LEARNING COMPOSITIONAL KOOPMAN OPERATORS FOR MODEL-BASED CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Finding an embedding space for a linear approximation of a nonlinear dynamical system enables efficient system identification and control synthesis. The Koopman operator theory lays the foundation for identifying the nonlinear-to-linear coordinate transformations with data-driven methods. Recently, researchers have proposed to use deep neural networks as a more expressive class of basis functions for calculating the Koopman operators. These approaches, however, assume a fixed dimensional state space; they are therefore not applicable to scenarios with a variable number of objects. In this paper, we propose to learn compositional Koopman operators, using graph neural networks to encode the state into object-centric embeddings and using a block-wise linear transition matrix to regularize the shared structure across objects. The learned dynamics can quickly adapt to new environments of unknown physical parameters and produce control signals to achieve a specified goal. Our experiments on manipulating ropes and controlling soft robots show that the proposed method has better efficiency and generalization ability than existing baselines.

1 INTRODUCTION

Simulating and controlling complex dynamical systems such as ropes or soft robots rely on the dynamics model's two key features: first, it needs to be *efficient* for system identification and motor control; second, it needs to be *generalizable* to a complex, constantly evolving environments.

In practice, computational models for complex, nonlinear dynamical systems are often not efficient enough for real-time control (Mayne, 2000). The Koopman operator theory suggests that identifying nonlinear-to-linear coordinate transformations allows efficient linear approximation of nonlinear systems (Williams et al., 2015; Mauroy & Goncalves, 2016). Fast as they are; however, existing papers on Koopman operators focus on a single dynamical system, making it hard for them to generalize to cases where there are a variable number of components.

In contrast, recent advances in approximating dynamics models with deep nets have demonstrated its power in characterizing complex, generic environments. In particular, a few recent papers have explored the use of graph nets in dynamics modeling, takes into account the state of each object as well as their interactions. This allows their models to generalize to scenarios with a variable number of objects (Battaglia et al., 2016; Chang et al., 2017). Despite their strong generalization power, they are not as efficient in system identification and control, because deep nets are heavily over-parameterized, making optimization time-consuming and sample-inefficient.

In this paper, we propose compositional Koopman operators, integrating Koopman operators with graph networks for generalizable and efficient dynamics modeling. We build on the idea of encoding states into object-centric embeddings with graph neural networks, which ensures generalization power. But instead of using over-parameterized neural nets to model state transition, we identify the Koopman matrix and control matrix from data as a linear approximation of the nonlinear dynamical system. The linear approximation allows efficient system identification and control synthesis.

The main challenge of extending Koopman theory to multi-object systems is scalability. The number of parameters in the Koopman matrix scales quadratically with the number of objects, which harms the learning efficiency and leads to overfitting. To tackle this issue, we exploit the structure of the underlying system and use the same block-wise Koopman sub-matrix for object pairs of the same

relation. This significantly reduces the number of parameters that need to be identified by making it independent of the size of the system.

Our experiments include simulating and controlling ropes of variable lengths and soft robots of different shapes. The compositional Koopman operators are significantly more accurate than the state-of-the-art learned physics engines (Battaglia et al., 2016; Li et al., 2019b), and is 20 times faster when adapting to new environments of unknown physical parameters. Our method also outperforms vanilla deep Koopman methods (Lusch et al., 2018; Morton et al., 2018) and Koopman models with manually-designed basis functions, which shows the advantages of using a structured Koopman matrix and graph neural networks. Please see our supplement video for an in-depth demonstration [*].

2 Related Work

Koopman operators. The Koopman operator formalism of dynamical systems is rooted in the seminal works of Koopman and Von Neumann in the early 1930s (Koopman, 1931; Koopman & Neumann, 1932). The core idea is to map the state of a nonlinear dynamical system to an embedding space, over which we can linearly propagate into the future. The linear representation will enable efficient prediction, estimation, and control using tools from linear dynamical systems (Williams et al., 2016; Proctor et al., 2018; Mauroy & Goncalves, 2017). People have been using hand-designed Koopman observables for various modeling and control tasks (Brunton et al., 2016; Kaiser et al., 2017; Abraham et al., 2017; Bruder et al., 2018; Arbabi et al., 2018). Some recent works have applied the method to the real world and successfully control soft robots with great precision (Bruder et al., 2019; Mamakoukas et al., 2019).

However, hand-crafted basis functions sometimes fail to generalize to more complex environments. Learning these functions from data using neural nets turns out to generate a more expressive invariant subspace (Lusch et al., 2018; Takeishi et al., 2017) and has achieved successes in fluid control (Morton et al., 2018). Morton et al. (2019) has also extended the framework to account for uncertainty in the system by inferring a distribution over observations. Our model differs by explicitly modeling the compositionality of the underlying system with graph networks. It generalizes better to environments of a variable number of objects or soft robots of different shapes.

Learning-based physical simulators. Battaglia et al. (2016) and Chang et al. (2017) first explored learning a simulator from data by approximating object interactions with neural networks. These models are no longer bounded to hard-coded physical rules, and can adapt to scenarios where the underlying physics is unknown. Please refer to Battaglia et al. (2018) for a full review. Recently, Mrowca et al. (2018) and Li et al. (2019a) extended these models to approximate particle dynamics of deformable shapes and fluids. Flexible as they are, these models become less efficient during model adaptation in complex scenarios, because the optimization of neural networks usually needs a lot of samples and compute, which limits its use in an online setting. Nagabandi et al. (2018a;b) proposed to use meta-learning for online adaptation, and have shown to be effective in simulated robots and a real legged millirobot. However, it is not clear whether their methods can generalize to systems with variable numbers of instances. The use of graph nets and Koopman operators in our model allows better generalization ability and enables efficient system identification as we only need to identify the transition matrices, which is essentially a least-square problem and can be solved very efficiently.

People have also used the learned physics engines for planning and control. Many previous papers in this direction learn a latent dynamics model together with a policy in a model-based reinforcement learning setup (Racanière et al., 2017; Hamrick et al., 2017; Pascanu et al., 2017; Hafner et al., 2019); a few alternatives use the learned model in model-predictive control (MPC) (Sanchez-Gonzalez et al., 2018; Li et al., 2019b; Janner et al., 2019). In this paper, we leverage the fact that the embeddings in the Koopman space are propagating linearly through time, which allows us to formulate the control problem as quadratic programming and optimize the control signals much more efficiently.

3 Approach

We first present the basics of Koopman operators: for a nonlinear dynamical system, the Koopman observation functions can map the state space to an embedding space where the dynamics become linear. We then discuss the compositional nature of physical systems and show how graph networks can be used to capture the compositionality.

[*] Video: https://youtu.be/idFH4K16cfQ

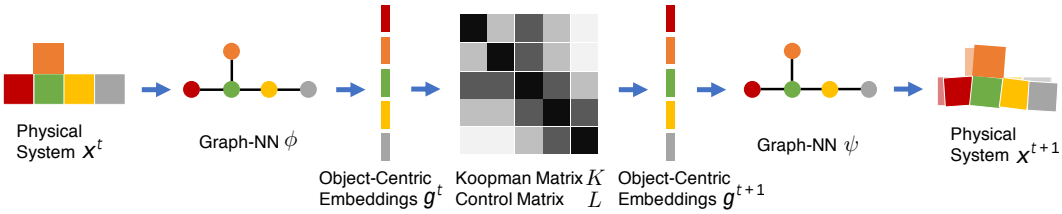

Figure 1: **Overview of our model.** A graph neural network ϕ takes in the current state of the physical system x^t, and generates object-centric representations in the Koopman space g^t. We then use the block-wise Koopman matrix K and control matrix L identified from equation 4 or equation 6 to predict the Koopman embeddings in the next time step g^{t+1}. Note that in K and L, object pairs of the same relation share the same sub-matrix. Another graph neural network ψ maps g^{t+1} back to the original state space, i.e., x^{t+1}. The mapping between g^t and g^{t+1} is linear and is shared across all time steps. We can iteratively apply K and L to the Koopman embeddings and roll multiple steps into the future, which enables efficient system identification and control synthesis.

3.1 The Koopman Operators

Let $x^t \in X \subseteq \mathbb{R}^n$ be the state vector for the system at time step t. We consider a non-linear discrete-time dynamical system described by $x^{t+1} = F(x^t)$. We call any function $g : X \to \mathbb{R}$ an *observable* of the system, and we note that the set of all observables forms an infinite-dimensional Hilbert space. The Koopman operator (Koopman, 1931), denoted by $\mathcal{K}$, is a linear transformation on this vector space given by $\mathcal{K}g = g \circ F$, so that $\mathcal{K}g(x^t) = g(F(x^t)) = g(x^{t+1})$.

Although the theory guarantees the existence of the Koopman operator, its use in practice is limited by its infinite dimensionality. Most often, we assume there is an invariant subspace $\mathcal{G}$ of the Koopman operator. It spanned by a set of base observation functions $\{g_1, \ldots, g_m\}$ and satisfies that $\mathcal{K}g \in \mathcal{G}$ for any $g \in \mathcal{G}$. With a slightly abuse of the notation, we now use $g(x^t) : \mathbb{R}^n \to \mathbb{R}^m$ to represent $[g_1(x^t), \ldots, g_m(x^t)]^T$. By constraining the Koopman operator on this invariant subspace, we get a finite-dimensional linear operator $K \in \mathbb{R}^{m \times m}$ that we refer as the *Koopman matrix*.

Traditionally, people hand-craft base observation functions from the knowledge of underlying physics. The system identification problem is then reduced to finding the Koopman matrix K, which can be solved by linear regression given historical data of the system. Recently, researchers have also explored data-driven methods that automatically find the Koopman invariant subspace via representing the base observation functions $g(x)$ via deep neural networks.

Above is the Koopman theory on modeling unforced dynamics. Now consider a system with an external control input u^t and a dynamics model $x^{t+1} = F(x^t, u^t)$. We aim to find the Koopman observation functions and the linear dynamics model in the form of $g(x^{t+1}) = Kg(x^t) + Lu^t$, where we assume the control signal has linear effects in the observation space. Here the coefficient matrix L is referred to as the *control matrix*.

3.2 Compositional Koopman Operators

The dynamics of a physical system are governed by physical rules, which are usually shared across different subcomponents in the system. Explicitly modeling such compositionality enables more efficient system identification and control synthesis and provides better generalization ability.

Motivating example. Consider a system with N balls moving in a 2D plane, each pair connected by a linear spring. Assume all balls have mass 1 and all springs share the same stiffness coefficient k. We denote the i's ball's position as (x_i, y_i) and its velocity as $(\dot{x}_i, \dot{y}_i)$. For ball i, equation 1 describes its dynamics, where $x_i = [x_i, y_i, \dot{x}_i, \dot{y}_i]^T$ denotes ball i's state:

$$\dot{x}_i = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{\dot{x}}_i \\ \dot{\dot{y}}_i \end{bmatrix} = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \sum_{j=1}^{N} k(x_j - x_i) \\ \sum_{j=1}^{N} k(y_j - y_i) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ k & -Nk & 0 & 0 \\ 0 & k & -Nk & 0 & 0 \end{bmatrix}}_{\triangleq A} \begin{bmatrix} x_i \\ y_i \\ \dot{x}_i \\ \dot{y}_i \end{bmatrix} + \sum_{j \neq i} \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \end{bmatrix}}_{\triangleq B} \begin{bmatrix} x_i \\ y_i \\ \dot{x}_i \\ \dot{y}_i \end{bmatrix}. \tag{1}$$

We can represent the state of the whole system using the union of every ball's state, where $x = [x_1, \ldots, x_N]^T$. Then the transition matrix is essentially a block matrix, where the matrix parameters

are shared among the diagonal or off-diagonal blocks as shown in equation 2:

$$\underline{x} = \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_N \end{bmatrix} = \begin{bmatrix} A & B & & B \\ B & A & & B \\ \vdots & \vdots & \ddots & \vdots \\ B & B & & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}. \tag{2}$$

Based on the linear spring system, we make three observations for multi-object systems.

The system state is composed of the state of each individual object. The dimension of the whole system scales linearly with the number of objects. We formulate the system state by concatenating the state of every object, corresponding to an object-centric state representation.

The transition matrix has a block-wise substructure. After assuming an object-centric state representation, the transition matrix naturally has a block-wise structure as shown in equation 2.

The same physical interactions share the same transition block. The blocks in the transition matrix encode actual interactions and generalize across systems. A and B govern the dynamics of the linear spring system, and are shared by systems with a different number of objects.

These observations motivate us to exploit the structure of multi-object systems, instead of learning separate models for systems that contains different numbers of balls.

Compositional Koopman operators. Motivated by the linear spring system, we want to inject a good inductive bias to incorporate compositionality when applying the Koopman theory. This allows better generalization ability and more efficient system identification and better controller design. Figure 1 shows an overview of our model.

Considering a system with N objects, we denote x^t as the system state at time t and x_i^t is the state of the i'th object. We further denote g^t , $g(x^t)$ as the embedding of the state in the Koopman invariant space. In the rest of the paper, we call g^t the *Koopman embedding*. Based on the observation we made in the case of linear spring system, we propose the following assumptions on the compositional structure of the Koopman embedding and the Koopman matrix.

The Koopman embedding of the system is composed of the Koopman embedding of every objects. Similar to the decomposition in the state space, we assume the Koopman embedding can be divided into object-centric sub-embeddings, i.e. $g^t = [g_1^{t\top}, \quad , g_N^{t\top}]^\top$ 2 $\mathbb{R}^{Nm}$, where we overload the notation and use $g_i^t = g_i(x^t)$ 2 $\mathbb{R}^m$ as the Koopman embedding for the i'th object.

The Koopman matrix has a block-wise structure. It is natural to think the Koopman matrix is composed of block matrices after assuming an object-centric Koopman embeddings. In equation 3, K_{ij} 2 $\mathbb{R}^{m \quad m}$ and L_{ij} 2 $\mathbb{R}^{m \quad ju_ij}$ are blocks of the Koopman matrix and the control matrix, while $u^t = [u_1^{t\top}, \quad , u_N^{t\top}]^\top$ is the control signal at time t:

$$\begin{bmatrix} g_1^{t+1} \\ \vdots \\ g_N^{t+1} \end{bmatrix} = \begin{bmatrix} K_{11} & & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & & K_{NN} \end{bmatrix} \begin{bmatrix} g_1^t \\ \vdots \\ g_N^t \end{bmatrix} + \begin{bmatrix} L_{11} & & L_{1N} \\ \vdots & \ddots & \vdots \\ L_{N1} & & L_{NN} \end{bmatrix} \begin{bmatrix} u_1^t \\ \vdots \\ u_N^t \end{bmatrix}. \tag{3}$$

As we have seen in the case of linear spring system, those matrix blocks are not independent, but some of them share the same set of values.

The same physical interactions shall share the same transition block. The equivalence between the blocks should reflect the equivalence of the objects, where we use the same transition sub-matrix for object pairs of the same relation. For example, if the system is composed of N identical objects, then, by symmetry, all the diagonal blocks should be the same while all the off-diagonal blocks should also be the same. The repetitive structure allows us to efficiently identify the values using least square regression.

3.3 Learning the Koopman Embeddings Using Graph Neural Networks

For a physical system that contains N objects, we represent the system at time t using a directed graph $G^t = (O^t, R)$, where vertices $O^t = \{o_i^t\}_{i=1}^N$ represent objects and edges $R = \{r_k\}_{k=1}^{N^2}$ represent pair-wise relations. Specifically, $o_i^t = (x_i^t, a_i^o)$, where x_i^t is the state of object i and a_i^o is a one-hot vector indicating the object type, e.g., fixed or movable. For relation, we have $r_k = (u_k, v_k, a_k^r), 1 \quad u_k, v_k \quad N$, where u_k and v_k are integers denoting the end points of this directed edge, and a_k^r is a one-hot vector denoting the type of the relation k.

We use a graph neural network similar to Interaction Networks (IN) (Battaglia et al., 2016) to generate object-centric Koopman embeddings. IN defines an object function f_O and a relation function f_R to model objects and their relations in a compositional way. Similar to a message passing procedure, we calculate the edge effect $e_k^t = f_R(o_{u_k}^t, o_{v_k}^t, a_k^r)_{k=1:::N^2}$, and node effect $g_i^t = f_O(o_i^t, \sum_{k2N_i} e_k^t)_{i=1:::N}$, where N_i denotes the relations that related to object i and $fg_i^t g$ are the derived Koopman embeddings. We use this graph neural network, denoted as ϕ, to represent our Koopman observation function.

System identification. For a sequence of observations $\tilde{x} = [x^1, \quad , x^T]$ from time 1 to time T, we first map them to the Koopman space as $\tilde{g} = [g^1, \quad , g^T]$ using the graph encoder ϕ, where $g^t = \phi(x^t)$. We use $g^{i:j}$ to denote the sub-sequence $[g^i, \quad , g^j]$. To identify the Koopman matrix, we solve the linear regression $\min_K kKg^{1:T \ 1} \quad g^{2:T}k_2$. As a result, $K = g^{2:T}(g^{1:T \ 1})^y$ will asymptotically approach the Koopman operator K with an increasing T. For cases where there are control inputs $\tilde{u} = [u^1, \quad , u^{T \ 1}]$, the calculation of the Koopman matrix and the control matrix is essentially solving a least square problem w.r.t. the objective

$$\min_{K;L} kKg^{1:T \ 1} + L\tilde{u} \quad g^{2:T}k_2. \tag{4}$$

As we mentioned in the Section 3.2, the dimension of the Koopman space is linear to the number of objects in the system, i.e., $\tilde{g} \ 2 \ R^{Nm \ T}$ and $K \ 2 \ R^{Nm \ Nm}$. If we do not enforce any structure on the Koopman matrix K, we will have to identify N^2m^2 parameters. Instead, we can significantly reduce the number by leveraging the assumption on the structure of K. Assume we know some blocks ($fK_{ij}g$) of the matrix K are shared and in total there are h different kinds of blocks, which we denote as $\hat{K} \ 2 \ R^{h \ m \ m}$. Then, the number of parameter to be identified reduce to hm^2. Usually, h does not depend on N, and is much smaller than N^2. Now, for each block K_{ij}, we have a one-hot vector $\sigma_{ij} \ 2 \ f0, 1g^h$ indicating its type, i.e., $K_{ij} = \sigma_{ij}\hat{K}$. Finally, as shown in equation 5, we represent the Koopman matrix as the product of the index tensor σ and the parameter tensor $\hat{K}$:

$$K = \sigma \quad \hat{K} = \begin{bmatrix} \sigma_{11}\hat{K} & & \sigma_{1N}\hat{K} \\ \vdots & \ddots & \vdots \\ \sigma_{N1}\hat{K} & & \sigma_{NN}\hat{K} \end{bmatrix}, \text{where} \quad \sigma = \begin{bmatrix} \sigma_{11} & & \sigma_{1N} \\ \vdots & \ddots & \vdots \\ \sigma_{N1} & & \sigma_{NN} \end{bmatrix} \ 2 \ R^{N \ N \ h}. \tag{5}$$

Similar to the Koopman matrix, we assume the same block structure in the control matrix L and denote its parameter as $\hat{L} \ 2 \ R^{h \ N \ ju_ij}$. The least square problem of identifying $\hat{K}$ and $\hat{L}$ becomes

$$\min_{\hat{K};\hat{L}} k(\sigma \quad \hat{K})g^{1:T \ 1} + (\sigma \quad \hat{L})\tilde{u} \quad g^{2:T}k_2. \tag{6}$$

Since the linear least square problems described in equation 4 and equation 6 have analytical solutions, performing system identification using our method is very efficient.

Training GNN models. To make predictions on the states, we use a graph decoder ψ to map the Koopman embeddings back to the original state space. In total, we have three losses to train the graph encoder and decoder. The first term is the auto-encoding loss $L_{\text{ae}} = \frac{1}{T}\sum_i k\psi(\phi(x^i)) \quad x^ik_2$. The second term is the prediction loss. To calculate it, we rollout in the Koopman space and denote the embeddings as $\hat{g}^1 = g^1$, and $\hat{g}^{t+1} = K\hat{g}^t + Lu^t$, for $t = 1, \quad , T \quad 1$. The prediction loss is defined as the difference between the decoded states and the actual states, i.e., $L_{\text{pred}} = \frac{1}{T}\sum_{i=1}^{T} k\psi(\hat{g}^i) \quad x^ik_2$. Third, we employ a metric loss to encourage the Koopman embeddings preserving the distance in the original state space. The loss is defined as the absolute error between the distances measured in the Koopman space and that in the original space, i.e., $L_{\text{metric}} = \sum_{ij} \left| kg^i \quad g^jk_2 \quad kx^i \quad x^jk_2 \right|$. Having Koopman embeddings that perserves the distance in the state space is important as we are using the distance in the Koopman space to define the cost function for downstream control tasks.

The ultimate training loss is simply the combination of all the terms above: $L = L_{\text{ae}} + L_{\text{pred}} + L_{\text{metric}}$. We then minimize the loss L by optimizing the parameters in the graph encoder ϕ and graph decoder ψ using stochastic gradient descent. Once the model is trained, it can be used for system identification, future prediction, and control synthesis.

3.4 CONTROL

For a control task, the goal is to synthesize a sequence of control inputs $u^{1:T}$ that minimize $C = \sum_{t=1}^{T} c_t(x^t, u^t)$, the total incurred cost, where $c_t(x^t, u^t)$ is the instantaneous cost. For example, considering the control task of reaching a desired state x^* at time T, we can design the following instantaneous cost, $c_t(x^t, u^t) = 1_{[t=T]}kx^t \quad x^* \ k_2^2 + \lambda ku^tk_2^2$. The first term promotes the control sequence that matches the state to the goal, while the second term regularizes the control signals.

Open-loop control via quadratic programming (QP). Our model maps the original nonlinear dynamics to a linear dynamical system. We can then solve the control task by solving a linear control problem. With the assumption that the Koopman embeddings preserve the distance measure, we define the control cost as $c_t(g^t, u^t) = 1_{[t=T]} \|g^t - g\|_2^2 + \lambda \|u^t\|_2^2$. As a result, we reduce the problem to minimizing a quadratic cost function $C = \sum_{t=1}^{T} c_t(g^t, u^t)$ over variables $\{g^t, u^t\}_{t=1}^{T}$ under linear constrains $g^{t+1} = Kg^t + Lu^t$, where $g^1 = \phi(x^1)$ and $g = \phi(x)$.

Model predictive control (MPC). Solving the QP gives us control signals, which might not be good enough for long-term control as the prediction error accumulates. We can combine it with Model Predictive Control, assuming feedback from the environment every τ steps.

4 EXPERIMENTS

Environments. We evaluate our method by assessing how well it can simulate and control ropes and soft robots. Specifically, we consider three environments. (1) **Rope** (Figure 2a): the top mass of a rope is fixed to a specific height. We apply force to the top mass to move it in a horizontal line. The rest of the masses are free to move according to internal force and gravity. (2) **Soft** (Figure 2b): we aim to control a soft robot that is consist of soft blocks. Blocks in dark grey as rigid and those in light blue are soft blocks. Each one of the dark blue blocks is soft but have an actuator inside that can contract or expand the block. One of the blocks is pinned to the ground, as shown using the red dots. (3) **Swim** (Figure 2c): instead of pinning the soft robot to the ground, we let the robot swim in fluids. The colors shown in this environment have the same meaning as in **Soft**.

Baselines. We compare our model to the following baselines: Interaction Networks (Battaglia et al., 2016) (**IN**), Propagation Networks (Li et al., 2019b) (**PN**) and Koopman method with hand-crafted Koopman base functions (**KPM**). IN and PN are the state-of-the-art learning-based physical simulators. For control, we first finetune the parameters in IN and PN to adapt to the test environment of unknown physical parameters. We then apply gradient descent to the control signals by minimizing the distance between the prediction from IN/PN and the target, following Li et al. (2019b). The generated control sequence is fed to the original simulator to evaluate the performance. Similar to our method, KPM fits a linear dynamics in the Koopman space. Instead of learning Koopman observations from data, KPM uses polynomials of the original states as the basis functions. In our setting, we set the maximum order of the polynomials to be three to make the dimension of the hand-crafted Koopman embeddings match our model's.

Data generation. We generate 10,000 samples for Rope and 50,000 samples for Soft and Swim. Among them, 90% are used for training and the rest for testing. Each data sample has 100 time steps.

Training and evaluation protocols. Our model is trained on the sub-sequence of length 64 from the training set. For evaluation, we use two metrics: **simulation error** and **control error**. For a given data sample, the simulation error at time step t is defined as the mean squared error between the model prediction $\hat{x}^t$ and the ground truth x^t. For control, we pick the t_0'th frame x^{t_0} and the $t_0 + t$'th frame x^{t_0+t} from a data sample. Then we ask the model to generate a control sequence of length t to transfer the system from the initial state x^{t_0} to the target state x^{t_0+t}. The control error is defined as the mean squared distance between the target state and the state of the system at time t. For our experiments we have $t = 64$.

4.1 SIMULATION

Figure 2 shows qualitative results on simulation. Our model accurately predicts system dynamics for more than 100 steps. For Rope, the small prediction error comes from the slight delay of the force propagation inside the rope; hence, the tail of the rope usually has a larger error. For Soft, our model captures the interaction between the body parts and generates accurate prediction over the global movements of the robot. The error mainly comes from the misalignment of some local components.

Figure 3 shows quantitative results. IN and PN do not work well due to poor system identification ability. IN and PN rely on gradient descent to adapt to the environments of unknown physical parameters and on average takes over 10 seconds. Our model takes around 0.43 seconds per instance in the identification stage which is about 20 times faster. The KPM baseline with simple hand-crafted Koopman basis works reasonably well in the Rope and Swim environment, partly because KPM uses the same system identification algorithm as our model, leveraging the prior knowledge on the structure of the system. Our model significantly outperforms all the baselines except in the Swim environment, where we are on par with KPM.

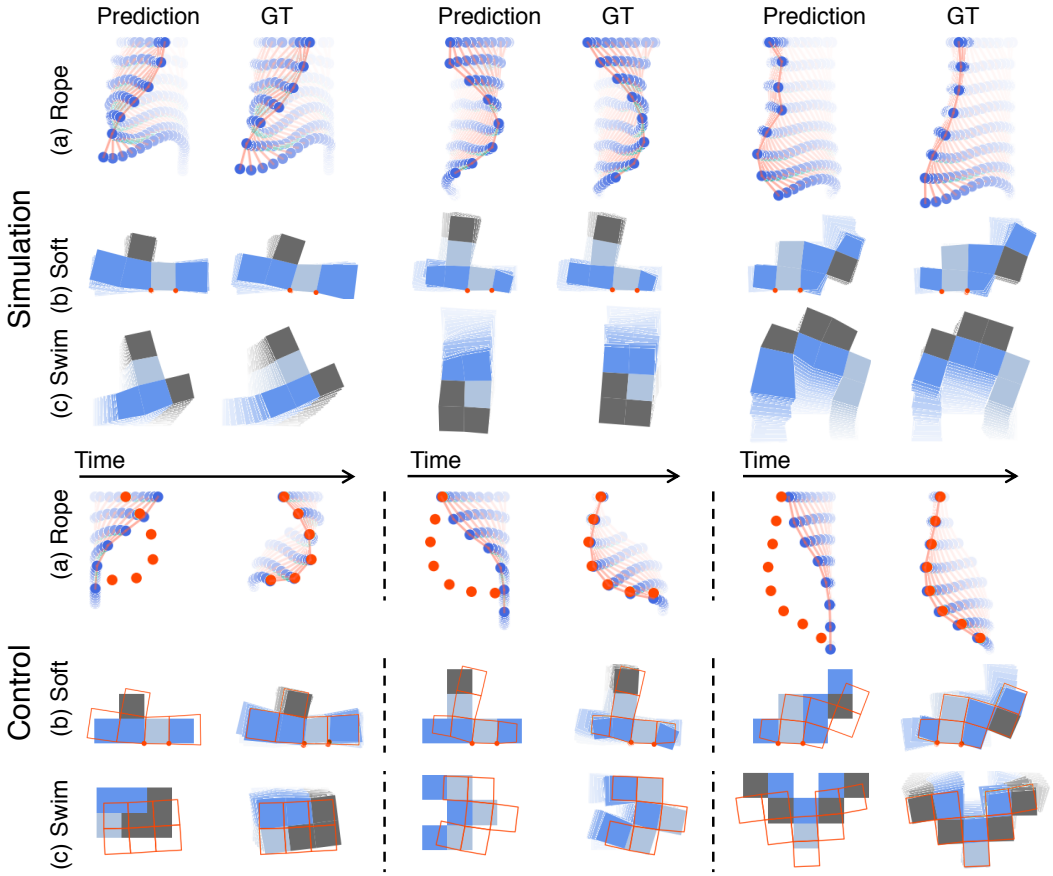

Figure 2: **Qualitative results.** Top: our model prediction matches the ground truth over a long period. Bottom: for control, we use red dots or frames to indicate the goal. We apply the control signals generated from our identified model to the original simulator, which allows the agent to achieve the goal accurately. Please refer to our supplementary video for more results.

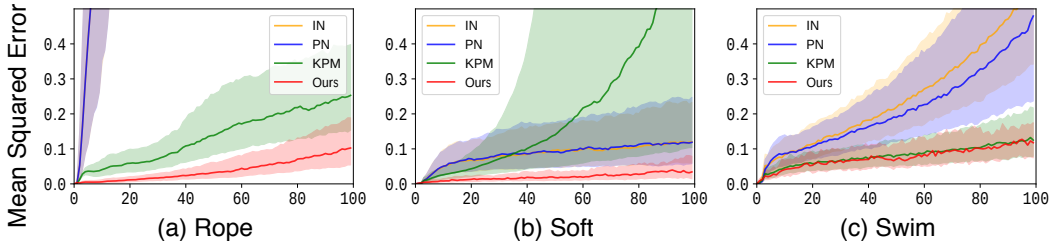

Figure 3: **Quantitative results on simulation.** The x axis shows time steps. The solid lines indicate medians and the transparent regions are the interquartile ranges of simulation errors. Our method significantly outperforms baselines in both Rope and Soft.

4.2 Control

As the simulation errors of IN and PN are too large for control, we compare our model with KPM. In Rope, we ask the models to perform open-loop control where it only solves the QP once at the beginning. The length of the control sequence is 40. When it comes to Soft/Swim, each model is asked to generate control signals of 64 steps, and we allow the model to receive feedback after 32 steps. Thus every model has a second chance to correct its control sequence by solving the QP again at the time step 32.

As shown in Figure 2, our model leverages the inertia of the rope and matches the target state accurately. As for controlling a soft body swinging on the ground or swimming in the water, our model can move each part (the boxes) of the body to the exact target position. The small control error

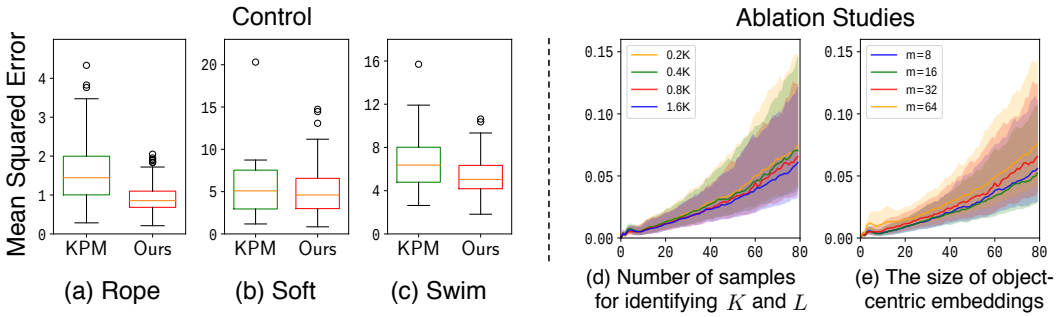

Figure 4: **Quantitative results on control and ablation studies on model hyperparameters.** Left: box-plots show the distributions of control errors. The yellow line in the box indicates the median. Our model consistently achieves smaller errors in all environments against KPM. Right: our model's simulation errors with different amount of data for system identification (d) and different dimensions of the Koopman space (e).

comes from the slight misalignment of the orientation and the size of the body parts. Figure 4 shows that quantitatively our model outperforms KPM, too.

4.3 Ablation Study

Structure of the Koopman matrix. We explore three different structures of the Koopman matrix, *Block*, *Diag* and *None*, to understand its effect on the learned dynamics. *None* assumes no structure in the Koopman matrix. *Diag* assumes a diagonal block structure of K: all off-diagonal blocks (K_{ij} where $i \neq j$) are zeros and all diagonal blocks share the same values. *Block* predefines a block-wise structure, decided by the relation between the objects as introduced in Section 3.3.

Table 1 includes our model's simulation error and control error with different Koopman matrix structures in Rope. All models are trained in the Rope environment with 5 to 9 masses. Besides the result on the test set, we also report models' extrapolation performance in parentheses, where the model is evaluated on system with more masses than training, i.e., 10 to 14 masses.

Table 1: Ablation study results on the Koopman matrix structure (Rope environment).

	Simulation	Control
Diag	0.052 (0.075)	2.337 (2.809)
None	0.056 (0.043)	1.522 (1.288)
Block	**0.046 (0.041)**	**0.854 (1.101)**

Our model with *Block* structure consistently achieves a smaller error in all settings. *Diag* assumes an overly simplified structure, leading to larger errors and failing to make reasonable controls. *None* has comparable simulation errors but larger control errors. Without structure in the Koopman matrix, it overfits the data and makes the resulting linear dynamics less amiable to the control.

Hyperparameters. In our main experiments, we set the dimension of the Koopman embedding to $m = 32$ per object. Online system identification requires 800 data samples for each training/test case. To understand our model's performance under different hyperparameters, we vary the dimension of the Koopman embedding from 8 to 64 and the number of data samples used for system identification from 200 to 1,600. Figure 4d shows that more data for system identification leads to better simulation results. Figure 4e shows that dimension 16 gives the best results on simulation. It may suggest that the intrinsic dimension of the Koopman invariant space of the Rope system is around 16 per object.

5 Conclusion

Compositionality is common in our daily life. Many ordinary objects contain repetitive subcomponents: ropes and soft robots, as shown in this paper, granular materials such as coffee beans and lego blocks, and deformable objects such as cloth and modeling clay. These objects are known to be very challenging for manipulation using traditional methods, while our formulation opens up a new direction by combining deep Koopman operators with graph neural networks. By leveraging the compositional structure in the Koopman operator via graph neural nets, our model can efficiently manipulate deformable objects such as ropes and soft robots, and generalize to systems with variable numbers of instances. We hope this work could encourage more endeavor in modeling larger and more complex systems by integrating the power of the Koopman theory and the expressiveness of neural networks.

REFERENCES

Ian Abraham, Gerardo De La Torre, and Todd D Murphey. Model-based control using koopman operators. *arXiv preprint arXiv:1709.01568*, 2017.

Hassan Arbabi, Milan Korda, and Igor Mezic. A data-driven koopman model predictive control framework for nonlinear flows. *arXiv preprint arXiv:1804.05291*, 2018.

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *NIPS*, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Daniel Bruder, C David Remy, and Ram Vasudevan. Nonlinear system identification of soft robot dynamics using koopman operator theory. *arXiv preprint arXiv:1810.06637*, 2018.

Daniel Bruder, Brent Gillespie, C David Remy, and Ram Vasudevan. Modeling and control of soft robots using the koopman operator and model predictive control. In *RSS*, 2019.

Steven L Brunton, Bingni W Brunton, Joshua L Proctor, and J Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PloS one*, 11(2):e0150171, 2016.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *ICLR*, 2017.

Herve Delingette. Toward realistic soft-tissue modeling in medical simulation. *Proceedings of the IEEE*, 86(3):512–523, 1998.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

Jessica B Hamrick, Andrew J Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W Battaglia. Metacontrol for adaptive imagination-based optimization. In *ICLR*, 2017.

Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. In *ICLR*, 2019.

Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of koopman eigenfunctions for control. *arXiv preprint arXiv:1707.01146*, 2017.

Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5):315, 1931.

BO Koopman and J v Neumann. Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences of the United States of America*, 18(3):255, 1932.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019a.

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *ICRA*, 2019b.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

Giorgos Mamakoukas, Maria Castano, Xiaobo Tan, and Todd Murphey. Local koopman operators for data-driven control of robotic systems. In *RSS*, 2019.

Alexandre Mauroy and Jorge Goncalves. Linear identification of nonlinear systems: A lifting technique based on the koopman operator. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6500–6505. IEEE, 2016.

Alexandre Mauroy and Jorge Goncalves. Koopman-based lifting techniques for nonlinear systems identification. *arXiv preprint arXiv:1709.02003*, 2017.

David Mayne. Nonlinear model predictive control: Challenges and opportunities. In *Nonlinear model predictive control*, pp. 23–44. Springer, 2000.

Jeremy Morton, Freddie D Witherden, Antony Jameson, and Mykel J Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. *arXiv preprint arXiv:1805.07472*, 2018.

Jeremy Morton, Freddie D Witherden, and Mykel J Kochenderfer. Deep variational koopman models: Inferring koopman observations for uncertainty-aware dynamics modeling and control. In *IJCAI*, 2019.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. In *NIPS*, 2018.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018a.

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018b.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv:1707.06170*, 2017.

Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *NIPS*, 2017.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, pp. 1130–1140, 2017.

Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Matthew O Williams, Maziar S Hemati, Scott TM Dawson, Ioannis G Kevrekidis, and Clarence W Rowley. Extending data-driven koopman analysis to actuated systems. *IFAC-PapersOnLine*, 49 (18):704–709, 2016.

A ADDITIONAL EXPERIMENTS

Ablation study on the sample efficiency. To estimate our model's data-efficiency, we train our model with a smaller training set whose size is $X\%$ of the standard size. In Table 2, we report relative errors, which are our model's simulation errors divided by PN (trained with standard training set)'s error. As we can see, even if with only 5% of training data, our model still outperforms PN trained on 100% data, which demonstrates that our model is more sample efficient.

Table 2: Study on the sample efficiency.

Env	The amount of training data				
	100%	40%	20%	10%	5%
Rope	0.015	0.021	0.028	0.026	0.035
Swim	0.328	0.345	0.406	0.378	0.403
Soft	0.238	0.351	0.409	0.439	0.438

Comparison with a classical physical simulator optimized using back-box optimization. We have performed comparisons with a classical physical simulator optimized using black-box optimization (Delingette, 1998) by assuming different levels of knowledge over the ground truth model.

If we assume that we know the ground truth model, where we only need to identify relevant physical parameters during the system identification stage, Bayesian Optimization (Snoek et al., 2012) (BO) can give us a reasonable estimate of the physical parameters. However, BO requires much more time to achieve comparable performance with our method in the Rope environment: 0.43 vs. 180 seconds averaged over 100 trails (Ours vs. BO).

If we are unsure about the ground truth model and we approximate the system using a set of points linked by springs and dampers, BO might not work as well. In our additional experiments, we approximate the Rope environment using a chained spring-mass system, say n masses and n 1 springs. While taking much more time, BO still cannot give us a satisfying result: simulation error 0.046 vs. 0.084 and control error 0.854 vs. 2.547 (Ours vs. BO).