the standard SINDy loss in the latent space, the loss on the reconstructed SINDy prediction in the real space and an L_1 regularization on the SINDy coefficients Ξ to promote sparsity. For model refinement, a further sparsity-promoting coefficient mask, consisting of 0s and 1s, is applied to Ξ. This selectively zeroes coefficients beneath a specific threshold, yielding a sparser and more simplified final model.

IV. Methodology

This section will now present the main changes and contributions of this work to the model learning procedure.

A. Dynamic system interpretation

In line with the floating base dynamics described in Subsection III-A, the robot's state q and the applied torques τ are measurable, but the constraint forces λ remain unobservable, presenting a significant obstacle to identifying the model of the system.

For the machine learning aspect of this work, the system is simplified and approximated. It is assumed that the constraint forces on the joints q_j are negligible. Furthermore, the body's constraint forces are considered to be solely reliant on the ground reaction forces (GRFs) acting on the robot's feet during ground contact.

To approximate the GRFs for a single foot in contact, each limb is modelled as a kinematic chain that exerts a force from the robot's base to the ground. This force, equal in magnitude and opposite in direction to the GRF, is calculated using the Jacobian J_k of the leg k, relating changes in foot position Δx_k to the change in joint configuration Δq_k, through $\Delta x_k = J_k \Delta q_k$. The force calculation is

$$F_k = J_k^{-T} \tau_k, \tag{4}$$

where $F_k \in \mathbb{R}^3$ is the leg's ground force, and J_k and τ_k are known for each timestep. More details on the derivation can be found in Appendix A.

The linear force acting on the robot's centre of mass is estimated by

$$F_{\text{lin}} = \sum_{k=1}^{4} F_k c_k. \tag{5}$$

Similarly, the angular torque acting on the robot's body is

$$F_{\text{ang}} = \sum_{k=1}^{4} (F_k \times x_k) c_k. \tag{6}$$

In these formulas, $c_k \in \{0, 1\}$ is a binary variable that represents the contact state of each foot, with the value of 1 denoting contact and 0 denoting no contact.

The total force $F \in \mathbb{R}^6$ acting on the robot's body, comprising both linear and angular components, is represented as

$$F = \begin{bmatrix} F_{\text{lin}} \\ F_{\text{ang}} \end{bmatrix}. \tag{7}$$

With these approximations, the dynamic system can be expressed as

$$\begin{cases} \ddot{q}_j = f_1(q, \dot{q}) + g_1(\tau) \\ \ddot{q}_b = f_2(q, \dot{q}) + g_2(F) \end{cases}, \tag{8}$$

where F takes the role of a virtual actuation on the unactuated coordinates, facilitating data preparation for the machine learning procedure without actually providing a theoretical description of the dynamic system.

B. Linear Actuated SINDy Autoencoder

Building upon the framework delineated in Subsection III-B, the SINDy autoencoder algorithm is modified to align with the objectives of this research.

The autoencoder was chosen to be linear, such that the latent space can retain an easily identifiable linear relationship with the real space, ensuring a more direct physical meaning. Therefore, now the autoencoder operations that occur at the start of every iteration are

$$z = W_e q + B_e, \tag{9}$$

$$q_{\text{dec}} = W_d z + B_d, \tag{10}$$

where W and B represent the autoencoder weights and biases.

To learn the dynamical model of the quadruped, a second-order model has to be considered to link the accelerations to the robot state. Therefore, both $\ddot{z}$ and $\dot{z}$ are calculated as

$$\dot{z} = W_e \dot{q}, \tag{11}$$

$$\ddot{z} = W_e \ddot{q}. \tag{12}$$

In the context of a robotic system, the control inputs u are incorporated into the dynamics model. Together with the system's state, the system inputs in the real space must also be transformed into their counterparts in the latent space, u_{lat}. This transformation relies on the assumption that the input space has the same dimension as the state space, as explained in Subsection IV-A. u_{lat} is then calculated as

$$u_{\text{lat}} = W_e^{-T} u. \tag{13}$$

more details on the derivation can be found in Appendix A.

The overall latent space dynamical model acquires the form

$$\ddot{z}_{\text{pred}} = g\left(z, \dot{z}, u_{\text{lat}}\right) = \Theta(z, \dot{z}, u_{\text{lat}})\Xi, \tag{14}$$

where Θ is the SINDy library encompassing first-order polynomial and trigonometric functions of the inputs, along with a constant. Further details about the library construction are documented in Appendix B.

The concluding step entails processing the predicted accelerations in the latent space $\ddot{z}_{\text{pred}}$ through the decoder to map these accelerations back to the real space as:

$$\ddot{q}_{\text{pred}} = W_d \ddot{z}_{\text{pred}} \tag{15}$$

The combined loss can now be formally defined. The first term is the standard autoencoder loss:

$$\mathcal{L}_{\text{dec}} = \|q - q_{\text{dec}}\|_2^2 \tag{16}$$

The second term is the standard SINDy loss in the latent space:

$$\mathcal{L}_{\ddot{z}} = \|\ddot{z} - \ddot{z}_{\text{pred}}\|_2^2 \tag{17}$$

The third term is the SINDy loss mapped to the real space:

$$\mathcal{L}_{\ddot{q}} = \|\ddot{q} - \ddot{q}_{\text{pred}}\|_2^2 \tag{18}$$

The final term is the L_1 regularization on the SINDy coefficients Ξ:

$$\mathcal{L}_{\text{reg}} = \|\Xi\|_1 \tag{19}$$

The overall loss is the sum of these terms:

$$\mathcal{L} = \mathcal{L}_{\text{dec}} + \lambda_1 \mathcal{L}_{\ddot{z}} + \lambda_2 \mathcal{L}_{\ddot{q}} + \lambda_3 \mathcal{L}_{\text{reg}} \tag{20}$$

where λ_1, λ_2, and λ_3 are hyperparameters that balance the contributions of each loss term.

C. Data collection

1) Go1 Simulation Data: Data for the experiment was sourced from simulations of a quadruped robot executing jumps of various lengths. The simulations were conducted using the PyBullet engine [15], modelled after the Unitree Go1 quadruped robot, shown in Figure 3.

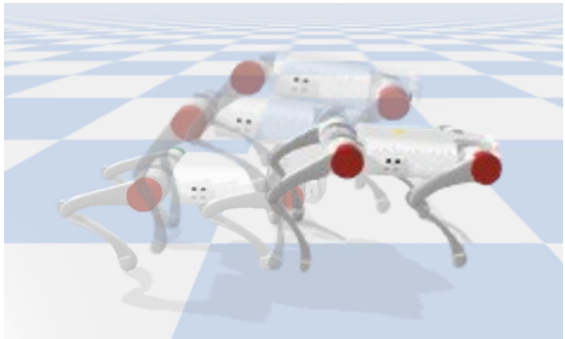

Fig. 3: Simulated Go1 quadruped performing a jumping motion.

The dataset includes joint angles and velocities $q_j, \dot{q}_j \in \mathbb{R}^{12}$, body position and velocity $q_b, \dot{q}_b \in \mathbb{R}^6$, input torques $\tau \in \mathbb{R}^{12}$, and the contact state of the feet $c \in \mathbb{R}^4$ for each timestep. Therefore, the complete state of the robot $q \in \mathbb{R}^{18}$ along with its time derivative is established.

In addition to the standard dataset, a variant with added noise is also generated to enhance the model's robustness for real-world applications. This noise includes three different components:

- Gaussian noise on the initial configuration
- Gaussian noise on the state readings
- External disturbance force applied to the robot's CoM

2) Go1 Real data: Together with the simulated data, hardware data was collected from an actual Go1 quadruped, shown in Figure 4.

Fig. 4: Real Go1 Quadruped performing a jumping motion.

The quadruped robot was operated using the same control algorithm as in the simulation, and data collection was performed similarly using the official Go1 SDK.

3) A1 Simulation Data: In exploring the algorithm's adaptability to different quadruped models besides the Go1, a distinct dataset was recorded from an alternative simulation [16].

The simulation involved the Unitree A1 quadruped, which is visualized in Figure 5, operating within the Pybullet simulation environment.

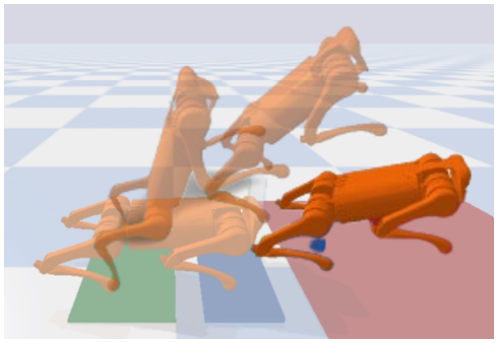

Fig. 5: Simulated A1 quadruped performing a jumping motion.

Observation of the figure reveals a variation in the jumping strategy compared to the Go1 scenario. Whereas the Go1 executes what is termed a synchronous jump, with all legs propelling off the ground simultaneously, the A1 employs an asynchronous approach. This method features an initial detachment of the front legs, followed by a subsequent powerful push using the rear legs. This behavioural distinction in the jumping technique serves to further contrast the two robotic models.

4) Data preprocessing: The state's acceleration is computed as

$$\ddot{q} = \frac{d\dot{q}}{dt}. \tag{21}$$

The force on the center of mass $F \in \mathbb{R}^6$ is calculated as outlined in Subsection IV-A:

$$F = \begin{bmatrix} \sum_{k=1}^{4} F_k c_k \\ \sum_{k=1}^{4} (F_k \times x_k) c_k \end{bmatrix}. \tag{22}$$

This process enables the formation of the final input to the system $u \in \mathbb{R}^{18}$:

$$u = \begin{bmatrix} \tau \\ F \end{bmatrix}. \tag{23}$$

At this point, all required data $q, \dot{q}, \ddot{q}, u, c$ is accounted for at each timestep.

The data is then segmented into distinct phases based on the number of grounded feet, which influences the robot's dynamics. For jumping motions. Three phases are identified:

- **Launch phase**: Preparing for the jump, with all feet grounded.
- **Flight phase**: Airborne, with no feet grounded.
- **Landing phase**: Landing after the jump, with all feet grounded.

Transitional timesteps with one to three grounded feet are included in either the launch or landing phase, depending on their occurrence.

Since both the launch and landing phases involve four grounded feet, they are merged into a 'contact phase'. Consequently, the data is divided into:

- **Contact data**: All four feet are grounded.
- **Flight data**: No feet are grounded.

To standardize phase durations for machine learning, which operates in batches of 100 timesteps, data is interpolated to ensure each phase's duration is a multiple of 100. As a result, each contact phase lasts 700 timesteps and each flight phase 200 timesteps.

Each dataset is then split into training, testing, and validation sets for machine learning applications.

D. Training procedure

When training the algorithm on the jumping dataset, the inherent hybrid aspect of the system has to be taken into account. In fact, during the different phases, the system behaves differently due to the change in contact with the ground, leading to necessarily different differential equations belonging to different system phases.

Also, regarding the autoencoder, training it independently on data from each phase before merging them for the final model may introduce inconsistencies in latent space representation, potentially compromising the physical relevance of the combined model's latent space. On the other hand, training only on a single phase could result in inaccurate reconstructions for the full jump dataset, mainly due to decoder biases. To mitigate these challenges, transfer learning [17] is applied.

The initial training step prioritizes the contact data due to its pivotal role in jump dynamics modelling. The focus here is on refining the encoder to achieve a precise latent representation of this critical phase.

In the second training step, the decoder is specifically targeted for improvement to adeptly reconstruct the entire jump dataset, not just the contact phase. This phase proceeds with the encoder's parameters fixed, ensuring the initial phase's accurate encoding is preserved while expanding the decoder's reconstruction capabilities to encompass the full jump data.

The final training step shifts focus to the SINDy parameters, adjusting them for both the contact and flight data. This stage is distinguished by keeping both the encoder and decoder parameters fixed, allowing for the precise learning of distinct SINDy coefficients. These coefficients represent different governing differential equations for each phase, ensuring that the consistent encoding and decoding framework established previously remains intact while accurately capturing the diverse dynamics of the jump.

E. Testing Method

To validate the final model M, a single dataset is selected from multiple test jumps. The initial system state $[q_0, \dot{q}_0]$ and inputs u are transformed by the trained encoder and the input transformation equation to the latent initial state $[z_0, \dot{z}_0]$ and $u_{\text{lat}} = W_e^{-T} u$, respectively. Model M then predicts latent accelerations $\ddot{z}_{\text{pred}} = M(z, \dot{z}, u_{\text{lat}})$, which are then integrated using the LSODA method [18] to obtain predicted latent positions and velocities. This process is repeated for each timestep:

$$\ddot{z}_{\text{pred}} = M(z_{\text{pred}}, \dot{z}_{\text{pred}}, u_{\text{lat}}) \tag{24}$$

At the end, the predicted latent states z_{pred} are decoded into real-space $q_{\text{pred}} = W_d z_{\text{pred}} + B_d$ for direct comparison with actual jump data.

An alternative testing approach simulates the learned system dor shorter intervals (e.g., 0.1 seconds) with periodic resets to the original latent state $[z, \dot{z}]$ rather than the predicted state $[z_{\text{pred}}, \dot{z}_{\text{pred}}]$. This strategy reduces error accumulation over time, enhancing precision, especially for short-term forecasts critical in model-based control. Meanwhile, the primary method remains preferred for in-depth, long-term evaluations like path planning, offering a broader assessment.

In hybrid systems, data points with unchanged contact states are individually simulated according to the specific model and then decoded with the established common decoder, ensuring accurate and consistent evaluation across different phases.

V. Experimental Validation

In this section, the results of the experiments and their respective implications will be explored.

A. 2D model analysis

The initial exploration of the latent model for a jumping quadruped robot employs two latent dimensions. This decision aligns with the common dimensional choice in most template models, such as the Actuated Spring-Loaded Inverted Pendulum (ASLIP) [19].

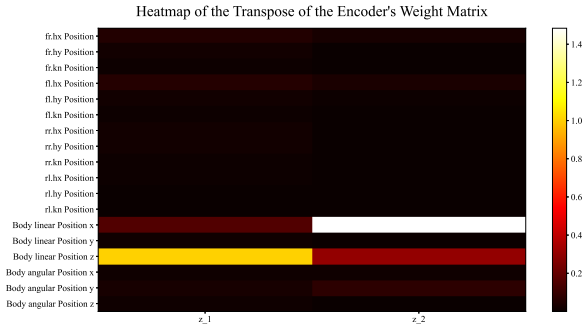

Fig. 6: Heatmap visualization of the transposed weight matrix from the trained encoder, reflecting weight magnitudes. Brighter colours denote higher absolute values of the weights, with the vertical axis showing the 18 real-space dimensions and the horizontal axis representing the two latent dimensions z_1 and z_2.

Figure 6 presents a heatmap of the transposed weight matrix from the trained encoder.

The heatmap clearly shows that the predominant weight values associate the two latent dimensions with the robot's linear x and z positions in real space, indicating a strong linear relationship.

This observation corroborates the dimensionality choice in template models like the ASLIP. The existence of such models validates the findings here, and simultaneously, these results empirically justify the dimensional choices made in the template models. Nevertheless, it is crucial to note that this selection of latent dimensions, though unbiased and algorithmically determined, is likely limited by the linearity of the autoencoder, potentially restricting the discovery of more complex latent space structures.

Regarding the final learned equations, the model for the contact phase is described by

$$\begin{cases} \ddot{z_1} = & 0.36 - 0.16\dot{z_1} - 0.91\dot{z_2} - 0.75\sin(\dot{z_1}) + 0.11u_1 - 0.05u_2 \\ \ddot{z_2} = & -0.16 - 0.23\dot{z_1} - 0.75\dot{z_2} - 0.96\sin(\dot{z_2}) + 0.14u_2 \end{cases} \tag{25}$$

and for the flight phase by:

$$\begin{cases} \ddot{z_1} = 0.29 - 1.22\dot{z_1} - 1.00\sin(\dot{z_1}) + 51.05u_1 \\ \ddot{z_2} = 0.42\dot{z_1} + 0.52\sin(\dot{z_1}) \end{cases} \tag{26}$$

These formulations differ significantly from the ASLIP model, expressed as:

$$\begin{cases} \ddot{x} & = \dfrac{k_s(l_{0_x} + p_{f_x})}{m} - \frac{k_s}{m}x + u_x \\ \ddot{z} & = \dfrac{k_s(l_{0_z} + p_{f_z})}{m} - g - \frac{k_s}{m}z + u_z \end{cases} \tag{27}$$

for the contact phase, and

$$\begin{cases} \ddot{x} & = 0 \\ \ddot{z} & = -g \end{cases} \tag{28}$$

for the flight phase, where:

- k_s is the spring constant
- m is the mass of the body
- l_{0_x} and l_{0_z} are the leg rest lengths along the x and z dimensions
- p_{f_x} and p_{f_z} are the positions of the leg base along the x and z dimensions

Nevertheless, the enhanced complexity of the 2D model derived from data leads to a more accurate reconstruction of real dimensions. Unlike most template models, including ASLIP, which primarily track the linear x and z dimensions, the proposed model achieves precise reconstruction of most real dimensions, as depicted in Figure 7. Coupling the learned

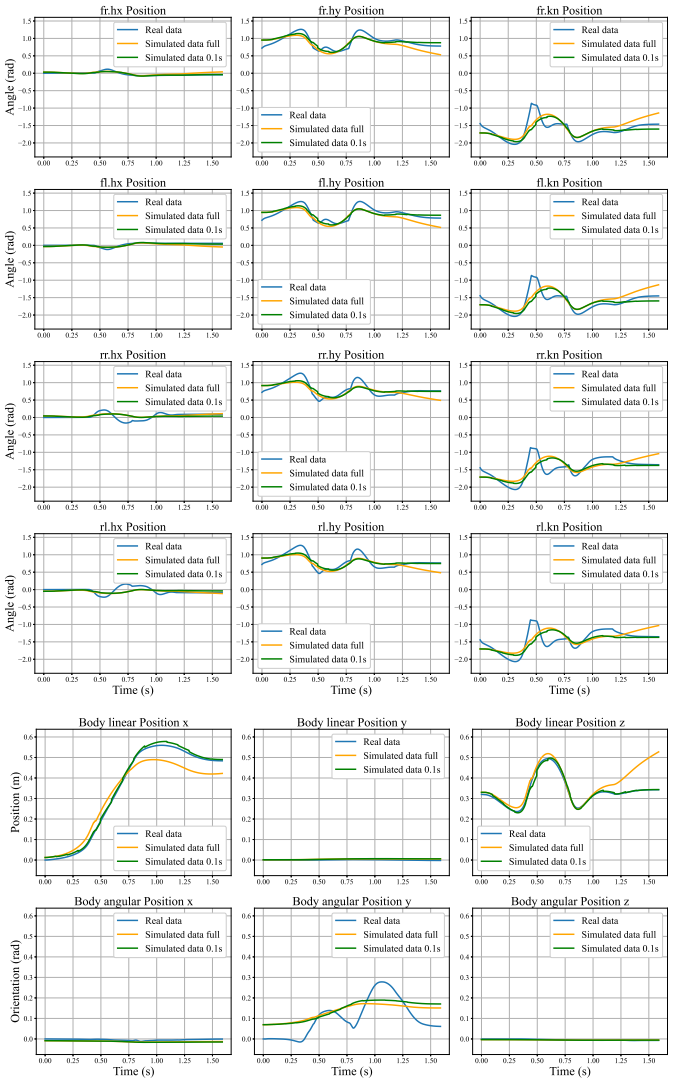

Fig. 7: Comparison of joint angles (top) and body positions (bottom) during a quadruped robot's jumping motion. Blue lines represent real data measurements, orange lines depict full interval simulated reconstructions, and green lines show 0.1-second interval simulated reconstructions, all within a 2D latent space model.

2D model with a decoder trained for accurate reconstruction of real space from the latent space endows it with greater versatility compared to standard template models, albeit at a slightly increased complexity.

Moreover, when analysing only the body's linear position along the x and z axes, the learned model presents significant advantages compared to a traditional ASLIP, as depicted in Figure 8. In fact, while the virtual spring is suitable to model the robot's elasticity for the launch phase, it does not work

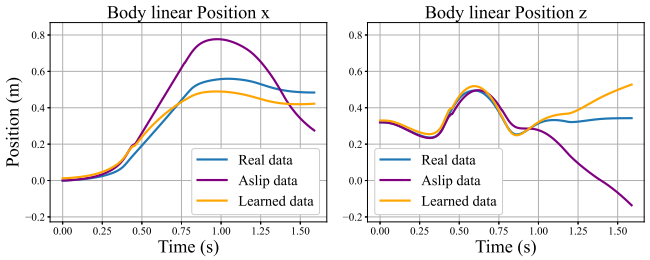

Fig. 8: Comparison of the body's linear positions along the x and z axes during a quadruped robot's jump. The blue line indicates real data, the purple line represents the ASLIP model simulation, and the orange line shows the learned model simulation.

well for the landing phase as the lack of a damping factor in the model prevents energy dissipation and therefore the model would keep oscillating indefinitely. On the other hand, the learned model presents some damping terms that allow the model to track correctly the trajectories in the landing phase (despite some propagating error) without excessively limiting the system's kinetic energy during the launch phase.

B. Latent dimensionality analysis

The choice of latent space dimensionality is a crucial aspect of the model's design. An increase in the number of latent dimensions leads to both enhanced accuracy and added complexity.

It is observed that, due to the linear characteristics of the autoencoder, a higher dimensional latent space captures similar real states as for lower dimensional cases. This is demonstrated in Figure 9, where the increase in the number

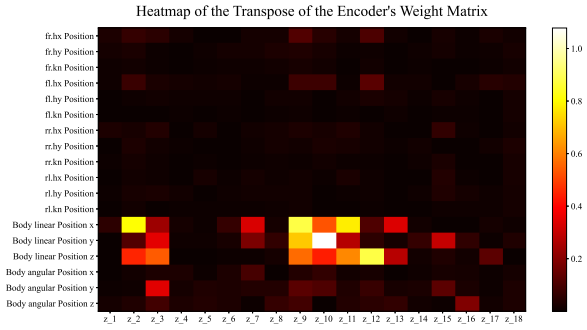

Fig. 9: Heatmap visualization of the transposed weight matrix from the trained encoder in the case of 18 latent dimensions, reflecting weight magnitudes. Brighter colours denote higher absolute values of the weights, with the vertical axis showing the 18 real-space dimensions and the horizontal axis representing the 18 latent dimensions.

of latent dimensions, in this case up to 18, does not alter the primary influence of the same real dimensions on the latent state. More examples are in Appendix C.

This phenomenon might stem from the more nonlinear behaviour of joint coordinates, complicating the establishment of a linear relationship with the latent dimensions.

This necessitates a thorough analysis of the accuracy versus complexity tradeoff based on the latent space dimension. Complexity is represented by the final reconstruction error $\mathcal{E}_{\text{dec}}$ on the validation set, while the final number of active SINDy coefficients $|\Xi|$ indicates accuracy.

Figure 10 displays the results of 18 different contact phase models, each with a varying number of latent dimensions, under identical training and testing conditions.

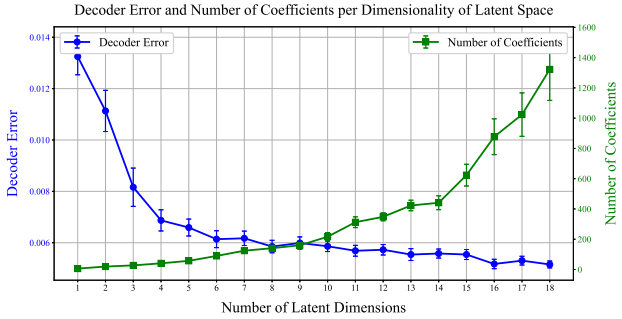

Fig. 10: Graph showing the trade-off between accuracy and complexity as the number of latent dimensions increases. The blue line indicates the average decoder error, while the green line shows the number of active SINDy coefficients. Error bars represent the standard deviation from five different random seeds, reflecting the consistency of the model's performance across varying latent dimensions.

For a more quantitative evaluation of this tradeoff, a combined loss for the models $\mathcal{L}_{\text{mod}}$ can be calculated using a formula that takes inspiration from the Akaike Information Criterion (AIC) for model selection [20]:

$$\mathcal{L}_{\text{mod}} = 2\mathcal{E}_{\text{dec}} + 2\lambda \log\left(|\Xi|\right). \tag{29}$$

Here, $|\Xi|$ denotes the number of active SINDy coefficients, and λ is a parameter that balances the significance of each loss component.

With $\lambda = 0.001$, a value appropriate for the dimensionality of the two metrics, the final combined loss is illustrated in Figure 11. According to this graph, models with 4 to 6

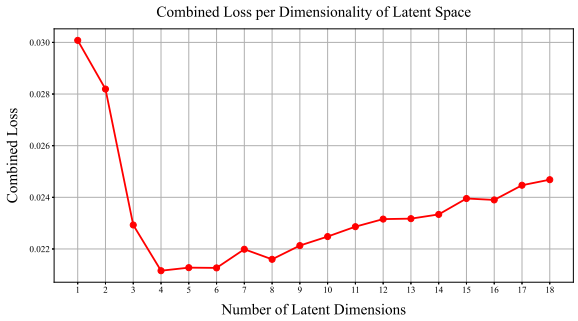

Fig. 11: Plot illustrating the combined loss across varying numbers of latent dimensions. The trend indicates how the model's loss responds to changes in dimensionality.

latent dimensions appear to offer the optimal balance between accuracy and complexity. This aligns with the number of visibly significant dimensions shown in Figure 9. For greater latent dimensionality, the incremental gain in accuracy does not seem to justify the heightened complexity.

It is worth mentioning that the choice between a simpler but less accurate model and a more complex but accurate one may vary depending on the specific application.

C. Results on Hardware Data

Following the approach outlined in Subsection IV-C, besides the standard dataset from the Go1 simulation, a noisy dataset was collected to enable more robust training for real-world scenarios.

A model with four latent dimensions was selected for this evaluation, based on the discussion in Subsection V-B.

The impact of noise on the dataset and its extension into the latent space is depicted in Figure 12. The model effectively

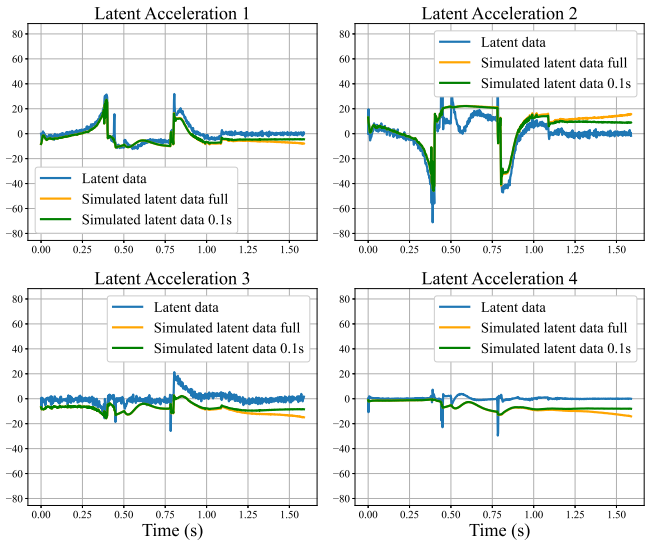

Fig. 12: Latent space accelerations for a quadruped robot's jumping motion with noisy data. The direct encoding of real dimensions is shown in blue, the learned latent accelerations over the full jump in orange, and the learned latent accelerations at 0.1-second intervals in green.

smooths out latent accelerations, demonstrating satisfactory performance on the noisy dataset.

When transitioning to the real hardware, given the limited availability of this data type compared to the simulated one, the previously trained model was maintained, with minimal refinement using hardware data.

Figure 13 illustrates the model's ability to replicate a jump on the actual hardware.

Notably, the model's performance on actual hardware data, especially during the flight and initial landing phases, was not as accurate as with simulated data. This discrepancy is attributed to differences in the real system's behaviour compared to the training data, especially regarding the Body angular Positions.

D. Learning Different Simulated Systems

To assess the algorithm's effectiveness across varied platforms, training and testing were also conducted on data from an A1 quadruped, as outlined in Subsection IV-C.

This scenario featured the robot performing what is termed an asynchronous jump. This jump includes a sequence starting with all legs grounded, transitioning to only the rear legs making contact, and culminating in a flight phase. Notably, the simulation excluded the landing phase, terminating the data

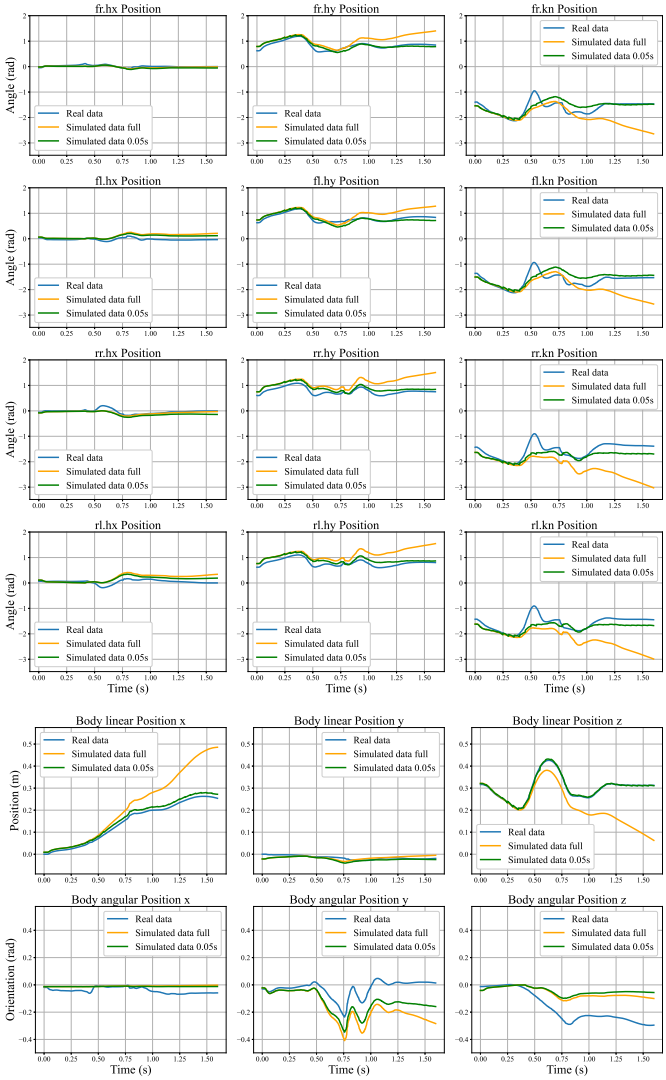

Fig. 13: Comparison of hardware-measured joint angles (top) and body positions (bottom) throughout a quadruped robot's jump. Blue lines indicate actual hardware data, orange lines represent reconstructions from a 4D latent space model over the full jump interval, and green lines depict reconstructions at 0.05-second intervals.

capture just before ground contact. This jump type necessitated a tailored learning process, segmenting the motion into full contact (four feet on the ground), partial contact (two feet on the ground), and flight (no feet on the ground) phases. This approach required an additional model within the hybrid system to address the distinct phases, yet the overarching training and testing procedure mirrored that used for the Go1, with results displayed in Figure 14.

A notable difference is observed in the Body angular Position y between the A1 and the Go1. The asynchronous jump induces a more pronounced angular rotation along the y-axis in the A1's case. Despite the increased significance of this angular dimension for the jump motion, the four-dimensional model successfully captures this dynamic and most joint positions, which exhibit smoother behaviour, easier to track than in the Go1 scenario.

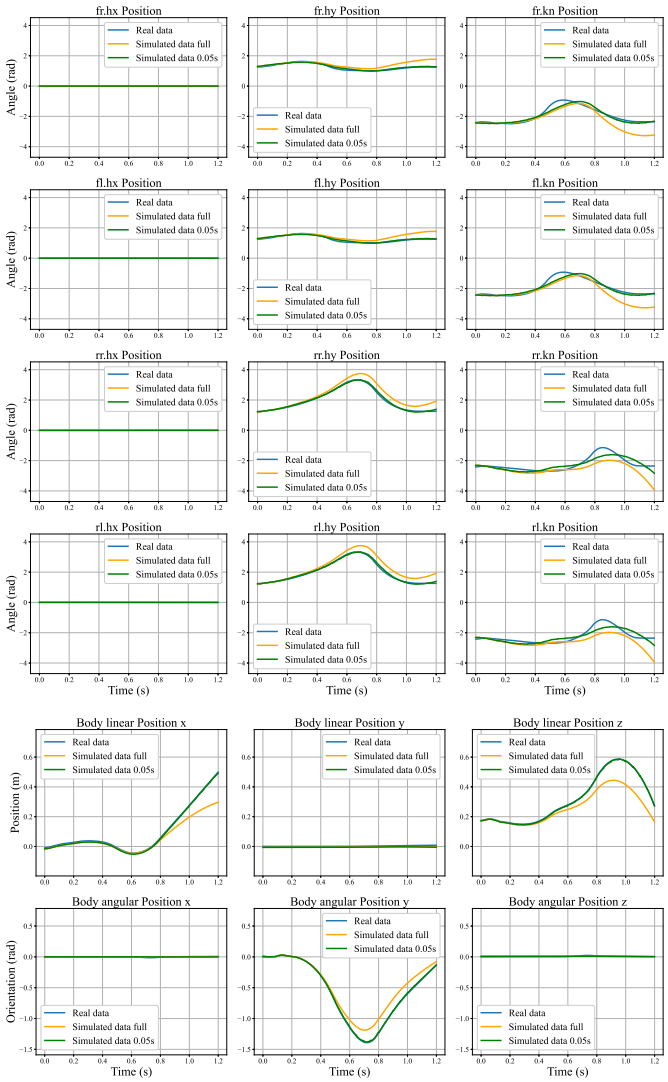

Fig. 14: Comparison of joint angles (top) and body positions (bottom) during an A1 quadruped robot's jump using a 4D latent space model. Blue lines display the actual measured data, orange lines represent full interval simulated reconstructions, and green lines illustrate reconstructions at 0.05-second intervals.

VI. Discussion and Conclusion

The findings from this study demonstrate that the methodology introduced herein is capable of generating reduced-order, interpretable dynamic models for jumping quadruped robots from data. These models surpass traditional template models in reconstructing real dimensions with higher accuracy, and can also be applied to hardware data with satisfactory performances. Furthermore, the robustness of the algorithm was validated across different simulations, proving its effectiveness on different types of quadruped robots provided appropriate phase segmentation and training are applied.

Researchers focusing on quadruped robot control can utilize these models in a manner akin to template models. The control algorithm can be implemented within the latent space model, thereby substantially reducing computational complexity. Consequently, the desired latent inputs u_{lat} can be determined for each timestep, and the actual inputs u to be applied to the robot can be deduced using the inverse transformation $u = W_e^T u_{\text{lat}}$. Similarly, the robot's real state at each timestep can be estimated as $q_{\text{pred}} = W_d z_{\text{pred}} + B_d$. This approach ensures that both the necessary input and the robot's state in real space are ascertainable at every timestep, despite the control operations being performed in the simplified latent space. It is important to highlight, however, that the stability of a control loop in latent space does not automatically guarantee its stability when transformed back to real space. Nonetheless, the encoding's linearity is expected to aid in maintaining stability.

The aspiration of this work is to contribute to the advancement of legged robotics by introducing a methodology for deriving dynamic models of quadruped robots from data that are both efficient and interpretable. The aim is to facilitate an additional step towards making this area of robotics more accessible and beneficial to society at large.

VII. Future Works

As highlighted earlier, this study marks a pioneering effort in symbolic model learning for quadruped robots. Consequently, there exist several opportunities for enhancement that could augment the method's effectiveness and broaden its research applicability.

1) Non-linear Autoencoder: One potential further exploration involves the structure of the autoencoder. As indicated in Subsection IV-B, a linear autoencoder was employed. This choice simplified the training process and facilitated a clearer physical interpretation of the latent space. However, adopting a more complex non-linear structure for the autoencoder could substantially enhance performance, albeit with some loss of clarity in interpreting the latent space.

2) SINDy Library Improvement: The SINDy library, as detailed in Subsection IV-B and Appendix B, was created using very general terms for simplicity and to avoid introducing biases or limitations. A library tailored to the specific application could enhance the accuracy of tracking real trajectories while simultaneously reducing model complexity. Consequently, refining the SINDy library with targeted physical knowledge could markedly benefit the machine learning method described.

3) Backpropagation Through Time: A prospective avenue for enhancing algorithmic performance involves devising methods to accurately track real trajectories over the entire duration of a jump. A potential solution might be the adoption of a technique inspired by Backpropagation Through Time (BPTT) [21], with a detailed explanation available in Appendix D.

4) Application to other quadruped tasks: A forthcoming objective is to broaden the scope of tasks for which the algorithm is applicable. Currently, the algorithm is specifically designed for jumping motions, chosen for their relative simplicity in terms of distinct phases. More intricate movements, like walking, present additional challenges, such as rapidly changing contact modes and a greater number of contact phases, each characterized by fewer data points due to the frequency of change. Adapting the algorithm to accommodate a

broader spectrum of the robot's actions may necessitate refined hyperparameter tuning and potentially some modifications to the algorithm's structure. However, achieving this adaptability would significantly expand the range of applications for the model learning method.

ACKNOWLEDGMENTS

Heartfelt gratitude is extended to all who contributed to the success of this research. Special appreciation is directed to Professor Cosimo Della Santina, whose supervision and profound knowledge were indispensable throughout this study. Equally, Jingyue Liu's support as a co-supervisor, providing timely advice and assistance, was invaluable.

I am thankful to Maximilian Stölzle and all the members of the Cognitive Robotics (CoR) department for their insightful feedback and collaborative efforts that greatly enriched this work. Gratitude is also owed to my colleague, Edoardo Panichi, for his assistance in data collection and the constructive critiques that significantly improved this thesis.

Last but not least, special thanks to Chuong Nguyen for providing the essential code for running the A1 quadruped simulation.

The combined expertise, encouragement, and support from these individuals were crucial to the research and writing process, making this achievement possible.

REFERENCES

[1] R. J. Full and D. E. Koditschek, "Templates and anchors: Neuromechanical hypotheses of legged locomotion on land," *Journal of experimental biology*, 1999.

[2] E. Grinke, C. Tetzlaff, F. Wörgötter, and P. Manoonpong, "Synaptic plasticity in a recurrent neural network for versatile and adaptive behaviors of a walking robot," *Frontiers in neurorobotics*, 2015.

[3] S. Zimmermann, R. Poranne, and S. Coros, "Go fetch!-dynamic grasps using boston dynamics spot with external robotic arm," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021.

[4] J. Wu, J. Wang, and Z. You, "An overview of dynamic parameter identification of robots," *Robotics and computer-integrated manufacturing*, 2010.

[5] K. Chatzilygeroudis and J.-B. Mouret, "Using parameterized black-box priors to scale up model-based policy search for robotics," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018.

[6] G. Pillonetto, A. Chiuso, and G. De Nicolao, "Prediction error identification of linear systems: A nonparametric gaussian regression approach," *Automatica*, 2011.

[7] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, 2017.

[8] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017.

[9] ETH Zurich, *Lecture 6: Dynamics 2 - robot dynamics*, Online, 2017.

[10] L. Righetti, J. Buchli, M. Mistry, and S. Schaal, "Inverse dynamics control of floating-base robots with external constraints: A unified view," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011.

[11] M. Mistry, J. Buchli, and S. Schaal, "Inverse dynamics control of floating base systems using orthogonal decomposition," in *2010 IEEE international conference on robotics and automation*, IEEE, 2010.

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, 2006.

[13] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, 2016.

[14] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, 2019.

[15] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[16] C. Nguyen and Q. Nguyen, "Contact-timing and trajectory optimization for 3d jumping on quadruped robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[17] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010.

[18] A. Hindmarsh and L. Petzold, "Lsoda, ordinary differential equation solver for stiff or non-stiff system," 2005.

[19] K. Green, R. L. Hatton, and J. Hurst, "Planning for the unexpected: Explicitly optimizing motions for ground uncertainty in running," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020.

[20] H. Bozdogan, "Model selection and akaike's information criterion (aic): The general theory and its analytical extensions," *Psychometrika*, 1987.

[21] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, 1990.

APPENDIX A
MATHEMATICAL DERIVATIONS

A. Ground Reaction Force for a Single Leg

The Ground Reaction Force (GRF) for a single leg is calculated as follows:

$$\tau_k^T \Delta q_k = F_k^T \Delta x_k \tag{30}$$

$$\tau_k^T \Delta q_k = F_k^T J_k \Delta q_k \tag{31}$$

$$\tau_k^T = F_k^T J_k \tag{32}$$

$$F_k = J_k^{-T} \tau_k \tag{33}$$

where:

- q_k represents the joint angles of leg k
- x_k denotes the foot position of leg k
- τ_k is the set of joint torques acting on leg k
- F_k is the force exerted at the foot of leg k
- J_k refers to the Jacobian of leg k

B. Latent Space Input

The latent space input is calculated to ensure that the work done in both real and latent spaces is equivalent, i.e., $u^T\dot{x} = u_{\text{lat}}^T\dot{z}$. The derivation is as follows:

$$u^T\dot{x} = u_{\text{lat}}^T\dot{z} \tag{34}$$
$$u^T\dot{x} = u_{\text{lat}}^T W_e\dot{x} \tag{35}$$
$$u^T = u_{\text{lat}}^T W_e \tag{36}$$
$$u = W_e^T u_{\text{lat}} \tag{37}$$
$$u_{\text{lat}} = W_e^{-T} u \tag{38}$$

where:

- x symbolizes the real space coordinates
- z represents the latent space coordinates
- u indicates the real space inputs
- u_{lat} is the set of latent space inputs
- W_e is the encoder weight matrix

APPENDIX B
SINDY LIBRARY GENERATION

In SINDy applications, generating the library is a critical aspect. Ideally, the exact basis functions should be known a priori, so that the training process merely fine-tunes their coefficients. However, often the exact basis functions are not known, and the system's physical properties provide only a general idea of the terms to include.

The primary challenge is to include all necessary basis functions to fully describe the system while avoiding excessive complexity due to an overly large library.

The library for this project is standard and does not include prior knowledge to avoid biasing the training and subsequent discussions. It comprises a constant term, first-order polynomials, and trigonometric functions of both the state positions and velocities, as well as the input terms.

Considering a latent space of dimension n with state variables $z = [z_1, z_2, \ldots, z_n]$ and inputs $u_{\text{lat}} = [u_{\text{lat}1}, u_{\text{lat}2}, \ldots, u_{\text{lat}n}]$, the library includes:

1. The constant term:

$$1$$

2. Linear terms for each state variable:

$$z_1, z_2, \ldots, z_n$$
$$\dot{z}_1, \dot{z}_2, \ldots, \dot{z}_n$$

3. Trigonometric functions, sine and cosine, for each state variable:

$$\sin(z_1), \sin(z_2), \ldots, \sin(z_n)$$
$$\cos(z_1), \cos(z_2), \ldots, \cos(z_n)$$
$$\sin(\dot{z}_1), \sin(\dot{z}_2), \ldots, \sin(\dot{z}_n)$$
$$\cos(\dot{z}_1), \cos(\dot{z}_2), \ldots, \cos(\dot{z}_n)$$

4. The input terms:

$$u_{\text{lat}1}, u_{\text{lat}2}, \ldots, u_{\text{lat}n}$$

For a system with n state variables, the library thus contains $7n + 1$ terms.

When applying the library to the following equation,

$$\ddot{z}_{\text{pred}} = \Theta(z, \dot{z}, u_{\text{lat}})\Xi \tag{39}$$

the set of basis functions is generated for each latent acceleration prediction. The total number of terms, and hence initial SINDy coefficients, becomes $7n^2 + n$.

This library composition is designed to capture linear relationships, constant offsets, and periodic behaviour in system dynamics, commonly found in many physical systems. Larger libraries could capture the dynamics more precisely but risk significantly increasing the model's complexity.

APPENDIX C
LATENT SPACE COMPOSITIONS FOR HIGHER DIMENSIONS

Figure 15 gives an overview of the encoder weights in the cases of a latent space of 3, 8, 13 and 18 dimensions. The purpose of this image is to give overall visual support to see the lack of change of significant dimensions with the increase in latent dimensionality.

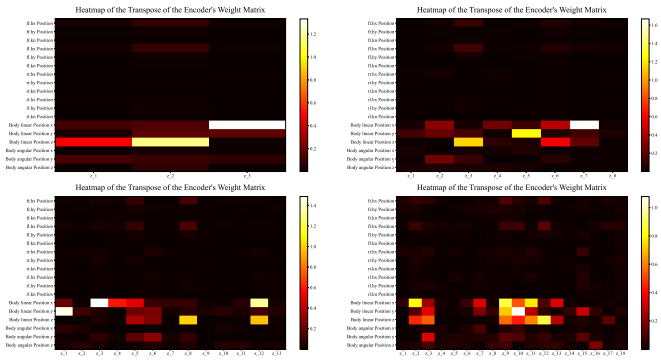

Fig. 15: Heatmaps of the transposed weight matrix for the encoders of four different models. The brighter the colour of the tile, the higher the absolute value of the corresponding weight. In the top left a model with 3 latent dimensions, in the top right a model with 8 latent dimensions, in the bottom left a model with 13 latent dimensions, in the bottom right a model with 18 latent dimensions.

APPENDIX D
BPTT-INSPIRED TECHNIQUE

This paragraph describes a method intended to mitigate the error propagation observed when simulating the learned system over extended time periods.

The core idea is to extend the calculation of the SINDy loss beyond the latent space accelerations $\ddot{z}$, incorporating an additional loss for the predicted latent positions z_{pred}. These positions are derived by simulating the state at each timestep using the current model and projecting m steps into the future, where m denotes the prediction window size.

For clarity, the notation $\ddot{z}_{\text{pred}}{}_i^j$ refers to the latent acceleration predicted from the real latent state at timestep i, extrapolated

j timesteps into the future. Integrating this acceleration twice yields the latent position ${z_{\text{pred}}}_i^j$.

Given a total time of n timesteps and a prediction window of m timesteps, the matrix of predictions P is constructed as follows:

$$P = \begin{bmatrix} {z_{\text{pred}}}_0^1 & z_1 & z_1 & \cdots & z_1 \\ {z_{\text{pred}}}_1^1 & {z_{\text{pred}}}_0^2 & z_2 & \cdots & z_2 \\ {z_{\text{pred}}}_2^1 & {z_{\text{pred}}}_1^2 & {z_{\text{pred}}}_0^3 & \cdots & z_3 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ {z_{\text{pred}}}_{n-1}^1 & {z_{\text{pred}}}_{n-2}^2 & {z_{\text{pred}}}_{n-3}^3 & \cdots & {z_{\text{pred}}}_{n-m}^m \end{bmatrix} \tag{40}$$

In this matrix, each column corresponds to a future timestep, and each row to all predicted positions for a given timestep.

Denoting the element of P at the i-th row and j-th column as p_i^j, the BPTT loss is calculated by summing the squared differences between each element of a row from the real value of z for the corresponding timestep, and then aggregating these errors across all timesteps:

$$\mathcal{L}_{\text{BPTT}} = \sum_{i=1}^{n-1} \sum_{j=1}^{m} \left(z_i - p_i^j \right)^2 \tag{41}$$

Incorporating this term into the combined loss function, with suitable weighting, aids in curbing the cumulative errors resulting from the temporal propagation of the model.

An important consideration in this process is the choice of integration method to derive ${z_{\text{pred}}}_i^j$ from ${\ddot{z}_{\text{pred}}}_i^j$. Employing rapid integration methods, such as the Euler method, can reduce the algorithm's time consumption, albeit at a potential cost to accuracy. Conversely, higher-order Runge-Kutta methods yield more precise results but significantly increase time complexity. A similar trade-off exists concerning the size of the prediction window.

If necessary, a decaying weighting scheme can be applied to the prediction matrix to diminish the impact of distant future predictions. This is achieved by multiplying progressively smaller weights to the columns as they extend further to the right in the matrix.