Dynamic Inference on Graphs using Structured Transition Models

Saumya Saxena[1] and Oliver Kroemer[1]

Abstract— **Modelling and learning the dynamics of intricate dynamic interactions prevalent in common tasks such as pushing a heavy door or picking up an object in one sweeping motion is a challenging problem. One needs to consider both the dynamics of the individual objects and of the interactions among objects. In this work, we present a method that enables efficient learning of the dynamics of interacting systems by simultaneously learning a *dynamic* graph structure and a *stable* and *locally linear* forward dynamic model of the system. The dynamic graph structure encodes evolving contact modes along a trajectory by making probabilistic predictions over the *edge activations*. Introducing a temporal dependence in the learned graph structure enables incorporating contact measurement updates which allows for more accurate forward predictions. The learned stable and locally linear dynamics enable the use of optimal control algorithms such as iLQR for long-horizon planning and control for complex interactive tasks. Through experiments in simulation and in the real world, we evaluate the performance of our method by using the learned interaction dynamics for control and demonstrate generalization to more objects and interactions not seen during training. We also introduce a control scheme that takes advantage of contact measurement updates and hence is robust to prediction inaccuracies during execution.**

I. INTRODUCTION

Common tasks such as pushing a heavy door using the momentum of our bodies or picking up an object while walking past it involve complex dynamic interactions with these objects. To learn how to effectively perform such tasks using a robot, we need to consider the dynamics of the robot, the individual objects, as well as the interactions between them. These learned interaction models can then be used to intentionally utilize dynamic interactions to accomplish complex tasks.

Learning interaction dynamics becomes more and more challenging as the number of interacting bodies in the scene increases, resulting in a combinatorial explosion in the number of interactions to reason about. Recent developments use graph neural networks (GNNs) [1], [2] to model and learn object-centric interaction dynamics of such systems. These networks provide a principled way of approaching this problem by incorporating a *relational inductive bias* in the learned dynamics. However, many GNN based approaches suffer from scaling issues as the number of interactions increases [3] which makes learning the dynamics of interactive tasks harder.

In this work, we develop a method for efficiently learning the dynamics of interacting systems by simultaneously learning a relational *dynamic* graph structure and a *stable locally linear* forward dynamic model of the system. The learned dynamic graph structure encodes information about dynamics and contact modes evolving along the task trajectory by adding and removing edges from the graph as contact is made and broken. This allows for strong generalization of the learned dynamics when more nodes are added to the graph. Starting by representing our system as a fully connected graph, we learn time-varying and state dependent *edge activations* using a *graph inference module*. An edge is inferred to be active when the nodes connected via the edge interact and influence the forward dynamics, thus making it critical for the edge activations and the forward model to be learned together. When a new object is added to the scene, the *graph inference module* can predict whether or not the new nodes influences the system dynamics based on edge predictions. This allows the learned model to generalize more effectively to new objects and new interactions in the environment.

We use the spring-mass-damper model as a structural prior over the *local* object-centric dynamics of our system. This provides us with two benefits: linearity and stability. For positive values of mass, stiffness and damping parameters, the linear spring-mass-damper model is always stable about the equilibrium point (which is also learned) [4]. Assuming a locally linear and stable prior structure over our dynamics model allows us to effectively learn the dynamics and to use the learned dynamics for long-horizon planning and control by precluding the rollout trajectories from growing unbounded. The learned locally linear dynamics also enable the use of optimal control algorithms, such as iLQR, to exploit the interaction dynamics and perform complex tasks optimally by predicting future edge activations and thus contact modes as well.

Control strategies that use a learned model are naturally sensitive to model inaccuracies, especially near the contact regions. This can lead to catastrophic failures during execution. To alleviate this issue, we use gated recurrent units (GRUs) to learn a probabilistic model that predicts edge activations. These units introduce a temporal dependency of the current edge activation on the previous edge activations. During execution, observed contact modes are used to evaluate a posterior over the edge activations and update them which enables more accurate future rollouts. We use model predictive control (MPC) to replan using the observed current state and contact modes.

We explore an exciting application of this work in the field

of apprenticeship learning wherein the learned locally linear model can be used for learning the quadratic cost function underlying expert demonstrations using differentiable LQR as the policy class [5], [6]. The learned behavior can then be generalized to unseen goal conditions.

The key contributions of this work are two-fold: 1) a method for learning stable locally linear dynamics for non-linear interactive systems using graph neural networks by encoding changing dynamics and contacts as part of the graph structure, enabling strong generalization properties to more objects in the scene, and 2) using the learned locally linear dynamics to devise a robust control scheme that utilizes the recurrent nature of the learned graph structure to adapt the model predictions and the policy to observed contact events.

We evaluate our results in simulation on tasks such as multiple object dynamic pickup and dynamic door-opening. We also perform real world experiments using the 7DOF Franka-Emika Panda robot arm for a dynamic pickup task.

II. RELATED WORK

Learning the interaction dynamics of contact-rich tasks is a problem of interest in many areas of robotics research such as manipulation [7], collaborative robotics and assistive systems. In some recent works the dynamics of such interactive systems is learned implicitly [8], [9], while in others [6], [10], [11], [12] the focus is on modelling these changing dynamics as explicit dynamic modes. In our method, we aim to exploit the inherent structure underlying interactions in physical systems by modelling them as graphs and taking advantage of the generalization benefits of graph based approaches.

There has been tremendous progress towards the development of graph neural networks for modelling and learning the forward interaction dynamics of physical systems. Approaches such as [1], [13], [2], [14], [15] learn the dynamics model assuming a static fully connected or known graph structure. Such models can require a large amount of interaction data and training time to accurately learn the interaction dynamics. There has also been work towards using static graphs for planning and control [16], [17], [18]. We focus on the problem of simultaneously learning the dynamic graph structure and the forward dynamic model of the system in a purely unsupervised fashion and using that model for control.

Some approaches [19], [20], [21] for learning the graph structure assume that the graph structure is static for a task, or along a single task trajectory, while other methods learn to actively predict edge interactions [3], [22], [23] using attention mechanisms. We take inspiration from these approaches for learning probabilistic predictions of edge activations that evolve with the state of the system.

Learning to control non-linear systems using locally optimal control algorithms such as iLQR is an exciting area of research. Such methods require learning locally linear and stable dynamics models of the system. Some recent approaches that aim to learn locally linear dynamics of

the system for control include [24], [25], [26], [27] but these approaches have rarely been extended to graph based interactive dynamic systems. We derive inspiration from these works and model our system dynamics as locally linear which enables the use of optimal control algorithms such as iLQR. In our work, we also explore methods developed in [5], [6], [28] for utilizing the learned linearized dynamics for learning simple quadratic cost functions of differentiable controllers such as LQR.

III. METHOD

In this section, we discuss in detail our method for learning stable locally linear dynamics for non-linear interactive systems using graph neural networks. Our training pipeline is composed of two main modules that are trained together, 1) a graph inference module that learns the dynamic graph structure, and 2) a forward dynamics module that learns the stable locally linear forward dynamics given the graph structure.

A. Graph Inference Module

This module predicts the dynamic interaction graph structure — which edges in the dynamic graph are *active* or *inactive* at a certain time step. In our domain where physical objects are represented as nodes in the graph, this module infers whether or not, at a certain time step, objects influence each other's dynamics. For example, in a scenario where a robot gripper picks up an object, an edge connects the gripper and the object only after the object has been grasped.

The state of the system at time t is represented as a graph $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{E}_t)$ where nodes $\mathcal{N}_t$ represent objects in the scene and edges $\mathcal{E}_t$ connect interacting nodes. Node $n_t^i \in \mathcal{N}_t$ is represented using a feature vector $\mathbf{x}_t^i$ where $\mathbf{x}_t^i = [\mathbf{q}_t^i, \dot{\mathbf{q}}_t^i]$. $\mathbf{q}_t^i$ and $\dot{\mathbf{q}}_t^i$ are the position and velocity of the ith object respectively. The edge $e_t^{ij} \in \mathcal{E}_t$ connecting nodes i and j is represented with edge features $\mathbf{d}_t^{ij}$ where $\mathbf{d}_t^{ij}$ is the distance between nodes n_t^i and n_t^j.

An overview of the graph inference module is shown in Fig. 1. We start with a set of trajectories of fully connected graphs $(\mathcal{G}_0, \mathbf{u}_0, \ldots, \mathcal{G}_T, \mathbf{u}_T)$ where $\mathbf{u}_t$ is the control applied at time t, and T is the length of the trajectory. We consider only one actuated node (agent) in all our tasks and share the control effort applied to the agent with all other nodes. Unactuated objects are influenced by the control indirectly via the agent; for example, when an object is picked up by a gripper. This is a design choice and one can apply different actions to all the actuated nodes in the scene and apply zero actions to unactuated objects if desired.

In order to learn the dynamic graph structure, we first use a graph neural network to propagate information spatially over each of the graphs $(\mathcal{G}_0, \ldots, \mathcal{G}_T)$ via message passing between the nodes using a node feature network f_n^{inf} and an edge feature network f_e^{inf}:

$$\mathbf{m}_t^{ij} = f_e^{\mathrm{inf}}(\mathbf{x}_t^i, \mathbf{x}_t^j, \mathbf{d}_t^{ij})$$

$$\mathbf{z}_t^i = f_n^{\mathrm{inf}}(\mathbf{x}_t^i, \sum_j \mathbf{m}_t^{ij})$$
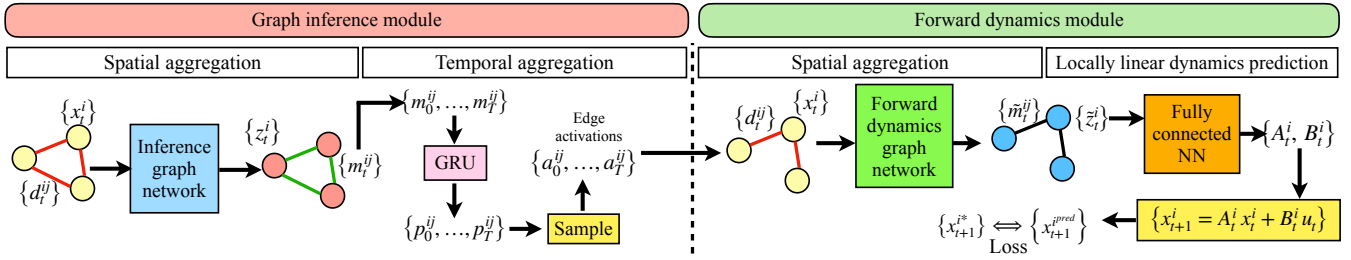
Fig. 1. Method overview: Our method has two main parts 1) Graph inference module: we start with a set of fully connected graphs and pass them through the inference graph network that performs message passing between the nodes and outputs embedded graphs. Next, to aggregate information temporally, we pass the edge embeddings $\mathbf{m}^{ij}_{0,...,T}$ through a GRU that outputs a discrete probability distribution $\mathbf{p}^{ij}_t$ over the edge types (active or inactive) for each edge. We sample from this distribution to get the edge activations $\mathbf{a}^{ij}_t$ (see III-A). 2) Forward dynamics module: using the edge activations, we remove the inactive edges from the fully connected input graphs and then pass them through the forward dynamics graph network that outputs another set of embedded graphs. The node embeddings $\tilde{\mathbf{z}}^{ij}_t$ are passed though a fully connected neural network that outputs stable locally linear transition dynamics (see III-B). We forward propagate the learned dynamics using the input control and get the next state which is used for loss calculation during training (4).

where f^{inf}_e and f^{inf}_n are fully connected neural networks representing the *Inference GNN* and $\mathbf{z}^i_t$ and $\mathbf{m}^{ij}_t$ are the output node and edge embeddings respectively. After two steps of message passing, the output edge embeddings $\mathbf{m}^{ij}_{0:T}$ for each edge are passed though a gated recurrent unit (GRU) to allow temporal information flow:

$$\mathbf{p}^{ij}_{0:T} = \text{GRU}_\phi(\mathbf{m}^{ij}_{0:T})$$
$$\mathbf{a}^{ij}_{0:T} = \text{sample}(\mathbf{p}^{ij}_{0:T}) \tag{1}$$

where ϕ are the trainable parameters of the GRU and $\mathbf{p}^{ij}_t$ is a discrete probability distribution over the edge types. We consider two edge types: active and inactive. Sampling from probability distribution $\mathbf{p}^{ij}_t$ gives us a one hot vector representing the *edge activation* $\mathbf{a}^{ij}_t$. If an edge is active at time t, $\mathbf{a}^{ij}_t = [1,0]$ else $\mathbf{a}^{ij}_t = [0,1]$. Naturally, this representation can be extended to include more than two edge types. Differentiability of this sampling procedure can be ensured by employing techniques such as Gumbel softmax [29]. If an edge e^{ij}_t is sampled to be inactive it is removed from the graph $\mathcal{G}_t$ for the subsequent calculations and no messages pass through that edge. The output of the graph inference module is a trajectory of graphs $(\tilde{\mathcal{G}}_0, \ldots, \tilde{\mathcal{G}}_T)$ where $\tilde{\mathcal{G}}_t = (\mathcal{N}_t, \tilde{\mathcal{E}}_t)$ and $\tilde{\mathcal{E}}_t \subseteq \mathcal{E}_t$ is the set of active edges. Given this inferred graph structure, our next step is to model the forward dynamics of the system.

B. Forward Dynamics Module

An overview of the forward dynamics module is shown in Fig. 1. To learn the object-centric interaction dynamics of the system, we use another GNN to perform two steps of message passing over each of the graphs $(\tilde{\mathcal{G}}_0, \ldots, \tilde{\mathcal{G}}_T)$ to propagate information spatially between connected nodes.

$$\tilde{\mathbf{m}}^{ij}_t = f^{\text{dyn}}_e(\mathbf{x}^i_t, \mathbf{x}^j_t, \mathbf{d}^{ij}_t)$$
$$\tilde{\mathbf{z}}^i_t = f^{\text{dyn}}_n(\mathbf{x}^i_t, \sum_j \tilde{\mathbf{m}}^{ij}_t)$$

where f^{dyn}_e and f^{dyn}_n are fully connected neural networks representing the *Forward Dynamics GNN*. Here, for scenarios where more than two edge types are considered, for

example to capture varied behavior over active edges, one can use a different edge feature network f^{dyn}_e for each edge type. The output node embeddings $\tilde{\mathbf{z}}^i_t$ now encode interaction information between the connected nodes and can be used to learn the forward dynamic parameters of the interactive system independently for each node. To learn stable locally linear dynamics, we approximate the dynamics of each node to resemble a spring-mass-damper system, the parameters of which are derived from the node embeddings at each time step. This representation was chosen since the dynamics of a spring-mass-damper system can be constrained to be stable by constraining the mass, stiffness and damping parameters to be greater than zero [4]. This is easily achieved by using activations such as sigmoid and relu. We use a fully connected neural network g^{dyn} to output the dynamic parameters for each node as,

$$\boldsymbol{\alpha}^i_t = g^{\text{dyn}}(\tilde{\mathbf{z}}^i_t) \tag{2}$$

where $\boldsymbol{\alpha}^i_t = [\frac{1}{m^i_t}, k^i_t, c^i_t, \hat{x}^i_t]$ where m^i_t is the mass, k^i_t is the stiffness of the spring, c^i_t is the damping, $\hat{x}^i_t$ is the equilibrium point. These parameters can then be used to write the stable locally linear dynamics of the system as:

$$\mathbf{x}^i_{t+1} = \mathbf{A}^i_t \mathbf{x}^i_t + \mathbf{B}^i_t \mathbf{u}_t + \mathbf{o}^i_t \tag{3}$$

where

$$\mathbf{A}^i_t = \begin{bmatrix} 1 & dt \\ \frac{-k^i_t\,dt}{m^i_t} & 1 - \frac{c^i_t\,dt}{m^i_t} \end{bmatrix}, \mathbf{B}^i_t = \begin{bmatrix} 0 \\ \frac{dt}{m^i_t} \end{bmatrix}, \mathbf{o}^i_t = \begin{bmatrix} 0 \\ \frac{k^i_t\,\hat{x}^i_t\,dt}{m^i_t} \end{bmatrix}$$

for a discrete-time one degree of freedom system.

C. Training

The graph inference module and the forward dynamics module are trained together to simultaneously learn the dynamic graph structure and the locally linear forward dynamics. We collect a set of N trajectories containing multiple interacting objects and represent them as graphs $\{(\mathcal{G}^*_0, \mathbf{u}_0, \ldots, \mathcal{G}^*_{T-1}, \mathbf{u}_{T-1})_n\}_{n=0,\ldots,N}$. The loss function for training the model is written as,

$$\mathcal{L}_{\text{dyn}} = \sum_i ||\mathbf{x}^{i*}_{1:T} - \mathbf{x}^{i\text{pred}}_{1:T}||_2 + \sum_{i,j} \text{KL}[\mathbf{p}^{ij}_{0:T-1} \,||\, \mathbf{q}^{ij}_{0:T-1}] \tag{4}$$

where $\mathbf{x}^{i*}_{t+1}$ is the observed next state and $\mathbf{x}^{i\,\mathrm{pred}}_{t+1} = \mathbf{A}^i_t \mathbf{x}^{i*}_t + \mathbf{B}^i_t \mathbf{u}_t + \mathbf{o}^i_t$ is the predicted next state for node n^i. $\mathbf{p}^{ij}_{0:T-1}$ is the probability distribution over predicted edge activations and $\mathbf{q}^{ij}_{0:T-1}$ is a prior on the edge activations. In prior work [19], [20], where the focus is on learning a static graph structure, the prior is designed such that a sparse graph is learned. However, in our scenario, since the graph structure is not static and can be sparse or dense depending upon the state, we design a prior using the relative distances between the nodes. In particular, when two nodes are spatially close to each other, the prior suggests that the probability of the edge connecting these nodes being active, is high. In particular, for a given distance threshold of d_{th}, the prior over edge e^{ij}_t at time t is given by:

$$\mathbf{q}^{ij}_t = \mathrm{Softmax}\left(\begin{bmatrix} d_{th} - d^{ij}_t \\ d^{ij}_t - d_{th} \end{bmatrix}\right)$$

where the edge feature d^{ij}_t is the distance between nodes n^i_t and n^j_t. The neural networks f^{inf}_e, f^{inf}_n, GRU_ϕ, f^{dyn}_e, f^{dyn}_n and g^{dyn} are trained simultaneously using stochastic gradient descent to minimize the loss $\mathcal{L}_{\mathrm{dyn}}$.

D. Execution

While it is common practice to use model predictive control for replanning using the current observed state, for interactive tasks where contacts are critical, it is essential to utilize observed contact information for replanning as well. Since we use GRU to learn the edge activations, it introduces a temporal dependency of the current edge activation on the previous edge activations. For one time step prediction, this recurrent process looks as follows, $\mathbf{p}^{ij}_{t+1} = \mathrm{GRUCell}(\mathbf{p}^{ij}_t, \mathbf{d}^{ij}_t)$. During execution, using the observed contact modes, we update the current probabilistic edge activations by calculating a posterior on the edge activations as follows,

$$\tilde{\mathbf{p}}^{ij}_t = z\, \mathbf{p}^{ij}_t\, \mathbf{c}^{ij}_t$$

where $\mathbf{c}^{ij}_t$ is a discrete probability distribution representing the observed contact between node n_i and n_j at time t and z is the normalization constant. Since GRU aggregates past information for future rollouts, the updated edge activation $\tilde{\mathbf{p}}^{ij}_t$ allows for more accurate future predictions. In particular, we use the posterior edge activations $\tilde{\mathbf{p}}^{ij}_t$ to make updated future predictions as follows, $\mathbf{p}^{ij}_{t+1} = \mathrm{GRUCell}(\tilde{\mathbf{p}}^{ij}_t, \mathbf{d}^{ij}_t)$.

IV. EXPERIMENTS

The proposed method is evaluated in simulation using Box2D and Isaac Gym environments and in the real-world on the 7DOF Franka-Emika Panda robot arm. Through our experiments, we aim to evaluate 1) How useful is our graph inference module in enabling efficient and accurate learning of the dynamics of the interactive system? 2) What are the generalization properties of the learned dynamics model as more objects (not seen during training) are added to the system? 3) How robust is our execution strategy to inaccuracies in the learned model? 4) Can our learned stable linearized dynamics be used for learning the cost function of a differentiable LQR controller using expert demonstrations of a task?

Ablation studies: To answer the first three questions we perform the following three ablation studies: 1) Training the dynamic model without the graph inference module i.e. learning the forward dynamics module given fully connected graphs. The dataset for this study is the same as the dataset used for training the full model. We call this model **No-GIM**. 2) Training the dynamic model without the graph inference module but with an augmented dataset containing more objects and interactions. We call this model **No-GIM-Aug**. 3) Training the full dynamic model with the graph inference module but replacing the GRU in (1) with a fully connected neural network that takes as input only the current edge embedding $\mathbf{m}^{ij}_t$ to output the edge activation i.e. there is no temporal relationship between the predicted edge activations. We call this model **GIM-Non-Temp**. Due to this modification in the network architecture, we will no longer be able to utilize the robust control scheme elaborated on in III-D. We call our model **GIM-Temp**. In addition to the above ablation studies, we also train a model that uses our graph inference module but instead of using a spring-mass-damper system based stable dynamics model in the forward dynamics module, learns the full $\mathbf{A}$ and $\mathbf{B}$ matrices from scratch.

Application: Apprenticeship learning. To answer question 4, given the learned stable locally linear interaction dynamics, we learn the cost matrices $\mathbf{Q}$ and $\mathbf{R}$ of a discrete-time linear quadratic regulator (LQR) using expert demonstrations. We demonstrate generalization of learned expert behavior to goal regions not visited by the expert. We closely follow the cost learning method in [6], but instead of learning a different cost function and goal condition for each dynamic *mode*, modeled as a discrete global variable, we learn a single cost function for a certain task with a fixed goal.

Evaluation metrics: To test for generalization, all the ablation models mentioned above, including our model, are tested with more objects and interactions than seen during training. For evaluating the accuracy of the learned model, the generalization properties and the effectiveness of the robust control scheme, we use N-step prediction error as the metric. This error is calculated upon using the learned locally linear models for control using iLQR-MPC (receding horizon control) and calculating the root mean square error between the predicted trajectory and the executed trajectory for a rollout length of N. For each environment, N is calculated such that the rollout time is 0.5 seconds i.e $N = 0.5/dt$. The prediction error is calculated over all the nodes in the graph. For **GIM-Non-Temp**, in addition to N-step prediction error, we also evaluate the accuracy of the predicted edge activations for a rollout length of N and compare it with our method **GIM-Temp**. We get the ground truth edge activations by observing contacts during execution. Contact information is not used during training. For **No-GIM-Aug**, we also present results for the sample complexity, i.e how much more interaction data is needed in order to achieve similar level of performance as with our method. This will
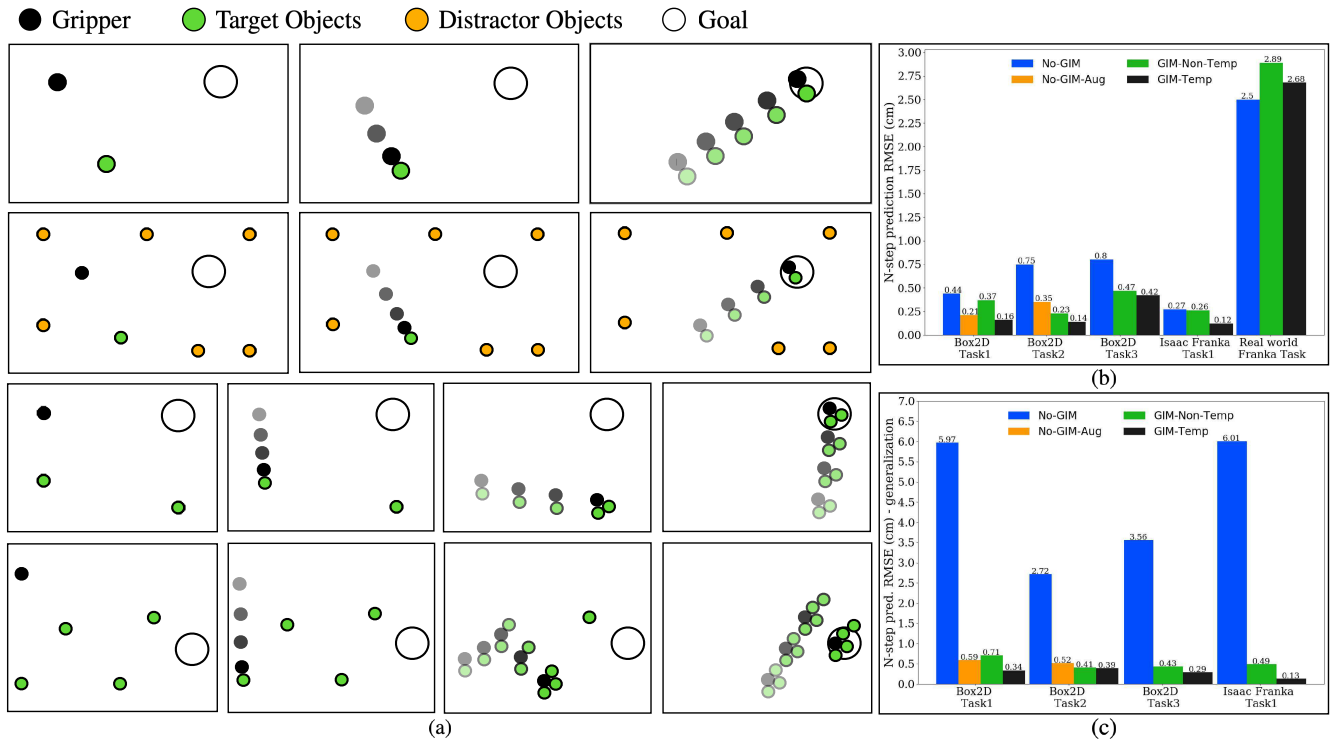
Fig. 2. (a) Row 1: Box2D Task 1 with same number of objects as seen during training. Row 2: Box2D Task 1 generalization scenario with many distractor objects (orange) in the scene. Row 3: Box2D Task 2 with same number of objects as seen during training. Row 4: Box2D Task 2 generalization scenario for picking up many more objects than during training. (b) N-step root mean square error between the predicted and executed trajectories. These experiments are performed with the same number of objects and interactions as seen during training. (c) N-step root mean square error between the predicted and executed trajectories. These experiments are performed to test for generalization by including many more objects and interactions in the task than seen during training.

demonstrate the efficiency of our method as compared to models where the graph inference module is not used.

Fig. 2(b) shows the N-step prediction error when the learned models are tested in environments with the same number of objects and interactions as seen during training. Fig. 2(c) shows the N-step prediction error for generalization scenarios i.e. testing with more interactions and more distractor objects in the scene. Fig. 3(b) shows the N-step edge activation prediction accuracy.

Box2D Tasks: We consider three tasks in this environment: **Task 1:** Dynamically (without stopping) picking up a 2D object using a 2D gripper and taking it to a goal region (Fig. 2(a) Rows 1 & 2), **Task 2:** Dynamic pickup of multiple objects (Fig. 2(a) Rows 3 & 4), and **Task 3:** Dynamic door opening (Fig. 3(a)). For task 1, **No-GIM**, **GIM-Non-Temp** and **GIM-Temp** are trained on a dataset consisting of 40 trajectories of the gripper picking up *one* target objects and taking it to goal location. Initial positions of the gripper and object and goal locations are sampled to collect data. **No-GIM-Aug** is trained with an augmented dataset consisting of up to three distractor objects which do not interact (160 trajectories). For task 2, **No-GIM**, **GIM-Non-Temp** and **GIM-Temp** are trained on a dataset consisting of 40 trajectories of the gripper picking up *two* target object and taking it to a goal. **No-GIM-Aug** is trained with an augmented dataset consisting of trajectories where the

gripper picks up a maximum of three objects and takes them to a goal (120 trajectories). For task 3, **No-GIM**, **GIM-Non-Temp** and **GIM-Temp** are trained on a dataset consisting of 40 trajectories of the gripper dynamically opening a door. All these learned models are then tested for generalization. For task 1, we test in environments with up to ten distractor objects. For task 2, we test for generalization by picking up up to five objects. For task 3, we test in environments with up to ten distractor doors.

Isaac Franka Task: We consider the task of dynamically picking up an object of mass 0.5kg using a 7DOF Franka-Emika Panda arm and taking it to a goal location (Fig. 4(a) and (b)). The robot is controlled using task space impedance control [30] during data collection wherein the orientation of the gripper is kept fixed. The initial cartesian position of the gripper, object (on a table) and goal are sampled for data collection. **No-GIM**, **GIM-Non-Temp** and **GIM-Temp** are trained on a dataset consisting of 200 trajectories of the gripper picking up the target object and taking it to a goal. A cartesian space model of the robot is learned while assuming fixed orientation. We test for generalization in environments with up to five distractor objects.

Real robot experiments: Real world experiments are performed for the task of dynamically picking up an object of mass 0.83kg and taking it to a goal location using a 7DOF Franka-Emika Panda robot arm (Fig. 4(c)). The setup,
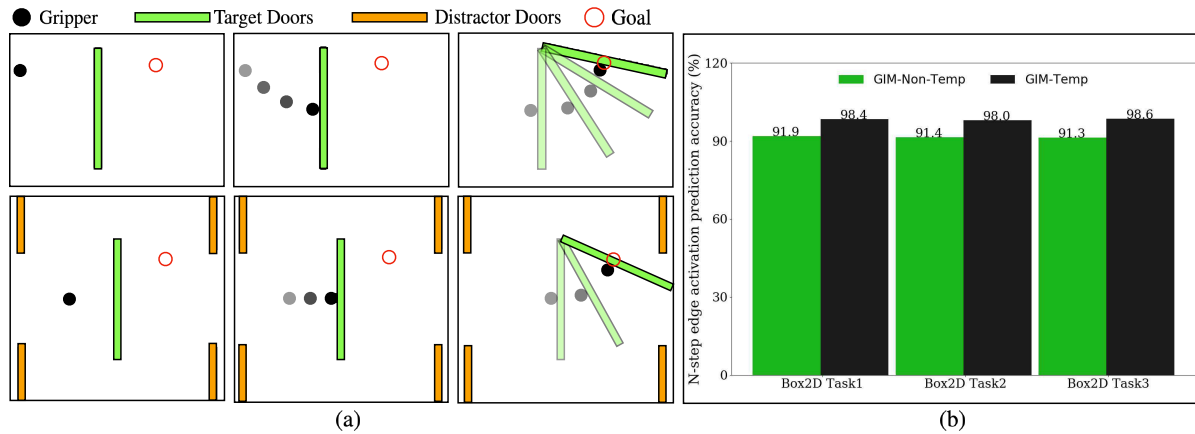
Fig. 3. (a) Row 1: Box2D Task 3 with same number of doors as seen during training. Row 2: Box2D Task 3 generalization scenario with many distractor doors in the scene. (b) N-step edge activation prediction accuracy.

controller, data collection and training strategy is the same as for the Isaac Gym Franka pickup task. The location of the object was tracked using April tags and a Kinect depth sensor [31]. Due to lack of reliable contact sensing, we do not update edge activation predictions using observed contacts during execution but still rely solely on GRU for edge activation predictions. **No-GIM**, **GIM-Non-Temp** and **GIM-Temp** are trained on a dataset of 25 trajectories.

Training parameters and network architecture: All the models are trained using ADAM [32] with default parameters and a learning rate of $1e-5$. The inference GNN and forwards dynamics GNN are composed of two fully connected neural networks each having two layers with 64 units each and ReLU activation. g^{dyn} in (3) is also a fully connected neural network having two layers with 64 units each and ReLU activation. We use *GRUCell* in PyTorch [33] and apply a softmax to its output for learning the edge activation probabilistic model. The reader is encouraged to refer to the supplementary material for additional details regarding the experiments and results (link).

V. RESULTS AND DISCUSSION

For each of the tasks, we use the learned dynamic models for control using iLQR-MPC and calculate the N-step prediction error as shown in Fig. 2(b & c). We observe that, when tested with the same number of object interactions as seen during training, the N-step prediction error for all the models trained using simulation data is less than a centimeter and around 2.5cm for the real world Franka task (Fig. 2(b)). However, when tested for generalization, the performance of **No-GIM** deteriorates significantly across all the tasks as shown in Fig. 2(c). For the Isaac Gym Franka task, control using the learned model is not able to successfully pick up the object in many trials when distractor objects are present in the scene. This is because in the absence of the graph inference module, **No-GIM** cannot infer that edges to additional objects that do not interact can be rendered inactive. This model also fails to infer the implicit mechanism behind the dynamic interaction that occurs when an edge becomes

active and hence cannot generalize to additional objects. We also observe that **No-GIM-Aug** generalizes well and that **No-GIM-Aug** and **GIM-Non-Temp** have comparative performance. Training with augmented data consisting of multiple interactions and distractor objects helps **No-GIM-Aug** ignore distractor objects and generalize to more interactions, however, this comes at the cost of three to four times more data and longer training time. In comparison, **GIM-Non-Temp** and our model **GIM-Temp**, trained on the same dataset as **No-GIM** generalize well to scenarios involving more interactions and distractor objects that seen during training. Overall, the N-step prediction performance of **GIM-Temp** exceeds all the other models in all tasks except for the real world Franka task. Since we do not use contact information for updating the edge predictions for the real world Franka task, the performance of our method, compared to other ablation studies, is not as good as in the other tasks. However, using iLQR-MPC with our learned linear model, we were able to perform the pickup task successfully as shown in Fig. 4(c). This demonstrates the benefit of learning a dynamic graph structure that encodes temporal dependence in the edge activations and of using a robust execution strategy that adapts to observed contact events. This can be quantitatively observed in Fig. 3(b). Both **GIM-Non-Temp** and our model **GIM-Temp** show interpretable results where edges were learned to be active only between objects and agents in contact. However, the edge activation prediction accuracy of our model is higher as compared to **GIM-Non-Temp**. This is because **GIM-Non-Temp** shows inaccuracies in regions close to contact and, in the absence of a mechanism for updating future predictions based on contact observations, its performance suffers.

For the model trained using our graph inference module but instead of using a spring-mass-damper system based stable dynamics model learns the full **A** and **B** matrices from scratch, we observed that even though the model trained to convergence, using this learned model for LQR was infeasible. This is because, without any constraints or prior structure in **A** and **B** ensuring stability, the learned
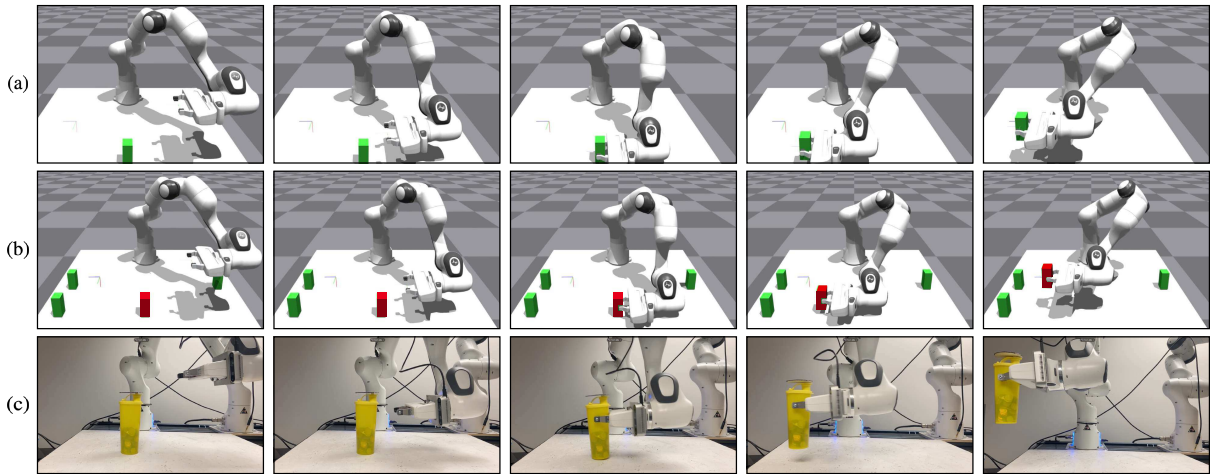
Fig. 4. (a) Isaac Gym Task with same number of objects as seen during training. (b) Isaac Gym Task generalization scenario with distractor (green) objects in the scene. (c) Real world experiments with the 7DOF Franka-Emika Panda arm performing a dynamic pickup task using our learned model.

model causes the predicted trajectories to become unstable, resulting in unbounded behavior upon forward rollout.

We also trained a model similar to our model **GIM-Temp** and using the same data but without including the KL divergence loss between the predicted edge activations and the distance-based prior in the loss function (4). We observe that without an informative prior, the model tends to quickly fall into a local minima where the graph stays fully connected for all time steps. This results in a model learned similar to **No-GIM** with poor generalization performance.

Finally, we use our learned model **GIM-Temp** to learn the quadratic cost matrices of a LQR controller using the method described in [6] for the Box2D pickup task 1. We learn the cost function underlying this dynamic task using expert demonstrations collected by sampling the initial position of the gripper but keeping the goal and object locations fixed. Our aim is to evaluate if the learned controller mimics the behavior of the expert and if it generalizes to unseen goal and object locations. We evaluate the performance of the learned controller using task success as the metric. If the gripper is able to pick up the object and take it within a goal region in time T then the task is considered successful. We observe that the learned controller can perform the task with 98% success rate.

VI. LIMITATIONS AND FUTURE WORK

One of the limitations of our method is that it cannot, in the present state, incorporate high-dimensional sensor inputs such as images. Similar to prior work [24], [25], [26], [27], we plan to extend our method to end-to-end learn a low-dimensional feature vector from sensory inputs along with the dynamic graph structure and the forward dynamic model. Using long-horizon predictions to reason about occluded or invisible objects is another exciting direction for future work [15]. The dependence of our control strategy on reliable contact measurement is another direction of improvement, since reliable contact measurements are not always available. One direction that can be considered is using the approach

in [34] to evaluate the probability of seeing the observed transition for all possible edge activations and use that to calculate the posterior. Another exciting future direction is learning a graph network based LQR controller that encodes relative costs between connected nodes and a goal node and implicitly infers the combined cost over tasks involving varying number of objects.

VII. CONCLUSION

In this work, we demonstrated the benefits of learning the forward model of interactive systems by simultaneously learning a dynamic interaction graph and a stable locally linear forward dynamic model given the graph. We showed that using a spring-mass-damper system as an approximation to the local dynamics allows us to effectively learn the parameters of a stable system that can be used for long-horizon planning. We then elaborated on the generalization benefits of learning a dynamic graph structure over using a fully connected graph and demonstrated that by introducing a temporal dependency of the current edge activation on the previous edge activations, we allow for contact measurement updates, ensuring more accurate future predictions. As an application of our work, we also discussed how our learned model can be used to learn the quadratic cost function underlying expert demonstrations using the method described in [5], [6]. The learned behavior can then be generalized to unseen goals.

REFERENCES

[1] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," *Advances in neural information processing systems*, vol. 29, 2016.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[3] Y. Hoshen, "Vain: Attentional multi-agent predictive modeling," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[4] K. J. Åström and R. M. Murray, *Feedback systems, Chapter 8.* Princeton university press, 2010.

[5] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *Advances in neural information processing systems*, vol. 31, 2018.

[6] S. Saxena, A. LaGrassa, and O. Kroemer, "Learning reactive and predictive differentiable controllers for switching linear dynamical models," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7563–7569.

[7] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges," *Representations, and Algorithms.(2019)*, 2019.

[8] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik, "Learning visual predictive models of physics for playing billiards," *arXiv preprint arXiv:1511.07404*, 2015.

[9] A. Lerer, S. Gross, and R. Fergus, "Learning physical intuition of block towers by example," in *International conference on machine learning*. PMLR, 2016, pp. 430–438.

[10] S. A. Khader, H. Yin, P. Falco, and D. Kragic, "Data-efficient model learning and prediction for contact-rich manipulation tasks," *IEEE robotics and automation letters*, vol. 5, no. 3, pp. 4321–4328, 2020.

[11] M. Toussaint and S. Vijayakumar, "Learning discontinuities with products-of-sigmoids for switching between local models," in *Proceedings of the 22nd international conference on Machine Learning*, 2005, pp. 904–911.

[12] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1503–1510.

[13] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8459–8468.

[14] Z. Weng, F. Paus, A. Varava, H. Yin, T. Asfour, and D. Kragic, "Graph-based task-specific prediction models for interactions between deformable and rigid objects," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 5741–5748.

[15] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti, "Visual interaction networks: Learning a physics simulator from video," *Advances in neural information processing systems*, vol. 30, 2017.

[16] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4470–4479.

[17] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," in *International conference on learning representations*, 2018.

[18] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu, "Reasoning about physical interactions with object-oriented prediction and planning," *arXiv preprint arXiv:1812.10972*, 2018.

[19] Y. Li, A. Torralba, A. Anandkumar, D. Fox, and A. Garg, "Causal discovery in physical systems from videos," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9180–9192, 2020.

[20] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2688–2697.

[21] F. Alet, E. Weng, T. Lozano-Pérez, and L. P. Kaelbling, "Neural relational inference with fast modular meta-learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[22] S. Van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber, "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions," *arXiv preprint arXiv:1802.10353*, 2018.

[23] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf, "Recurrent independent mechanisms," *arXiv preprint arXiv:1909.10893*, 2019.

[24] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," *Advances in neural information processing systems*, vol. 28, 2015.

[25] E. Banijamali, R. Shu, H. Bui, A. Ghodsi *et al.*, "Robust locally-linear controllable embedding," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 1751–1759.

[26] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine, "Solar: Deep structured representations for model-based reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7444–7453.

[27] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," *Advances in neural information processing systems*, vol. 27, 2014.

[28] H. Yin, M. C. Welle, and D. Kragic, "Policy learning with embedded koopman optimal control," 2021.

[29] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[30] K. Zhang, M. Sharma, J. Liang, and O. Kroemer, "A modular robotic arm control stack for research: Franka-interface and frankapy," *arXiv preprint arXiv:2011.02398*, 2020.

[31] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4193–4198.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[34] N. Sugimoto, J. Morimoto, S.-H. Hyon, and M. Kawato, "The emosaic model for humanoid robot control," *Neural Networks*, vol. 29, pp. 8–19, 2012.