2024 IEEE International Conference on Robotics and Automation (ICRA)
May 13-17, 2024. Yokohama, Japan

Towards Robotic Tree Manipulation: Leveraging Graph Representations

Chung Hee Kim, Moonyoung Lee, Oliver Kroemer, George Kantor

2024 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-8457-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICRA57147.2024.10611327

***Abstract*— There is growing interest in automating agricultural tasks that require intricate and precise interaction with specialty crops, such as trees and vines. However, developing robotic solutions for crop manipulation remains a difficult challenge due to complexities involved in modeling their deformable behavior. In this study, we present a framework for learning the deformation behavior of tree-like crops under contact interaction. Our proposed method involves encoding the state of a spring-damper modeled tree crop as a graph. This representation allows us to employ graph networks to learn both a forward model for predicting resulting deformations, and a contact policy for inferring actions to manipulate tree crops. We conduct a comprehensive set of experiments in a simulated environment and demonstrate generalizability of our method on previously unseen trees. Videos can be found on the project website: https://kantor-lab.github.io/tree_gnn**

I. Introduction

In response to labor shortages in agriculture, there is growing interest in adopting labor-saving mechanization, notably robotics, for automating agricultural tasks [1]. Many of these tasks such as harvesting, pruning, and inspecting involve contact interactions like pushing or pulling on branches to reveal obstructed objects as illustrated in Fig 1. However, the deformable nature of tree branches presents a significant challenge for robot manipulation. Deformable objects require characterization in a higher-dimensional configuration space to accurately represent their states. Consequently, modeling the geometry and dynamics of branches becomes a non-trivial task [2]. Even more challenging is the task of selecting actions for manipulating trees, such as determining optimal contact points and the appropriate direction for perturbation.

One approach to modeling tree branch deformation under applied force is to use Finite Element Analysis (FEA) [3]. While FEA allows precise offline analysis, real-time deployment poses difficulties and demands an accurate 3D model beforehand. Alternatively, we use a method that approximates tree motions through kinematics and dynamics models. Past studies have constructed geometric models of trees with rigid links articulated by spring-damper joints [4] to determine the equations of motion. We adopt this tree modeling approach to simulate data of the robot interacting with the tree. With this dataset, we train a *forward model* which approximates the resulting state of a deformed tree given its initial state and applied action, as well as a *contact policy* which generates a set of candidate actions to be applied given the tree's initial state and target state. A key aspect of our approach is representing the physical structure of a tree as well as its kinematic and dynamic parameters as a graph. This approach is particularly intuitive as it involves a straightforward conversion of the tree geometry in the form of a Euclidean graph. Moreover, this enables the usage of Graph Neural Networks (GNNs), known for their inherent inductive bias properties that allow them to learn latent relationships between nodes and edges in a graph. This is especially beneficial for tasks like ours, where understanding the interaction between different components in the tree's structure is crucial for accurate modeling and prediction.

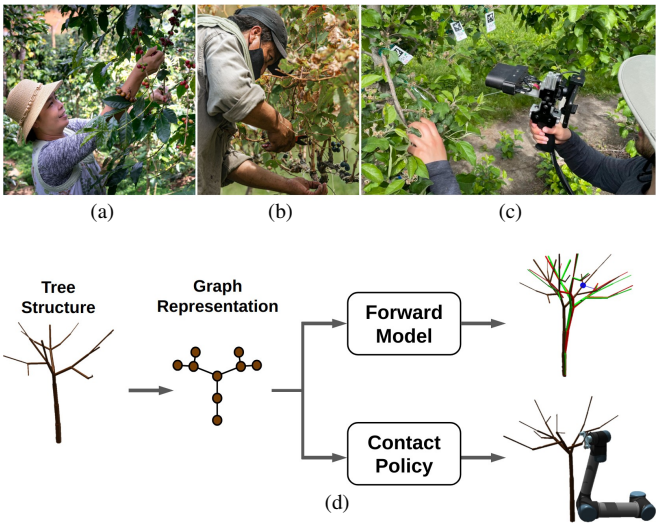


Fig. 1: Laborious agricultural tasks often require delicate contact interaction with crops. For example, a worker is (a) harvesting, (b) pruning, and (c) inspecting [5] crops with their right hand, while pulling/pushing on branches with their left hand. (d) Overview of our proposed framework for learning the deformation behavior of tree-like crops under contact interaction.

The goal of this study is to establish a framework for modeling trees in the context of learning crop manipulation. Through this line of research, we aim to contribute towards agricultural task automation that demands precise contact interaction with crops. The key contributions of this paper are:

- Novel representation of tree crops as graphs, facilitating a GNN-based tree model trained in simulation using mass-spring-damper tree models.
- GNN-based policy that infers actions for manipulating trees towards target states while maintaining adaptability to previously unseen tree structures.
- Validation of models and policies in a series of experiments and ablation studies conducted in a simulation environment.

C. H. Kim, M. Lee, O. Kroemer and G. Kantor are with Carnegie Mellon University, Pittsburgh PA, USA {chunghek, moonyoul, okroemer, kantor}@andrew.cmu.edu

This work was supported in part by NSF Robust Intelligence 1956163, and NSF/USDA-NIFA AIIRA AI Research Institute 2021-67021-35329.

II. Related Work

There are several previous studies on automating agricultural tasks with varying degrees of interactions with crops. For instance, [6]–[8] automates pruning tasks by treating branches as rigid collisions and avoids interacting with crops except with the cutting tool. Yandun et al. [9] leverages reinforcement learning by penalizing collisions with grapevines to reach pruning locations. Drawing inspiration from how humans use both arms to unclutter crops from leaves, dual-arm robotic systems have been proposed for crop harvesting [10]–[12]. As leaves are soft, this degree of interaction can be heuristically addressed [13] without explicitly modeling interaction dynamics to plan robot actions. In our work, we propose a general method aimed at intentionally manipulating tree-like crops, with the goal of automating agricultural tasks.

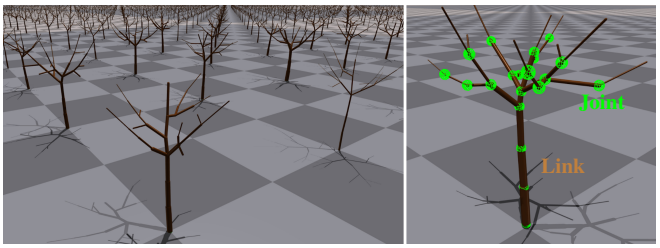
To automate tasks that involve physical interactions with crops, it is essential to understand and model its dynamics. 3D reconstructed crops are often used to reason about their responses to external influences [4], [14], [15]. One common approach to modeling tree dynamics is applying equations of motions based on the spring-damper system [16]. Yandun et al. [4] modeled joint-connected branches as a spring-damper system to predict deformation of trees subjected to external forces. Spatz and Theckes [17] studied the damped oscillating behavior of trees when blown by wind. Jacob et al. [18] also modeled trees in simulation as a mass-spring-damper system to estimate system parameters from branch trajectories obtained through active probing. Our work uses the same simulation setup proposed in [18], however, our framework leverages graph representations to learn forward models and manipulation policies.

We use graph neural networks to learn the dynamical behavior of tree-like crops. GNNs are a class of neural networks designed to operate on graphs, enabling them to propagate and aggregate information between nodes and edges to capture latent relationships [19]–[21]. Building upon this foundation, Battaglia et al. [22] demonstrated that GNN models are capable of reasoning about how objects interact within complex systems. They achieve this by making dynamical predictions and inferences regarding system parameters. Furthermore, Sanchez-Gonzalez et al. [23] extended the application of GNNs to simulate complex particle physics, and proposed the use of GNNs as learnable physics engines for dynamical systems [24]. In our work, we build upon the framework introduced by Sanchez-Gonzalez et al. by utilizing their graph2graph GNN layer [24] as the backbone of our model. This allows us to effectively capture the steady-state behavior of tree-like crops, as well as learning a policy for contact manipulation.

III. Methodology

The following section describes our simulation environment and our approach to learning the forward model to predict how trees deform and the contact policy for manipulating tree-like crops.

A. Simulation Environment



(a) (b)

Fig. 2: Example of our tree crop model visualized in simulation. (a) Varying tree sizes and structures are generated in a parallelized environment. (b) The tree is modeled as a series of rigid links (brown) articulated by spring-damper joints (green).

We utilize NVIDIA Isaac Gym [25] to simulate deformation of multiple tree structures concurrently as visualized in Fig. 2(a). To generate tree structures in simulation that reflect physical tree growth patterns, we use the Space Colonization Algorithm (SCA) as proposed in [26]. The SCA randomly distributes attraction points while iteratively linking tree skeletons to emulate the competition for space between branches that naturally occur in trees. To simulate realistic tree movements, we assume that the kinematics and dynamics of trees can be approximated through a series of rigid links as in [4]. These links are connected by spherical joints characterized as spring-damper systems. Specifically, branches are represented as cylindrical links, and spherical joints interconnect two or more branches within the tree's structure (see Fig. 2(b)). The motion of branches is simulated using a proportional-derivative (PD) controller, governed by adjustable stiffness and damping parameters based on the spring-damper model:

$$\tau = K_s\theta_p + K_d\theta_v \tag{1}$$

where τ represents the applied torque, K_s and K_d are the stiffness and damping coefficients respectively, θ_p is the angular position error, and θ_v is the angular velocity error.

We populate the stiffness coefficients K_s based on the beam deflection model [27]:

$$\theta = \frac{F\ell^2}{2EI} \tag{2}$$

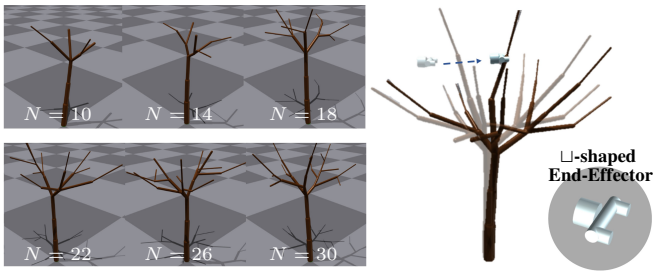
where θ is the angular deflection, F is the applied force, ℓ is the length of the beam, E is the elastic modulus of the material, and I denotes the second moment of area of the cross-section of the beam. Assuming that a tree with circular branch cross-sections undergoing deformation has reached steady-state, substituting equation (2) into equation (1) yields:

$$K_s = \frac{E\pi r^4}{2\ell} \tag{3}$$

where r and ℓ denotes the cross-sectional radius and length of the branch (generated from SCA), respectively. The damping coefficients are empirically set to be $K_d = K_s/10$. The topology of the tree, as well as the control (K_s, K_d) and geometric (r, ℓ) parameters are stored and loaded using the Unified Robotics Description Format.

B. Dataset Collection

We abstract the physical structure of a tree as a directed graph denoted by $G = (\mathbf{g}, \{\mathbf{n}_j\}_{j=1\cdots N}, \{\mathbf{e}_{ij}\})$, where $\mathbf{g}$ is a vector of global attributes, $\{\mathbf{n}_j\}_{j=1\cdots N}$ consists of N nodes representing branch locations in the tree with node attributes $\mathbf{n}_j$, and $\{\mathbf{e}_{ij}\}$ consists of edges representing cylindrical branches connecting parent node $\mathbf{n}_i$ and child node $\mathbf{n}_j$ with attributes $\mathbf{e}_{ij}$. Using our tree generation pipeline (Sec. III-A), we generate 100 unique tree topologies for each tree size ranging from 10 to 30 nodes (with increments of 1) as illustrated in Fig. 3(a). The height of the trees range between 0.6m and 1.7m. In order to simulate and acquire data pertaining to tree contact manipulation, we employ an unconstrained ⊔-shaped rigid end-effector that pushes the tree. The end-effector initiates contact with a tree node and subsequently follows a linear trajectory along the direction of the wrist as shown in Fig. 3(b). The selection of the contact node, as well as the direction and distance of the end-effector trajectory are sampled from a uniform distribution.



(a) (b)

Fig. 3: Visualization of the dataset collection. (a) Trees with varying sizes and randomized structures are generated as discussed in Sec. III-A. (b) A ⊔-shaped end-effector makes contact and pushes the tree by tracking a linear trajectory. The initial tree state, end-effector trajectory, contact node, and the final tree state is recorded for each push interaction episode.

Each data point is composed from a push interaction episode by recording the initial node absolute positions $\mathbf{p}_j = [p_x, p_y, p_z]$, displaced final node absolute positions $\mathbf{p}'_j = [p'_x, p'_y, p'_z]$, end-effector trajectory $\mathcal{T} = [dx, dy, dz]$, contact node flag c_j, and stiffness coefficients of joints K_s. The contact node flag indicates which tree node the end-effector makes contact with ($c_j = 1$ if node is in contact, 0 otherwise), hence only one node is set as the contact node in each episode. We collected a training dataset comprising 6,000 push interactions for each tree size category with node counts of $N \in \{11, 13, 15, 17, 19\}$, and a testing dataset comprising 1,000 push interactions for each tree size category spanning the range of $N \in [10, 30]$, resulting in a combined dataset of 51,000 push interactions.

C. Learning the Forward Model & Contact Policy

1) Forward Model: The forward model problem involves predicting the tree's future state, based on its initial state and the actions performed by the end-effector (see Fig. 4(a)). We encode the input graph G_{in} with the following global, node, and edge attributes:

$$\mathbf{g} = [\mathcal{T}] \tag{4}$$

$$\mathbf{n}_j = [c_j] \tag{5}$$

$$\mathbf{e}_{ij} = [\mathbf{p}_j - \mathbf{p}_i, v_{ij}, K_s] \tag{6}$$

where $v_{ij} = 1$ if the direction of the edge is along the direction from the tree's root-to-leaf, or -1 otherwise. The forward model is trained to predict the positional differences $\Delta\mathbf{p}_j = [\Delta p_x, \Delta p_y, \Delta p_z]$ for each node at steady-state. Hence, the absolute node positions are obtained by updating the initial node positions with the predicted differences.

2) Contact Policy: The contact policy problem is to predict the contact location and action required from the end-effector, given information on the trees current state and its desired target state (see Fig. 4(b)). The input graph G_{in} for the contact policy shares the same edge attributes as the forward model in equation (6), however, the global attributes is null and the node attributes is defined as:

$$\mathbf{n}_j = [\mathbf{p}'_j - \mathbf{p}_j] \tag{7}$$

which encodes per node position difference between the target state and current state. The target states are set to be the final node positions recorded in simulation which ensures feasibility. We train our contact policy to predict a *per-node* linear end-effector trajectory $\mathcal{T}_j^{\text{pred}} = [dx, dy, dz]$, and a *per-node* affordance score s_j. The affordance score ranges between 0 and 1 such that $\sum_{j=1}^{N} s_j = 1$, quantifying which node the end-effector should make contact with relative to all other nodes in the tree.

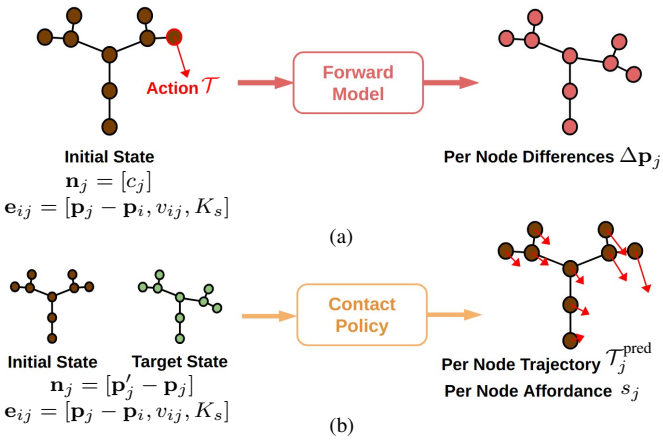


Fig. 4: The input and output diagrams of the (a) forward model and (b) contact policy.

D. GNN Model

We use the graph2graph layer proposed in [24] for our GNN model to learn both the forward model as well as the contact policy. The graph2graph layer consists of two sub-functions, f_e and f_n implemented as multilayer perceptrons (MLP), to compute edge and node feature embeddings as follows:

1) For each edge $\mathbf{e}_{ij}$, compute new edge feature embedding $\mathbf{e}^*_{ij}$:

$$\mathbf{e}^*_{ij} = f_e(\mathbf{g}, \mathbf{n}_i, \mathbf{n}_j, \mathbf{e}_{ij}) \tag{8}$$

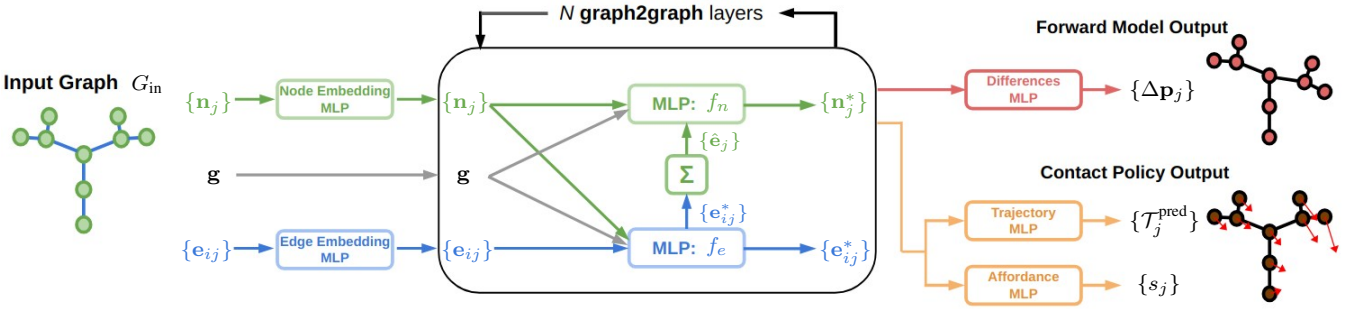


Fig. 5: Our GNN model architecture utilizes stackable graph2graph layers [22], placed between input embedding layers and output prediction heads. The model takes as input a graph G_{in} and outputs per-node predictions. Different output heads are used for the forward model and the contact policy.

2) For each node $\mathbf{n}_j$, aggregate incoming edge embeddings:

$$\hat{\mathbf{e}}_j = \sum_i \mathbf{e}^*_{ij} \tag{9}$$

and subsequently compute new node embeddings $\mathbf{n}^*_j$:

$$\mathbf{n}^*_j = f_n(\mathbf{g}, \mathbf{n}_j, \hat{\mathbf{e}}_j) \tag{10}$$

The graph2graph layer is stacked such that the output node and edge embedding of the current layer is passed as input to the next layer. We empirically set our GNN model to contain five graph2graph layers placed between preprocessing MLPs that extract node/edge feature's from their raw attributes, and post-processing MLP output heads. The overall architecture of our GNN model is shown in Fig. 5, where the forward model and the contact policy shares the same GNN backbone, and differs only by their input attributes and output heads.

We further note that the input graph is preprocessed into a fully connected graph (as opposed to having a partially connected graph with edges only at physical branches) prior to being passed into the model. Information of the tree structure is preserved by setting $K_s = 0$ for non-branch edges, while v_{ij} preserves the tree growth direction from root-to-leaf. The reason for this is to facilitate direct message passing between all nodes and edges. Preliminary tests suggested this provides significant performance advantages over a partially connected graph, further discussed in our ablation studies in Sec. IV-D.

The GNN model is implemented using Pytorch Geometric [28]. All MLPs in Fig. 5 are composed of three fully connected layers with a hidden-size of 128. The forward model and contact policy are trained separately, hence the two models do not share weights. We use Adam with a batch size of 64 and learning rate of 0.001 to optimize the model.

E. Contact Policy with a Robot Arm

Given an initial state and target state of a tree, the contact policy outputs a set of candidate actions represented as per-node trajectories $\{\mathcal{T}^{\text{pred}}_i\}$ and affordance scores $\{s_i\}$. The contact policy, however, cannot be directly executed by a robot arm because the training data is collected using a free-floating end-effector. In order to address the robot's joint configuration space, workspace limitations and collision constraints that may arise from the disparity between a free-floating end-effector and a robot arm, we check the feasibility of candidate actions and select the best action. We achieve this by making use of the multimodal solution offered by the per-node trajectories $\{\mathcal{T}^{\text{pred}}_i\}$ and affordance scores $\{s_i\}$ as presented in Algorithm 1. We iteratively apply the RRT* algorithm to each node, prioritized by decreasing affordance scores, aiming to identify a valid joint space trajectory for a robot arm to reach and interact with the candidate node. If a feasible path is identified, the policy can be executed. Otherwise, we proceed to the next candidate node and repeat the process.

Algorithm 1 Contact Policy Trajectory Planner

Input: Input Graph G_{in}
Output: Joint Space Trajectory $\mathcal{J}$
$\{\mathcal{T}^{\text{pred}}_i\}, \{s_i\} \leftarrow \texttt{ContactPolicy}(G_{\text{in}})$
$i_{\text{order}} \leftarrow \texttt{ArgSortDescending}(\{s_i\})$
for i **in** i_{order}
  $\mathcal{J} \leftarrow \texttt{RRTStar}(\mathcal{T}^{\text{pred}}_i)$
  if $\mathcal{J}$ is feasible
    return $\mathcal{J}$

IV. Experiment Results

A. Metric & Baselines

To assess our method, we train and test both the forward model and the contact policy using the dataset described in Sec. III-B. Our evaluation metric is the error between the predicted node position $\mathbf{p}^{\text{pred}}_j$ and target node position $\mathbf{p}'_j$ for each tree, averaged within the tree size category N:

$$e_N = \frac{\sum_{d=1}^{D} \max_{j \in N} ||\mathbf{p}'_j - \mathbf{p}^{\text{pred}}_j||}{D} \tag{11}$$

where D is the total number data points (episodes) in category N. For each tree, we only measure the maximum node position error to capture the error upper-bound. The forward model directly provides predicted node positions $\mathbf{p}^{\text{pred}}_j$ by summing the predicted differences to the initial states. However, the contact policy predicts actions instead. Hence, we execute the predicted action in simulation and compare the resulting node positions with the target state for contact policy evaluation.

For baseline methods, we compare our forward model and contact policy with a learned PointNet [29] baseline model, which we trained to make predictions on the same data as the GNN model. One important distinction between these two model architectures is that PointNet is specifically designed to handle point cloud data without any edges.

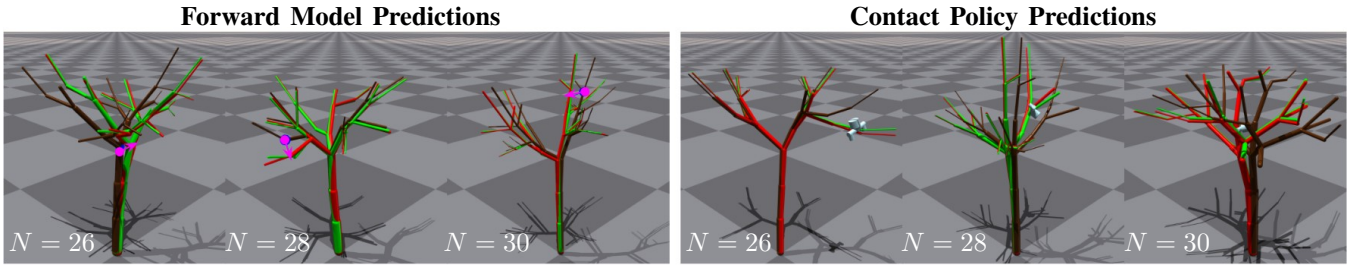


Fig. 6: Visualization of forward model and contact policy predictions on tree sizes of $N \in \{26, 28, 30\}$. The initial tree, ground truth target tree, and predicted tree is shown in the colors brown, green, and red, respectively. The magenta arrow in the forward model predictions depict the applied action.

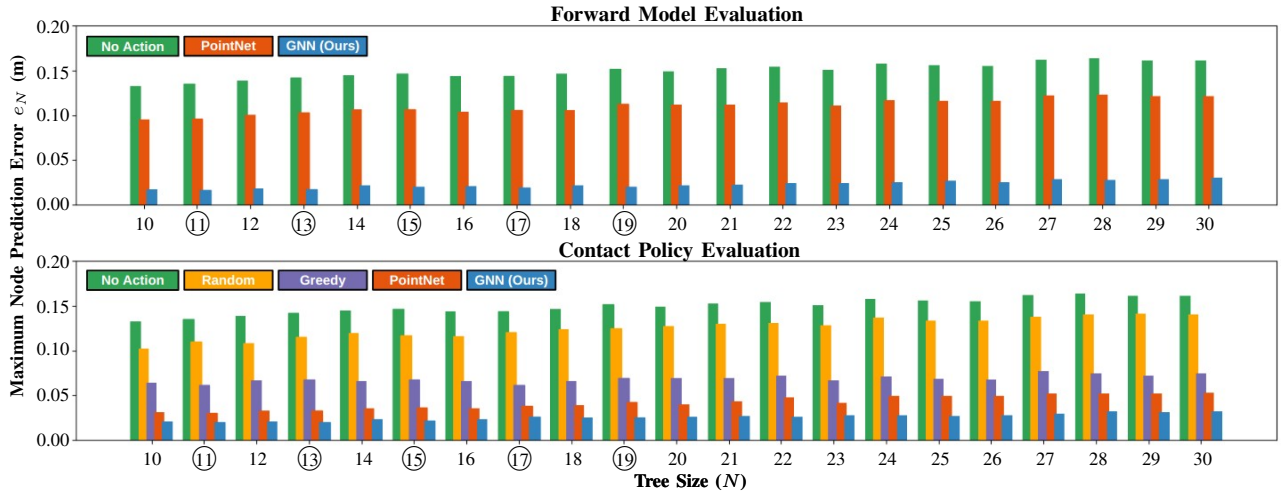


Fig. 7: Performance of the Forward Model (Top) and Contact Policy (Bottom) against baseline methods. The *No Action* baseline is the error measured when the tree remains still. The bars represent the maximum node prediction error per tree, averaged within the tree size category (refer to equation (11)). Circled tree sizes (N) were seen during training, while the rest are predicted in a zero-shot manner indicating generality of the model to unseen systems.

Hence, the PointNet model inherently lacks awareness of the underlying tree structure and its associated attributes (recall equation (6)). We additionally compare the contact policy with two heuristic baselines:

- **Greedy Baseline:** The end-effector makes contact with the tree node $\mathbf{n}_j$ that is farthest from its target position, then pushes the tree along the vector $\mathcal{T} = \mathbf{p}'_j - \mathbf{p}_j$.
- **Random Baseline:** The end-effector makes contact with a randomly chosen node $\mathbf{n}_j$, then pushes the tree along the vector $\mathcal{T} = \mathbf{p}'_j - \mathbf{p}_j$.

B. Evaluation Results

Using our method, we can accurately replicate the forward tree model with the GNN, and learned policies can push trees to a desired target position. Both forward model and policies generalize to previously unseen trees. Fig. 6 illustrates the forward model and contact policy predictions, while Fig. 7 plots the error metric against tree size category. The GNN-based model outperformed baselines for both the forward model and contact policy predictions. Specifically, our method achieves an average node position error of 2.2cm (GNN) as opposed to 11cm (PointNet) for the forward model predictions, and 2.5cm (GNN) compared to 4.2cm (PointNet) for the contact policy predictions. We also show that the GNN model can generalize to make zero-shot predictions on more complex tree structures that were held out during training. This is shown in Fig. 7, where the model was trained on a training dataset with a maximum tree size of $N = 19$,

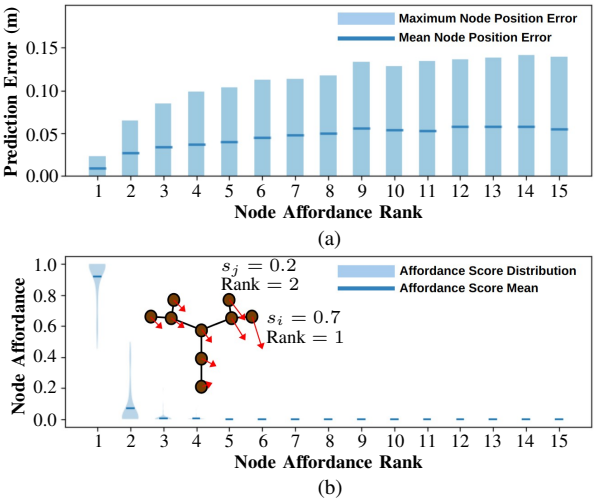


Fig. 8: (a) Bar plot showing the prediction error versus the node affordance rank. (b) Violin plot showing the distribution of node affordance scores versus the affordance rank. The embedded tree graph depicts an example of a tree's affordance score and its resulting rank for two of its nodes.

while consistently performing zero-shot predictions even up to trees of $N = 30$.

C. Multimodal Solution with a Simulated Robot Arm

The affordance scores and actions predicted by the contact policy offer a multimodal solution for manipulating a tree to its target state. We demonstrate this by executing the predicted action for each node and measuring the resulting node position error. Fig. 8(a) illustrates the maximum and average node position errors, while Fig. 8(b) displays the distribution

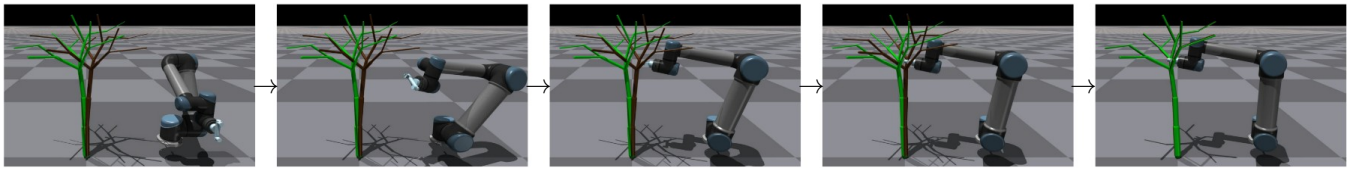


Fig. 9: The UR5 robot arm manipulates the tree-crop (brown) to its target state (green) by executing the trajectory obtained from Algorithm 1.

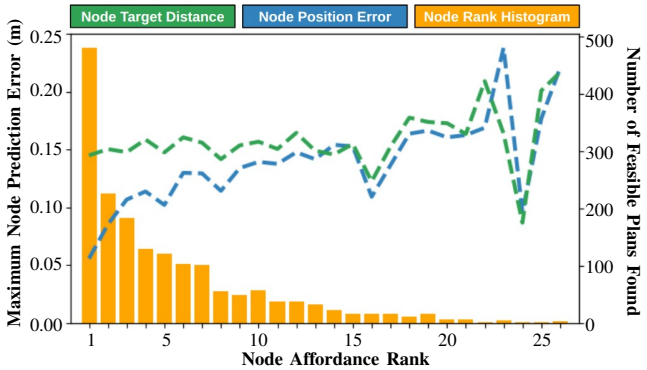


Fig. 10: Evaluating the Contact Policy using a simulated UR5 Robot Arm. The yellow bars display the histogram depicting the number of feasible plans discovered per node affordance rank. A total of 2,100 episodes were tested. The associated initial node target distance (green dashed line) and resulting node position error (blue dashed line) are also plotted.

of affordance scores corresponding to these errors. These metrics are plotted against the node affordance rank (ordered based on decreasing affordance scores), averaged over 100 testing data points for trees with 15 nodes. The plot indicates that the position error increases as the affordance score decreases until the sixth highest-scoring node, after which the error tends to plateau. This behavior can be attributed to the tree's branching structure, as contacting one sub-tree versus another may not significantly affect the outcome if neither of them is the primary target for manipulation.

We apply Algorithm 1 to execute the actions obtained from the contact policy on a simulated UR5 robot arm as illustrated in Fig. 9. To assess the method, we tested the algorithm on 100 episodes for each tree size ranging from $N \in [10, 30]$, resulting in a planning success rate of 82.9%. Fig. 10 depicts the histogram of successfully planned contact node ranks, along with its corresponding node position errors. The algorithm returned a valid trajectory for the highest ranking node 22.9% of the time, gradually decreasing for lower ranking nodes. The error metric is also consistent with the results obtained in Fig. 8(a), where the error gradually increases as the node rank increases.

D. Ablation Study

1) Ablation of Edge Attributes: In this study, we aim to show that the GNN model is effectively learning the relationships intrinsic to the tree structure as a graph. To this end, we perform experiments by training the forward model using the same architecture as our proposed method while removing all edge attributes, as well as systematically removing a single edge attribute for each experiment (see Table I, Rows 1-5). The results indicate that the removal of any one edge attribute leads to a higher node prediction error. This underscores the critical role of each edge attribute in enabling the model to accurately predict nodes by learning latent relationships intrinsic to the tree structure.

TABLE I: Ablation Study of Edge Attributes & Input Graph Connectivity

Ablation of	Number of G2G Layers	G_{in} Connectivity	Included Edge Attributes $\mathbf{e}_{ij}$	Prediction Error (m)
Edge Attributes	5	Full	$[\cdot]$	0.110
	5	Full	$[v_{ij}, K_s]$	0.067
	5	Full	$[\mathbf{p}_j - \mathbf{p}_i, v_{ij}]$	0.038
	5	Full	$[\mathbf{p}_j - \mathbf{p}_i, K_s]$	0.026
Ours	**5**	**Full**	$[\mathbf{p}_j - \mathbf{p}_i, v_{ij}, K_s]$	**0.023**
Input Graph Connectivity	5	Partial	$[\mathbf{p}_j - \mathbf{p}_i, v_{ij}, K_s]$	0.047
	10	Partial	$[\mathbf{p}_j - \mathbf{p}_i, v_{ij}, K_s]$	0.039
	15	Partial	$[\mathbf{p}_j - \mathbf{p}_i, v_{ij}, K_s]$	0.043
	20	Partial	$[\mathbf{p}_j - \mathbf{p}_i, v_{ij}, K_s]$	0.044

2) Ablation of Input Graph Connectivity: We also demonstrate the benefit of preprocessing the input graph into a fully connected form, as opposed to a partially connected one with edges exclusively at physical branches. We train the forward model on partially connected graphs with varying numbers of graph2graph layers, which is then compared to our proposed model trained on fully connected graph inputs (see Table I, Rows 5-9). Even with an increased number of graph2graph layers, employing a fully connected graph as input for a model with only 5 graph2graph layers (ours) surpasses the performance of models with up to 20 graph2graph layers trained on partially connected input graphs. This indicates that the fully connected graph representation significantly enhances the model's ability to learn and capture latent relationships within the tree graph.

V. Conclusion

In this study, we presented a framework that encodes tree crops, modeled as spring-damper systems, into a graph representation. This enables the learning of both a forward model to predict resulting deformations, and a policy to execute non-prehensile contact actions for tree manipulation, using graph neural networks. Our proposed framework has been comprehensively evaluated in simulation using a simplistic model of tree dynamics. However, validating the method in the real world remains as future work. This involves extensive system identification of the tree's dynamic parameters, as well as consideration of more complex properties (such as the anisotropic characteristics of branches). For example, estimating parameters by probing on real trees [18] may lead to realistic simulation for nonlinear dynamical deformation of tree crops. These steps are imperative to achieve Sim2Real policy transfers for practical field applications, which we intend to pursue in our next steps. Applying our learned forward model to implement model-predictive control or model-based reinforcement learning for task-specific crop manipulation also remains as future work.

REFERENCES

[1] P. Martin, "Covid and us farm labor," *Journal of Immigrant & Refugee Studies*, vol. 21, no. 1, pp. 45–58, 2023.

[2] P. Mitrano, A. LaGrassa, O. Kroemer, and D. Berenson, "Focused adaptation of dynamics models for deformable object manipulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5931–5937.

[3] "Finite element analysis of trees in the wind based on terrestrial laser scanning data," *Agricultural and Forest Meteorology*, vol. 265, pp. 137–144, 2019.

[4] F. Yandun, A. Silwal, and G. Kantor, "Visual 3d reconstruction and dynamic simulation of fruit trees for robotic manipulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 54–55.

[5] H. Freeman, M. Qadri, A. Silwal, P. O'Connor, Z. Rubinstein, D. Cooley, and G. Kantor, "Autonomous apple fruitlet sizing and growth rate tracking using computer vision," *arXiv preprint arXiv:2212.01506*, 2022.

[6] A. You, N. Parayil, J. G. Krishna, U. Bhattarai, R. Sapkota, D. Ahmed, M. Whiting, M. Karkee, C. M. Grimm, and J. R. Davidson, "An autonomous robot for pruning modern, planar fruit trees," *arXiv preprint arXiv:2206.07201*, 2022.

[7] A. Silwal, F. Yandun, A. Nellithimaru, T. Bates, and G. Kantor, "Bumblebee: A path towards fully autonomous robotic vine pruning. arxiv 2021," *arXiv preprint arXiv:2112.00291*, 2021.

[8] A. You, H. Kolano, N. Parayil, C. Grimm, and J. R. Davidson, "Precision fruit tree pruning using a learned hybrid vision/interaction controller," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2280–2286.

[9] F. Yandun, T. Parhar, A. Silwal, D. Clifford, Z. Yuan, G. Levine, S. Yaroshenko, and G. Kantor, "Reaching pruning locations in a vine using a deep reinforcement learning policy," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2400–2406.

[10] D. SepúLveda, R. Fernández, E. Navas, M. Armada, and P. Gonzalez-De-Santos, "Robotic aubergine harvesting using dual-arm manipulation," *IEEE Access*, vol. 8, pp. 121 889–121 904, 2020.

[11] S. Stavridis, D. Papageorgiou, L. Droukas, and Z. Doulgeri, "Bimanual crop manipulation for human-inspired robotic harvesting," *arXiv preprint arXiv:2209.06074*, 2022.

[12] X. Zhang and S. Gupta, "Push past green: Learning to look behind plant foliage by moving it," *arXiv preprint arXiv:2307.03175*, 2023.

[13] Y. Xiong, Y. Ge, L. Grimstad, and P. J. From, "An autonomous strawberry-harvesting robot: Design, development, integration, and field evaluation," *Journal of Field Robotics*, vol. 37, no. 2, pp. 202–224, 2020.

[14] C. H. Kim and G. Kantor, "Occlusion reasoning for skeleton extraction of self-occluded tree canopies," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9580–9586.

[15] S. Katyara, F. Ficuciello, D. G. Caldwell, F. Chen, and B. Siciliano, "Reproducible pruning system on dynamic natural plants for field agricultural robots," in *Human-Friendly Robotics 2020: 13th International Workshop*. Springer, 2021, pp. 1–15.

[16] E. Quigley, Y. Yu, J. Huang, W. Lin, and R. Fedkiw, "Real-time interactive tree animation," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 5, pp. 1717–1727, 2017.

[17] H.-C. Spatz and B. Theckes, "Oscillation damping in trees," *Plant science*, vol. 207, pp. 66–71, 2013.

[18] J. Jacob, T. Bandyopadhyay, J. Williams, P. Borges, and F. Ramos, "Learning to simulate tree-branch dynamics for manipulation," *arXiv preprint arXiv:2306.03410*, 2023.

[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[21] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[22] 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.

[23] 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.

[24] 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.

[25] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[26] A. Runions, B. Lane, and P. Prusinkiewicz, "Modeling trees with a space colonization algorithm." *Nph*, vol. 7, no. 63–70, p. 6, 2007.

[27] S. Timoshenko, "History of strength of materials mcgraw-hill book company," *Inc., New York/Toronto/London*, 1953.

[28] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[29] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.