Compensating for Unmodeled Forces using Neural Networks in Soft Manipulator Planning

Scott Chow[1], Gina Olson[2], and Geoffrey A. Hollinger[1]

***Abstract*— Soft manipulators made of deformable materials have great promise in applications that require additional flexibility and compliance; however, these characteristics also make them difficult to simulate accurately and quickly. The lack of a fast and accurate simulator prevents motion planners from generating feasible plans, which would enable soft robots to achieve more complex tasks, such as manipulation. In this work, we propose combining a simplified quasistatic model with a neural network that learns to compensate for unmodeled forces, such as friction and loads, in order to create a fast forward model for soft manipulators with multiple segments. We show that the resulting neural network model reduces average end effector position error by 62% compared to the quasistatic model, while still being fast enough for motion planning. We also incorporate this model into an RRT*-based planner and demonstrate that the plans generated using our model are more likely to be feasible when executed on hardware than plans generated with a simulator using the quasistatic model.**

I. Introduction

Currently, robotic manipulators are widely used in highly structured environments, such as factory assembly lines, but a recent goal has been to expand their usage into more challenging unstructured environments, such as underwater manipulation and human-robot cooperative assembly. Soft robots made of deformable materials have great potential for tasks requiring safety and flexibility such as manipulating delicate objects. However, a motion planner with an accurate simulator for soft robots is necessary to generate plans for these robots to accomplish more complex tasks.

Motion planning is a critical component of robotic systems that enables efficient navigation around obstacles. Sampling-based motion planning is a family of planners commonly used in robots ranging from vehicles to manipulators. However, sampling-based planners can potentially query a simulator thousands of times to generate a solution; this requires a fast model that can keep up with the planner.

Creating fast, yet accurate, models of complex systems is difficult. Pneumatic soft robotic manipulators are particularly challenging to model. The entire shape of the manipulator changes as it actuates, which complicates the internal forces within the manipulator. Chaining multiple segments together to increase the degree of freedom of the arm also introduces load, as segments further down the chain act as loads on base segments. Reduced-order, closed form solutions must neglect loads such as friction, which can cause large errors between predicted and observed behavior, as seen in Figure 1.

Previous approaches have examined modeling soft manipulators using simplified kinematics models, which offers speed at the cost of accuracy. Physics-based approaches are significantly more realistic and accurate, but are too slow for use in motion planning. Data-driven approaches offer a promising middle ground between speed and accuracy.

In this work, we present a data-driven approach for modeling pneumatic soft manipulators with multiple segments. We propose using a neural network that leverages the output of a recently developed quasistatic no load model [1] to accurately model the manipulator's behavior while maintaining faster than real time prediction speed to prevent bottlenecking the motion planner. This model is shown to more accurately simulate our soft manipulator compared to the quasistatic no load model. Using this model in a simulator for a sampling-based planner, we show that the resulting plans are more likely to be feasible and achieve the goal position while avoiding obstacles more reliably than plans generated using the no load model.

II. Background

In this section, we review relevant work in modeling soft arms and motion planning techniques.

1) Modeling Soft Robots: Prior work for modeling soft robots can be viewed in two categories. Kinematic models parameterize soft and continuum robots with a set of parameters that describe the robot's shape. The most prominent of such models is the piecewise constant curvature (PCC) model [2], which makes the assumption that each segment can be

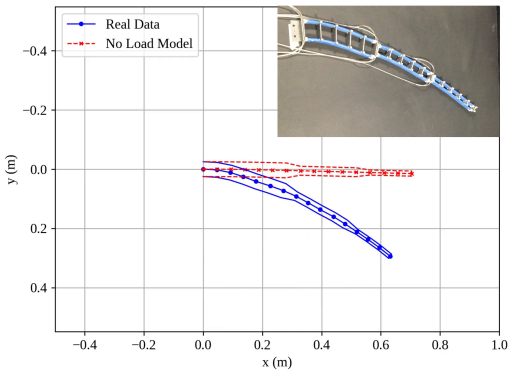

Fig. 1: The final state of the arm after a cycle of inflating, then deflating all right side segments (blue). The arm does not return to its outstretched state due to friction and load, which are not considered by the quasistatic model (red).

This work was funded in part by NSF award IIS-1734627 and ONR award N00014-21-1-2052.

[1] Collaborative Robotics and Intelligent Systems (CoRIS) Institute at Oregon State University, Corvallis, OR 97331, USA {chows, geoff.hollinger}@oregonstate.edu,

[2]Department of Mechanical Engineering at Carnegie Mellon University, Pittsburgh, PA 15213. golson@andrew.cmu.edu

parameterized by a length and curvature. The PCC model is very fast to compute and thus can be used for motion planning. However, the PCC assumption is often violated in practice, especially when external forces act on the arm.

Physics-based models examine the internal forces within the arm to model soft arms. Previous approaches have used the Discrete Cosserat Model [3] and Euler-Bernoulli Beam Theory [4]. Additionally, finite element analysis-based approaches have also been applied to model soft robots [5]. While these models tend to be accurate, they are also relatively slow to compute, requiring minutes to solve for a second of simulation [3]. This slow computation time makes these models unsuitable for use with motion planners.

Recently, data-driven approaches have been used for modeling soft robots. For example, deep networks have been used to learn a mapping of data from piezoresistive sensors to arm shape to enable better proprioception [6]. Other works have used a hybrid of kinematic or physics-based models and neural networks to compute the forward and inverse kinematics of soft [7] and continuum arms [8]. These approaches offer fast simulation while providing reasonably accurate predictions; however, they are primarily used in controllers and for modeling arms with two or fewer segments.

To address the limitations of previous data-driven approaches, we draw inspiration from faster than real-time soft material simulation in computer graphics. One promising approach [9] uses a combination of a linear least squares model and a neural network model to predict how a deformable object reacts to forces in real time. Our work combines a simplified physics-based model for a single segment under no load with a neural network that learns to account for unmodeled forces to accurately simulate a multi-segment arm while maintaining fast computation speed.

2) Planning for Soft Manipulators: Previous approaches to soft manipulator planning have examined using genetic algorithms and trajectory optimization. Xiao et al. [10] proposed the Real-time Adaptive Motion Planner that uses a genetic algorithm for online planning for a tendon-driven continuum manipulator modeled using the Piecewise-Constant Curvature model. While genetic algorithms allows for online planning in dynamic environments, the resulting trajectories have few guarantees and many suboptimal actions. Marchese et al. [11] takes a trajectory optimization-based approach to move their soft arm through a sequence of human-specified waypoints through the maze. Unlike our approach, they do not address the generation of these waypoints.

3) Sampling-Based Motion Planning: Sampling-based motion planning utilizes samples of the robot's configuration space to generate a plan. Rapidly-exploring Random Trees (RRTs) are a common formulation of sampling-based motion planners [12] that involves repeatedly sampling the arm configuration space to find a path from the start to goal. RRT* is an asympotically optimal variant of RRT that takes into account path cost in plan formulation [13] in order to find the lowest cost path to the goal.

In prior work, RRTs were used to generate plans for soft manipulators by planning in curvature space using the constant curvature model [4]. However, one of the primary challenges with that approach was finding consistent mappings between curvatures and actuation pressures. In this work, we adapt the RRT* algorithm for use with our neural network model to generate plans for our soft manipulator. Not only do we now consider path cost in planning, but we also plan in pressure space using the neural network model, which improves the feasibility of our plan.

III. Problem Formulation

This section details the forward simulation problem and the motion planning problem for an n segment soft arm.

A. Forward Simulation Problem

In order for a sampling-based motion planner to generate plans, it is critical to have the ability to quickly forward simulate how the shape of the arm changes given some change in pressures. For our pneumatically driven arm, the configuration space at time t is given by the pressures of the actuators for each segment $q_t = [p_0, \ldots, p_n]$, where positive/negative pressures denote pressurizing the right/left sides of the arm respectively. We make the assumption that only one side of the arm will be pressurized at once.

The shape of the arm in the workspace is defined by the poses $\mathbf{x} = [x, y, \theta]$ of m points along the backbone of the arm. Thus, the backbone of the arm at time step t is $B_t = [\mathbf{x}_{1,t}, \mathbf{x}_{2,t}, \ldots, \mathbf{x}_{m,t}]$. The end effector pose is defined as the pose of the distal-most point of the arm, $\mathbf{x}_m$.

The forward simulation problem can be given as follows: given the arm's previous configuration q_{t-1}, previous backbone pose B_{t-1}, and the arm's change in configuration $\Delta q = q_t - q_{t-1}$, we want to find the backbone at the current time step B_t. That is, we want to find a forward simulation function F such that:

$$B_t = F(q_{t-1}, \Delta q, B_{t-1}) \tag{1}$$

B. Motion Planning Problem

Next, let us define the motion planning problem for the soft arm. Let B denote the set of possible backbone positions, and $B_{free} \subseteq B$ denote the subset of backbones in which the arm is not in collision with obstacles in the workspace. Let X denote the space of possible end effector poses. Let the path, $P = (\Delta q_1, \ldots, \Delta q_T)$, be a set of pressure changes similar to the action space representation used in [14]. The motion planning problem is defined as follows: given an initial arm pressure configuration q_0, initial backbone pose B_0, and a goal end effector pose region $X_{goal} \subseteq X$, find the optimal path P^* that satisfies the following:

$$\begin{aligned} P^* = \underset{P \in \mathcal{P}}{\arg\min}\, \Pi(P)\ s.t. \\ \forall B_i \in \Psi(P, B_0, q_0), B_i \in B_{free} \\ \mathbf{x}_{m,t} \in X_{goal} \end{aligned}$$

where $\mathcal{P}$ is the set of all possible paths, Π is the cost function defined in Equation 2, and $\Psi(P, B_0, P_0)$ is a function that recursively maps F onto the vector of pressure changes P beginning from the initial backbone pose B_0 and initial

pressure q_0, yielding the set of backbone poses for each timestep $t \in 1 \dots T$.

Our cost function is defined to be the absolute sum of change in pressures across all segments in the plan:

$$\Pi(P) = \sum_{\Delta q \in P} \sum_{\Delta p \in \Delta q} |\Delta p| \tag{2}$$

IV. Methods

In this section, we present the quasistatic no load model that is leveraged in our approach, our neural network model, and a RRT*-based planning algorithm for soft manipulators.

A. Quasistatic No Load Model

The quasistatic no load model is derived by first characterizing the actuator by mapping the pressure of an inflated actuator to its strain via a fourth order polynomial fit. Olson et al. derived a quasistatic bending model using an Euler-Bernoulli formulation [1]. For a planar segment composed of McKibben actuators under no load, the curvature of a segment is related to the strain of the actuator by:

$$\kappa = \left(-\frac{t}{\epsilon} - \frac{t}{2} \right)^{-1} \tag{3}$$

and the length of segment is related to the strain by:

$$l = (1 + 0.5\epsilon) l_0 \tag{4}$$

where ϵ is the strain, t is the width of the segment, κ is the curvature of the segment, l_0 is the length of the segment when unpressurized, and l is the shortened length of the segment. Using our polynomial fit, we can find curvature and segment length given pressure. Under the piecewise constant curvature assumption, we can then compute the backbone of the segment using these parameters.

This quasistatic no load model makes the simplifying assumption that the segments are frictionless and under no load, which are violated with real-world arms. First, the quasistatic assumption is violated since our arm is continuously actuated without stopping. Second, there are frictional forces between the planar arm and the work surface. Finally, all segments of the arm are subject to loads because distal segments in the arm act as additional mass that must moved by proximal segments. Thus, we use our neural network to adjust for these violated assumptions.

B. Neural Network Model

To address the forward simulation problem, we propose a neural network-based approach (shown in Figure 2) for finding how the backbone changes as segments are inflated and deflated. The inputs to the network are the previous and current arm configuration pressures q_{t-1} and q_t, the previous backbone of the arm B_{t-1}, and the predicted output of the no load quasistatic model $\tilde{B}_t = \tilde{F}(q_t)$. The output of the network is a compensation vector that encodes the deviation of the no load model from reality due to friction and other load forces that have been ignored in the no load model. We train a multi-layer feedforward neural network Φ such that:

$$B_t = \Phi(q_{t-1}, q_t, B_{t-1}, \tilde{B}_t) + \tilde{B}_t \tag{5}$$

Because the physics of the internal forces of the arm are complex, it is difficult to train a network to predict the backbone purely from collected data. Thus, our approach uses the neural network to learn to predict the difference between the backbone predicted by the fast, but less accurate, simplified quasistatic model, $\tilde{F}$, and the true backbone.

Because the sequential queries of a sampling-based planner are often not related to each other, it is critical that the the output of the network to one query must only be dependent on the current input into the network. To account for this, we instead embed limited recurrence indirectly by allowing the network to take the pressure configuration and backbone at the previous time step as input. This backbone is either known at $t = 0$ or is the previous output of network.

C. Network Training

One of the difficulties of training this network is the implicit recurrence due to the network taking in the output at a previous time step as input. The traditional training process of iterating through minibatches of the dataset results in instabilities in both training and execution, as a single misprediction would result in divergence from the training set. In previous works [9], it has been shown that training across windows of time and backpropagating the average error of the entire trajectory leads to more stable training.

Thus, to train the network, we split our dataset into windows of time. For each window, we use our network to predict the backbone of the arm repeatedly across the entire window. The error for a particular frame is calculated as the sum of L1 errors between the expected pose and the predicted pose along the entire length of the backbone of the arm. The overall error across a window is the average of the frame errors. This window error is then backpropagated using stochastic gradient descent. Using this approach, we are able to reduce instability caused by using our network output as input for a subsequent step during training.

D. Pressure Space RRT*

Now that we have a forward model for our simulator, we can now utilize sampling-based planners to generate plans for our arm. In this work, we modify the traditional RRT* algorithm to account for using our compensation network.

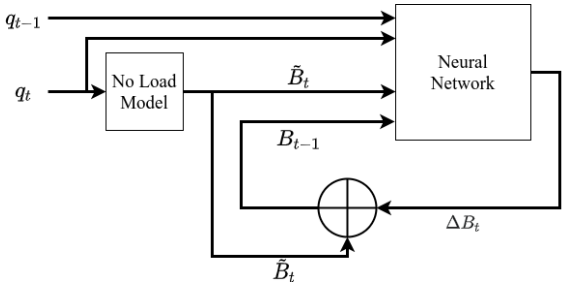

Fig. 2: A diagram of the proposed neural network model. q is the current set of pressures, $\tilde{B}$ is the prediction of the no load model, ΔB is the compensation to the no load model backbone computed by the neural network, and B is the adjusted backbone configuration of the arm.

The main adjustment that must be made is that every state on the tree must account for both the configuration q as well as the backbone B. This is because it is possible for the same set of pressures to result in different backbones because of how that set of pressures was reached.

This leads to two adjustments to the RRT* algorithm. First, when choosing a parent node to grow towards the sampled state, we instead expand all the parent nodes towards the sampled state without collision checking and sort the resulting potential new nodes by cost. Then from lowest to highest cost, we check for collisions and choose the lowest cost state that is collision-free to keep in our tree. Second, when rewiring, we must check that it is possible to get to not just the same pressure configuration, but also the same backbone configuration before allowing rewiring to occur.

E. Goal Biasing

Goal biasing is a common technique in which a set of potential goal configurations are used to bias the search and reduce planning time [15]. In order to compute potential goal configurations for goal biasing, we cannot use our trained network as it is not invertible. We instead invert the no load model to generate potential goal solutions to bias our search.

The process for generating goal solutions is as follows: we first compute the spatial Jacobian matrix that relates change in curvature to change in end effector position. We derived the general form of the Jacobian for a piecewise-constant curvature arm by following a similar process as [16]. However, for our arm, segment length is related to curvature through strain. To address this, we find the relationship between change in length and change in curvature by combining equations (3) and (4) and taking the derivative with respect to time:

$$\dot{l} = D\dot{\kappa} = \begin{cases} \frac{-2l_0 t}{4}\dot{\kappa}, & \kappa = 0 \\ \frac{-2l_0 t\kappa}{|\kappa|(t|\kappa|+2)^2}\dot{\kappa}, & \text{else} \end{cases} \tag{6}$$

Accounting for this relationship, the generalized form of our spatial Jacobian for a single segment is:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J_j \begin{bmatrix} \dot{\kappa} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -(l+\kappa D)\sin\Delta\phi & 0 \\ (l+kD)\cos\Delta\phi & 0 \\ D + \frac{\sin\kappa l - \kappa l}{\kappa^2}\eta & 0 \\ \frac{1-\cos\kappa l}{\kappa^2}\eta\cos\Delta\phi & 0 \\ \frac{1-\cos\kappa l}{\kappa^2}\eta\sin\Delta\phi & 0 \end{bmatrix} \begin{bmatrix} \dot{\kappa} \\ \Delta\dot{\phi} \end{bmatrix} \tag{7}$$

where v, w are the linear velocity and angular velocity of the end effector respectively, $\eta = -\dot{\kappa}l^2$ and D is the coefficient term defined in (6). $\Delta\phi$ and $\Delta\dot{\phi}$ which correspond to the change in bend plane angle and its velocity are both 0 in the planar case. As discussed in [16], we can find the Jacobian for an n-segment arm by stacking the Jacobians:

$$J = [J_0|Ad_{T_0}J_1|...|Ad_{T_{n-1}}J_n] \tag{8}$$

where J_j is the jth segment Jacobian and Ad_{T_i} is the adjoint of the transform from the base to the end of segment i.

Once the Jacobian has been computed, we use the Newton-Rhapson method [17] to solve the inverse kinematics problem. The result is a set of curvatures that would result in the end effector at the goal according to the piece-wise constant curvature model. We then invert the polynomial mapping from the quasistatic no load model and solve the resulting linear system of equations to obtain a set of pressures that represent a potential goal configuration. We repeat this process multiple times in order to sample the goal region to generate a set of approximate goal solutions to bias towards.

V. Experimental Setup

In this section, we describe the three segment arm used in our experiments, the training and validation data set collection process, and the parameters of our neural network.

1) Three Segment Soft Manipulator: In this work, we apply our neural network forward prediction model and pressure space RRT* to the planar three segment pneumatically driven soft manipulator shown in Figure 3. Each segment consists of 4 McKibben contracting actuators, two along the left side of the segment and two along the right. The actuators are composed of air bladders made from EcoFlex 00-30 and wrapped with expandable polyester sheathing. To construct a manipulator using these actuators, we utilize 3D-printed PLA support plates along the backbone of arm. Actuators are secured onto the left and right side of the support plates. Our three segment arm features 16 support plates. By tracking the pose of these plates using Optitrack markers, we compute the points along the backbone of the arm. More details on the fabrication and assembly of the arm can be found in [4].

Each segment is 235 mm long. The three segment arm has a staged configuration with decreasing widths. The widths of each segment from base to end are 50, 30, and 17.2 mm respectively. In order to move the arm, actuators on one side of a segment are inflated, causing them to contract and the segment to bend towards the corresponding side. Thus the resulting planar arm has 3 controllable degrees of freedom. Each segment is inflated and deflated using a series of pumps and inflation/deflation valves that are controlled by a PD controller with hand-tuned gains to hold a set point pressure.

2) Data Set Collection: To collect a dataset for training and validation of our compensation network, we placed our soft manipulator into an Optitrack motion capture system collecting backbone pose data at 20 Hz. Each segment can have either the left actuator fully inflated, the right actuator fully inflated, or have neither actuator inflated. This results in 27 different sets of pressure configurations. From these 27 configurations, there are 729 ways to transition from one configuration to another. For our training set, we executed each of these transitions on our arm and recorded actuator pressure and backbone points. For our test set, we generated 100 random goal arm configurations and executed them sequentially on hardware.

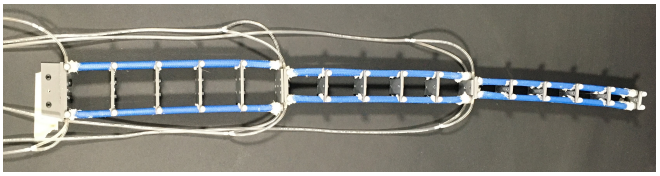

Fig. 3: Three segment soft manipulator used in experiments.

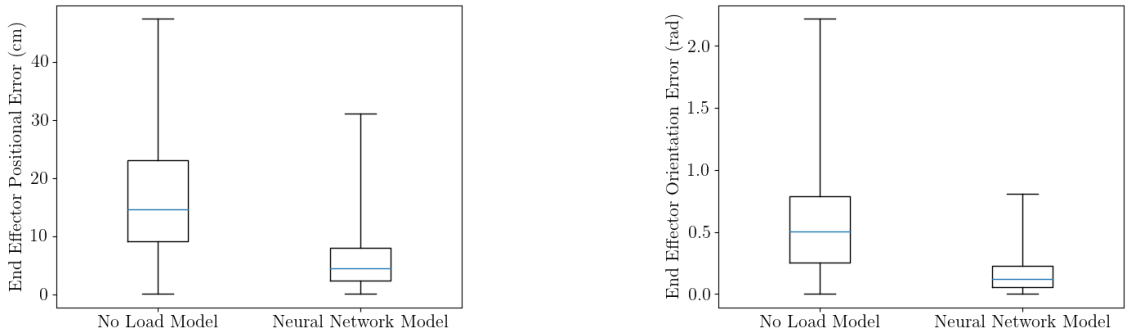

Fig. 4: The neural network model achieves lower position and orientation end effector error across the test set compared to the no load model.

3) Network Parameters and Training: For our compensation network, we instantiated a 3 hidden layer feedforward neural network with 100 hidden units each layer that uses the ReLU activation function. To prepare our training data, we normalized our data between 0 and 1. Then we split our training set into windows of 32 frames to train across windows of time as described in Section IV-C. The windows are grouped into batches of 128, which are then fed into the network for training using stochastic gradient descent with a learning rate of 10^{-4}. The network was trained until the test error converged after 6000 epochs.

VI. Model Validation Results

To validate our model, we applied both the no load model and the neural network model on the test set created by executing 100 random arm configurations. We compare the pose of the end effector predicted by the models to the actual pose from ground truth data collected from the Optitrack for the arm in Figure 3. The results are provided in Figure 4. The no load model has an average positional error of 16.2 ± 8.8 cm and an average orientation end effector error of 0.61 ± 0.48 radians. In comparison, the neural network model had an average positional error of 5.8 ± 4.7 cm and an average orientation error of 0.16 ± 0.13 radians. This corresponds to a 62% decrease in positional error and a 73% decrease in orientation error. This suggests that the neural network model is able to more accurately predict the pose of the end effector compared to the no load model.

We benchmarked both models by querying each model 10^6 times and measuring the speed at which the models responded. The no load model ran at 978 queries per second while the neural network model ran at 510 queries per second. Even though the neural network is slower, it is still operating at faster than real time simulation speeds and is fast enough for motion planning.

These experiments demonstrate that our data-driven approach significantly improves the accuracy of the model while maintaining fast query speeds necessary for planning.

VII. Planning Results

1) Simulation Plan Generation: First, we demonstrate that we are able to utilize our neural network to generate motion plans. We constructed eight environments shown in Figure 5. These environments were chosen to illustrate situations where the additional accuracy of the neural network model is crucial. For each environment, we using two simulators, one that uses the no load model, and one that uses our proposed neural network model.

Next we used the pressure space RRT* method to generate plans. For each simulator, we ran our planner ten times per environment, recording whether the plan generation was successful within the allotted planning time of 10 minutes, the time it took to generate the plan, as well as the cost of the path generated as defined in Equation 2. The results of these experiments are given in Table I under the Planning in Simulation column. We report the number of successful plans generated, average time to generate a successful plan, and average plan cost with standard errors out of 20 trials.

The plan generation success rate for the two planners is similar. Across the eight environments, the neural network model has a higher success rate on three of the environment, a lower rate on four, and an equivalent rate on one.

While the no load model produces paths with lower cost, these paths may not be realizable, since they require traversing states that are not actually reachable according to our neural network model. Since our neural network model has a more accurate model of the true transition function, it must expand more states to avoid the falsely-feasible states used by the no load model. The expansion of these additional states is also reflected in the higher planning times for the planner using the neural network model simulator.

2) Hardware Plan Execution: Next, to evaluate the feasibility of the plans, we took a successfully generated plan for each simulator, environment pair and executed it on hardware. The results of these hardware trials are provided in Table I under the Execution in Hardware column. We report whether a collision occurred as well as how much the end effector pose deviates from the goal in terms of position and orientation at the end of plan execution. The final backbone from executing the plans are visualized in Figure 5.

We observe that the plans generated using the neural network-based simulator enable the end effector to reach closer to the goal while avoiding obstacles. In one case

TABLE I: Using the neural network model increases planning time and path cost; however, the resulting plans are more feasible and get the end effector closer to the goal without collisions.

		Planning in Simulation			Plan Execution in Hardware		
Environment	Model Used	Successfully Generated Plans (out of 20)	Successful Plan Time (s)	Successful Plan Cost (psi)	Collision Occurred	Distance to Goal (cm)	Orientation to Goal (rad)
1	No Load	20	51.2 ± 17.9	19.4 ± 0.9	No	3.26	0.28
	Neural Network	20	126.1 ± 34.3	26.8 ± 0.9	No	4.09	0.86
2	No Load	20	70.1 ± 15.7	27.6 ± 1.2	No	5.07	0.64
	Neural Network	17	249.8 ± 36.4	40.7 ± 1.3	No	4.62	0.41
3	No Load	17	212.7 ± 42.2	30.3 ± 1.6	No	5.35	0.42
	Neural Network	20	155.7 ± 16.4	40.0 ± 1.0	No	0.96	0.02
4	No Load	18	215.5 ± 43.7	23.1 ± 1.4	No	14.86	0.35
	Neural Network	19	115.1 ± 38.8	30.4 ± 1.2	No	3.93	0.71
5	No Load	20	70.5 ± 19.9	24.5 ± 1.0	No	10.43	0.96
	Neural Network	17	221.2 ± 39.8	41.1 ± 1.5	No	1.69	0.57
6	No Load	17	184.8 ± 42.1	42.1 ± 2.0	No	7.98	0.31
	Neural Network	13	246.3 ± 49.5	52.8 ± 2.3	No	2.15	0.52
7	No Load	17	144.5 ± 24.2	42.5 ± 1.5	Yes	N/A	N/A
	Neural Network	10	229.8 ± 31.3	52.7 ± 4.5	No	6.39	0.5
8	No Load	18	156.6 ± 39.8	43.7 ± 2.1	No	13.05	1.08
	Neural Network	20	184.1 ± 16.5	57.4 ± 2.0	No	3.11	0.27

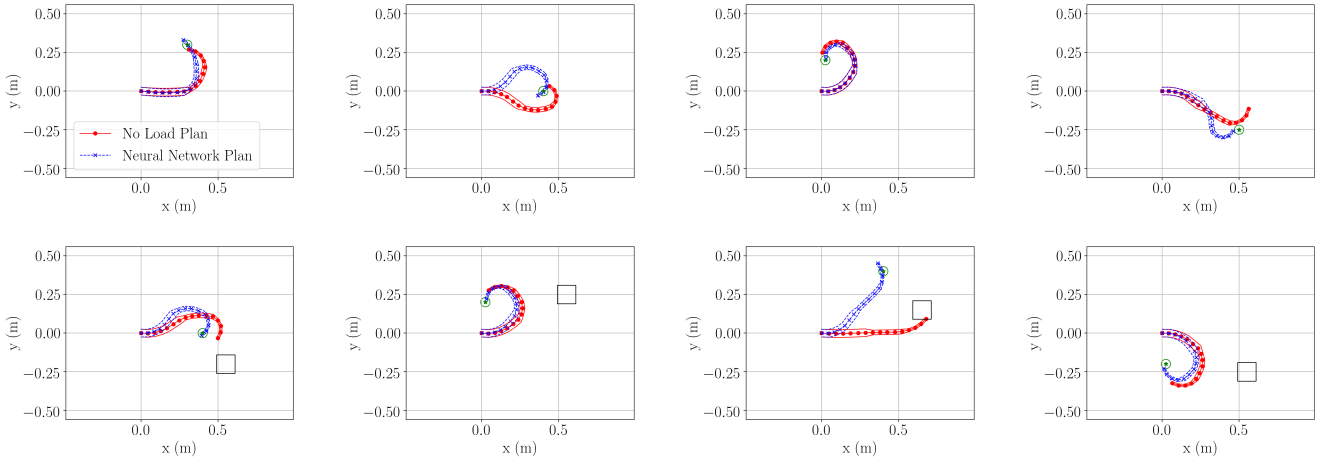

Fig. 5: The eight environments used for planning and hardware trials using the three segment arm shown in Figure 3. The goal location is indicated by the green star. The final backbone from executing plans using the no load model simulator (red) and the neural network simulator (blue) during hardware trials are provided.

(environment 7), the no load model plan actually resulted in a collision with an obstacle. In the remaining trials where there were no collisions, using our neural network model reduced the final positional error of the end effector by an average of 5.7 cm while achieving similar average orientation error to the no load model. These experiments demonstrate that plans from the neural network model simulator reflect the real world behavior of the arm more accurately and are thus more likely to be feasible compared to plans from the no load model.

VIII. Discussion

In this paper, we combine a simplified quasistatic, no load model with a neural network that learns to compensate for unmodeled forces such as friction and loads. This network is used to model a three segment soft manipulator. Our neural network model is shown to be more accurate than the quasistatic model, while still being fast enough for motion planning. The RRT* algorithm was adapted to plan in pressure space using our model. The resulting plans from using the neural network-based simulator were shown to be more likely to be successfully executed, and generally bring the end effector closer to the goal than plans generated using the no load model.

An avenue for future work is to adapt our approach to forward simulate interactions with the environment. In order to properly take advantage of soft robots' ability to safely interact with the world, an accurate simulation of the resulting deformations is necessary. By creating more accurate but fast models for soft arms, we can generate feasible plans that will enable soft robotics to perform more complex tasks in a greater range of environments.

REFERENCES

[1] G. Olson, R. L. Hatton, J. A. Adams, and Y. Mengüç, "An euler-bernoulli beam model for soft robot arms bent through self-stress and external loads," *International Journal of Solids and Structures*, vol. 207, pp. 113–131, 2020.

[2] R. J. Webster III and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1661–1683, 2010.

[3] F. Renda, F. Boyer, J. Dias, and L. Seneviratne, "Discrete cosserat approach for multisection soft manipulator dynamics," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1518–1533, 2018.

[4] G. Olson, S. Chow, A. Nicolai, C. Branyan, G. Hollinger, and Y. Mengüç, "A generalizable equilibrium model for bending soft arms with longitudinal actuators," *The International Journal of Robotics Research*, vol. OnlineFirst, 2019.

[5] F. Largilliere, V. Verona, E. Coevoet, M. Sanz-Lopez, J. Dequidt, and C. Duriez, "Real-time control of soft-robots using asynchronous finite element modeling," in *Proc. IEEE International Conference on Robotics and Automation*, 2015, pp. 2550–2555.

[6] R. L. Truby, C. Della Santina, and D. Rus, "Distributed proprioception of 3d configuration in soft, sensorized robots via deep learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3299–3306, 2020.

[7] A. Nicolai, G. Olson, Y. Mengüç, and G. A. Hollinger, "Learning to control reconfigurable staged soft arms," in *Proc. IEEE International Conference on Robotics and Automation*, 2020, pp. 5618–5624.

[8] O. Lakhal, A. Melingui, and R. Merzouki, "Hybrid approach for modeling and solving of kinematics of a compact bionic handling assistant manipulator," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 3, pp. 1326–1335, 2015.

[9] D. Holden, B. C. Duong, S. Datta, and D. Nowrouzezahrai, "Subspace neural physics: Fast data-driven interactive simulation," in *Proc. of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2019, pp. 1–12.

[10] J. Xiao and R. Vatcha, "Real-time adaptive motion planning for a continuum manipulator," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 5919–5926.

[11] A. D. Marchese, R. K. Katzschmann, and D. Rus, "Whole arm planning for a soft and highly compliant 2d robotic manipulator," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 554–560.

[12] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[14] D. Jones, G. A. Hollinger, M. J. Kuhlman, D. A. Sofge, and S. K. Gupta, "Stochastic optimization for autonomous vehicles with limited control authority," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 2395–2401.

[15] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[16] R. J. Webster III, J. P. Swensen, J. M. Romano, and N. J. Cowan, "Closed-form differential kinematics for concentric-tube continuum robots with application to visual servoing," in *Experimental Robotics*. Springer, 2009, pp. 485–494.

[17] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.