LEARNING KOOPMAN EIGENFUNCTIONS FOR NONLINEAR SYSTEMS USING DEEP AUTOENCODERS

Sriram Krishnamoorthy
Department of Mechanical Engineering
Clemson University
Clemson, SC 29630
sriramk@clemson.edu

Dakota Rufino
Department of Mechanical Engineering
Clemson University
Clemson, SC 29630
drufino@clemson.edu

Andrew Zheng
Department of Mechanical Engineering
Clemson University
Clemson, SC 29630
azheng@clemson.edu

ABSTRACT

Nonlinear dynamical systems are notoriously difficult to analyze due to the existence of multiple equilibrium points and limit cycles. These difficulties make it infeasible to design controllers for nonlinear systems using existing well established linear systems approach. In this paper, we use Koopman operator theory to lift the nonlinear dynamics to a higher dimensional space using an autoencoder. We added custom losses during the training process to enforce the learned Koopman operator to propagate the dynamics linearly. We demonstrate the efficacy of our approach on quadruped locomotion. We present experimental and simulation results along with spectral analysis of the learned Koopman operator. The learned Koopman eigenfunctions are able to predict periodic orbits and distinguish between the swing and stance phase.

Keywords Quadruped locomotion · Koopman operator · Autoencoder

1 Introduction

The study of dynamical systems and control has a wide range of applications in the field of engineering such as robotics, aerospace, automotive etc. They can be very useful to predict the evolution of future states $\mathbf{x}_{t+1}$ of the system such as position, velocity etc., over time based on a functional mapping of its current states $\mathbf{x}$ and control actions u given by $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ (in discrete time setting). Obtaining a solution for such systems is important to design suitable controllers for stabilization or trajectory tracking applications. Most engineering systems are nonlinear in nature and are defined by a nonlinear differential equations. For simple nonlinear systems, the solution can be obtained through direct integration. However, for most real life applications, this method is often complicated and highly inefficient. Analysis and control of nonlinear dynamical systems is still on active area of research.

One of the common approaches to controlling nonlinear systems is the application of Hartman Grobman's Theorem. It provides an effective way to linearize about a hyperbolic fixed point to arrive at an approximate linear model of the dynamical system, so long as the system does not diverge too far from that fixed point. Powerful tools and techniques from linear system analysis [Hespanha, 2018], [Nise, 2020] can be employed to develop standard linear control algorithms such as Proportional-Integral-Derivative (PID) controllers, Linear Quadratic Regulators (LQR), and Model Predictive Control (MPC). The latter two make use of optimal control theory to assign a cost function that sets the dynamical system to the desired state. However, such methods tend to perform poorly for complex systems where these controllers still face the underlying issue of a simplified linear model, limited control around the local equilibrium points. To address these issues, tremendous efforts have been made in the recent past to develop nonlinear control

strategies that are global in nature. However, they tend to be very complicated and highly dependent on the type and structure of the dynamics. Techniques such as feedback linearization, passivity and Lyapunov based control [Khalil, 1996] can be applied to a class of nonlinear systems which follow certain restrictive assumptions. These controllers are typically sub optimal and it requires a lot of domain knowledge to develop a nonlinear controller that performs better than a linear controller.

Although linear controllers can meet a wide range of performance metrics ensuring stability and robustness criteria, using linear control techniques for stabilization and tracking of nonlinear systems is inherently limiting and leads to sub optimal controllers. Nonlinear systems have fundamentally different properties with varying degrees of complexities which linear controllers +cannot handle [Khalil, 1996], [Vidyasagar, 2002]. Firstly, nonlinear systems have multiple isolated fixed points, limit cycles, bifurcations, etc. The linearized model is only valid around a small region around the fixed point of the system. To extend this linear design technique to deal with multiple fixed points, gain scheduling is used where the system is linearized over a set of desired fixed points and a set of linear controllers are designed around them. The control objective is then to determine how to transition between fixed points by designing a suitable gain for each controller. However, for complex nonlinear systems with a wide range of fixed points, it is not efficient to design multiple controllers for each fixed point of interest. Further, it may require a lot of energy to switch between different fixed points. Secondly, many systems in the robotics industry have high degrees of freedom, making modeling the system extremely difficult and virtually impossible. Even if some complex systems have been modeled, these equations of motion are subject to assumptions and do not fully capture the true system dynamics. Efforts to control these systems are mitigated by these inherent inaccuracies. Therefore, data driven methods are being utilized to characterize and control highly nonlinear systems.

In the current research area of Dynamics and Controls, there is a surge in data driven modeling and control of complex large scale dynamical systems. With advancement in technological hardware, the challenges that were previously faced by data driven policies due to computation time are becoming less of an issue. One area of data driven modeling of nonlinear dynamics makes use of the Koopman operator. Given the states of your original nonlinear system, a basis function is used to map the states to an infinite dimension where the dynamics of the system evolve linearly in time. The Koopman operator acts on these lifted states, known as observables, by propagating them forward linearly in time

In this project, we analyze quadruped locomotion using Koopman operator theory. The analysis is performed on Champ, a quadruped which was developed based on MIT cheetah [Bledt et al., 2018] and the Clemson Tigger, a quadruped developed by our lab, Distributed Intelligence Robot Autonomy (DIRA). We develop an auto-encoder based approach to lift the state space of the robot to a higher dimensional space (called the observable space) where the dynamics are linear. The Koopman operator is learned based on the linear evolution of observables in the lifted space and the eigenfunctions are used for analysis of the quadruped's system dynamics.

2 Preliminaries

Notation: $\mathbb{R}^n$ denotes the n dimensional Euclidean space and $\mathbb{R}^n_{\geqslant 0}$ is the positive orthant. Given $\mathbf{X} \subseteq \mathbb{R}^{\mathbf{n}}$ and $\mathbf{Y} \subseteq \mathbb{R}^{\mathbf{m}}$, let $\mathcal{L}_1(\mathbf{X}, \mathbf{Y})$, $\mathcal{L}_\infty(\mathbf{X}, \mathbf{Y})$, and $\mathcal{C}^k(\mathbf{X}, \mathbf{Y})$ denote the space of all real valued integrable functions, essentially bounded functions, and space of k times continuously differentiable functions mapping from $\mathbf{X}$ to $\mathbf{Y}$ respectively. $\mathbf{s_t}(\mathbf{x}, \mathbf{u})$ denotes the solution of dynamical system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ starting from initial condition $\mathbf{x_0}$.

2.1 Quadruped Dynamics

Several methods have been developed to model the dynamics of quadruped locomotion. The quadruped system is underactuated in nature. The input torques are applied at the joints of the legs which can directly control the joint angles. However, there is no control input which acts directly on the robot's body to control the center of mass of the system. Assuming the robot is rigid, the most used approach is the use of Euler-Lagrange equations. It is a first principles based approach where the potential energy, kinetic energy, and constraint forces for each link in the robot is calculated. By defining the Lagrangian operator as the difference of the kinetic and potential energies to arrive at the robot's equation of motion as a set of second order differential equations based on generalized coordinates q as shown in equation 1.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{B}\mathbf{u} + \mathbf{J}(\mathbf{q})^\top \lambda \tag{1}$$

Here M is the inertia matrix, h captures the effect of centrifugal, Coriolis and gravity terms, S^T is a selection matrix that determines how the input torques τ interact with each joint and J is the Jacobian matrix which maps the foot forces f to generalized forces. However, this approach can become very tedious with increasing joints and including the presence of non-conservative forces such as friction which makes it nonlinear in nature. The resulting model is very bulky with large number of states. Further, there is no explicit coordination between high level control tasks, such as walking in a particular gait, and the low-level joint controller.

By representing the change of momentum through a frame fixed at the center of mass (CoM) of the robot, a simplified equation of motion can be achieved. Such a representation is also known as the centroidal dynamics [Orin et al., 2013]. Here, $\mathbf{A}$ maps the momentum of each element from a stationary frame to the CoM frame of reference, $\mathbf{r}$ is the CoM position and $\mathbf{p_i}$ is the location of each foot. This results in a 6 degree of freedom system of equations which relate the linear and angular momentum of each link to the sum of forces and moments acting on the link.

$$\mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{A}}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} m\mathbf{g} + \sum_{\mathbf{i=1}}^{\mathbf{n_1}} \mathbf{f_i} \\ \sum_{i=1}^{n_1} \mathbf{f_i} \times (\mathbf{r}(\mathbf{q}) - \mathbf{p_i}(\mathbf{q})) \end{bmatrix} \tag{2}$$

However, the solution to this equation is not unique and there exists multiple physically feasible states for the same desired configuration. Further, the inclusion of joint angles $\mathbf{q}$ makes the model nonlinear and not very suitable for control design. This model can be simplified further by neglecting the momentum produced by joint velocities and assumption the full body inertia remains constant. Using these assumptions, the model can be decoupled into linear and angular parts as shown in equation 3,4

$$m\ddot{\mathbf{r}} = m\mathbf{g} + \sum_{\mathbf{i=1}}^{\mathbf{n_i}} \mathbf{f_i} \tag{3}$$

$$\mathbf{I}(\theta)\omega + \omega \times \mathbf{I}(\theta)\omega = \sum_{\mathbf{i=1}}^{\mathbf{n_i}} \mathbf{f_i} \times (\mathbf{r} - \mathbf{p_i}) \tag{4}$$

This approach is referred as the Single Rigid Body Dynamics (SRBD) model [Bledt et al., 2017], [Winkler et al., 2018]. Here too, the model produces a system of 6 differential equations but, it is expressed using the total mass m and the total inertia I of the robot, is the base orientation and is the combined angular velocity of the robot. However, the assumptions made here can be very restrictive. The joint momentum can be neglected only when the robots have a negligible leg mass while the constant inertia means that the robots can only operate very close to their nominal configuration. There is still nonlinearities introduced by the cross product and the presence of non-conservative forces at the constraints.

To achieve a fully linearized model, further assumptions are made such as restricting the height of the robot to remain constant with the base having zero angular velocity and angular acceleration and the ground is planar. This leads to the linearized inverted pendulum model (LIPM) as shown

$$m\ddot{\mathbf{r}}_x = \sum_{i=1}^{n_i} \mathbf{f_{i,x}} = \frac{\mathbf{mg}}{\mathbf{h}}(\mathbf{r_x} - \mathbf{p_{c,x}}) \tag{5}$$

Here, $\mathbf{x}$ is the horizontal motion, $\mathbf{h}$ is the constant height of CoM of the robot, $\mathbf{p}(\mathbf{c}, \mathbf{x})$ is the x position of the Center of Pressure (CoP) of the robot. The CoP position can be manipulated to a desired location by suitable contact forces and foot positions. Hence this model relates the change in the horizontal acceleration of the robot as a result of contact forces though CoP positions. Since this results in a linear model, it is highly suitable for current control design approaches such as LQR and PID. This approach has been used in [Winkler et al., 2015], [Kajita et al., 2003], [Kalakrishnan et al., 2010], [Bellicoso et al., 2017] . However, it is very restrictive since it does not account for changes in the height of the robot or uneven terrain. Hence, they don't generalize well in real world applications.

In order to account for impact dynamics during different gait patterns, a hybrid model is required which can switch between swing and stance phase effectively. In recent works [Ma et al., 2019], [Ma and Ames, 2020] propose a nonlinear hybrid model for quadruped location. However, the control approach is to iteratively solve the complex optimization problem to get the desired joint angles as the solution to the nonlinear hybrid dynamics. This approach can become computationally expensive leading to local minima and suboptimal controllers.

2.2 Koopman Operator Theory

Consider a dynamical system

$$\dot{\mathbf{x}} = \mathbf{f}(x, u), \quad \mathbf{x} \in \mathbf{X} \subseteq \mathbb{R}^{\mathbf{n}}, \mathbf{u} \in \mathbf{U} \subseteq \mathbb{R}^{\mathbf{m}} \tag{6}$$

where the vector field is assumed to be $\mathbf{f}(\mathbf{x}) \in \mathcal{C}^{\mathbf{1}}(\mathbf{X}, \mathbb{R}^{\mathbf{n}})$. The nonlinear dynamics in the state space can be lifted to infinite dimension space of functions using the Koopman operator. The Koopman operator $\mathbb{U}_t : \mathcal{L}_\infty(\mathbf{X}) \rightarrow \mathcal{L}_\infty(\mathbf{X})$ for dynamical system is an infinite-dimensional linear operator (6) is defined as

$$[\mathbb{U}_t\varphi](\mathbf{x}, \mathbf{u}) = \varphi(\mathbf{s_t}(\mathbf{x}, \mathbf{u})), \quad \varphi \in \mathcal{L}_\infty. \tag{7}$$

The infinitesimal generator for the Koopman operator

$$\lim_{t\to 0} \frac{\mathbb{U}_t\varphi - \varphi}{t} = (\mathbf{x}, \mathbf{u}) \cdot \nabla\varphi(\mathbf{x}, \mathbf{u}) =: \mathcal{K}_{\mathbf{f}}\varphi. \tag{8}$$

Since $\mathbb{U}_t$ is semi-group with generator $\mathcal{K}$ it satisfies

$$\frac{d}{dt}\mathbb{U}_t\varphi = \mathcal{K}_t\varphi. \tag{9}$$

For a dynamical system, measurement data can be discretely sampled. Therefore, the system can be modeled as a discrete time dynamical system in the form

$$x_{k+1} = \mathbf{F}(x_k) \tag{10}$$

where **F** is the nonlinear *flow map* that propagates the states $\mathbf{x} \in \mathbb{R}^n$ forward in time [Brunton et al., 2021]. The goal is to model the system dynamics as a linearized system through a transformation of coordinates defined as z = φ(x) such that the system is simplified by the following formulation

$$z_{k+1} = \mathbf{K}z_k \tag{11}$$

where the matrix **K** is the finite approximation of the infinite time matrix, $\mathbb{U}$. The Koopman operator $\mathbb{U}$ is defined as a discrete infinite dimensional operator that advances measure functions g: X$\to \mathbb{C}$ forward in time represented as follows:

$$[\mathbb{U}g](x_k) := g \circ \mathbf{F}(x) = g(x_{k+1}) \tag{12}$$

For an eigenfunction φ(x) corresponding to the an eigenvalue λ where φ: X$\to \mathbb{C}$, the Koopman operator would be the following

$$\mathcal{K}(\varphi(x_k)) = \lambda\varphi(x_k) = \varphi(x_{k+1}) \tag{13}$$

where this formulation gives more structure to ensure linearity. Thus, to get a matrix representation of $\mathbb{U}$, the matrix **K** is represented by approximating the span of the eigenfunctions, thus finite dimension. More recently, with the increase in computational capacity, a large amount of effort have been invested in approximating the spectral decomposition of $\mathbb{U}$ from measurement data.

2.3 Finite Dimensional Approximation of the Koopman Operator from Data

Consider a continuous-time dynamical system with collected data as snapshots in the form

$$\begin{aligned} \mathbf{X} &= [x_1, x_2, ..., x_{M-1}] \\ \mathbf{Y} &= [x_2, x_3, ..., x_M] \end{aligned} \tag{14}$$

Where $x_i, y_i \in X$. Let $\mathcal{D} = \{\psi_1(x), \psi_2(x), ..., \psi_N(x)\}$ be the set of linearly independent dictionary functions used to construct the states in the lifted dimension, called observables. The dynamics in the lifted space, are propagated forward linearly in time by **K**. The approximated finite dimensional representation of the Koopman operator, **K** can be found as such

$$\mathbf{K} = \mathbf{\Psi}(\mathbf{Y})\mathbf{\Psi}(\mathbf{X})^{\dagger} \tag{15}$$

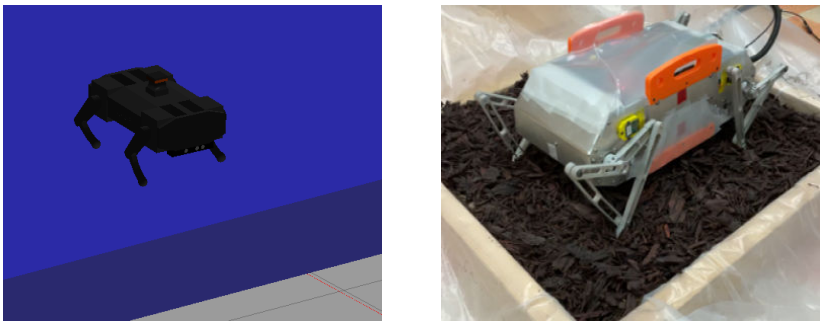
where the eigenfunction of the Koopman operator is defined by the left eigenvectors of the operator

$$\varphi(\mathbf{x}) = \mathbf{w}^{\mathbf{T}}\mathbf{\Psi}(\mathbf{X}) \tag{16}$$

However, as mentioned previously, this requires domain knowledge and expertise to curate the system to be of the form shown in 2.2. With neural networks being more frequently used for data driven approaches in engineering applications, the approach we use in this paper uses the computation power of neural networks and the structure preserving mapping of autoencoders to define a physics informed Koopman Autoencoder neural network architecture that maps the state space dynamics to the lifted space and vice versa while learning the Koopman operator.

3 Data generation and network architecture

Without a standard dataset to work with for quadruped locomotion, we conducted the following experiments on both simulation and experiment to analyze the Koopman eigenfunctions of the system as shown in 1. The following sections below will cover our data collection procedure, simulation and experiment, and the network architecture used to study the eigenfunction of the quadrupeds.



(a) (b)

Figure 1: Data collection a) Experiment b) Simulation

3.1 Simulation data

The quadruped's locomotion, based on MIT cheetah, was simulated in Gazebo under the default physics engine. Simulation consists of commanding the quadruped to trot at a desired velocity under different stiffness and damping ground parameters. Joint states of the quadruped and the corresponding force of impact were collected during the locomotion of the robot. In addition, the dataset includes different gait motion under these ground parameters shown in Table 1 where rigid ground is the default gazebo ground parameters. Currently, there are approximately 800,000 points within the dataset. To our knowledge, this is one of the first dataset that includes quadruped state locomotion for different ground parameters and we hope to expand and make the dataset available within the near future [1].

Table 1: Simulation Data Parameters

	Ground Parameters		
Gait Motion	Stiffness (k_p)	Damping (k_d)	Velocity (m/s)
Forward	100×10^{10}	0	0.1
Left	100×10^{8}	100	0.2
Right	100×10^{6}	250	0.3
Backwards	Rigid Ground	100	0.4+

3.2 Experiment data

The mechanical structure of the quadruped robot is based on the Stanford Doggo [Kau et al., 2019]. The body of 20 cm wide and 13 cm in height with a mass of 6 kg. The legs of the robot are designed as a symmetric five-bar linkage with a length of 9 cm and 16 cm for the upper and lower links respectively. The actuation of each leg is driven by coaxial drive assemblies which involves two motors independently controlling the concentric leg shafts. Each drive assembly is connected to a drive belt of ratio 3:1 that transfers the power from the motors to the independent coaxial drive shafts.

An off-board computer was used to send commands to the Teensy 3.5, the on-board microcontroller, via USB serial communication. The joint states and control inputs were sampled at 35 Hz through real time ROS messages using an XBee wireless module. Odrive 3.5 motor controllers are used to recieve leg position commands and actuates the motors accordingly. Low level control is achieved through a PD controller at 100 Hz to execute the desired gait trajectories.

For each experiment, the robot was set on different terrain such as rigid ground, sand, pebbles on sand, soil, and mulch on soil as shown in Figure 1. We execute a static "dance" gait where the robot flexes up and down with the foot in full contact with the ground. This results in a range of contact force responses acting on the robot foot depending on the terrain properties. The states for the robot are θ, which is the angle of the foot with respect to the vertical, γ, which is the leg separation angle, and then their derivatives. Therefore the states are formulated as $\mathbf{X} = [\theta, \gamma, \dot{\theta}, \dot{\gamma}]^T$. Each experiment generated roughly 1500 snapshots of data points.

[1] https://github.com/sriram-2502/Deep_Koopman_AutoEncoder

3.3 Network Architecture

3.3.1 Encoder

The data snapshots in the form of $\mathbf{X}$ is given as input to the autoencoder. Each layer of the encoder has the trainable weights (W_l, b_l) where l is the layer index. Each layer performs transforms its inputs as $\sigma(W_l^T\mathbf{x} + b_l)$ where $\sigma(.)$ is the activation of each layer. The encoder layers has widths w_l such that $w_{l+1} > w_l$. The architecture of the layer is constructed using a custom keras layer using subclassing. The output of the encoder takes the inputs $\mathbf{X}$ to the lifted space $\mathbf{Z}$ (observable space). Finally, the layer class was wrapped into a custom keras model which will be used in the autoencoder model

3.3.2 Koopman layer

The Koopman layer is implemented as a pass-through layer with a custom loss function within the layer. The layer takes the observables $\mathbf{Z}$ as inputs with the Koopman matrix $\mathbf{K}$ as a trainable weight. The layer performs a the following computations. A new tensor is constructed to represent the one time step prediction obtained using $\mathbf{K}$ using equation 17

$$\hat{\mathbf{Z}}_\mathbf{2} = \mathbf{K}\mathbf{Z}_\mathbf{1} \tag{17}$$

Next, the layer inputs $\mathbf{Z}$ is time shifted to get $\mathbf{Z}_1$ and $\mathbf{Z}_2$. Further, the initial condition from each trajectory in $\mathbf{X}$ is extracted as $\mathbf{z}_0$ using tensor slices. A new tensor $\mathbf{Z}_{linear}$ is constructed in order to learn $\mathbf{K}$ as a dynamical system which ensures linearity as shown in equation 18

$$\mathbf{Z}_\mathbf{linear} = \begin{bmatrix}\mathbf{z_0} & \mathbf{K}\mathbf{z_0} & \mathbf{K}^2\mathbf{z_0} & \dots & \mathbf{K}^m\mathbf{z_0}\end{bmatrix} \tag{18}$$

where m is the number of snapshots in each trajectory. Note that $\mathbf{Z}_{linear}$ is used to predict all of the m snapshots in the trajectory starting with the initial condition $\mathbf{z}_0$. The linearity loss is implemented within the layer as shown in equation 19

$$\mathcal{L}_{linear1} = ||\mathbf{Z}_2 - \mathbf{Z}_{linear}||_{MSE} \tag{19}$$

Finally the Koopman layer returns $\mathbf{Z}$, $\hat{\mathbf{Z}}_\mathbf{2}$ and $\mathbf{Z}_{linear}$ as the layer outputs along with $\mathbf{L}_{linear}$ as its layer loss. Like the encoder, the Koopman layer is also wrapped into a custom keras model class to be used in the autoencoder model.

3.3.3 Decoder

Similar to the encoder, the decoder layer was also implemented using a custom keras layer. Each decoder layer takes three inputs $\mathbf{Z}$, $\hat{\mathbf{Z}}_\mathbf{2}$, $\mathbf{Z}_{linear}$ and applies $\sigma(W_l^T\mathbf{z} + b_l)$ to each inputs (z represents inputs). The encoder layers has widths w_l such that $w_{l+1} < w_l$. The output layer width is constrained to be the dimension of the inputs. The decoder outputs the reconstructed version of the inputs from the observable space back to the state space as $\hat{\mathbf{X}}$, $\hat{\mathbf{X}}_\mathbf{2}$ and $\hat{\mathbf{X}}_\mathbf{linear}$ respectively. Finally, the decoder layers are wrapped into a custom keras model to be used in the autoecoder model. $\hat{\mathbf{X}}$ is used for reconstruction loss, $\hat{\mathbf{X}}_\mathbf{2}$ is used for one time step prediction loss and $\hat{\mathbf{X}}_\mathbf{linear}$ is used for linear prediction loss over the entire trajectory.

3.3.4 Autoencoder Model

The autoencoder model is developed as a custorm keras model via subclassing using the encoder, Koopman and decoder models respectively. This style of architecutre design allows for modularity and hence can be wrapped around a hyperparamter tuning class from keras tuner to optimize over widths, layers, activations etc. Finally, wrapping everything into a keras model will benefit from several built in methods such as the save and load model methods. The overall architecture is illustrated in Figure 2. The parameters of the network are listed in table 2

3.3.5 Custom losses

The loss functions are defined such that the autoencoder model learns a Koopman matrix $\mathbf{K}$ which represents a dynamical system. To achieve this, the linearity losses are added along with the traditional prediction and reconstruction loss. Since $\mathbf{K}$ is high dimensional, a small error in the elements of $\mathbf{K}$ can lead to huge prediction errors in $\hat{\mathbf{X}}_\mathbf{2}$ and $\hat{\mathbf{X}}_\mathbf{linear}$. Hence a $\mathcal{L}_2$ regularization loss is added. Further, to penalize the data points with highest error (to prevent over-fitting), $\mathcal{L}_\infty$ loss is added. Each of the loss function is weighted by $\alpha_1 = 0.1$, $\alpha_2 = 10^{-7}$ and $\alpha_3 = 10^{-15}$ respectively. The explicit loss functions are defined as follows

Table 2: Parameters for each layer of the network

	units	activation
Encoder		
Layer 1	16	tanh
Layer 2	32	tanh
Layer 3	64	tanh
Layer 4	64	tanh
Layer 5	16	tanh
Layer 6	16	tanh
Layer 7	16	tanh
Koopman Layer	16	none
Decoder		
Layer 1	64	tanh
Layer 2	64	tanh
Layer 3	32	tanh
Layer 4	64	tanh
Layer 5	2	tanh

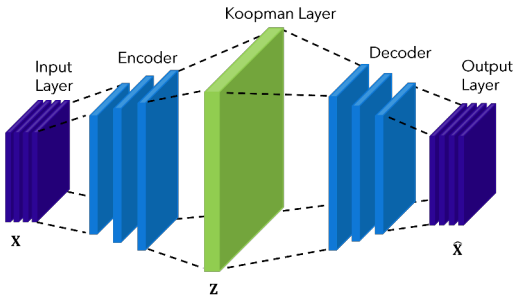


Figure 2: Deep Koopman Autoencoder architecture

$$\mathcal{L}_{reconstruction} = ||\mathbf{X} - \hat{\mathbf{X}}||_{\mathbf{MSE}} \tag{20}$$

$$\mathcal{L}_{prediction} = ||\mathbf{X_2} - \hat{\mathbf{X}}_\mathbf{2}||_{\mathbf{MSE}} \tag{21}$$

$$\mathcal{L}_{linear1} = ||\mathbf{Z}_2 - \mathbf{Z}_{linear}||_{MSE} \tag{22}$$

$$\mathcal{L}_{linear2} = ||\mathbf{X_2} - \hat{\mathbf{X}}_{\mathbf{linear}}||_{\mathbf{MSE}} \tag{23}$$

$$\mathcal{L}_{\mathbf{W}} = ||W_l|| \tag{24}$$

$$\mathcal{L}_{\infty} = ||\mathbf{X} - \hat{\mathbf{X}}||_{\infty} + ||\mathbf{X_2} - \hat{\mathbf{X}}_\mathbf{2}||_{\infty} \tag{25}$$

$$\mathcal{L}_{total} = \alpha_1(\mathcal{L}_{reconstruction} + \mathcal{L}_{prediction}) + (\mathcal{L}_{linear1} + \mathcal{L}_{linear2}) + \alpha_2\mathcal{L}_{\mathbf{W}} + \alpha_3\mathcal{L}_{\infty} \tag{26}$$

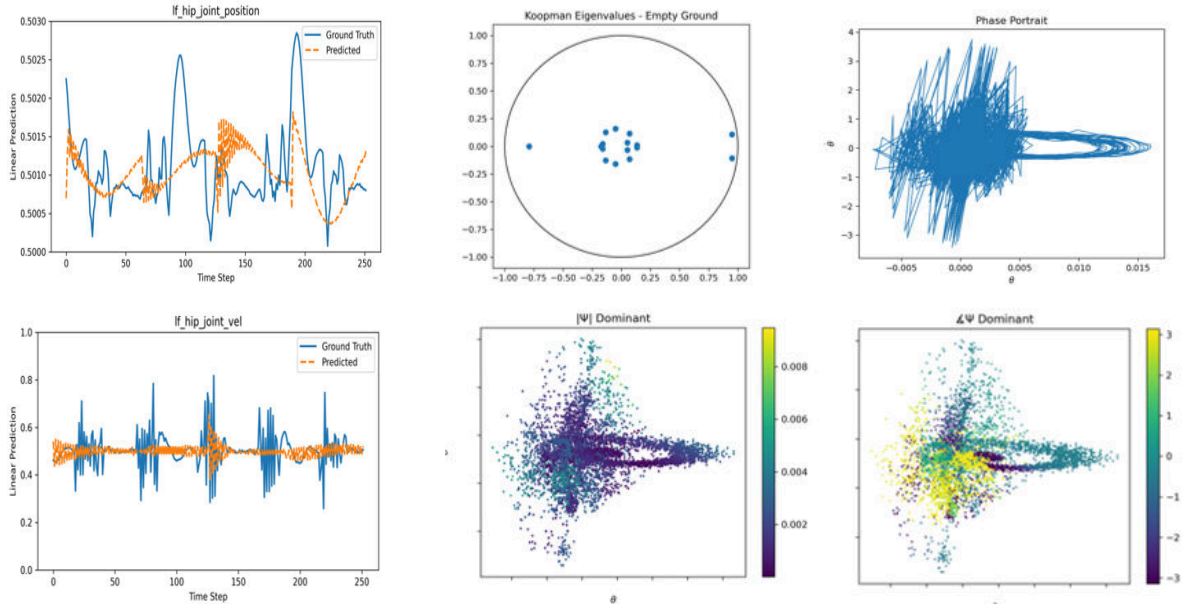
Further information of the parameterization of these loss functions can be referenced [Lusch et al., 2018]

4 Results and Discussion

4.1 Simulation Results

The Koopman Autoencoder model was trained on the quadruped's trot motion simulated on the Gazebo platform. Joint states and impact forces were collected using installed sensors on different ground parameters. As shown in Figure 3 the Koopman autoencoder model is not able to predict the trotting trajectory of the hybrid system. However, although prediction is poor, much of the results calculated from the Koopman operator can still be useful in the context of system analysis for quadruped locomotion. The Koopman operator for empty ground learns a stable operator, although not quite cyclic. This is supported by the phase portrait of champ's trotting motion. The phase portrait can be divided into stance

and swing phase where the swing phase is seen between $\theta = 0.005^\circ$ and $\theta = 0.0015^\circ$ and the stance phase can be seen to the left of the swing phase, evident of a hybrid system. This is furthermore supported by the eigenfunction plots where the cyclic motion defines the zero level sets of the eigenfunction, separating the stable and unstable manifolds. Note, the eigenfunction plots are in high dimensional space, more specifically 16 dimensions with our given network architecture.



(a) (b)

Figure 3: Simulation Results with Koopman Autoencoder model a) Linear Prediction Results b) Phases Portrait with Koopman Eigenfunctions

4.2 Experimental Results

The auto encoder network was trained on experimental data collected from a quadruped robot developed at Clemson DIRA lab. The gait pattern was selected based on a predefined library and the robot was set to "dance" on rigid ground and off road terrain like mulch and pebbles using a suitable sized test bed corresponding to each terrain type. The auto encoder was able to learn a stable Koopman operator for all three different terrain types with reasonable prediction results. For the rigid ground experiment (see Figure 4), reconstruction of the original states by the decoder is fairly accurate with about 0.05% error. The one time step prediction of θ seem to overestimate the ground truth while the full trajectory linear prediction tends to perform better with a conservative estimate. Note that the full trajectory prediction takes only the initial condition of the trajectory and is able to predict all time steps till the end the trajectory based on linearity. For γ, since the dynamics are very small, both the one time step prediction and full trajectory prediction are reasonably good. The prediction results for the angular velocities $\dot{\theta}$ and $\dot{\gamma}$ tend to have a smoothing effect compared to the ground truth. This could be due to the fact that the velocity measurement are generally noisy and have sharp abrupt changes due to the hybrid nature of the quadruped dynamics. Extending the loss functions of the auto encoder network to capture such hybrid dynamics will be investigated in future works.

For off road terrain like mulch-soil and pebbles, we observe a similar pattern with very accurate reconstruction. The one time step prediction and full trajectory linear prediction has an error well under 0.5% and 5% error respectively for both mulch-soil and pebbles terrain (see Figure 5 and 6). The phase portraits of the predicted systems for different ground parameters are listed in Figure 7. Although the errors in reconstruction and predictions are small, the auto encoder is still not able to recover the original phase portrait accurately. For linear predictions, the phase portrait of mulch and pebbles tend to exhibit well defined periodic orbits compared to rigid ground. This could be attributed to the damping effect induced by setting the robot on mulch-soil and pebble-sand terrains. Such off road terrain tend of have compliance and hence leads to a more prolonged contact force compared to rigid ground where the contact forces are instantaneous and impulsive in nature.

Spectral properties of the Koopman operator for each of the terrain is analyzed. From Figure 8, all eigenvalues of the Koopman operator lie within the unit circle which indicates that the learned operator is stable (for discrete time case). For the dance gait, the eigenvalue of one indicates the presence of a periodic orbit. The corresponding eigenfunctions can be interpreted as an invariant of motion. Such invariant sets can be used for designing passivity based energy shaping control. For the trot gait, the zero level sets of the eigenfunctions can be used to distinguish different phases present in the gait such as swing and stance phase. Further, since the learned Koopman operator is linear in the latent space, it can be used to construct well established controllers such as MPC and LQR with strict theoretical and performance guarantees.

5 Conclusions

In this project, we apply the Koopman operator theory to study the dynamics of a quadruped robot on off road and rigid terrain. Data is collected from both simulation and experiment and is used as the input to a deep auto encoder. The encoder "lifts" the non linear states to a higher dimensional space where the dynamics evolve linearly. The Koopman operator which governs this linear dynamics is learned during the training process. The decoder reconstructs the higher dimension space back to state space. Based on this architecture, the prediction error for the full trajectory is well under 5% for experimental data and 10% in simulation. The predicted phase portraits exhibits well defined periodic orbits for deformable terrain compared to rigid ground (as indicated by the dominant eigenvalues of the learned Koopman operator). For the dance gait, the dominant eigenfunctions corresponding to the period orbit can be used as an invariant measure to design energy based control. For the trotting gait, the dominant eigenfunctions can be used to distinguish different phases such as swing and stance phase. Future works will implement additional terms to the loss functions such that it enforces the learning of true eigenfunctions of the lifted system dynamics. Future works will also explore control design based on these new findings reported in this project.

References

Joao P Hespanha. Linear systems theory. In *Linear Systems Theory*. Princeton university press, 2018.

Norman S Nise. *Control systems engineering*. John Wiley & Sons, 2020.

Hassan K Khalil. *Nonlinear Systems*. Prentice Hall, 1996.

Mathukumalli Vidyasagar. *Nonlinear systems analysis*. SIAM, 2002.

Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252. IEEE, 2018.

David E Orin, Ambarish Goswami, and Sung-Hee Lee. Centroidal dynamics of a humanoid robot. *Autonomous robots*, 35(2):161–176, 2013.

Gerardo Bledt, Patrick M Wensing, and Sangbae Kim. Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the mit cheetah. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4102–4109. IEEE, 2017.

Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5148–5154. IEEE, 2015.

Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1620–1626. IEEE, 2003.

Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Fast, robust quadruped locomotion over challenging terrain. In *2010 IEEE International Conference on Robotics and Automation*, pages 2665–2670. IEEE, 2010.

C Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter. Dynamic locomotion and whole-body control for quadrupedal robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3359–3365. IEEE, 2017.

Wen-Loong Ma, Kaveh Akbari Hamed, and Aaron D Ames. First steps towards full model based motion planning and control of quadrupeds: A hybrid zero dynamics approach. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5498–5503. IEEE, 2019.

Wen-Loong Ma and Aaron D Ames. From bipedal walking to quadrupedal locomotion: Full-body dynamics decomposition for rapid gait generation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4491–4497. IEEE, 2020.

Steven L. Brunton, Marko Budišić, Eurika Kaiser, and J. Nathan Kutz. Modern koopman theory for dynamical systems. 2021. doi:10.48550/ARXIV.2102.12086.

Nathan Kau, Aaron Schultz, Natalie Ferrante, and Patrick Slade. Stanford doggo: An open-source, quasi-direct-drive quadruped. *CoRR*, abs/1905.04254, 2019.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

Appendix

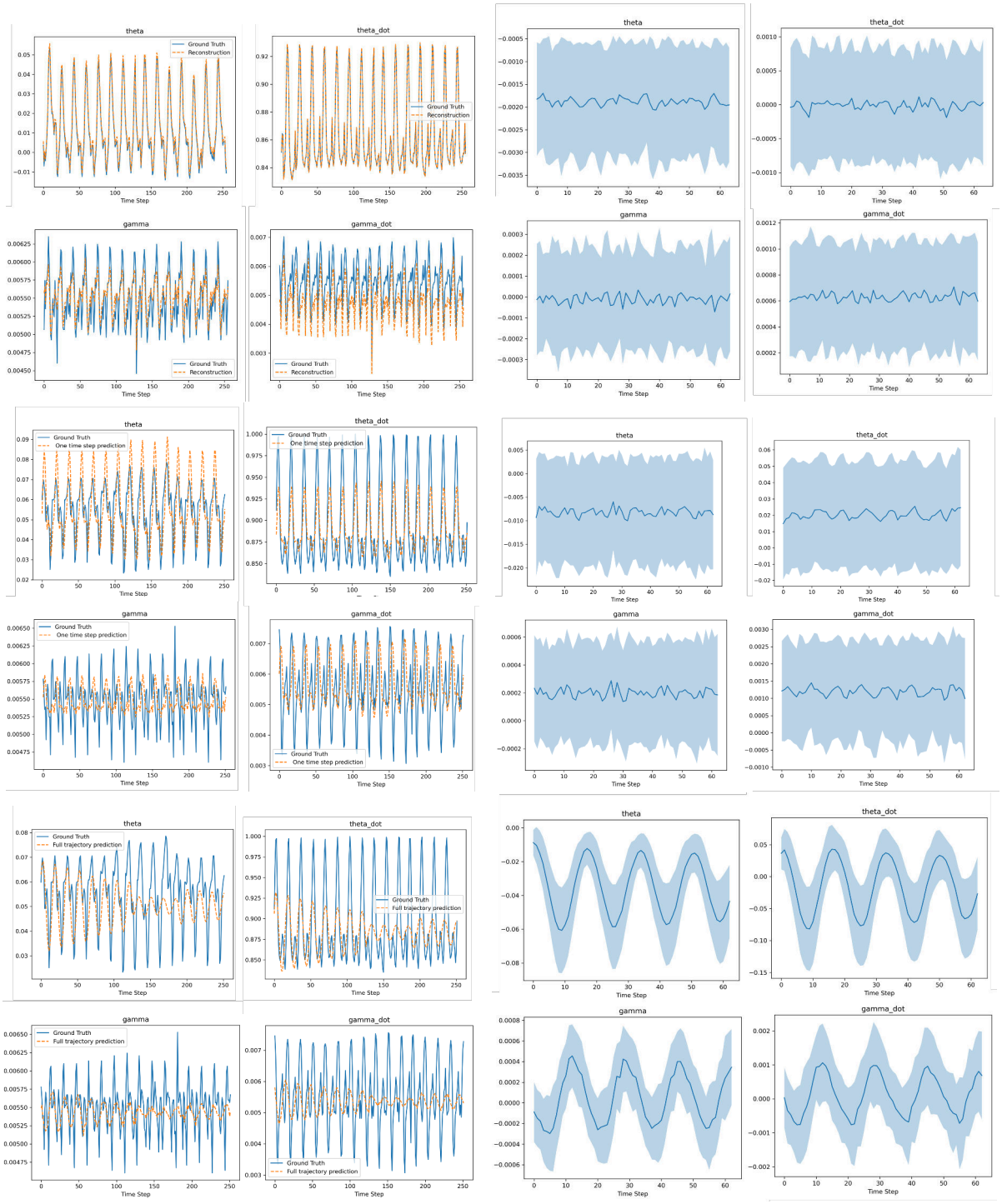


Figure 4: Reconstruction, one time step and full trajectory predictions on rigid ground.

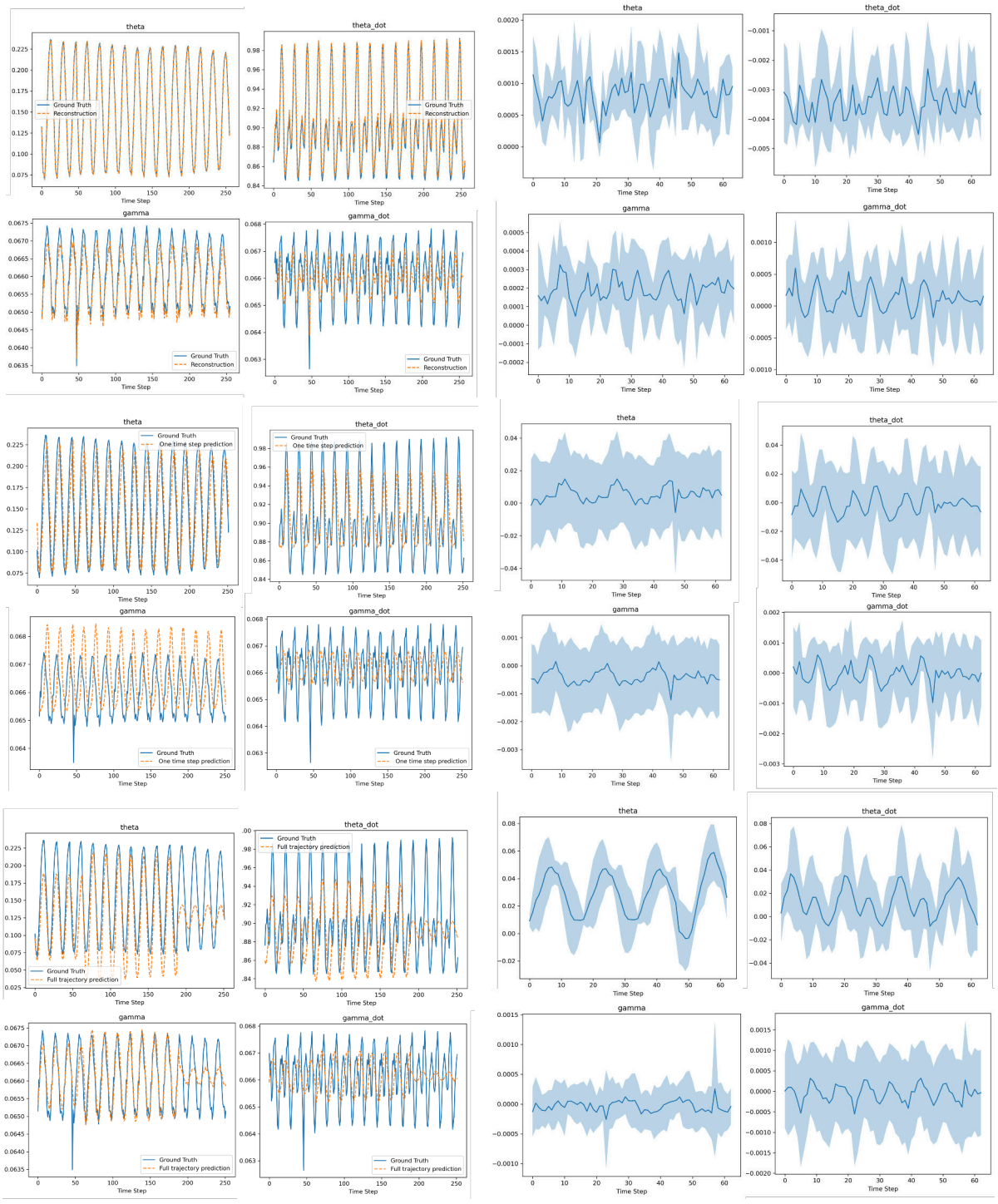


Figure 5: Reconstruction, one time step and full trajectory predictions on mulch soil.

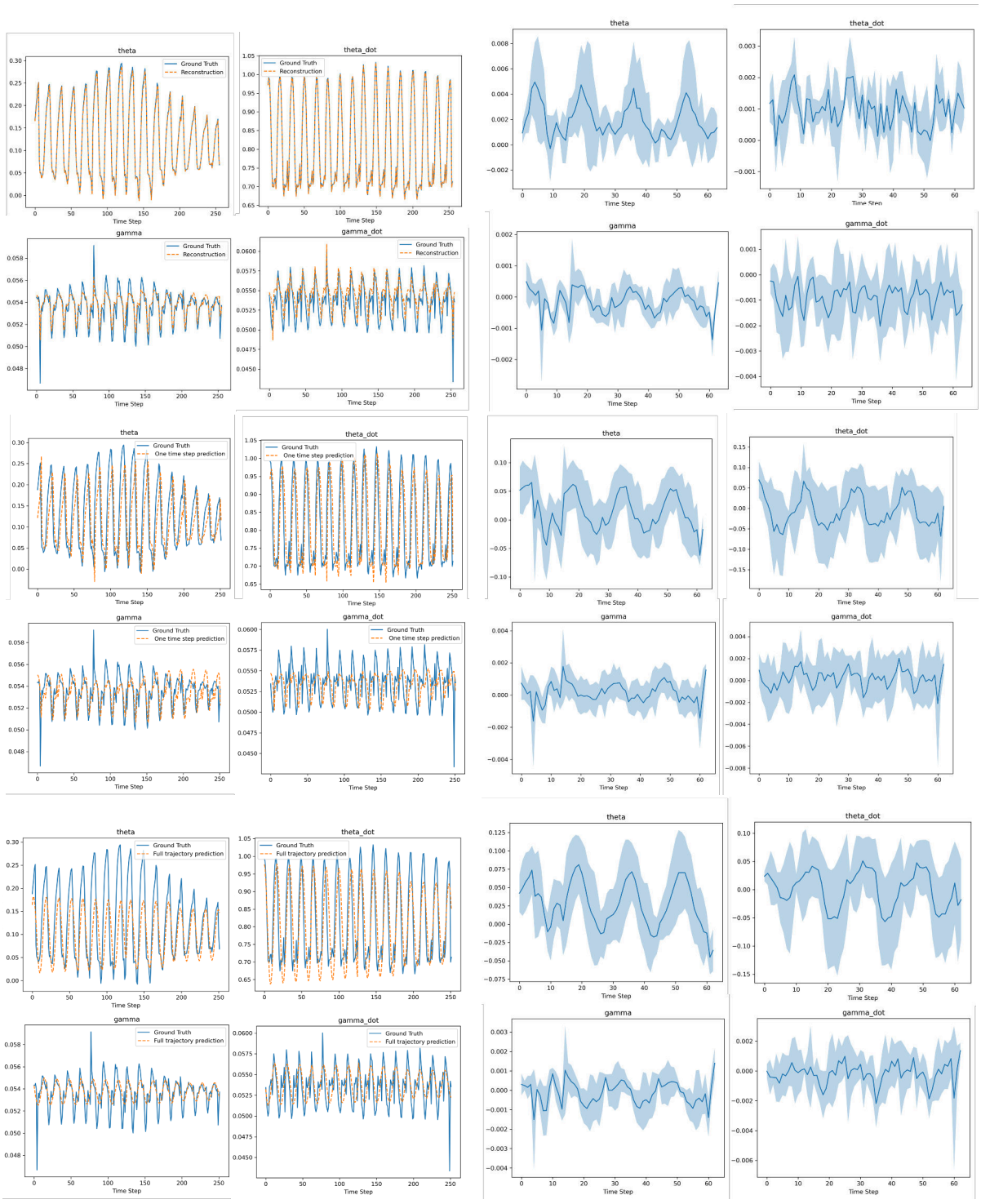


Figure 6: Reconstruction, one time step and full trajectory predictions on Pebbles soil.

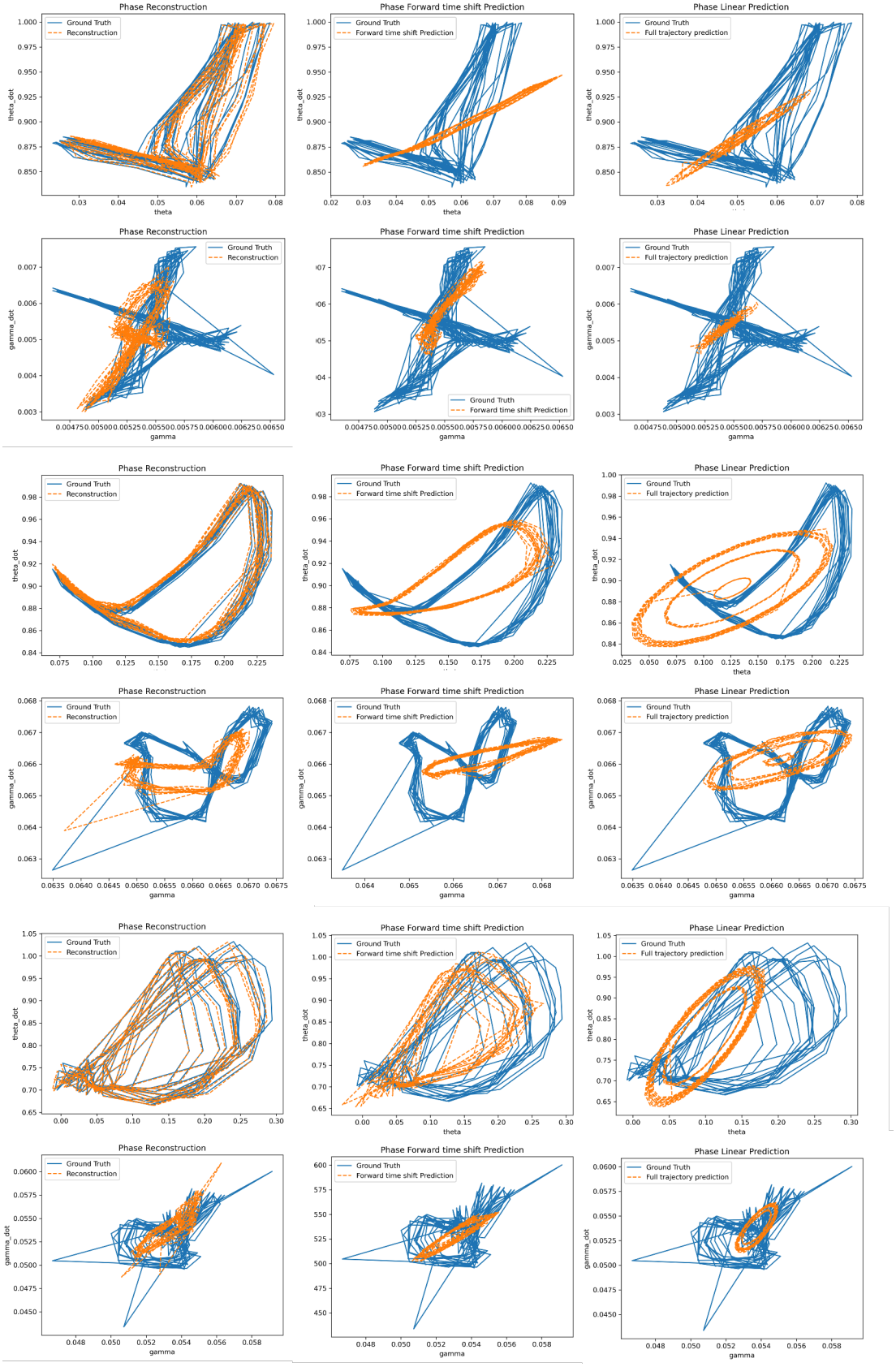


Figure 7: Phase portraits of reconstruction, one time step and full trajectory predictions on experimental data.

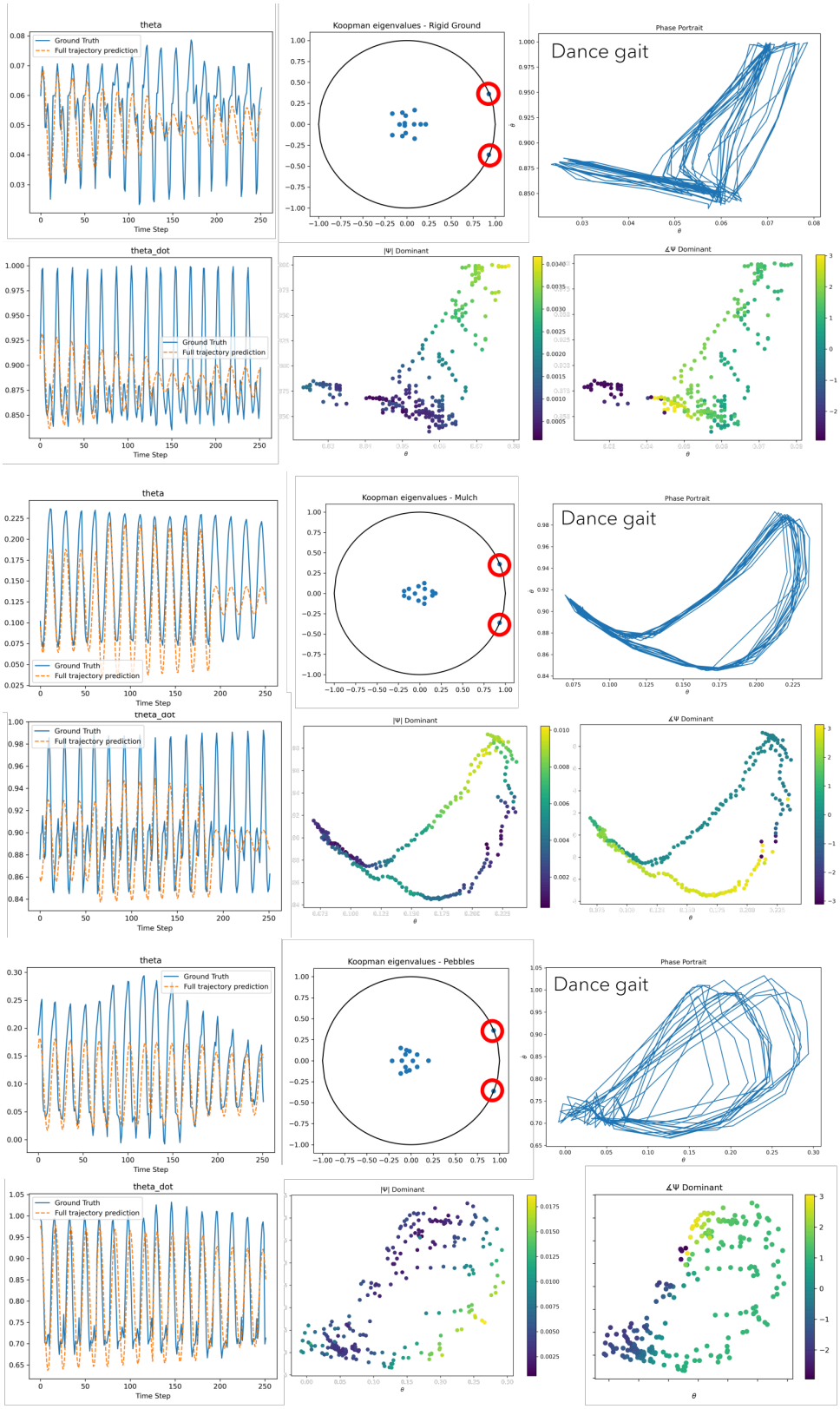


Figure 8: Koopman eigenfunctions for experiments conducted on each terrain.

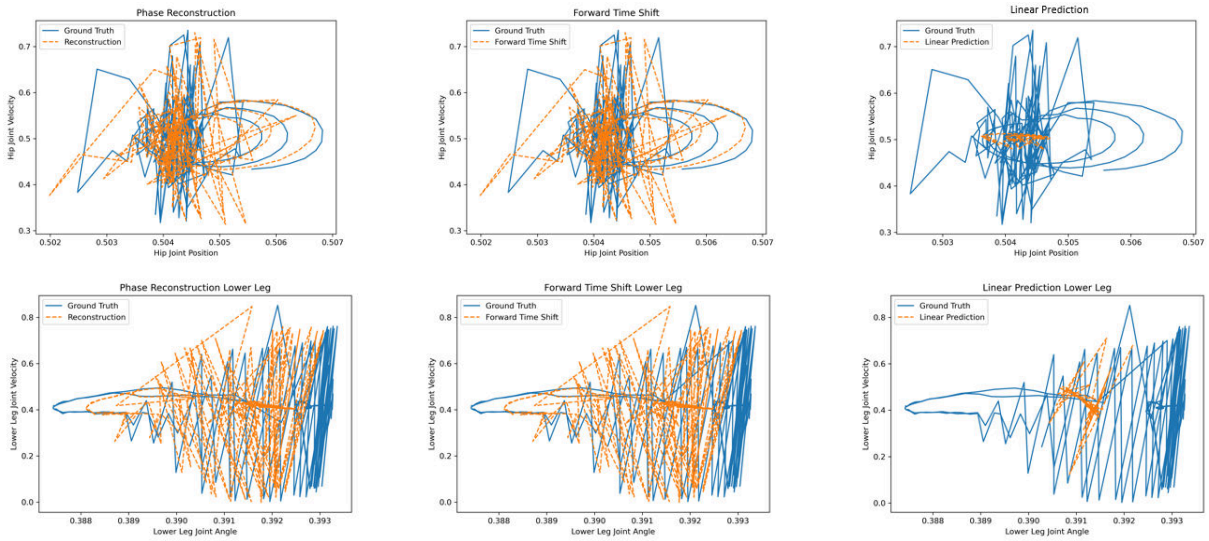
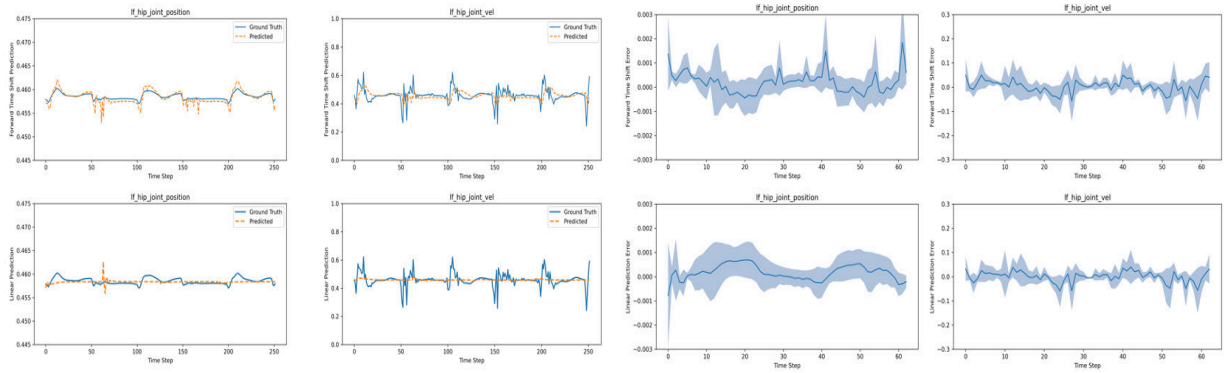


Figure 9: Rigid Ground Phase Portrait of Champ Locomotion



(a) (b)

Figure 10: Simulation Results for Ground Parameters $K_p = 10^{12}$ and $K_d = 0$ a) Prediction Plots b) Error Graphs

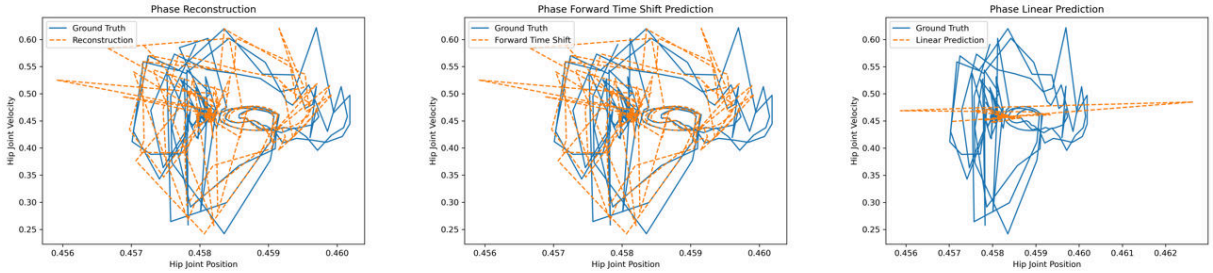


Figure 11: Phase Portrait of Champ Locomotion for Ground Parameters $K_p = 10^{12}$ and $K_d = 0$