2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561420

A Dynamics Simulator for Soft Growing Robots*

Rianna Jitosho[1], Nathaniel Agharese[1], Allison Okamura[1], and Zac Manchester[2]

***Abstract*— Simulating soft robots in cluttered environments remains an open problem due to the challenge of capturing complex dynamics and interactions with the environment. Furthermore, fast simulation is desired for quickly exploring robot behaviors in the context of motion planning. In this paper, we examine a particular class of inflated-beam soft growing robots called "vine robots," and present a dynamics simulator that captures general behaviors, handles robot-object interactions, and runs faster than real time. The simulator framework uses a simplified multi-link, rigid-body model with contact constraints. To bridge the sim-to-real gap, we develop methods for fitting model parameters based on video data of a robot in motion and in contact with an environment. We provide examples of simulations, including several with fit parameters, to show the qualitative and quantitative agreement between simulated and real behaviors. Our work demonstrates the capabilities of this high-speed dynamics simulator and its potential for use in the control of soft robots.**

I. Introduction

Dynamics simulators are widely used in robotics for applications such as design exploration, motion planning, controller tuning, and human operator training. However, simulating soft robots in cluttered environments remains an open problem due to the challenge of modeling complex dynamics and handling environment interactions. There are simulators that use Finite Element Analysis to achieve high accuracy [1], but they are often slow and difficult to use. In this work, we are interested in a simulator that is suitable for motion planning, in which it is common to opt for a low-fidelity model in order to compute dynamics quickly. In particular, we are interested in a dynamics simulator that: (1) captures bending and growing behaviors, (2) handles robot-object interactions, (3) runs faster than real time, and (4) is amenable to classical motion planning strategies.

Real-time simulators currently exist for elastomer-based soft robots [2], [3]. In contrast, our work examines a particular class of soft growing robots called "vine robots," which has shown potential for use in applications such as medical devices as well as navigation and exploration [4], [5]. Unlike elastomer-based robots, these inflated-beam robots are built from flexible but inextensible materials. These robots can grow over 100 times their original length via pneumatic eversion. In Fig. 1, we provide an example of a vine robot in motion, and overlay a simulated trajectory that was generated using the framework presented in this paper.

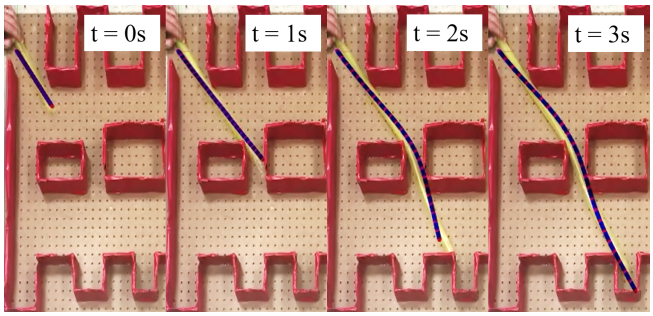

Fig. 1. Trajectory of a simulated robot (blue) overlaid on video frames of a soft growing robot (yellow) in a cluttered environment.

Prior work has contributed models characterizing vine robot behavior [6]–[9], and recent papers have explored models of vine robot interactions with the environment. Haggerty, et al. modeled transverse and axial buckling modes that result from environment interaction to predict the pressure required for eversion [10]. Greer, et al. developed a kinematic model for a vine robot growing in a cluttered environment [11]. Selvaggio, et al. examined robot-obstacle interactions in the context of workspace analysis and presented an object-interaction model that accounts for point contacts [12]. These models are for planar motion, as is the case in our work. Our work differs in that we present a simplified rigid-body dynamic model that can handle interaction with objects of various geometries. This is more general than the existing kinematic and object-interaction models. In addition, we incorporate this dynamic model into a simulator framework.

Simulation of rigid-body dynamics is possible with existing software [13]–[16], and vine robot growth has previously been simulated in Unity [17]. Most of these simulators use minimal coordinate representation. In contrast, our work uses maximal coordinates due to the benefits discussed in Section II. Moreover, because we aim to use our framework in motion planning, it is beneficial to have access to the internal details of the dynamics model, including constraint Jacobians and other derivatives. This low-level access enables tighter integration with optimization software for motion planning, system identification, and trajectory optimization.

The main contributions of our work are:

1) A vine robot simulator framework that captures general behaviors, handles robot-object interactions, runs faster than real time, and is amenable to motion planning.
2) A method for fitting model parameters based on video data of a vine robot in motion.
3) Validation of the framework by comparing simulations

*This material is based upon work supported in part by the National Science Foundation (NSF), including the Graduate Research Fellowship Program and Awards 1637446 and 2024247, and the Air Force Office of Scientific Research under Award FA2386-17-1-4658.

[1]R. Jitosho, N. Agharese, and A. M. Okamura are with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA.
[2]Z. Manchester is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA.
Corresponding author e-mail: rjitosho@stanford.edu

to real vine robot behaviors.

The remainder of the paper is organized as follows: Section II describes the details of the simulator framework. Section III presents a method for fitting model parameters to bridge the sim-to-real gap. Section IV characterizes the performance of the simulator framework in order to validate the methods proposed in Sections II and III. Finally, Section V summarizes our conclusions.

II. Simulator Framework

We draw from previous work on dynamic simulation of rigid bodies with contact to design a simulator that is computationally efficient. We make three key design choices: First, we use a dynamic model with a constant-size state vector so that computational complexity remains fixed regardless of robot growth over time. Moreover, constant-size state vectors are amenable to classical motion planning strategies. In contrast, a variable length state vector would add significant complexity during motion planning (e.g., implementing a hybrid system where the model changes at specific timesteps in the trajectory). Second, we use an impulse-velocity dynamics formulation. Because collisions between rigid bodies result in velocity discontinuities, instantaneous accelerations and forces at the moment of impact become infinite. To address this, dynamics are written in terms of velocities and impulses. Third, we use the Lagrange multiplier formulation originally presented by Baraff [18] to define the model and equations of motion. The benefits of the Lagrange multiplier formulation are in its simplicity: (1) all coordinates are in the world frame, and (2) no recursive computations are required (as is the case with the recursive Newton-Euler and Featherstone algorithms [19], [20]). Furthermore, these benefits can be achieved while matching the computational efficiency of traditional minimal-coordinate techniques [18], and outperforming them in numerical robustness and scalability [21].

In the remainder of this section, we present the details of the simulator framework. In II-A, we provide additional background information on the Lagrange multiplier method. In II-B, we explain the simplified rigid-body model used to abstract the vine robot. In II-C to II-E, we describe the constraints that must be satisfied at any given time. In II-F and II-G, we write out the stiffness and damping models as well as the full robot dynamic model. Finally, in II-H, we combine all previous subsections into a single algorithm for simulating vine robot dynamics.

A. Lagrange Multiplier Formulation

In the Lagrange multiplier formulation, rigid bodies are described with maximal coordinates, which equates to 3 degrees of freedom for each rigid body (assuming planar motion). Joints between bodies are modeled with explicit constraints, and Lagrange multipliers associated with constraint forces (or in this work, constraint impulses) are calculated at each time step along with the next state.

Constraints are written in the form $c(\mathbf{q}) = \mathbf{0}$, where the function $c(\mathbf{q})$ defines a vector of constraint errors given a robot configuration $\mathbf{q}$. The constraint impulses act in opposition to the constraint errors. Specifically, the constraint impulse vector $\boldsymbol{\lambda}$ is defined in a particular basis that depends on $c(\mathbf{q})$, and the mapping of $\boldsymbol{\lambda}$ from the constraint basis to maximal coordinates is the Jacobian of the constraint equation, i.e. $J = \frac{\partial c}{\partial \mathbf{q}}(\mathbf{q})$. This yields the following dynamics:

$$M\Delta\mathbf{v} = J^\top\boldsymbol{\lambda} + \mathbf{F}\Delta t, \tag{1}$$

where M is the (constant) mass matrix, $\Delta\mathbf{v} = \mathbf{v}_{k+1} - \mathbf{v}_k$ is the change in velocity from timestep k to $k+1$, $J^\top$ maps constraint impulses $\boldsymbol{\lambda}$ into maximal coordinates, F contains other forces such as gravity, and Δt is the timestep length.

We note two important characteristics of the dynamics formulation: First, dynamics are at the level of velocities and impulses rather than accelerations and forces due to the discontinuities that arise in contact dynamics. Second, this is an implicit formulation wherein simulating dynamics requires finding $\boldsymbol{\lambda}$ and $\mathbf{v}_{k+1}$ such that the resulting motion satisfies the system constraints at timestep $k+1$. Thus, this formulation does not suffer from the constraint drift problem common in naive maximal-coordinate implementations [21], [22].

B. Rigid-Body Model

We define a dynamic model that abstracts the vine robot as a series of rigid bodies. Since we use maximal coordinates, the configuration for each body is (x, y, θ), where (x, y) is the position of the center of mass and θ is the orientation, all expressed in the world frame (Fig. 2). The configuration $\mathbf{q}$ for the entire vine robot is the concatenation of the configurations for each body, ordered from the base to the tip. Similarly, the velocity vector $\mathbf{v}$ for the entire vine robot is the concatenation of the velocity vectors for each body.

The series of rigid bodies are connected via alternating pin and prismatic joints. The first pin joint is between the first body (at the base of the robot) and the world frame origin. An illustration of these alternating joints is shown in Fig. 2. The use of prismatic joints is an abstraction of the actual mechanism for growth. In reality, vine robots grow via eversion, i.e. lengthening is achieved by adding new material to the tip. However, our abstraction enables a fixed-length state vector, which is important for fast computation and compatibility with motion planning.

C. Joint Constraints

In this section, we define the joint constraints of the rigid-body model.

1) Pin Joint: Let bodies 1 and 2 be connected via a pin joint, where body 2 is more distal. The corresponding pin joint constraint is that the distal endpoint of body 1 must have the same coordinates as the proximal endpoint of body 2 as illustrated in Fig. 3. In algebraic form, the pin constraint is the vector equation:

$$\begin{bmatrix}(x_1 + d\cos(\theta_1)) - (x_2 - d\cos(\theta_2))\\ (y_1 + d\sin(\theta_1)) - (y_2 - d\cos(\theta_2))\end{bmatrix} = \mathbf{0}, \tag{2}$$

where (x_i, y_i, θ_i) is the configuration of body i and d is the distance from the center of mass to the body's endpoint.

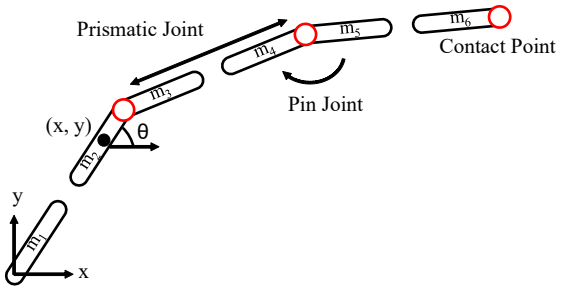

Fig. 2. Illustration of the rigid-body model used in the simulator. The vine robot is abstracted as a series of rigid bodies $(m_1, \ldots, m_n)$ with alternating pin and prismatic joints. For each rigid body, the configuration contains the position of the center of mass and the orientation in the world frame. Contact points (red) are used for handling environment interactions.

2) Prismatic Joint: Let bodies 1 and 2 be connected via a prismatic joint, where body 2 is more distal. The corresponding prismatic joint constraint is that the two bodies must be collinear. For this to be true, the following vector equation must be satisfied:

$$\begin{bmatrix} \mathbf{b}_1 \cdot \mathbf{h} \\ \mathbf{b}_2 \cdot \mathbf{h} \end{bmatrix} = \begin{bmatrix} -\sin(\theta_1) & \cos(\theta_1) \\ -\sin(\theta_2) & \cos(\theta_2) \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} = \mathbf{0}, \quad (3)$$

where $\mathbf{b}_i$ is the vector normal to body i and $\mathbf{h}$ is the vector from body 1's center of mass to that of body 2 (Fig. 3).

3) Total Joint Constraint Equation: The joint constraint for the entire robot is the concatenation of Equations (2) and (3) for all bodies:

$$c(\mathbf{q}) = \mathbf{0}, \quad (4)$$

where $c(\mathbf{q})$ computes the left-hand sides of the joint constraint equations for each joint given robot configuration $\mathbf{q}$.

D. Contact Constraints

We enforce contact constraints at specific "contact points" along the robot. The contact points in Figs. 2 and 3 are shown in red. In our implementation, contact points correspond to pin joints, but this choice is arbitrary, and number and location of contact points may be freely chosen.

Each contact point must satisfy the contact complementary constraints, which can be summarized as: (1) no interpenetration (a contact point cannot be inside an object); (2) normal forces exerted by objects must be non-negative (an object cannot pull the robot towards it); and (3) normal forces can only be exerted when the distance between the contact point and the object is zero,

$$\begin{aligned} \Phi(\mathbf{q}) &\geq \mathbf{0} \\ \mathbf{n} &\geq \mathbf{0} \\ \Phi(\mathbf{q})^\top \mathbf{n} &= 0, \end{aligned} \quad (5)$$

where Φ computes the vector of signed distances between each contact point and each object in the environment, and $\mathbf{n}$ is a vector of the normal forces exerted by the environment at the contact points. The direction of each normal force is the outward-pointing surface normal vector of each object, so $\mathbf{n}$ need only store a scalar value for each normal force.

This method for handling contact interactions is an approximation of actual contact interactions whereby contact

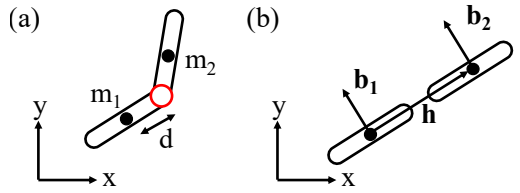

Fig. 3. Illustrations of joint constraints. (a) For a pin joint, endpoints from body 1 (m_1) and body 2 (m_2) must be coincident (red circle). (b) For a prismatic joint, we define normal vectors $\mathbf{b}_i$ for body i, as well as vector $\mathbf{h}$, which goes from body 1's center of mass to that of body 2. The prismatic joint constraint is that the two bodies be collinear, i.e. $\mathbf{b}_1 \perp \mathbf{h}$ and $\mathbf{b}_2 \perp \mathbf{h}$.

can occur at any point along the robot. Subsets of contact points can approximate line contacts, and sliding contact is also captured. Potential problems arise when the spacing between contact points is large enough that simulations become unrealistic (e.g. an object can pass through the robot). Thus, it is important to place contact points such that the distance between neighboring points is small compared to the size of objects in the environment. This results in a trade-off where increasing the number of contact points improves the accuracy of contact behavior but increases computational complexity. As a rule of thumb, the maximum distance between contact points should be smaller than the minimum corner radius of all objects.

E. Growth Constraints

In our framework, we treat the growth rate of the robot as a control input $\mathbf{u}$. In doing so, we introduce a growth rate constraint and a corresponding constraint impulse $\mathbf{w}$. While we could have defined growth forces and included this in the external force vector F in Equation (1), we chose to add a growth rate constraint because prior work with hardware implementations often used a feedback controller for the growth rate [5], [23]. With a feedback controller, actuation is adjusted to reach a target growth rate. Similarly, with the growth rate constraint in our simulator framework, constraint impulses are computed to achieve a target growth rate. The growth constraint is represented as:

$$g(\mathbf{q}, \mathbf{v}) - \mathbf{u} = \mathbf{0}, \quad (6)$$

where the function $g(\mathbf{q}, \mathbf{v})$ computes the growth rate of each prismatic joint via robot kinematics.

While there is no limit on the length of prismatic joints, contact points must remain well-spaced. In practice, we split the overall desired growth rate equally amongst prismatic joints. While this is different from an actual vine robot (growing from the tip), it keeps contact points evenly spaced.

F. Bending Stiffness and Damping

We model the system response to bending with torsion springs and dampers at the pin joints. We express this as:

$$\tau(\mathbf{q}, \mathbf{v}) = -K\theta(\mathbf{q}) - C\dot{\theta}(\mathbf{v}), \quad (7)$$

where K is the stiffness matrix, $\theta(\mathbf{q})$ computes the pin joint angles given a robot configuration, C is the damping matrix, and $\dot{\theta}(\mathbf{v})$ computes the pin joint velocities given a robot velocity. Incorporating this into the robot dynamics given

by Equation (1) requires a matrix R that maps these torques into maximal coordinates:

$$\mathbf{F} = R\tau(\mathbf{q}, \mathbf{v}). \tag{8}$$

G. Dynamics

Having defined the system constraints and external forces, we can write out the robot dynamics. The system constraints include joint constraints, contact constraints, and growth constraints. Per the Lagrange multiplier formulation, we introduce three corresponding constraint impulses that we refer to as $\boldsymbol{\lambda}$, $\mathbf{n}$, and $\mathbf{w}$. Combining these with the spring and damping forces yields:

$$M(\mathbf{v}_{k+1} - \mathbf{v}_k) = J^\top \boldsymbol{\lambda} + L^\top \mathbf{n} + G^\top \mathbf{w} + \mathbf{F}\Delta t, \tag{9}$$

where M is the mass matrix, $\mathbf{v}_i$ is the velocity at timestep i, $\boldsymbol{\lambda}$, $\mathbf{n}$, and $\mathbf{w}$ are constraint impulses, $J^\top$, $L^\top$, and $G^\top$ are Jacobians that map the constraint impulses into maximal coordinates, $\mathbf{F}$ captures spring and damping torques, and Δt is the length of the timestep.

For the configuration dynamics, we use the semi-implicit Euler method commonly used for simulating rigid-body dynamics with contact [24]. Specifically,

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{v}_{k+1}\Delta t, \tag{10}$$

where $\mathbf{q}_k$ and $\mathbf{q}_{k+1}$ are the configurations at timestep k and $k+1$ respectively.

H. Simulator Algorithm

We now synthesize the information from the previous sections and present the simulator algorithm. The high-level steps are listed in Algorithm 1.

Simulating the dynamics requires finding values for the constraint impulses such that the resulting motion satisfies the system constraints at the end of the time step. This is equivalent to solving a feasibility problem satisfying:

$$\begin{aligned} M(\mathbf{v}_{k+1} - \mathbf{v}_k) &= J^\top \boldsymbol{\lambda} + L^\top \mathbf{n} + G^\top \mathbf{w} + \mathbf{F}\Delta t \\ \mathbf{q}_{k+1} &= \mathbf{q}_k + \mathbf{v}_{k+1}\Delta t \\ c(\mathbf{q}_{k+1}) &= \mathbf{0} \\ \Phi(\mathbf{q}_{k+1}) &\geq \mathbf{0} \\ \mathbf{n} &\geq \mathbf{0} \\ \Phi(\mathbf{q}_{k+1})^\top \mathbf{n} &= 0 \\ g(\mathbf{q}_{k+1}, \mathbf{v}_{k+1}) - \mathbf{u} &= \mathbf{0}, \end{aligned} \tag{11}$$

where the decision variables are $\mathbf{q}_{k+1}$, $\mathbf{v}_{k+1}$, $\boldsymbol{\lambda}$, $\mathbf{n}$, and $\mathbf{w}$.

Algorithm 1 Simulate Dynamics

Require: $\mathbf{q}_k$, $\mathbf{v}_k$, $\mathbf{u}_k$, Δt
- compute $J(\mathbf{q}_k)$ and $c(\mathbf{q}_k)$
- compute $L(\mathbf{q}_k)$ and $\Phi(\mathbf{q}_k)$
- compute $G(\mathbf{q}_k, \mathbf{v}_k)$ and $g(\mathbf{q}_k, \mathbf{v}_k)$
- solve Quadratic Program (QP)
- $\mathbf{v}_{k+1} \leftarrow$ QP optimal solution
- $\mathbf{q}_{k+1} \leftarrow \mathbf{q}_k + \mathbf{v}_{k+1}\Delta t$
- **return** $\mathbf{q}_{k+1}$, $\mathbf{v}_{k+1}$

We apply two simplifications that reduce the computation cost. First, we linearize the constraints and use Equation (10) to approximate $\delta\mathbf{q}$ as $\mathbf{v}_{k+1}\Delta t$. For the joint constraint,

$$c(\mathbf{q}_{k+1}) \approx c(\mathbf{q}_k) + J\mathbf{v}_{k+1}\Delta t, \tag{12}$$

where $J = \frac{\partial c}{\partial \mathbf{q}}(\mathbf{q}_k)$. For the contact constraint,

$$\Phi(\mathbf{q}_{k+1}) \approx \Phi(\mathbf{q}_k) + L\mathbf{v}_{k+1}\Delta t, \tag{13}$$

where $L = \frac{\partial \Phi}{\partial \mathbf{q}}(\mathbf{q}_k)$. For the growth constraint,

$$\begin{aligned} g(\mathbf{q}_{k+1}, \mathbf{v}_{k+1}) &\approx g(\mathbf{q}_k, \mathbf{v}_k) + g_q\mathbf{v}_{k+1}\Delta t + g_v\Delta\mathbf{v} \\ &\approx \bar{g}(\mathbf{q}_k, \mathbf{v}_k) + \bar{G}\mathbf{v}_{k+1}, \end{aligned} \tag{14}$$

where

$$\begin{aligned} g_q &= \frac{\partial g}{\partial \mathbf{q}}(\mathbf{q}_k), \\ g_v &= \frac{\partial g}{\partial \mathbf{v}}(\mathbf{v}_k), \\ \Delta\mathbf{v} &= \mathbf{v}_{k+1} - \mathbf{v}_k, \\ \bar{g}(\mathbf{q}_k, \mathbf{v}_k) &= g(\mathbf{q}_k, \mathbf{v}_k) - g_v\mathbf{v}_k, \text{and} \\ \bar{G} &= g_q\mathbf{v}_{k+1}\Delta t + g_v. \end{aligned}$$

These linearizations eliminate the dependence of the constraints on $\mathbf{q}_{k+1}$. Thus, we remove $\mathbf{q}_{k+1}$ from the feasibility problem and compute it with Equation (10) once $\mathbf{v}_{k+1}$ is known. Second, we reformulate the linearized feasibility problem as the following quadratic program (QP):

$$\begin{aligned} \underset{\mathbf{v}_{k+1}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{v}_{k+1}^\top M\mathbf{v}_{k+1} - \mathbf{v}_{k+1}^\top(M\mathbf{v}_k + \mathbf{F}\Delta t) \\ \text{subject to} \quad & c(\mathbf{q}_k) + J\mathbf{v}_{k+1}\Delta t = \mathbf{0}, \\ & \Phi(\mathbf{q}_k) + L\mathbf{v}_{k+1}\Delta t \geq \mathbf{0}, \\ & \bar{g}(\mathbf{q}_k, \mathbf{v}_k) + \bar{G}\mathbf{v}_{k+1} - \mathbf{u}_k = \mathbf{0}. \end{aligned} \tag{15}$$

The feasibility problem (11) corresponds to the Karush-Kuhn-Tucker (KKT) conditions of (15). This allows us to leverage fast, efficient QP solvers like the Operator Splitting Quadratic Program (OSQP) solver [25] used in this work.

III. Fitting Model Parameters

The framework presented in Section II is sufficient for generating qualitatively realistic vine robot behaviors. However, it is necessary to bridge the sim-to-real gap by fitting model parameters to data. In our work, the rigid-body model is an abstraction of the vine robot, so we are unable to collect explicit measurements of the state vector defined in Section II-B. Thus, we are faced with a joint state *and* parameter estimation problem. A common technique for estimating parameters given missing or hidden variables is the Expectation-Maximization algorithm (EM) [26]. However, we can collect video data of a vine robot in motion, which provides an observation of the hidden state variables. By leveraging this additional information, we can use a batch nonlinear least-squares method and avoid complexities from the recursion in EM.

We pose the joint state and parameter estimation problem as a single optimization problem and simultaneously

fit model parameters as well as a trajectory. This method assumes the presence of errors in both the dynamic model as well as the measurements (i.e. video data). Thus, rather than treating the measurements as ground truth and placing hard constraints on satisfying dynamics, we form an objective function with two goals: (1) minimize deviations from the reference trajectory, and (2) minimize deviations from the dynamic model. Specifically, we solve:

$$\begin{aligned} \underset{Z,K,C,\mathbf{r}}{\text{minimize}} \quad & \sum_{i=1}^{N} ||\hat{\mathbf{p}}_i - p(\mathbf{q}_i)||^2 + \sum_{i=1}^{N-1} \mathbf{r}_i^\top R \mathbf{r}_i \\ \text{subject to} \quad & Z_{i+1} - f(Z_i, K, C) = \mathbf{r}_i, \quad i = 0, \ldots, N-1, \\ & c(\mathbf{q}_i) = \mathbf{0}, \quad i = 0, \ldots, N, \\ & g(\mathbf{q}_i, \mathbf{v}_i) = \mathbf{u}_i, \quad i = 0, \ldots, N, \\ & \phi(\mathbf{q}_i) = \mathbf{0}, \quad i = 0, \ldots, N. \end{aligned} \tag{16}$$

where $Z = \{Z_1, \ldots, Z_N\}$ is a robot trajectory over N timesteps, Z_i is the concatenation of configuration $\mathbf{q}_i$ and velocity $\mathbf{v}_i$, K and C are stiffness and damping matrices as defined in Equation (7), and $\mathbf{r}_i$ is the deviation from the dynamic model from timestep i to $i+1$. Measurements $\hat{\mathbf{p}}_i = \{\mathbf{p}_i^1, \ldots, \mathbf{p}_i^n\}$ are points that are evenly spaced along the length of the real robot in the i^{th} video frame. We use computer vision techniques to find these coordinates for each frame in the video (e.g., localization of fiducials on the vine robot). In order to compare this set of points with a simulated vine configuration $\mathbf{q}_i$, we define a function $p(\mathbf{q})$ that interpolates n points along the length of the simulated robot in order to correspond with the extracted measurements $\hat{\mathbf{p}}_i$. R is a diagonal matrix of weights that roughly translates to the level of trust in the dynamic model. This is because increasing the values in R will encourage lower values for $\mathbf{r}$ and thus smaller deviations from the model. $f(Z_i, K, C)$ computes the state at the next time step. The last three constraints ensure that the joint, contact, and growth constraints are satisfied at all timesteps. Because we have a video reference, we know *a priori* which rigid bodies are in contact with an object at each time step. Thus, we use a modified contact constraint $\phi(\mathbf{q}_i)$ to compute the signed distances for cases where the distance ought to be 0.

IV. Performance

In this section, we characterize the performance of our simulator framework. First, we validate runtime performance and measure how this scales with model size. Second, we discuss simulator accuracy by comparing real vine robot behaviors with simulated behaviors produced with fit models. Finally, we demonstrate general environment interactions by providing an example simulation in a cluttered environment.

A. Runtime and Scaling

To characterize the runtime performance of the simulator, we consider a vine robot growing at a constant growth rate in an environment with a single, circular object. Fig. 4 shows an example of a 4-second trajectory in this environment, with a discretization of the vine robot into 30 rigid bodies. In this example, the timestep Δt was 10 ms, and the average runtime per timestep was 3.0 ms.

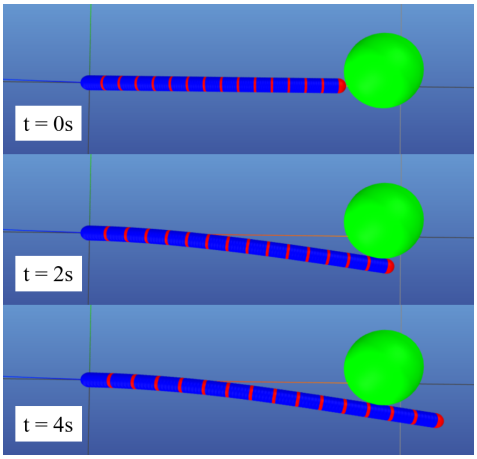

Fig. 4. Frames from a visualization of a simulated trajectory. A circular object in the environment is shown in green. The left end of the robot was fixed to the origin, and a constant growth rate control input was applied. The model discretizes the vine robot into 30 rigid bodies. Contact points are shown in red.

We repeated this procedure for different discretizations of the vine robot, which correspond to different numbers of bodies in the model, and plotted the average runtime per timestep for each model in Fig. 5. These results show that (1) there is a linear relationship between model size and simulator runtime, and (2) the simulator runs in real time for models with upwards of 70 bodies.

B. Accuracy

To characterize the accuracy of behaviors generated by the simulator framework, we present two examples of fitting model parameters using the optimization problem defined in Section III. In these examples, we specify a 10-body model and define a mass matrix that models the bodies as hollow cylinders. We fit scalar values for the stiffness and damping coefficients (i.e. we make simplifying assumptions that stiffness and damping are the same at each joint, and that these torques are linear in behavior). However, the fitting procedure can accommodate more general stiffness and damping models as well. Both examples use Interior

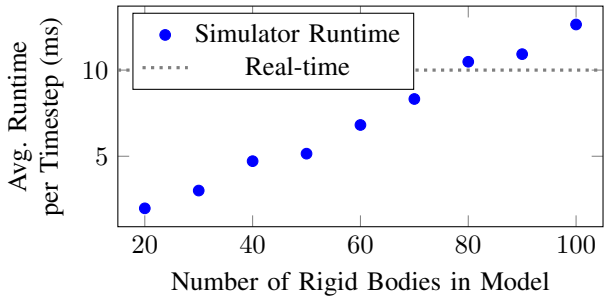

Fig. 5. Runtime results for different discretizations of the vine robot. For each model, we computed a 4-second trajectory, and recorded the average runtime per timestep. The plot shows a linear relationship between runtime and model size. The length of each timestep was 10 ms, demonstrating that the simulator is able to generate trajectories in real time for upwards of models with 70 bodies.

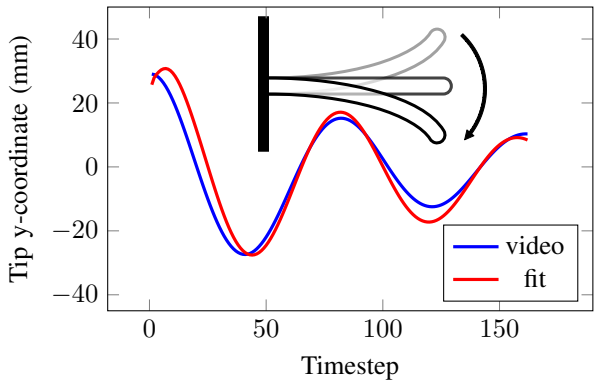

Fig. 6. Results from the fitting method presented in this paper for a cantilevered robot oscillating upon release from an initial displacement in the y-axis (illustrated). The plot compares the y-coordinates of the robot tip in the trajectory from the reference against that from the fit model. The mean absolute error is 3.3 mm.

Point OPTimizer (IPOPT) as the optimization solver [27].

1) Oscillation: As a baseline example, we consider the scenario of a cantilevered robot oscillating upon release from an initial displacement (Fig. 6). In this example, there are no objects in the environment, and there is no growth of the robot over time. The reference trajectory used in the fitting procedure came from experimental high-speed video of an inflated beam marked with fiducials. We extracted a section of the video corresponding to two oscillations of the beam. The full video showed third-order dynamics that are noticeable over a longer time horizon, and we plan to address modeling this more complex behavior in future work.

The results of the fitting procedure are plotted in Fig. 6. As a metric for accuracy, we compared the y-coordinate trajectory of the robot tip between the fit model and the real robot. The mean absolute error of the y-coordinate was 3.3 mm. Our results validate the ability of the simulator framework to capture bending modes of the vine robot.

2) Growth Into Wall: As an example of environment interaction, we consider the scenario of a vine robot that grows and comes in contact with a wall, causing the distal endpoint to slide along the wall as the robot bends and continues growing. We illustrate the contact interaction in Fig. 7. The reference trajectory used in the fitting procedure came from a video of the scenario described above. We extracted a section of the video during which the robot tip was in contact with the wall. In the fitting optimization, the modified contact constraint $\phi(\mathbf{q}_i)$ enforces the robot tip to have zero distance from the wall.

The results of the fitting procedure are plotted in Fig. 7. As a metric for accuracy, we compared the x-coordinate trajectory of the robot tip between the fit model and the real robot. The mean absolute error of the x-coordinate was 1.1 mm. Our results validate the ability of the fitting procedure to optimize parameters in more complicated scenarios involving contact and growth.

C. Cluttered Environment Demonstration

To demonstrate that the simulator can generate realistic behaviors with multiple contact interactions, we present a simulation of a robot growing in a cluttered environment.

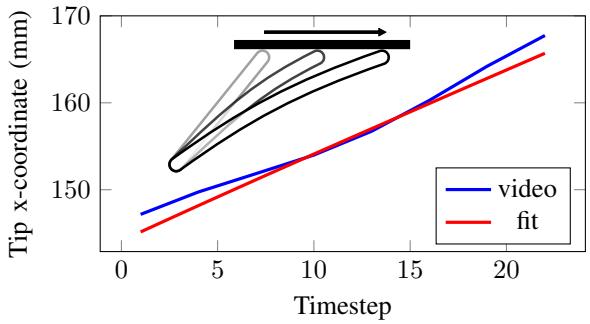

Fig. 7. Results from the fitting method presented in this paper for a vine robot growing into a wall (illustrated). The plot compares the x-coordinates of the robot tip in the trajectory from the reference against that from the fit model. The mean absolute error is 1.1 mm.

We based the environment structure on an existing video to show the qualitative agreement between the simulated and real trajectories. For the simulation, we used a model with 40 bodies and generated a 3-second trajectory. The results are overlaid on the original video in Fig. 1. This example features simultaneous contact interactions at multiple points along the vine robot as well as navigation through passageways.

V. Conclusion

In this work, we presented a dynamics simulator for soft growing robots that generates realistic behaviors, handles environment interactions, runs in real time, and is amenable to motion planning. We designed a framework that uses a simplified multi-link model and incorporates existing methods for rigid-body dynamics with contact. We provided a fitting procedure that bridges the sim-to-real gap. Finally, we characterized the performance of the simulator framework in terms of runtime, accuracy, and capability. The computational efficiency and amenability to motion planning of our high-speed simulator validates its potential for use in the control of soft robots. Our implementation of the simulator framework is available at `https://github.com/charm-lab/Vine_Simulator`.

We are interested in multiple areas of future work. First, if computational constraints are less conservative, the model fidelity can be increased to better capture vine robot behaviors. For example, vine robots grow by adding new material at the tip, so a time-varying mass matrix would more accurately model a growing vine robot. Other examples for improving the fidelity include using a soft-contact model and incorporating the slight deformation of the robot tube, fitting higher-order stiffness models to capture non-linear bending behaviors, and incorporating the effects of beam pressure. Another direction for future work is extending the dynamics model to 3D motion. Finally, we are interested in incorporating bending actuation for steering so that the presented simulator framework can be utilized in motion planning strategies.

Acknowledgment

The authors thank Joseph Greer, Laura Blumenschein, James Pillot IV, and Lauren Pitzer for contributing vine robot video data.

REFERENCES

[1] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, and S. Cotin, "SOFA: A Multi-Model Framework for Interactive Physical Simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, ser. Studies in Mechanobiology, Tissue Engineering and Biomaterials, Y. Payan, Ed. Berlin, Heidelberg: Springer, 2012, pp. 283–321.

[2] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "ChainQueen: A real-time differentiable physical simulator for soft robotics," in *International Conference on Robotics and Automation*, 2019, pp. 6265–6271.

[3] W. Huang, X. Huang, C. Majidi, and M. K. Jawed, "Dynamic simulation of articulated soft robots," *Nature Communications*, vol. 11, no. 1, p. 2233, 2020.

[4] P. Slade, A. Gruebele, Z. Hammond, M. Raitor, A. M. Okamura, and E. W. Hawkes, "Design of a soft catheter for low-force and constrained surgery," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 174–180.

[5] M. M. Coad, L. H. Blumenschein, S. Cutler, J. A. R. Zepeda, N. D. Naclerio, H. El-Hussieny, U. Mehmood, J.-H. Ryu, E. W. Hawkes, and A. M. Okamura, "Vine robots: Design, teleoperation, and deployment for navigation and exploration," *IEEE Robotics Automation Magazine*, vol. 27, no. 3, pp. 120–132, 2020.

[6] E. W. Hawkes, L. H. Blumenschein, J. D. Greer, and A. M. Okamura, "A soft robot that navigates its environment through growth," *Science Robotics*, vol. 2, no. 8, 2017.

[7] L. H. Blumenschein, N. S. Usevitch, B. H. Do, E. W. Hawkes, and A. M. Okamura, "Helical actuation on a soft inflated robot body," in *IEEE International Conference on Soft Robotics*, 2018, pp. 245–252.

[8] L. H. Blumenschein, A. M. Okamura, and E. W. Hawkes, "Modeling of bioinspired apical extension in a soft robot," in *Biomimetic and Biohybrid Systems*, ser. Lecture Notes in Computer Science, M. Mangan, M. Cutkosky, A. Mura, P. F. Verschure, T. Prescott, and N. Lepora, Eds. Cham: Springer International Publishing, 2017, pp. 522–531.

[9] J. D. Greer, T. K. Morimoto, A. M. Okamura, and E. W. Hawkes, "A soft, steerable continuum robot that grows via tip extension," *Soft Robotics*, vol. 6, no. 1, pp. 95–108, 2018.

[10] D. A. Haggerty, N. D. Naclerio, and E. W. Hawkes, "Characterizing environmental interactions for soft growing robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 3335–3342.

[11] J. D. Greer, L. H. Blumenschein, R. Alterovitz, E. W. Hawkes, and A. M. Okamura, "Robust navigation of a soft growing robot by exploiting contact with the environment," *The International Journal of Robotics Research*, vol. 39, no. 14, pp. 1724–1738, Dec. 2020.

[12] M. Selvaggio, L. A. Ramirez, N. D. Naclerio, B. Siciliano, and E. W. Hawkes, "An obstacle-interaction planning method for navigation of actuated vine robots," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 3227–3233.

[13] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[14] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 5026–5033.

[15] S. Miller, "Simscape Multibody Multiphysics Library." [Online]. Available: https://github.com/mathworks/Simscape-Multibody-Multiphysics-Library/releases/tag/20.2.4.0

[16] M. Müller, M. Macklin, N. Chentanez, S. Jeschke, and T.-Y. Kim, "Detailed rigid body simulation with extended position based dynamics," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '20. Goslar, DEU: Eurographics Association, Oct. 2020, pp. 1–12. [Online]. Available: http://doi.org/10.1111/cgf.14105

[17] H. El-Hussieny, U. Mehmood, Z. Mehdi, S.-G. Jeong, M. Usman, E. W. Hawkes, A. M. Okamura, and J.-H. Ryu, "Development and evaluation of an intuitive flexible interface for teleoperating soft growing robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 4995–5002.

[18] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 137–146.

[19] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, no. 2, pp. 69–76, 1980.

[20] R. Featherstone, "Inverse Dynamics," in *Rigid Body Dynamics Algorithms*, R. Featherstone, Ed. Boston, MA: Springer US, 2008, pp. 89–100.

[21] J. Brüdigam and Z. Manchester, "Linear-time variational integrators in maximal coordinates," in *Proceedings of 14th workshop on the algorithmic foundations of robotics (WAFR '20)*, Jun. 2020, pp. 194–209.

[22] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 1, no. 1, pp. 1–16, 1972.

[23] J. Luong, P. Glick, A. Ong, M. S. deVries, S. Sandin, E. W. Hawkes, and M. T. Tolley, "Eversion and retraction of a soft robot towards the exploration of coral reefs," in *IEEE International Conference on Soft Robotics*, 2019, pp. 801–807.

[24] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction," *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.

[25] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[26] J. Kokkala, A. Solin, and S. Särkkä, "Expectation maximization based parameter estimation by sigma-point and particle smoothing," in *International Conference on Information Fusion*, 2014, pp. 1–8.

[27] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.