Dynamic manipulation of deformable objects with implicit integration

Simon Zimmermann[1], Roi Poranne[2], Stelian Coros[1]

***Abstract*— Due to their complex dynamics and high-dimensional configuration spaces, non-rigid objects such as cables, garments, bedding and various food items remain notoriously challenging for robots to manipulate effectively. In this paper, we therefore develop, validate and analyze model-based optimal control techniques for *dynamic* manipulation of deformable objects.**

We study, in particular, the application of both the batch Newton method and the stagewise Differential Dynamic Programming (DDP) approach to this challenging problem domain. On a technical level, we derive analytic formulations for all necessary derivatives, noting that numerically stable simulation of deformable objects demands implicit integration schemes, which do not have closed form solutions. While both DDP and Newton's method converge quadratically, our experiments and analysis show that the relative overall performance of these two approaches depends heavily on the dimensions of the control problems being solved. We demonstrate the efficacy of our trajectory optimization formulations through a variety of simulation and real-world experiments.

I. Introduction

Interaction with non-rigid objects such as garments, bedding or various food items is routinely required in many of our daily activities. And while tasks like changing the sheets on a bed or kneading pizza dough may seem trivial to us, they pose a tremendous challenge to robots. Indeed, dexterous robotic manipulation of deformable objects remains a grand challenge in the field, a technological barrier that must be overcome in the quest for robot intelligence.

In this paper, we formalize, evaluate and analyze model-based trajectory optimization (TO) techniques for dynamic manipulation of non-rigid objects. In robotics, TO is commonly used to compute open-loop solutions to optimal control problems in locomotion, manipulation and path planning. Given a dynamical system, TO leverages numerical optimization routines to generate a control sequence that optimizes a certain objective. While many successful TO approaches have been presented in the literature (see Sec. II-B), by and large they are not targeted towards soft body manipulation, as deformable objects exhibit very complex dynamics and high-dimensional configuration spaces. The challenges inherent to dynamic manipulation of non-rigid items arise due to the need to employ advanced constitutive material models that relate deformations to internal restorative forces. These challenges are further exacerbated since these deformable body simulations suffer from numerical stiffness issues, which can only be stably handled by applying *implicit* integration schemes. Conversely, explicit schemes tend to be unstable and blow-up quickly, making them practically unusable.

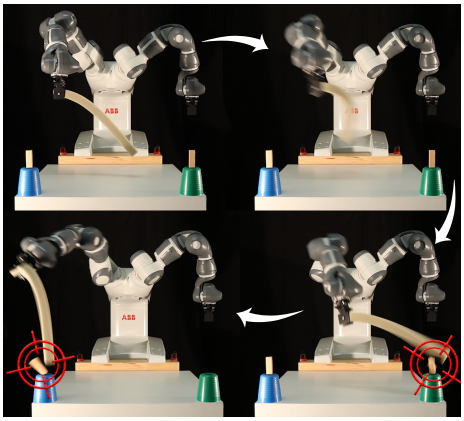

Fig. 1: A YuMi® IRB 14000 robot using a soft rod as a whip to hit two small wooden blocks. Starting from a statically stable configuration (top left), the robot performs a dynamic maneuver that begins by pulling the whip back to gain sufficient momentum (top right), and then swiftly knocking the two targets off the table (bottom).

We center our attention on the challenge of adapting two *single shooting* trajectory optimization strategies, Newton's method and Differential Dynamic Programming (DDP), to physical systems that are simulated forward in time using implicit integration schemes. Briefly, Newton's method is a batch approach that operates on the entire control sequence simultaneously, while DDP is a stagewise method that processes the control sequence recursively. Both of these methods use forward simulation as an integral part of their formulations. As mentioned, stable simulation of soft bodies (under reasonably large time steps) hinges on implicit integration schemes. Unlike explicit alternatives, implicit integration methods do not have per time-step solutions available in closed form. We therefore show how to leverage sensitivity analysis to analytically compute the first and second order derivatives which are essential for both Newton's method and DDP.

To evaluate the TO formulations we present in this paper, we make use of a dual-armed YuMi® IRB 14000 robot and different types of elastic objects made out of foam. YuMi can hold the foam objects with one or both grippers (the grasping configuration is prescribed as input), and it executes the joint-space trajectories computed with TO in an open-loop fashion. We model elastic objects using the Finite Element Method (FEM), and we employ the second-order accurate Backward Differentiation Formula (BDF2)

[1] The authors are with the Department of Computer Science, ETH, Zurich, Switzerland. simon.zimmermann@inf.ethz.ch; roi.poranne@inf.ethz.ch; scoros@gmail.com

[2]Roi Poranne is with the Department of Computer Science, University of Haifa, Haifa, Israel. roiporanne@cs.haifa.ac.il

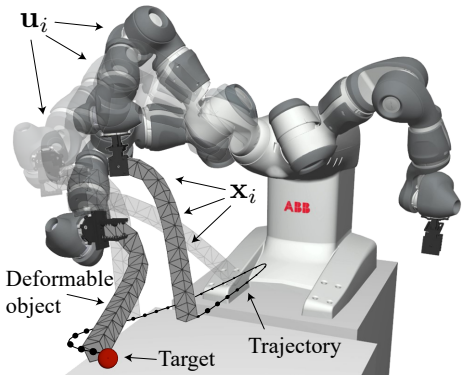

Fig. 2: An overview of the system. The YuMi robot is parameterized by its joint angles $\mathbf{u}$, and the deformable object by its node positions $\mathbf{x}$.

numerical integration scheme. The control objectives we experiment with are simple, taking on the form of target positions for selected points on the soft objects at different moments in time, yet they lead to rich, highly dynamic motions being performed. To demonstrate the efficacy of our TO formulations, we execute and analyze a variety of simulation and real-world experiments.

II. Background

A. FEM simulation

Similar to previous methods, we employ FEM to simulate the deformable object being manipulated. We refer the reader to closely related papers that discuss trajectory optimization for soft robot locomotion [1], [2], or quasi-static manipulation of deformable objects [3], [4], [5], [6]. [7] discusses an MPC approach for dynamic manipulation, using short time windows and linearized forces. For a review of earlier work, see [8].

In brief, FEM considers a mesh discretization of the manipulated object into a volumetric, tetrahedral mesh. We employ linear elements and a compressible neo-Hookean material model for our simulation. The specific choice of material model is not pivotal, so long as the simulation captures the behaviour of the real material well. While it cannot capture e.g. the viscoelastic behavior of the elastic foam we used, we find the neo-Hookean model to be satisfactory for our application.

The positions of all nodes at each time-step i are assembled into a vector $\mathbf{x}_i$ of dimension $3n$ (n being the number of nodes), constituting the *state* of the object. The deformation energy density of each element t, defined by its vertex coordinates $\mathbf{x}_{it}$, using a compressible Neo-Hookean material model is defined by

$$\Psi(\mathbf{x}_{it}) = \frac{\mu}{2}\text{tr}(\mathcal{F}_t^\mathsf{T}\mathcal{F}_t - I) - \mu \ln J_t + \frac{\kappa}{2}(\ln J_t)^2, \quad (1)$$

where $\mathcal{F}_t$ is the deformation gradient, $J_t = \det(\mathcal{F})$ is its determinant, and μ and κ are material parameters. The total elastic energy stored in the mesh $E_{\text{el}}(\mathbf{x}_i) = \sum_t \Psi(\mathbf{x}_{it})$ is the sum of the energies stored in each element. The parameters of the material model were identified experimentally.

The controls, which in our case are the joint angles of the robot, are stacked in a vector $\mathbf{u}_i$ (Fig. 2). A simple explicit Euler stepping rule is commonly formulated as follows:

$$\begin{aligned} \mathbf{v}_{i+1} &= \mathbf{v}_i - \frac{h}{m}\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}_i}W(\mathbf{x}_i, \mathbf{u}_i) \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + h\mathbf{v}_i, \end{aligned} \quad (2)$$

where h is the step size, and $W(\mathbf{x}_i, \mathbf{u}_i)$ is the total potential energy of the system at time step i. This energy comprises of the elastic energy of the foam $E_{\text{el}}(\mathbf{x}_i)$, gravity $E_{\text{g}}(\mathbf{x}_i)$ and soft contacts $E_{\text{con}}(\mathbf{x}_i)$ (see [2]). In addition, W contains a term that expresses the coupling of the foam and the robot's end effectors. This can be formulated as a quadratic penalty, i.e.

$$E_{\text{coup}}(\mathbf{x}, \mathbf{u}) = w_{\text{coup}}\|\mathbf{x}_i^j - \mathcal{K}(\mathbf{l}, \mathbf{u}_i)\|^2, \quad (3)$$

where w_{coup} is a weighting coefficient, $\mathbf{x}_i^j$ is the node in contact with the gripper, and $\mathcal{K}(\mathbf{l}, \mathbf{u}_i)$ is a standard forward kinematics function that computes the world coordinates of a point $\mathbf{l}$ expressed in the local coordinate frame of a robot's gripper. To summarize,

$$W(\mathbf{x}_i, \mathbf{u}_i) = E_{\text{el}}(\mathbf{x}_i) + E_{\text{g}}(\mathbf{x}_i) + E_{\text{con}}(\mathbf{x}_i) + E_{\text{coup}}(\mathbf{x}, \mathbf{u}).$$

Explicit integration schemes such as (2) are known to be unstable. While an explicit scheme might suffice for a simple problem like a pendulum on a cart, elastic FE meshes tend to blow-up after only a few steps. This makes explicit integration unviable for simulation and control, as they fail to reliably predict motions. Stabilization procedures for explicit integration are a matter of ongoing research. However, their computational footprint is either similar to implicit schemes, or their accuracy is reduced (see e.g. [9], [10], [11], [12]).

An implicit Euler time stepping scheme is equivalent to finding $\mathbf{x}_i^*$ that minimizes the functional [13],

$$U_i(\mathbf{x}_i, \ddot{\mathbf{x}}_i, \mathbf{u}_i) = \frac{h^2}{2}\ddot{\mathbf{x}}_i^T\mathbf{M}\ddot{\mathbf{x}}_i + W(\mathbf{x}_i, \mathbf{u}_i), \quad (4)$$

where $\ddot{\mathbf{x}}_i$ is an acceleration discretization, and $\mathbf{M}$ is the system's mass matrix. The discretizations we experimented with in this paper are the Backward Differentiation Formulas BDF1 approximation $\ddot{x}_i = (\frac{\mathbf{x}_i - 2\mathbf{x}_{i-1} + \mathbf{x}_{i-2}}{h^2})$ and BDF2, $\ddot{x}_i = (\frac{2\mathbf{x}_i - 5\mathbf{x}_{i-1} + 4\mathbf{x}_{i-2} - \mathbf{x}_{i-3}}{h^2})$. Eq. (4) means that we seek $\mathbf{x}_i^*$, a critical point of $U_i(\mathbf{x}_i)$, such that

$$\mathbf{g}(\mathbf{x}_i, \ddot{\mathbf{x}}_i, \mathbf{u}_i) = \nabla U_i(\mathbf{x}_i, \ddot{\mathbf{x}}_i, \mathbf{u}_i) = 0,$$

which is equivalent to Newton's law $\mathbf{f} = \mathbf{M}\ddot{\mathbf{x}}$. In the next sections we discuss how this condition can be integrated within a trajectory optimization problem.

B. Trajectory optimization algorithms

TO problems typically appear in the following form:

$$\min_{\mathbf{x},\mathbf{u}} \quad \ell_f(\mathbf{x}_n) + \sum_{i=0}^{n-1} \ell(\mathbf{x}_i, \mathbf{u}_i) \quad (5a)$$

$$\text{s.t.} \quad \mathbf{x}_{i+1} = f(\mathbf{x}_i, \mathbf{u}_i), \quad (5b)$$

where l and l_f are the running and final costs respectively, and f is an *explicit* transition function that can express, for

example, the stepping rule in (2). However, as noted, implicit integration is sometimes essential to avoid instability. TO problems with implicit integration would appear as follows:

$$\min_{\mathbf{x},\mathbf{u}} \quad \ell_f(\mathbf{x}_n) + \sum_{i=0}^{n-1} \ell(\mathbf{x}_i, \mathbf{u}_i) \tag{6a}$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{x}_{i+1}, \ddot{\mathbf{x}}_{i+1}, \mathbf{u}_i) = 0. \tag{6b}$$

We note that the inclusion of additional previous time steps (6b), e.g. $\mathbf{x}_{i-1}$ as part of $\ddot{\mathbf{x}}_{i+1}$ is uncommon in itself, and we further discuss this in Sec. IV-B.

Single shooting methods can be divided into *batch* or *stagewise*. The batch approach optimizes the entire trajectory simultaneously, often treating the problem as a general optimization problem. The stagewise approach on the other hand, starts from the end of the trajectory, and recursively computes a symbolic expression of the optimal controls at a certain time step as a function of the state in the preceding step. DDP is considered to first appear in [14] and become popular with [15]. The debate whether the stagewise approach is generally favorable has been ongoing for decades, usually citing superior performance of DDP in comparison to the more standard Newton's method [16], [17], [18]. Theoretically, DDP has been shown to exhibit quadratic convergence similar to Newton's method [18], [19], but experiments show that DDP iterations are faster, and less are required to reach a minimum. Previous work attributes the improved performance to two features DDP has: first, [17] shows that a DDP iteration requires fewer operations than a Newton iteration. However, there is an underlying assumption in [17] that the cost of evaluating the gradient of the transition function is linear in the number of variables, which is not true in the general case, nor in our specific case. Second, Newton's method can only use a second order approximation of the dynamics, while DDP employs fewer approximations in the forward pass, and therefore is "more accurate" [18]. While this statement is true, there is no proof this improved accuracy leads to improved convergence rates, and in our experiments, results vary.

We conclude this section by mentioning a few other stagewise methods. In [16], Pantoja introduced the stagewise Newton algorithm, which is equivalent to the batch Newton's method, but enjoys faster iterations. It was demonstrated that the method outperforms DDP, but it is later shown in [20] that the experiment contained an error, and in fact DDP and stagewise Newton are equivalent for the chosen problem, but not in general. DDP requires the second derivatives of the dynamics, which can be too costly to compute. Furthermore, they may cause some of the linear systems involved in DDP to become indefinite, which can halt the progress of the algorithm. Similar to the Gauss-Newton method, parts of the second derivatives can simply be discarded, reducing the cost and the risks of indefinite systems. In this case, DDP is usually referred to as iterative LQR (iLQR) [21]. Other variants enable the inclusion of control limits [22] and non-linear constraints [23].

III. Newton's method with implicit integration

As mentioned, the batch strategy ignores the structure of the problem (6). From this viewpoint it can be equivalently written as

$$\min_{\mathbf{X},\mathbf{U}} \quad \mathcal{O}(\mathbf{X}, \mathbf{U}) \tag{7a}$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{X}, \mathbf{U}) = 0, \tag{7b}$$

where $\mathbf{X}, \mathbf{U}$ are the stacked state and controls of the entire trajectory. While this problem can be solved directly using a constrained optimization method, the drawback of this approach is that the candidate solution can only be considered physically correct once the constraints are satisfied. The condensation approach would be to substitute the state variables with the control variables, and consequently reduce the size of the problem while transforming it to an unconstrained problem. In the presence of implicit constraints, this would seem to require inverting $\mathbf{g}$, but as noted in [24], for Newton's method we only require the total derivatives of the objective w.r.t. $\mathbf{U}$, and these are readily available via the implicit function theorem.

We bring here only the final expressions for the derivatives, and refer the reader to [25] for the full derivation. First, the gradient of (7a) can be computed from

$$\frac{\mathrm{d}\mathcal{O}}{\mathrm{d}\mathbf{U}} = \frac{\partial \mathcal{O}}{\partial \mathbf{X}}\mathbf{S} + \frac{\partial \mathcal{O}}{\partial \mathbf{U}}, \tag{8}$$

where

$$\mathbf{S} = -\left(\frac{\partial \mathbf{g}}{\partial \mathbf{X}}\right)^{-1} \frac{\partial \mathbf{g}}{\partial \mathbf{U}} \tag{9}$$

is the *sensitivity matrix*. We assume that $\frac{\partial \mathbf{g}}{\partial \mathbf{X}}$ is invertible, which is generally the case for elastic deformation. The Hessian is obtained by

$$\begin{aligned}\frac{\mathrm{d}^2\mathcal{O}}{\mathrm{d}\mathbf{U}^2} = \frac{\partial \mathcal{O}}{\partial \mathbf{X}}\left(\mathbf{S}^T\frac{\partial}{\partial \mathbf{X}}\mathbf{S} + \frac{\partial}{\partial \mathbf{U}}\mathbf{S}\right) + \\ + \mathbf{S}^T\frac{\partial^2 \mathcal{O}}{\partial \mathbf{X}^2}\mathbf{S} + \mathbf{S}^T\frac{\partial^2 \mathcal{O}}{\partial \mathbf{X}\partial \mathbf{U}} + \frac{\partial^2 \mathcal{O}}{\partial \mathbf{U}\partial \mathbf{X}}\mathbf{S} + \frac{\partial^2 \mathcal{O}}{\partial \mathbf{U}^2},\end{aligned} \tag{10}$$

where $\frac{\partial}{\partial \mathbf{X}}\mathbf{S}$, $\frac{\partial}{\partial \mathbf{U}}\mathbf{S}$ are tensor terms, and their precise expressions are found in [25]. Unfortunately, their computational cost is high, and they can cause the Hessian to become indefinite. To overcome that, we can apply the Gauss-Newton approach and simply drop these terms to get

$$\mathbf{H} = \mathbf{S}^T\frac{\partial^2 \mathcal{O}}{\partial \mathbf{X}^2}\mathbf{S} + \mathbf{S}^T\frac{\partial^2 \mathcal{O}}{\partial \mathbf{X}\partial \mathbf{U}} + \frac{\partial^2 \mathcal{O}}{\partial \mathbf{U}\partial \mathbf{X}}\mathbf{S} + \frac{\partial^2 \mathcal{O}}{\partial \mathbf{U}^2}. \tag{11}$$

The resulting iteration is both quicker to compute, and faster to converge. The same strategy is used within DDP, and when used, the process becomes equivalent to iLQR. Using the gradient and the Hessian (or its approximation), we can minimize (7) using Newton's method, e.g. by solving

$$\mathbf{H}\mathbf{d} = -\frac{\mathrm{d}\mathcal{O}}{\mathrm{d}\mathbf{U}} \tag{12}$$

to receive the search direction $\mathbf{d}$. We note one key difference: for each candidate $\mathbf{U}$, we must always compute the corresponding $\mathbf{X}$ to ensure that (7b) holds before evaluating

any of the derivatives. In the particular case of trajectory optimization, this can be achieved by simply running a forward simulation procedure, which guarantees that the constraints given by the dynamical system are satisfied.

IV. Differential dynamic programming

A. Preliminaries

We review DDP for the sake of completeness, following the introduction found in [22], [23]. Assuming we know what $\mathbf{x}_i$ is, we only need to optimize the control sequence $\mathbf{U}_i = (\mathbf{u}_i, \dots, \mathbf{u}_{n-1})$. In other words, we are required to minimize

$$\mathcal{J}_i(\mathbf{x}_i, \mathbf{U}_i) = \ell_f(\mathbf{x}_n) + \sum_{j=i+1}^{n-1} \ell(\mathbf{x}_j, \mathbf{u}_j), \tag{13}$$

where $\mathcal{J}_i(\mathbf{x}_i, \mathbf{U}_i)$ is the *cost-to-go* at time step i. All the subsequent states $\mathbf{x}_j, j = i+1, \dots, n$ are determined by the transition function $\mathbf{f}$. Eq. (13) can be written recursively as

$$\mathcal{J}_i(\mathbf{x}_i, \mathbf{U}_i) = \ell(\mathbf{x}_i, \mathbf{u}_i) + \mathcal{J}_{i+1}(\mathbf{x}_{i+1}, \mathbf{U}_{i+1}). \tag{14}$$

Next, assume that given $\mathbf{x}_{i+1}$, we can compute the optimal controls $\mathcal{J}_{i+1}(\mathbf{x}_{i+1}, \mathbf{U}_{i+1})$. Let

$$V_i(\mathbf{x}_i) = \min_{\mathbf{U}_i} \mathcal{J}_i(\mathbf{x}_i, \mathbf{U}_i) \tag{15}$$

be the *optimal cost-to-go*. By the assumption, we have an expression for $V_{i+1}(\mathbf{x}_{i+1}) = V_{i+1}(\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i))$. The dynamic programming principle, a.k.a. Bellman's equation, is then framed in the following formula:

$$V_i(\mathbf{x}_i) = \min_{\mathbf{u}_i} [\ell(\mathbf{x}_i, \mathbf{u}_i) + V_{i+1}(\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i))]. \tag{16}$$

DDP suggests to use a quadratic model for V_{i+1}. We follow the standard notation by removing the subscripts and defining $V' = V_{i+1}$. We define the argument of (16) by

$$\mathcal{Q}(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + V'(\mathbf{f}(\mathbf{x}, \mathbf{u})) \tag{17}$$

and expand it to second order

$$\begin{aligned} \mathcal{Q}(\mathbf{x} + d\mathbf{x}, \mathbf{u} + d\mathbf{u}) = \mathcal{Q} + \mathcal{Q}_{\mathbf{x}}^T d\mathbf{x} + \mathcal{Q}_{\mathbf{u}}^T d\mathbf{u} + \\ + \frac{1}{2} \begin{pmatrix} d\mathbf{x} \\ d\mathbf{u} \end{pmatrix}^T \begin{pmatrix} \mathcal{Q}_{\mathbf{xx}} & \mathcal{Q}_{\mathbf{xu}} \\ \mathcal{Q}_{\mathbf{ux}} & \mathcal{Q}_{\mathbf{uu}} \end{pmatrix} \begin{pmatrix} d\mathbf{x} \\ d\mathbf{u} \end{pmatrix}, \end{aligned} \tag{18}$$

where we use a subscript to denote the partial derivative w.r.t. that variable to be consistent with the standard DDP derivation. The derivatives can be evaluated to

$$\mathcal{Q}_{\mathbf{x}} = \ell_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^{\mathsf{T}} V'_{\mathbf{x}}, \quad \mathcal{Q}_{\mathbf{u}} = \ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}} V'_{\mathbf{x}} \tag{19a}$$

$$\mathcal{Q}_{\mathbf{xx}} = \ell_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^T V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}} \quad (+V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{xx}}) \tag{19b}$$

$$\mathcal{Q}_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^T V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{u}} \quad (+V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{uu}}) \tag{19c}$$

$$\mathcal{Q}_{\mathbf{ux}} = \ell_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^T V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}} \quad (+V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{ux}}). \tag{19d}$$

The terms in parentheses are the optional tensor terms mentioned before. They distinguish between DDP and iLQR. By taking the gradient of $\mathcal{Q}(\mathbf{x} + d\mathbf{x}, \mathbf{u} + d\mathbf{u})$ w.r.t. $d\mathbf{u}$ and setting it to zero we get

$$\mathcal{Q}_{\mathbf{u}} + \mathcal{Q}_{\mathbf{ux}} d\mathbf{x} + \mathcal{Q}_{\mathbf{uu}} d\mathbf{u} = 0. \tag{20}$$

Solving this equation for $d\mathbf{u}$ results in

$$\bar{\mathbf{u}} = \mathbf{u} + k + K(\bar{\mathbf{x}} - \mathbf{x}), \tag{21}$$

where the indices are again removed, and

$$k = -\mathcal{Q}_{\mathbf{uu}}^{-1} \mathcal{Q}_{\mathbf{u}}, \quad \text{and} \quad K = -\mathcal{Q}_{\mathbf{uu}}^{-1} \mathcal{Q}_{\mathbf{ux}}. \tag{22}$$

What remains to compute is V' and its derivatives, and we refer the reader again to [22], [23] for the full expressions.

B. Extended DDP

We extend DDP to the case where the cost and the transition function depends also on previous states and controls, which is the case in (6). Such problems are also known as higher-order Markov processes [26]. This has to be done to support implicit integration, that rely on velocities and accelerations. Similarly, we consider cost functions that depend on previous states and controls too. This allows us to put objectives on velocities and accelerations of the controls and states. We denote the sequence of controls starting from $\mathbf{u}_i$ and going back j steps by $\mathbf{U}_j^i$, i.e $\mathbf{U}_j^i = (\mathbf{u}_i, \dots, \mathbf{u}_{i-j})$ and likewise for $\mathbf{X}_j^i$. Then we define the trajectory optimization problem by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \ell_f(\mathbf{X}_j^n) + \sum_{i=0}^{n-1} \ell(\mathbf{X}_j^i, \mathbf{U}_j^i) \tag{23a}$$

$$\text{s.t.} \quad \mathbf{x}_{i+1} = \mathbf{f}(\mathbf{X}_j^i, \mathbf{u}_i). \tag{23b}$$

In contrast to the standard DDP, the recursive cost-to-go relation at time step i cannot be expressed solely as a function of the current state $\mathbf{x}_i$ and current and future controls $\mathbf{U}_i$. Rather, it must also depend on some earlier states and controls:

$$\mathcal{J}_i\left(\mathbf{X}_j^i, \mathbf{U}_{j-1}^{i-1}, \mathbf{U}_i\right) = \ell(\mathbf{X}_j^i, \mathbf{U}_j^i) + \mathcal{J}_i\left(\mathbf{X}_j^i, \mathbf{U}_{j-1}^i, \mathbf{U}_{i+1}\right). \tag{24}$$

The optimal cost-to-go is thus

$$V_i(\mathbf{X}_j^i, \mathbf{U}_{j-1}^{i-1}) = \min_{\mathbf{U}_i} \mathcal{J}_i\left(\mathbf{X}_j^i, \mathbf{U}_{j-1}^{i-1}, \mathbf{U}_i\right), \tag{25}$$

and Bellman's equation takes on the following form:

$$\begin{aligned} V_i(\mathbf{X}_j^i, \mathbf{U}_{j-1}^{i-1}) = \min_{\mathbf{u}_i} [\ell\left(\mathbf{X}_j^i, \mathbf{U}_j^i\right) + \\ + V_{i+1}\big(\underbrace{(\mathbf{f}(\mathbf{X}_j^i, \mathbf{u}_i), \mathbf{X}_{j-1}^i)}_{\mathbf{X}_j^{i+1}}, \mathbf{U}_{j-1}^i\big)]. \end{aligned} \tag{26}$$

To make this appear more familiar, we can define $\tilde{\mathbf{X}}_i = (\mathbf{X}_j^i, \mathbf{U}_{j-1}^{i-1})$, and then (26) becomes

$$V_i(\tilde{\mathbf{X}}_i) = \min_{\mathbf{u}_i} [\ell(\tilde{\mathbf{X}}_i, \mathbf{u}_i) + V_{i+1}(\tilde{\mathbf{f}}(\tilde{\mathbf{X}}_i, \mathbf{u}_i))], \tag{27}$$

where $\tilde{\mathbf{f}}(\tilde{\mathbf{X}}_i, \mathbf{u}_i) = \tilde{\mathbf{X}}_{i+1}$. In other words, we can treat the previous controls $\mathbf{U}_{j-1}^{i-1}$ simply as state variables, for the purpose of estimating the optimal $\mathbf{u}_i$.

An alternative derivation is obtained by explicitly adding auxiliary state variables, which equivalently removes the dependency on past states and controls. We replace each $\mathbf{x}_i$ by $\mathbf{x}_i^0, \dots, \mathbf{x}_i^{j-1}$, where $\mathbf{x}_i^0 = \mathbf{x}_i$ represents the current state, and $\mathbf{x}_i^l$ stores the previous l states. For each $\mathbf{u}_i$, we add state

Algorithm 1: Trajectory optimization: Newton

Input: Dynamical system, initial $\mathbf{u}$, initial $\mathbf{x}_0, \dot{\mathbf{x}}_0$
Output: Optimal control trajectory $\mathbf{u}$
while *criterion not reached* **do**
  Compute $\mathbf{x}(\mathbf{u})$ using forward simulation
  Compute $\frac{\mathrm{d}\mathcal{O}}{\mathrm{d}\mathbf{u}}$ (Eq. (7a))
  Compute $\mathbf{H}$ (Eq. (10) or (11))
  Solve $\mathbf{H}\mathbf{d} = -\frac{\mathrm{d}\mathcal{O}}{\mathrm{d}\mathbf{u}}$
  Run backtracking line search on α in $\mathbf{u} := \mathbf{u} + \alpha\mathbf{d}$

variables $\mathbf{y}_i^1, \ldots, \mathbf{y}_i^{j-1}$. We avoid adding $\mathbf{y}_i^0$ as we wish to preserve the standard DDP form. Then the problem becomes

$$\min_{\mathbf{x},\mathbf{u}} \quad \ell_f(\mathbf{x}_n, \mathbf{x}_n^1, \ldots, \mathbf{x}_n^{j-1}) + \tag{28a}$$

$$+ \sum_{i=0}^{n-1} \ell(\mathbf{x}_i, \mathbf{x}_i^1, \ldots, \mathbf{x}_i^{j-1}, \mathbf{y}_i^1, \ldots, \mathbf{y}_i^{j-1}, \mathbf{u}_i), \tag{28b}$$

$$\text{s.t.} \quad \mathbf{x}_{i+1}^0 = \mathbf{f}(\mathbf{x}_i, \mathbf{x}_i^1, \ldots, \mathbf{x}_i^{j-1}, \mathbf{y}_i^1, \ldots, \mathbf{y}_i^{j-1}, \mathbf{u}_i), \tag{28c}$$

$$\mathbf{y}_{i+1}^1 = \mathbf{u}_i, \tag{28d}$$

$$\mathbf{x}_{i+1}^l = \mathbf{x}_i^{l-1},\ \mathbf{y}_{i+1}^l = \mathbf{y}_i^{l-1},\ l = 1, \ldots, j-1, \tag{28e}$$

which can directly be solved with standard DDP.

C. DDP with implicit integration

With the foundation laid, we can finally solve (6) using DDP. The remaining challenge lies in computing the derivatives of $\mathcal{Q}$, since we do not have an explicit expression for $\mathbf{f}$. To be clear, we seek the derivatives of the implicit equivalent of (23b), and similar to (6b), i.e.

$$\mathbf{g}(\mathbf{x}_{i+1}, \mathbf{X}_j^i, \mathbf{u}_i) = 0. \tag{29}$$

These are readily available using the implicit function theorem again. To be consistent with the previous notation, we denote $\mathbf{x}' = \mathbf{x}_{i+1}$, and $\mathbf{x} = \mathbf{X}_j^i, \mathbf{u} = \mathbf{u}_i$. Then, we can state that

$$\mathbf{f}_\mathbf{x} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}'}^{-1} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}, \quad \mathbf{f}_\mathbf{u} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}'}^{-1} \frac{\partial \mathbf{g}}{\partial \mathbf{u}}. \tag{30}$$

Subsequently, $\mathcal{Q}_\mathbf{x}$ in (19) is

$$\mathcal{Q}_\mathbf{x} = \ell_\mathbf{x} + \left(\frac{\partial \mathbf{g}}{\partial \mathbf{x}'}^{-1} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \right)^\mathsf{T} V'_\mathbf{x}, \tag{31}$$

and similarly for the rest of the derivatives. We summarize the algorithms for Newton's method and DDP in the presence of implicit integration in Algorithm 1 and 2.

V. Analysis

A. Complexity

In this section we discuss the complexities of Newton's method and DDP, and the strengths and weaknesses of each method. We validate the analysis experimentally on a set of problems where we vary both the length of the planning horizon, and the dimensionality of the dynamical systems that are manipulated. This was discussed to some extent in [27], in the context of asymptotic complexity, and our finding generally supports their conclusion.

Algorithm 2: Trajectory optimization: DDP

Input: Dynamical system, initial $\mathbf{u}$, initial $\mathbf{x}_0, \dot{\mathbf{x}}_0$
Output: Optimal control trajectory $\mathbf{u}$
while *criterion not reached* **do**
  Compute $\mathbf{x}(\mathbf{u})$ using forward simulation
  `/* Backward pass */`
  $V_n \leftarrow \ell_f(\mathbf{x}_n)$
  for $i \leftarrow n-1$ *to* 1 **do**
    Compute derivatives of $\mathcal{Q}_i$ and V_i ((31),[22])
    Compute K_i and k_i (Eq. (22))
  `/* Forward pass */`
  $\alpha \leftarrow 1$
  while $\mathcal{O}$ *is not reduced* **do**
    for $i \leftarrow 0$ *to* $n-1$ **do**
      Compute $\bar{\mathbf{u}}_i$ (Eq. (32))
      Simulate $\mathbf{x}_{i+1}$.
    Evaluate $\mathcal{O}$
    $\alpha \leftarrow \alpha/2$

Recall that n_t is the number of time steps, n_c is the number of control variables, and let n_s be the number of state variables. Starting with the complexity of Newton's method, we note that it is dominated by the computation of the sensitivity matrix $\mathbf{S}$ in (9) and the solution to the system in (12). The Hessian in (12) signifies the contribution of each control variable in each time step, and is generally dense. Its dimensions are $n_t n_c \times n_t n_c$, and therefore, due to its dense structure, the time for solving (12) scales like $n_t^3 n_c^3$. As noted in [27], instead of the condensation approach the problem can be solved by a Riccati based factorization, which is equivalent to DDP, and which we discuss next.

The computational overhead of DDP is dominated by the terms k and K, which must be evaluated in each step of the backward pass. This involves the solution of a system including $\mathcal{Q}_{\mathbf{uu}}$, which has dimensions $n_c \times n_c$. To compute $\mathcal{Q}_{\mathbf{uu}}$, and the other derivatives, we require $\mathbf{f}_\mathbf{x}$, and $\mathbf{f}_\mathbf{u}$, which in turn, requires us to solve a square linear system with n_s equations (eq. (30)). Consequently, each DDP iteration scales cubically with n_s and n_c and linearly with n_t.

To conclude, the major difference is that Newton's method scales cubically with the number of time steps, while DDP scales linearly. This is also supported by the experiments shown in Fig. 3. However, there is a caveat, as for short trajectories, Newton seems to outperform DDP. The reason can be traced back to the computation of $\mathbf{f}_\mathbf{x}$ (30). It also scales linearly with the number of columns of $\frac{\partial \mathbf{g}}{\partial \mathbf{x}}$, which equals to the size of the state times the number of previous time steps.

B. Regularization and line-search

Since the problem is not convex, both Newton and DDP require a modification to ensure that each step produces a descent direction. A Newton step is guaranteed to result in a descent direction if the Hessian is positive definite. If an ascent direction is generated instead, then the Hessian must be transformed into a positive definite matrix. One strategy to achieve that is to add a regularization term in the form of a scaled identity λI to the Hessian. An equivalent result exists

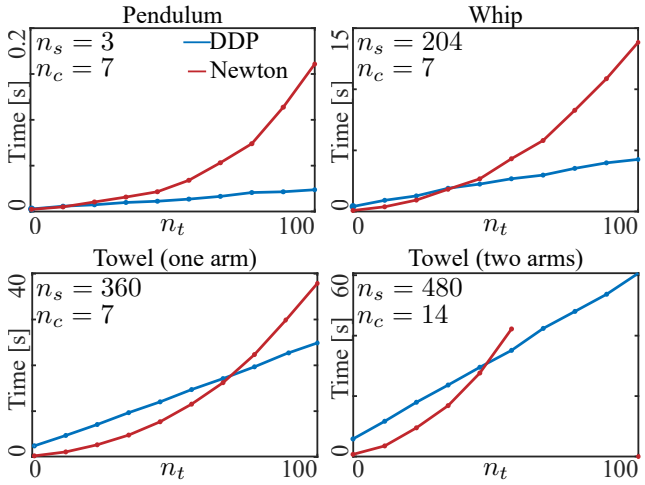

Fig. 3: Timing plot showing the average time per iteration for different experiments and different trajectory lengths (n_t). Newton's method seems to outperform DDP for shorter trajectories. The trajectory length at which both methods achieve similar performance depends on the number of state (n_s) and control (n_c) variables. DDP performs better with increasing control size, and worse with increasing state size. Note that in the dual-armed towel example (bottom right), Newton's methods can fail due to lack of memory.

for DDP: in this case, $\mathcal{Q}_{\mathbf{uu}}$ is required to be positive definite for each time step [18]. Likewise, when not the case, adding λI to $\mathcal{Q}_{\mathbf{uu}}$ should resolve the problem. In our experiments, we noted that DDP seems to be very sensitive to the choice of λ. We show the convergence plot for a typical example with different λ's in Fig. 5. In this case, best results were achieved for $\lambda \in [1e^{-3}, 1e^{-2}]$. Adaptive techniques that can find λ exist (e.g. [19]), but we instead found them empirically.

We also note that in addition to regularization, both Newton's method and DDP required a line-search procedure. We follow the simple backtracking strategy. For Newton's method, the next control candidate is given by $\bar{\mathbf{u}} = \mathbf{u} + \alpha \mathbf{d}$, where α is the line-search parameter that starts at 1, and is iteratively halved. The evaluation of $\mathcal{O}(\mathbf{x}, \bar{\mathbf{u}})$ in each line search iteration must be preceded by a forward simulation using $\bar{\mathbf{u}}$ in order to update $\mathbf{x}$ according to the dynamics. Standard DDP uses the following line-search iteration in the forward-pass

$$\bar{\mathbf{u}}_i = \mathbf{u}_i + \alpha k_i + K_i(\bar{\mathbf{x}}_i - \mathbf{x}_i), \tag{32}$$

similarly initializing α with 1.

VI. Results

We verify the efficacy of our implementation by experimenting with several tasks for the YuMi to carry out. On one hand we use these different scenarios to compare the performance of the batch Newton method with extended DDP. On the other hand we investigate the *dynamic* manipulation of deformable objects using robots. The implementation was done in C++ using Eigen [28], and all the experiments run on a computer with an Intel Core i7-7709K 4.2Ghz. We present results in simulation, and additionally execute some of the tasks on the physical platform to study to what extent they carry over to the real world. These experiments are presented in length in the accompanying video. Details regarding parameters used for the individual demonstrations appear in table I. The mass density was determined by measuring total mass and volume of the foam piece. The corresponding shear and bulk modulus were found empirically by matching the pose of the simulated mesh with its physical counterpart. The initial configuration of the robot was chosen such that all joints are reasonably far away from its physical limits. It is considered by the algorithms as a fixed initial condition. Therefore, choosing a different initial configurations would result in different optimal trajectories. The objective for all experiments include quadratic penalties that match a specific mesh node to a specified target in world coordinates at a given moment in time. To induce smooth control trajectories, we additionally add a regularizer that penalizes high joint accelerations. These are approximated using finite difference.

Whipping. One set of tasks that we experimented with was *whipping*. The goal is to find a trajectory such that the tip of the "whip" hits a predefined target with maximum speed. This setup generates fairly dynamic motions. We began by running preliminary experiments with a pendulum, modeled as a point mass connected to the robot's gripper by a unilateral cable. We demonstrate this small-scale *wrecking ball* example with one, two, and four targets (in form of wooden blocks). The setup of these experiments is shown in Fig. 4. Next, we increased the problem's complexity by manipulating a soft rod modeled with FEM instead. Again, the objective was to hit one or more targets with the tip of foam stick with maximum velocity. We present cases where the robot manages to whip one and two targets, respectively (see Fig. 1).

Laying a cloth on the dining table or a sheet on the bed is a task that typically requires dynamic manipulation. We use our framework to explore the robot's capabilities to perform this task. Fig. 6 shows strategies to lay the lower part of a strip of foam onto a table using one arm, as illustrated in. This task was formulated as an objective that measures the distance of the nodes of one end of the mesh with predefined positions on the table. Fig. 7 shows a similar example, but where the robot uses *both* arms to lay a larger piece of foam. We present a side-by-side comparison of the physical and simulated experiment in . In a third case it uses the same model to flip the soft body around (see accompanying video).

VII. Discussion and Future Work

We presented two trajectory optimization techniques based on Newton's method and DDP for robotic manipulation of deformable objects. Both methods rely on a forward simulation model of soft objects, which we developed based on the Finite Element Method and implicit integration. In this setting, we showed how to leverage sensitivity analysis to analytically compute all the derivatives that are required for the two trajectory optimization methods.

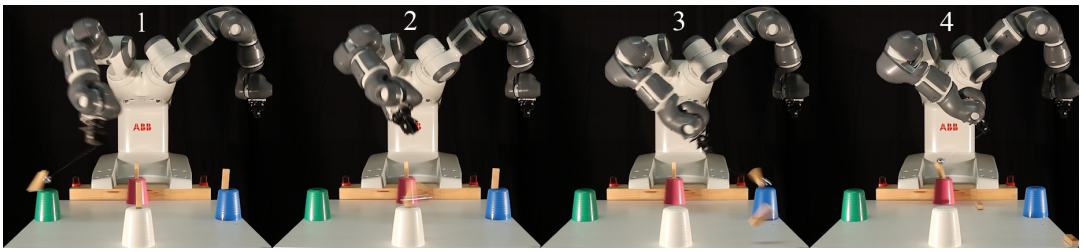

Fig. 4: Robot using a pendulum to hit four targets in a single dynamic motion.

TABLE I: Parameters that were used for the different experiments.

System and Objective	# Nodes	# Arms	# Steps	Overall time	Mass (Density)	Shear modulus	Bulk modulus
Pendulum: Hit 1 / 2 / 4 target(s)	1	1	60 / 60 / 100	0.667 / 1.5 / 1.667 s	0.045kg	–	–
Soft rod: Whip 1 / 2 target(s)	68	1	40	1.0 / 0.889 s	34.722 $\frac{kg}{m^3}$	21149 $\frac{kg}{ms^2}$	98696 $\frac{N}{m^2}$
Thin cloth: Toss onto table	120	1	40	0.8s	50.71 $\frac{kg}{m^3}$	35714 $\frac{kg}{ms^2}$	166667 $\frac{N}{m^2}$
Cloth: Toss / Flip onto table	160	2	40	1.0s	50.588 $\frac{kg}{m^3}$	35714 $\frac{kg}{ms^2}$	166667 $\frac{N}{m^2}$

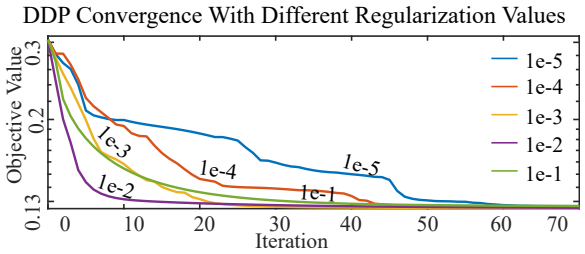

Fig. 5: Convergence plot showing the value of the total control objective per iteration for different regularizer values for DDP.

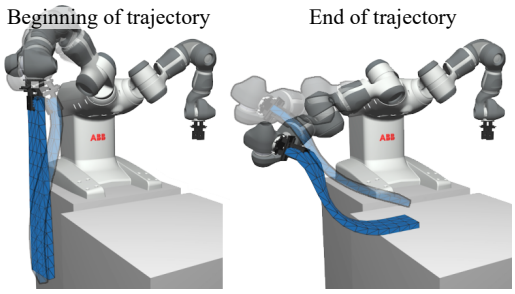

Fig. 6: Robot tossing a thin piece of foam onto the table using one arm. Several poses of the robot and the mesh are overlaid.

DDP vs Newton. With appropriate regularizers and line search mechanisms, both DDP and Newton's method converge quite reliably. Our analysis and experiments show that DDP scales better than Newton's method in terms of the length of the planning horizon. We have also observed that the overall computational cost heavily depends on the dimensionality of the object's state space and likely its sparsity structure. Overall, the relative performance seem to be highly dependent on the nature of the problem (Fig. 8). Nevertheless, we see opportunities when it comes to improving the performance of both methods. For example, the matrices used for sensitivity analysis exhibit a sparsity structure that could be exploited by specialized solvers that might not be equivalent to either approach. In the future, we plan to develop such specialized solvers, as well as adaptive regularization techniques that have the potential to drastically improve convergence rates.

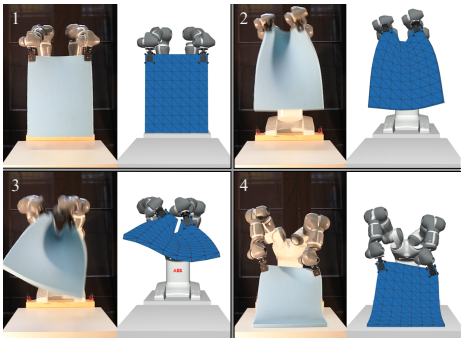

Fig. 7: Robot laying a piece of foam on a table. Four time instances are shown, comparing the physical and simulated motions on the left and right side, respectively. Note that although the motion is anticipated correctly, there is a mismatch between the simulation and the physical result in frame 3.

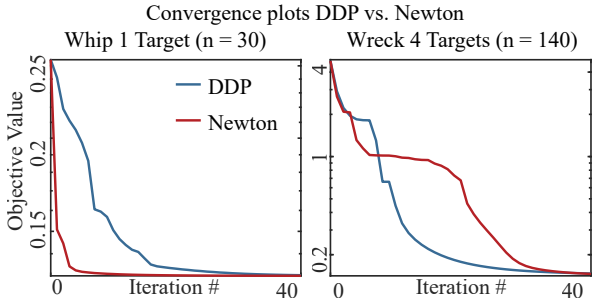

Fig. 8: Convergence plot comparing Newton's method and DDP. The relative performance of the two algorithms highly depends on the nature of the problem.

Direct collocation. An alternative TO approach is the direct collocation method, which solves (7) using constrained optimization. Direct collocation, using e.g. the interior point method, allows the constraints to be violated during optimization, where in some cases, this could speed up the optimization process. This largely depends on the heuristics employed by the solver. The advantage of single shooting is that every iteration is feasible, and is better than the previous iteration. Whether or not this is a desirable feature depends on the application. We implemented direct collocation using *Ipopt* [29], an open sources interior point

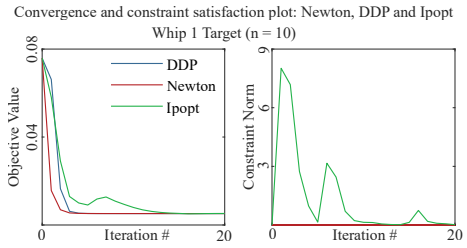

Fig. 9: Convergence and constraint violation plot comparing Newton's method, DDP and Ipopt. While the dynamics constraint is satisfied at each iteration for Newton and DDP, Ipopt only generates a feasible solution in the end.

optimizer. We tested several Ipopt settings, and found that using the Gauss-Newton approximation works better. Fig. 9 shows a comparisons of the three methods for a smaller scale version of the *whipping* experiment, in terms of convergence and constraint satisfaction. While all methods converge to the same solution, Ipopt requires more iterations. We also note that, as expected, the objective is not monotonically decreasing, and the dynamics are only satisfied at the very end of the optimization procedure. We note that our code has not been optimized. However, we made our best effort to run a fair comparison, which indicated that the time per iteration of Ipopt is similar.

Sim-to-real. To investigate the extent to which the results of our trajectory optimization methods carry over to the real world, we executed several experiments involving robotic manipulation of physical objects. We calibrated the real and simulated cameras so we can easily visualize the differences (Fig. 7). For a simple dynamical systems like a pendulum (Fig. 4), the simulated motions predict the behavior of the physical system well. For the foam we experimented with, the calibration of the underlying simulation models becomes more challenging. Fig. 7 shows mismatches between the simulation and the real world: the simulation model predicts that the lower end of the soft sheet will lift up higher than it does in reality. While parameter identification suffices in the quasi-static case, dynamical motions remains a challenge due to internal energy dissipation. Recent methods track and analyze the motion of physical prototypes in order to accurately reconstruct their viscoelastic material parameters [30] promise to alleviate such problems. Another interesting future avenue to cope with these problems is to include feedback control into the current feed-forward process. Combining a measurement technique for the mesh with either a model-based or a more traditional feedback loop applied in real-time is a challenge that deserves it own investigation.

References

[1] C. Duriez, "Control of elastic soft robots based on real-time finite element method," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[2] J. M. Bern, P. Banzet, R. Poranne, and S. Coros, "Trajectory optimization for cable-driven soft robot locomotion," in *Robotics: Science and Systems XV, University of Freiburg, Germany*, 2019.

[3] Y. Li, Y. Wang, Y. Yue, D. Xu, M. Case, S. F. Chang, E. Grinspun, and P. K. Allen, "Model-driven feedforward prediction for manipulation of deformable objects," *IEEE Trans. on Automation Science and Engineering*, 2018.

[4] T. Bretl and Z. McCarthy, "Quasi-static manipulation of a kirchhoff elastic rod based on a geometric analysis of equilibrium configurations," *IJRR*, 2014.

[5] S. Duenser, J. M. Bern, R. Poranne, and S. Coros, "Interactive robotic manipulation of elastic objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018.

[6] E. Yoshida, K. Ayusawa, I. G. Ramirez-Alpizar, K. Harada, C. Duriez, and A. Kheddar, "Simulation-based optimal motion planning for deformable object," in *IEEE ARSO, Lyon, France*, 2015.

[7] Y. Bai, W. Yu, and C. K. Liu, "Dexterous manipulation of cloth," in *Computer Graphics Forum (Eurographics), 2016*, ser. EG '16, 2016, p. 523–532.

[8] P. Jiménez, "Survey on model-based manipulation planning of deformable objects," *Robotics and computer-integrated manufacturing*, 2012.

[9] D. Dinev, T. Liu, and L. Kavan, "Stabilizing integrators for real-time physics," *ACM TOG*, vol. 37, no. 1, Jan. 2018.

[10] D. Dinev, T. Liu, J. Li, B. Thomaszewski, and L. Kavan, "Fepr: fast energy projection for real-time simulation of deformable objects," *ACM TOG*, vol. 37, no. 4, pp. 1–12, 2018.

[11] M. Zheng, Z. Yuan, Q. Tong, G. Zhang, and W. Zhu, "A novel unconditionally stable explicit integration method for finite element method," *Vis. Comput.*, vol. 34, no. 5, p. 721–733, May 2018.

[12] M. Macklin, K. Storey, M. Lu, P. Terdiman, N. Chentanez, S. Jeschke, and M. Müller, "Small steps in physics simulation," in *SCA*, 2019.

[13] S. Martin, B. Thomaszewski, E. Grinspun, and M. Gross, "Example-based elastic materials," *ACM TOG*, 2011.

[14] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, 1966.

[15] D. Jacobson and D. Mayne, *Differential Dynamic Programming*, ser. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1970.

[16] J. F. A. de O. Pantoja, "Differential dynamic programming and newton's method," *Int. J. Control*, 1988.

[17] L. Liao and C. Shoemaker, "Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems," Tech. Rep., 1992.

[18] D. M. Murray and S. J. Yakowitz, "Differential dynamic programming and newton's method for discrete optimal control problems," *J. Optim. Theory Appl.*, Jul 1984.

[19] L. Liao and C. A. Shoemaker, "Convergence in unconstrained discrete-time differential dynamic programming," *IEEE Transactions on Automatic Control*, June 1991.

[20] E. Mizutani, "On pantoja's problem allegedly showing a distinction between differential dynamic programming and stagewise newton methods," *Int. J. Control*, 2015.

[21] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO*. INSTICC Press, 2004.

[22] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *IEEE Int. Conference on Robotics and Automation (ICRA)*, May 2014.

[23] Z. Xie, K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *IEEE Int. Conference on Robotics and Automation (ICRA)*, 2017.

[24] R. H. Jackson and G. P. Mccormick, "Second-order sensitivity analysis in factorable programming: Theory and applications," *Math. Program.*, May 1988.

[25] S. Zimmermann, R. Poranne, and S. Coros, "Optimal control via second order sensitivity analysis," *arXiv:1905.08534*, 2019.

[26] M. Toussaint, "Newton methods for k-order markov constrained motion problems," *CoRR*, 2014.

[27] M. Diehl, J. Ferreau, and N. Haverbeke, *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*, Mai 2009, vol. 384.

[28] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[29] 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.

[30] D. Hahn, P. Banzet, J. Bern, and S. Coros, "Real2Sim: visco-elastic parameter estimation from dynamic motion," *ACM ToG*, Nov. 2019.