2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Closed-Loop Trajectory Optimization of Deformable Linear Objects for Dynamic Motions

Marc Kilian Klankers[1] and Jochen J. Steil[1]

***Abstract*— Tracking dynamic endpoint trajectories of deformable linear objects (DLOs) with a robotic manipulator remains challenging due to their complex non-linear behavior. While closed-loop Model Predictive Control (MPC) can account for these non-linearities, it requires an accurate dynamic model and precise state estimation. This paper introduces a closed-loop approach for controlling a DLO's endpoint to track dynamic 2D shapes. We model the DLO as a floating-base kinematic chain and present a new perspective on learning its dynamics using a data-driven approximation of its hybrid dynamics. Based on this model, we formulate an Optimal Control Problem (OCP), which we solve within the control loop using both linear MPC and DDP. We validate our approach with simulation and hardware experiments, demonstrating its ability to track dynamic endpoint motions.**

I. INTRODUCTION

Deformable linear objects (DLOs) are a common class of objects, characterized by one dimension being significantly larger than the other two. They are ubiquitous across a wide range of applications, including industrial (e.g., beams, cables), medical (e.g., needles, tubes), and domestic environments (e.g., foam materials, ropes) [1]. While the category of DLOs encompasses objects with diverse behavioral characteristics, this paper specifically focuses on those possessing significant bending stiffness, such as beams, tubes, and foam materials. This is in contrast to highly flexible objects like ropes, which can adopt a vast range of configurations. Most existing research on DLO interaction assumes slow, quasi-static motion. In contrast, our work focuses on modeling and controlling dynamic motions by solving an optimal control problem (OCP) in the loop. For instance, consider a robotic manipulator swinging a foam cylinder while attempting to track the DLO's endpoint position, as demonstrated in Fig. 1. This task is particularly challenging as it demands accurate and computationally efficient state estimation, dynamic modeling, and control.

Deformable objects impose significant challenges compared to rigid objects. Whereas the state of a rigid body can be uniquely described by a 6D pose, and that of a rigid body chain by a finite set of generalized coordinates, deformable objects like DLOs lack a generic spatial configuration representation. To address this, many configuration spaces have been studied, including meshes, pseudo-rigid body models, point models, and black-box models. Similarly, a variety of methods are used to model the dynamics of deformable objects, such as the Finite Element Method (FEM), rigid body dynamics, mass-spring-damper systems, data-driven methods, and hybrid approaches. [2] provides a detailed, high-level review of these common modeling approaches.

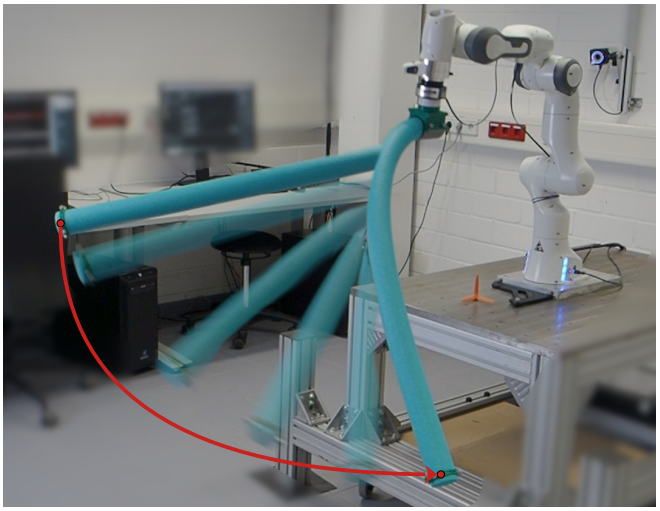

Fig. 1. Robot controlling end point of DLO object

The choice of configuration space and dynamic model significantly influences the resulting optimal control problem. For example, the authors of [3], [4] modeled the configuration of foam materials as a mesh and used the energy-based FEM. While [3] employs the quasi-static assumption, [4] solves an open-loop optimal control problem for dynamic tasks like whipping and cloth laying using Newton's method and Differential Dynamic Programming (DDP). Although FEM is a powerful modeling approach, it presents several challenges, including the sim-to-real gap, the selection of hyperparameters (such as state space dimensionality), computation times, and difficulties in state estimation [3], [4].

In this paper, we model the DLO as a floating-base rigid-body chain whose base is attached to the robot's end-effector. This approach allows the DLO's configuration to be described by the floating-base state, $\vec{\mathbf{x}}$, and the joint states of the kinematic chain, $\vec{\mathbf{q}}$. As this model is intended for closed-loop control of the DLO, a fast, accurate, and robust state estimation method is required.

The authors of [5] demonstrated that observing only the DLO's tip is sufficient for learning a data-driven inverse kinematics (IK) model that maps these observations to the joint space of the representative kinematic chain. In their work, an optical tracking system measured the DLO's tip position, and these measurements were used to train a neural-

[1]Marc Kilian Klankers and Jochen J. Steil are with the Institut für Robotik und Prozessinformatik, Technische Universität Braunschweig, 38106 Braunschweig, Germany m.klankers|j.steil@tu-braunschweig.de

network-based encoder that establishes the mapping to the joint space. Building on this concept, [6] proposes a fusion network that integrates the outputs of various data-driven IK solutions within the kinematic chain model's joint space using a linear Kalman filter. In addition to achieving fast state estimation (under 1 ms), this method uses force/torque measurements as an input to the data-driven IK. This enables the system to predict the state, maintaining estimation even when sensor data is lost, for instance, during tip occlusion. Consequently, this approach is highly suitable as it meets the requirements for fast, accurate, and robust state estimation.

Regarding control, [7] used a kinematic chain model to track the dynamic tip motions of a DLO. Although this is an efficient modeling strategy, the authors only applied open-loop control. They derived a partially input-output linearizing force-based controller using rigid body dynamics. In a subsequent work, [8] closed the control loop using an observer and developed a velocity-based controller to guide a specific point on the DLO along a reference trajectory. Similarly, [9] modeled the DLO with a floating base, represented its shape using curvature parameterization, and derived a controller from the equations of motion to track dynamic movements. A key limitation of these three works is the application of an instantaneous control law, which only allows the controller to react to current deviations and lacks a predictive planning horizon. In contrast, optimal control offers the potential to improve tracking performance because the planned control inputs can account for the long-term behavior of the DLO.

Optimal control was applied in [10] to track the tip position of a foam cylinder along fast trajectories. The authors employed a black-box modeling approach, training a LSTM network on input-output trajectories. Consequently, the internal state of the LSTM served as an abstract representation of the DLO's state. Although this approach was demonstrated to be sufficient for tracking various end-effector trajectories in 3D space, it had several limitations. First, their approach relied on the assumption that the system has a "unique globally exponentially stable equilibrium state," corresponding to the resting position from which all trajectories were initiated. This assumption restricts its applicability to other scenarios. Furthermore, they only used the pitch and yaw angles of the robot's end-effector for actuation, thereby neglecting the other available degrees of freedom for the tracking task. Another potential area for improvement is the choice of solver; the paper utilized only gradient descent for the optimal control problem, whereas more sophisticated solvers, such as DDP, are available. Moreover, because the DLO state in their model is not physically interpretable, integrating state or environmental constraints into the optimal control problem is not straightforward. This limitation further emphasizes the preferability of a physically meaningful state, such as the generalized coordinates of a floating-base kinematic chain, in the context of optimal control (OC).

This raises the question of how to model the dynamics of a DLO's within a kinematic chain representation. In [5], [6], the dynamics were modeled using a data-driven approach, learning a discrete mapping from the current to the subsequent joint state using RNN-based architectures. [5] demonstrates that RNN-based structures are superior to purely model-based approaches that utilize rigid body dynamics with spring-damping joints. Furthermore, [11] enhanced this concept by introducing hybrid dynamics combined with a data-driven network.

Hybrid dynamics is a combination of forward and inverse dynamics, where the accelerations of some joints are known, and the generalized forces for the remaining joints [12]. A typical scenario for floating-base rigid body chains involves a known base acceleration—which, in this work, is the variable to be controlled—while the acting spring-damping joint torques can be calculated from the current state. This hybrid approach enables the calculation of the necessary floating-base wrench and the resulting joint accelerations, allowing for forward predictions over time. The authors of [11] showed that using hybrid dynamics with spring-damping joints, in combination with a small data-driven network to predict joint torques, is sufficient to accurately learn the system dynamics from a single trajectory. However, a notable gap in the works of [5], [6], [11] is the application of these learned dynamics models to control problems.

This paper presents a novel perspective on learning a data-driven dynamics model for DLOs. We train a neural network to simultaneously predict the joint accelerations and the corresponding base wrenches of a kinematic chain model representing the DLO. While predicting joint accelerations is not new, the key novelty of our work is the concurrent prediction of the acting base wrench. This dual prediction offers two main advantages. First, the predicted accelerations enable the forward prediction of the DLO's joint motion using an integrator, potentially with different time steps (Δt) than those used during training. Second, the predicted base wrench is valuable for low-level robot control, as it can be used in a feed-forward manner to compensate for the forces and torques exerted by the DLO.

In summary, this paper focuses on tracking dynamic end-point trajectories of a DLO using closed-loop trajectory optimization. We model the DLO as a floating-base kinematic chain, which provides a physically interpretable state and allows for the introduction of constraints into the optimization problem. The system's dynamics are learned using a data-driven approximation of the hybrid dynamics, which enables fast evaluation. Consequently, this paper presents advancements in:

- A closed-loop trajectory optimization framework for dynamic DLO end-point manipulation.
- A data-driven approximation of hybrid dynamics that simultaneously predicts joint accelerations for motion forecasting and base wrenches for feed-forward control.

Section II provides relevant background information. We then describe our methodology in Section III and detail the experimental setting and implementation in Section IV. The results of our work are presented in Section V, followed by our concluding remarks.

II. BACKGROUND

This work models the DLO as a floating-base kinematic chain. The chain's floating base, denoted by $\mathbf{x}$, is rigidly grasped by a robot. As illustrated in Fig. 2, the kinematic chain consists of N rigid bodies connected by $N-1$ 2D rotational joints. Each joint provides rotational degrees of freedom around the y- and z-axes, with the x-axis oriented along the link's direction. Consequently, the DLO's spatial configuration can be described by $6+2(N-1)$ generalized coordinates. The state of the DLO is therefore represented by the base pose $\mathbf{x}$, its spatial velocity $\dot{\mathbf{x}}$, and the joint states $\vec{\mathbf{q}}$. Unless otherwise specified, the state of any quantity is defined by its position and velocity, denoted as $\vec{\mathbf{a}} = \begin{bmatrix}\mathbf{a} & \dot{\mathbf{a}}\end{bmatrix}^T$. We assume that the pose $\mathbf{x}$ and spatial velocity $\dot{\mathbf{x}}$ of the kinematic chain's base frame can be reliably measured throughout this work. This can be achieved, for instance, with a tracking system or calculated via forward kinematics since the base is rigidly grasped by the robot. To estimate the joint states $\vec{\mathbf{q}}$, we employ the latent space fusion model introduced in [6]. This model utilizes multiple data-driven inverse kinematic networks to estimate the joint states $\vec{\mathbf{q}}_1, \vec{\mathbf{q}}_2, \ldots, \vec{\mathbf{q}}_N$. These estimations are based on partial observations of the DLO, using various input combinations of the base state $\vec{\mathbf{x}}$, the tip state $\vec{\mathbf{t}}$, and the wrench acting at the base $\mathbf{F}_b$. The resulting joint state predictions serve as observations for a linear Kalman filter, which in turn computes a final joint state estimate, $\vec{\tilde{\mathbf{q}}}$, as depicted in Fig. 2. A key benefit of this approach is its robustness; the Kalman filter can provide an estimate even when some state information is unavailable, enhancing the system's reliability. The tip pose $\mathbf{t}$ and velocity $\dot{\mathbf{t}}$ can then be calculated using forward kinematics. Further details regarding the fusion network and the data-driven inverse kinematics are available in [6], [5].

III. METHODOLOGY

Based on the kinematic chain model described in Sec. II, the objective is to formulate a dynamical system suitable for optimal control. The system is modeled as a second order system and its state is defined by $\vec{\mathbf{s}} = [\vec{\mathbf{x}},\ \dot{\vec{\mathbf{q}}}]^T = [\mathbf{x},\ \dot{\mathbf{x}},\ \mathbf{q},\ \dot{\mathbf{q}}]^T$. Here $\mathbf{x}$ describes the pose of the floating base, $\dot{\mathbf{x}}$ its velocity, $\mathbf{q}$ the joint position and $\dot{\mathbf{q}}$ the joint velocity.

A. Data Driven Approximation of Hybrid Dynamics

Hybrid dynamics distinguishes between two joint types. Inverse-dynamics joints, where the accelerations are prescribed and the corresponding generalized forces must be computed and forward-dynamics joints, where the generalized forces are given, and the resulting accelerations are determined. In our use case, we choose the floating-base coordinates as inverse-dynamics joints. Their accelerations are known, and we compute the wrench at the base. The DLO's own joints are treated as forward-dynamics joints. Their applied torques are known, and we solve for the joint accelerations. Hence, the hybrid-dynamics equations for our system can be written as:

$$\begin{bmatrix}\mathbf{F}_{base}\\ \ddot{\mathbf{q}}_{joints}\end{bmatrix} = \mathbf{f}_{HD}\left(\begin{bmatrix}\vec{\mathbf{x}}\\ \vec{\mathbf{q}}\end{bmatrix}, \begin{bmatrix}\ddot{\mathbf{x}}_{base}\\ \tau_{joints}\end{bmatrix}\right) \tag{1}$$

For a more detailed description, we refer to [12].

We apply a data-driven approach to the hybrid dynamics, approximating the right-hand side of the equation with a neural network. For classical hybrid dynamics, we would need to provide the acting torques in the DLO's joints, which could be calculated from the state, for instance, by assuming spring-damper models. However, since we use a fully black-box approach, these joint torques are not available, and the network approximation must learn them from the provided data. This leads to the following structure for the data-driven approximation of the hybrid dynamics:

$$\begin{bmatrix}\mathbf{F}_{base}\\ \ddot{\mathbf{q}}_{joints}\end{bmatrix} = \mathbf{NN}\left(\begin{bmatrix}\vec{\mathbf{x}}\\ \vec{\mathbf{q}}\\ \ddot{\mathbf{x}}_{base}\end{bmatrix}\right), \tag{2}$$

where $[\vec{\mathbf{x}},\ \vec{\mathbf{q}},\ \ddot{\mathbf{q}}_{base}]$ is the input vector to the neural network. This data driven approximation enables forward simulation in time of the floating base kinematic chain.

$$\dot{\vec{\mathbf{s}}} = \mathbf{f}(\vec{\mathbf{s}}, \vec{\mathbf{u}}) \tag{3}$$
$$\begin{bmatrix}\dot{\mathbf{x}}\\ \ddot{\mathbf{x}}\\ \dot{\mathbf{q}}\\ \ddot{\mathbf{q}}\end{bmatrix} = \begin{bmatrix}\dot{\mathbf{x}}\\ \vec{\mathbf{u}}\\ \dot{\mathbf{q}}\\ \mathbf{NN}(\vec{\mathbf{x}},\ \vec{\mathbf{q}},\ \vec{\mathbf{u}})\end{bmatrix}$$

Here, the control input $\vec{\mathbf{u}}$ is the floating base acceleration $\ddot{\mathbf{x}}_{base}$. The forward roll-out in time is performed by a suitable integrator, starting from an initial condition $\vec{s}_0$ with a given sequence of DLO base accelerations $\ddot{\mathbf{x}}_{t:t+N-1}$.

The neural network for the hybrid dynamics is trained on collected DLO trajectory data. We also obtain the kinematic chain's joint space trajectories by applying the network from Sec. II, either during data collection or post-processing. The network learning is thus achieved by minimizing the following loss function for a single trajectory:

$$\begin{aligned}\mathscr{L} = &\frac{w_{q,all}}{2T}\sum_{t=1}^{T}\mathrm{WMSE}(\vec{\mathbf{q}}_{\mathrm{ref},t}, \vec{\hat{\mathbf{q}}}_t, w_q) + \\ &\frac{w_{\dot{q},all}}{2T}\sum_{t=1}^{T}\mathrm{WMSE}(\vec{\dot{\mathbf{q}}}_{\mathrm{ref},t}, \vec{\hat{\dot{\mathbf{q}}}}_t, w_{\dot{q}})\end{aligned} \tag{4}$$

The loss is a weighted mean square error (WMSE) calculated over a prediction horizon T. At each time step t in the horizon, we compute the WMSE between the predicted joint positions $\vec{\hat{\mathbf{q}}}_t$ and velocities $\vec{\hat{\dot{\mathbf{q}}}}_t$ and their target values $\vec{\mathbf{q}}_{\mathrm{ref},t}$ and $\vec{\dot{\mathbf{q}}}_{\mathrm{ref},t}$ using the weights w_q and $w_{\dot{q}}$. Subsequently, the mean position and velocity errors across the horizon T are weighted by $w_{q,all}$ and $w_{\dot{q},all}$

Our learning approach differs from that in [5]. Any data-driven IK must resolve kinematic redundancy by learning a consistent joint configuration. In [5], the IK and dynamics were learned jointly, with the resulting shape influenced by regularization on the predicted joint states. In contrast, we adopt a "separation of concerns" strategy, shifting the redundancy resolution problem entirely to the data-driven IK via joint position regularization. Therefore, the dynamics network is freed from the task of handling this redundancy.

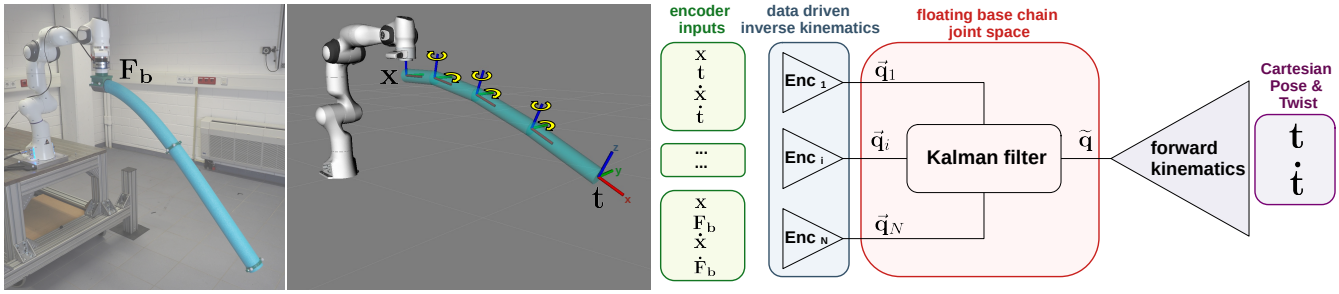

Fig. 2. Overview of floating base kinematic chain modeling and DLO state estimation via data driven inverse kinematics and Kalman filtering.

B. Optimal Control Problem

The state vector, $\vec{\mathbf{s}} = [\vec{\mathbf{x}},\ \vec{\mathbf{q}}]^T$, describes the DLO's configuration. To facilitate tracking, we augment this state vector with the DLO's tip position, $\mathbf{t}$, yielding the new state $\vec{\mathbf{s}}_t = [\vec{\mathbf{x}},\ \vec{\mathbf{q}},\ \mathbf{t}]^T$. This allows the tip position, which is computed via forward kinematics within the integration loop, to be directly included in the cost function. The trajectory optimization is formulated as the following optimal control problem with a standard quadratic cost function:

$$\vec{\mathbf{u}}^*_{t:t+N-1} = \underset{\vec{\mathbf{u}}_{t:t+N-1}}{\operatorname{argmin}} \ \frac{1}{2}\left(\vec{\mathbf{e}}_N^T \mathbf{T} \vec{\mathbf{e}}_N\right) + \frac{1}{2}\sum_{i=1}^{N-1}\left(\vec{\mathbf{e}}_i^T \mathbf{S} \vec{\mathbf{e}}_i + \vec{\mathbf{u}}_i^T \mathbf{R} \vec{\mathbf{u}}_i\right)$$

$$\textit{such that}: \quad \mathbf{H}(\vec{\mathbf{s}}_{t,j}) = 0, \qquad \mathbf{G}(\vec{\mathbf{s}}_{t,j}) \geq 0 \tag{5}$$

$$\textit{with}: \quad \vec{\mathbf{e}}_j = \vec{\mathbf{s}}_{t,j} - \vec{\mathbf{s}}_{t,\text{goal},j} \quad \forall j = 1,\ldots,N$$

Here $\vec{\mathbf{s}}_t$ is the augmented state and $\vec{\mathbf{s}}_{t,\text{goal},t}$ is the desired state at time index t. The control input is denoted by $\mathbf{u}_i$, in our case the floating base acceleration while $\mathbf{S}$ and $\mathbf{R}$ are weighting matrices and $\mathbf{T}$ is the conventional terminal weight for the final state $\vec{\mathbf{s}}_{t,N}$. The functions $\mathbf{H}()$ and $\mathbf{G}()$ introduce constraints to the optimization problem. This augmented state formulation conveniently allows for the inclusion of regularization terms for the floating base or joint states directly into the cost function or as additional constraints.

C. Control Architecture

We propose the control architecture for the robotic system depicted in Fig. 3. A *high-level* controller solves the trajectory optimization, generating reference trajectories for a *low-level* controller. The low-level control loop, which directly controls the robot, operates at a much higher frequency than the high-level optimization.

In each control cycle, the high-level controller receives the current system state and re-solves the optimization problem. The solver's significant computation time causes a delay, meaning the system state has evolved by the time a new trajectory is available—a common issue in receding horizon control. To prevent discontinuous jumps in the commanded trajectory, the high level controller sends the planned trajectory to a *trajectory buffer* which smoothly blends newly received trajectories with the active one [13]. Finally, the trajectory buffer sends the current commands to the low-level

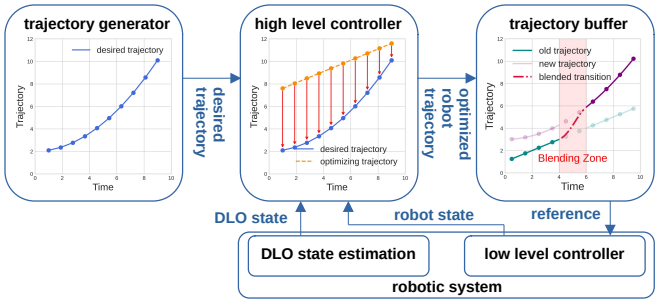

Fig. 3. Control architecture of the robotic system.

controller. While this method is simple and effective, more advanced approaches exist. For instance, in future works the solver could predict the time delay and perform a state prediction using the last optimized control input and perform a time-ahead optimization from the predicted current state as done in [14].

IV. Experimental Setting and Implementation

A. Experimental Setting

A Franka 7-DoF robotic arm [15] serves as the robotic platform for both simulation and hardware experiments.

Simulation: In the simulation, we model the DLO as a kinematic chain composed of six segments with lengths of 0.1 m, 0.1 m, 0.1 m, 0.2 m, 0.2 m and 0.3 m. This chain includes $2 \cdot 5$ spring-damper joints and has a total mass of 0.6 kg. The spring-damper parameters were heuristically tuned to achieve a suitable deformation behavior. We use Gazebo as the simulation tool.

Hardware: We use a 1.05 m long polyethylene foam cylinder of radius 3.15 cm as DLO in our hardware experiments. Reflective markers were attached to the DLO to visually track the location of the base and tip position of the DLO using the *Optitrack* tracking system. Further, we use the *JR3 85M35A-140-D 200N12* force-torque sensor to measure the wrench between the robot and the DLO base.

Data Collection: The robot performs 30 randomly sampled sinusoidal joint trajectories consisting of an initial wait time of 2 s, a moving period of 15 s and a resting phase of 7 s. We implemented a ramping up and down phase of 4 s for each trajectory to smoothly start and end the motion. To generate dynamic motions, the amplitude and frequency of the joint

trajectories are sampled from a heuristically defined range of values. We collect one dataset for the simulation and one for the real world experiments.

Data preparation: Before further processing, all measured poses are transformed into a common frame of reference, which we choose as the robot base frame. The derivatives of quantities, which cannot be measured directly, were calculated using central differences and smoothed with a rolling average with adjusted window.

B. Implementation

The neural networks were trained in Python using JAX [16] and the FLAX library [17]. For production deployment, we run the networks in C++ by translating the JAX models to ONNX format with the jax2onnx library [18]. A key design decision for the data-driven IK and fusion network (Sec. II) is how many segments to use when modeling the DLO as a kinematic chain. In this work we use 4 segments, giving a joint dimension of $2\ (4-1) = 6$. This choice follows the guideline in [11], which recommends 3–4 segments for stiffer objects and 5–7 for softer ones. Since our DLO is relatively short but rather soft, we choose 4 segments.

For the closed-loop optimal control problem in (5), we implemented several solvers. First, two variations of gradient-descent controller that computes the loss gradient with respect to the control inputs and updates them via the ADAM optimizer. One version performs numerical differentiation using simultaneous perturbation stochastic approximation (SPSA) [20] and the other automatic differentiation using CppAD [19]. We also implemented a linearizing MPC controller (*Linear MPC*), which linearizes the nonlinear dynamics (3) about the current state and solves the resulting QP using qpOASES [21]. Lastly, we developed a DDP solver with numerical differentiation, employing the iLQR variant rather than the full-Newton method. The DDP solver terminates after at most 20 iterations or when the maximum gradient norm falls below 0.0001. All solvers are implemented in C++. For the trajectory buffer, we use a path blending duration of t = 0.25 s for the DDP solver and t = 0.02 s for the Linear MPC controller.

The robot's low level controller is implemented as a joint space torque controller. Therefore, we modify the optimal control problem described in Sec. III-B. We consider the joint acceleration of the robot $\ddot{\mathbf{q}}_R$ as control input and replace the floating base state $\vec{\mathbf{x}}$ by the joint state of the robot $\vec{\mathbf{q}}_R$, resulting in the new state vector $\vec{\mathbf{s}}_{R,t} = [\vec{\mathbf{q}}_R,\ \dot{\vec{\mathbf{q}}},\ \mathbf{t}]^T$. Under this formulation, the dynamical system now iterates the robot's joint state directly. The dynamical system now represents the change in the joint state of the robot manipulator and the floating base trajectory can be calculated via forward kinematics such that the dynamical system internally calculates the values of $\vec{\mathbf{x}}$ and $\ddot{\mathbf{x}}$. This is just a low-level control design decision, a Cartesian controller could be implemented as well or inverse kinematics could be applied. The benefit is that regularization and constraints can be introduced at the joint-space level. On the other hand, this requires additional computational effort.

In our control experiments, we implement no constraints in the optimal control problem (5). Since, our state vector has dimension $\vec{\mathbf{s}}_{R,t} \in \mathbb{R}^{29}$, the 29×29 weighting matrices $\mathbf{T}$ and $\mathbf{S}$ are constructed the following way. For the tracking of the end point position, the lower right 3×3 part of the $\mathbf{T}$ and $\mathbf{S}$ matrices are set to a diagonal matrix containing the weights to track the tip position. We also applied joint space regularization, such that the robot does not drift to far away from its initial joint position. We set the 7×7 upper left part of $\mathbf{T}$ and $\mathbf{S}$ matrices with a regularization weight. All the other entities of the $\mathbf{T}$ and $\mathbf{S}$ matrices are set to zero. The cost matrices were initialized via Bryson's rule and empirically tuned to balance tracking accuracy with control effort. We utilized consistent weights for $\mathbf{S}$ and $\mathbf{T}$ to prioritize tracking and penalize deviations from a nominal joint configuration. The input cost $\mathbf{R}$ was tuned to mitigate actuator saturation and ensure smooth trajectories.

V. Results

The proposed approach is evaluated in three steps. First, we assess the prediction capabilities of the data-driven approximation of the hybrid dynamics. Second, we test and compare the different controllers in simulation. Third, we validate their performance through experiments on hardware.

A. Data Driven Approximation of Hybrid Dynamics

For the following evaluation, we use the dataset collected on the real robotic system. As a reference, we first trained two discrete-map dynamic models following [6], one based on an LSTM network [22] and one on a GRU network [23]. In each case, we initialize the cell's hidden state with the current DLO joint state $\vec{\mathbf{q}}$, and then roll it out by predicting the next state. Next, we trained our data-driven hybrid-dynamics (DDHD) models using three different architectures: LSTM, GRU, and a feed-forward MLP. For the LSTM and GRU versions, we adopt the same hidden-state initialization as in the discrete maps. We then append a linear output layer that maps from the hidden state (i.e., the current joint state) to the joint accelerations, $\ddot{\mathbf{q}}$. An explicit Euler integration step converts those accelerations into the next joint state, which becomes the new hidden state for the subsequent time step. The MLP model comprises two hidden layers of 128 neurons each, using tanh activations, and directly outputs the joint accelerations. All networks are trained to predict N = 100 steps into the future using Δt = 0.01 s resulting in a 1 s look ahead window. Finally, we trained each DDHD architecture in two variants: one that also predicts the base wrench and one that does not. The complete evaluation results are given in Table I.

Our results show that the DDHD networks outperform the discrete-mapping models in prediction accuracy. Among the DDHD architectures, the MLP yields the best performance, surpassing both RNN-based (LSTM and GRU) models. These networks also successfully learn the base wrenches acting on the DLO, as illustrated for an example trajectory in Fig. 6. Note that the F/T sensor was zeroed before executing this trajectory. Incorporating wrench prediction into the

TABLE I

EVALUATION DATA DRIVEN HYBRID DYNAMICS

dynamics network	tip position error mean ± std (max.) cm	tip velocity error mean ± std (max.) cm/s	force error mean ± std (max.) N	torque error mean ± std (max.) N
LSTM (discrete map)	4.77 ± 4.2 (30.6)	31.2 ± 36.7 (302)	—	—
GRU (discrete map)	5.13 ± 4.2 (29.6)	33.0 ± 38.3 (269)	—	—
DDHY LSTM without wrench	3.80 ± 4.8 (43.9)	29.4 ± 43.1 (397)	—	—
DDHY LSTM	4.45 ± 4.9 (35.2)	33.2 ± 40.8 (299)	0.6 ± 0.5 (4.0)	1.2 ± 1.1 (10.3)
DDHY GRU without wrench	3.50 ± 3.0 (19.9)	26.2 ± 27.4 (171)	—	—
DDHY GRU	4.06 ± 4.1 (33.8)	32.2 ± 38.2 (300)	0.5 ± 0.5 (8.1)	0.8 ± 0.8 (8.6)
DDHY MLP without wrench	2.54 ± 2.2 (19.1)	19.5 ± 21.98 (162)	—	—
DDHY MLP	2.59 ± 3.0 (26.8)	20.9 ± 27.8 (237)	0.3 ± 0.2 (2.5)	0.6 ± 0.5 (5.3)

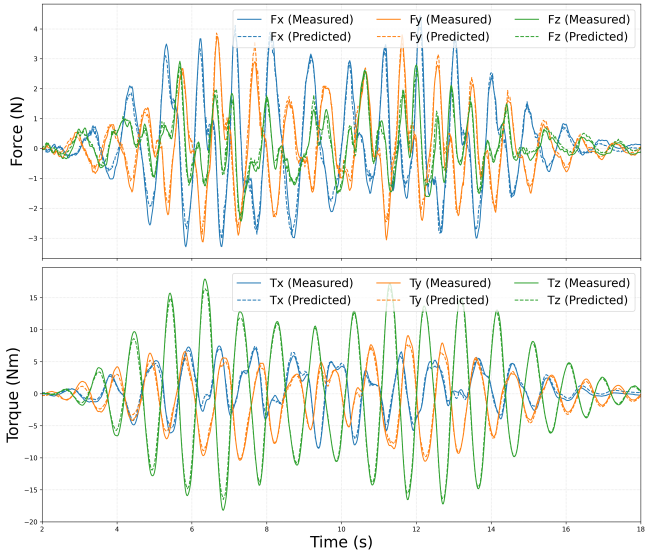

Fig. 4. Prediction of floating base wrench for one recorded trajectory using learned approximation of hybrid dynamics.

learning task introduces a slight increase in joint-trajectory prediction error. Since our control application does not use the predicted wrench, we employ only the DDHD networks without wrench prediction in the following experiments.

B. Tracking in simulation

In our tracking experiments, we track various 2D geometric shapes in 3D space. The shapes include a circle of radius 0.25 m, a rectangle of width 0.8 m and height 0.3 m, and a Lissajous figure in the y–z plane of the robotic base frame (neglecting the x component). The durations for completing one shape were set to 1.2 s for the circle, 2 s for the rectangle, and 2 s for the Lissajous figure. We command 10 complete trajectories but implement a startup phase in which we gradually increase the speed to avoid rapid initial motions. For trajectory optimization, we use different planning horizons. The gradient descent and DDP controllers plan over a horizon of N = 200 steps with $\Delta t = 0.01$ s , resulting in a 2 s look-ahead window. Since the linear MPC linearizes around the current state, we use a shorter horizon of N = 40 to avoid larger errors from the linearization.

Fig. 5 shows the convergence of the gradient descent solver with a) numerical differentiation and b) auto differentiation, alongside the DDP controller. The figure presents the convergence for the initial call on one trajectory—where the

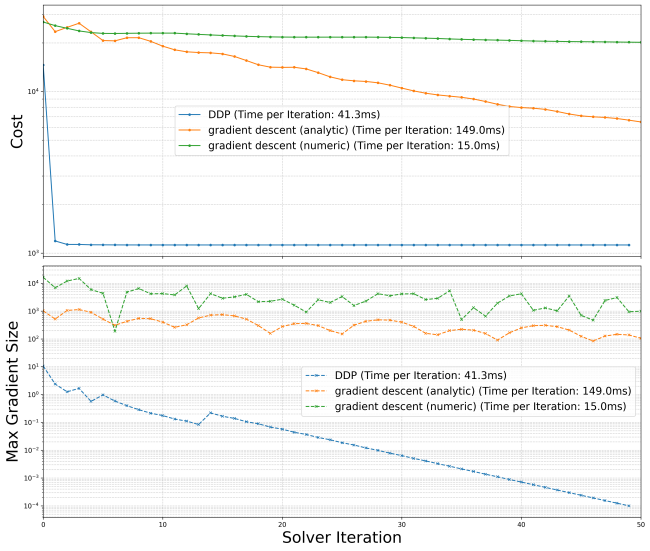

Fig. 5. Comparison of cost curve and max. gradient size for gradient descent controller vs DDP

initial control input is set to zero. Subsequent calls would use warm starting. Additionally, the mean solver times are reported. As expected, DDP converges much faster than gradient descent despite a longer solver time per iteration. The gradient descent controller using analytic differentiation has slower iteration, but overall converges more quickly thanks to more accurate gradient information.

The gradient-descent controllers converge far too slowly for closed-loop control. In [10], gradient descent was sufficient for closed-loop operation at around 40 Hz . We believe the main reasons for our slower convergence are the larger input dimension, $\mathbf{u} \in \mathbb{R}^7$ vs $\mathbf{u} \in \mathbb{R}^2$ and our much longer prediction horizon N = 200 compared to N = 25. Consequently, our tracking experiments use only the DDP and the linear MPC solvers. Fig. 6 shows the tracking of the geometric shapes, the corresponding tracking error, and the tip velocity.

Both controllers successfully tracked the reference trajectories, but the DDP-based solver consistently produced smaller tracking errors than the linear MPC. In particular, when following the rectangular path the linear MPC exhibited a noticeable oscillation of the tip, whereas the DDP solution remained smooth and closely aligned with the target. This behavior suggests that the linear MPC, with its limited horizon and reliance on local linearization, is less able to anticipate and compensate for longer-term effects of its control actions. On the computation side, the DDP solver required on average 446 ms (±157 ms) per optimization, which corresponds to roughly 30 ms per iteration, and converged in about 15 iterations to meet the gradient-based stopping criterion. By contrast, the linear MPC solver completed each optimization in just 18 ms (±3 ms). All timing measurements were taken on an AMD Ryzen 7 1800X running Ubuntu 24.04.

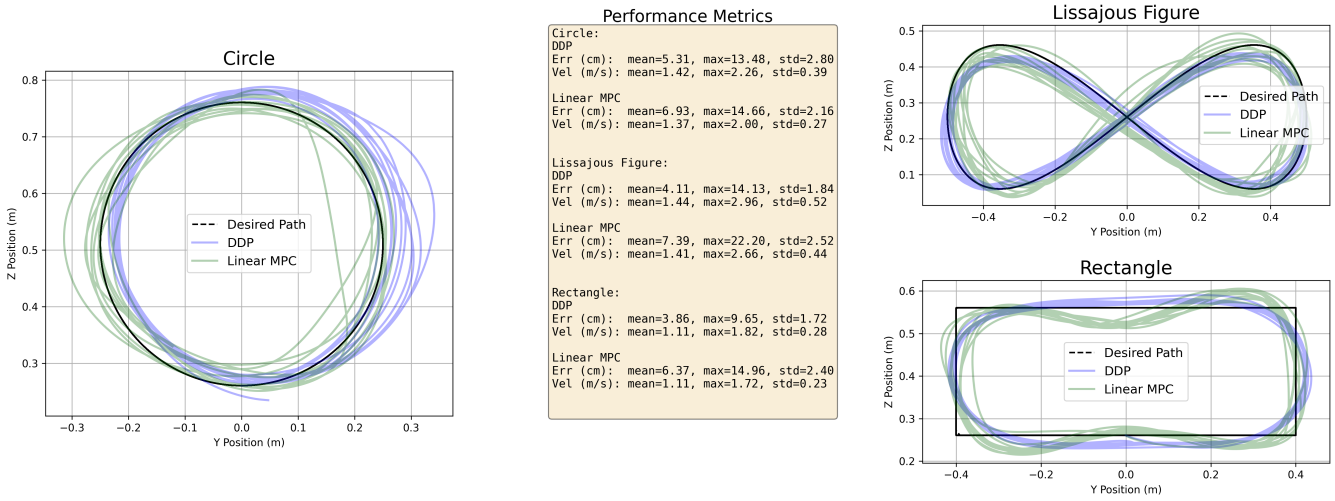

Fig. 6. Simulation: Tracking of different shapes in y-z-axis.

C. Tracking in real world

Fig. 7 shows the tracking of the different geometrical shapes on real hardware using the Linear MPC and DDP solver and the corresponding performance metrics. Additionally, footage of the experiments is available in the supplemental material. The visual tracking quality is noticeably worse than in the simulation, and the tracking performance metrics confirm this degradation. Overall, the DDP solver still outperforms the linear MPC solver. Both solvers exhibit oscillatory behavior when following the rectangular trajectory. Linear MPC in particular produces very large errors, indicating that its local linearization cannot compensate for the DLO's strong nonlinearities. The DDP solver, while better, also struggles to compensate for rapid changes in movement direction.

In terms of computation time, the DDP solver requires on average 825 ms ($\pm$ 72 ms) per optimization, with 42 ms per iteration over a mean of 19.5 iterations. This suggests it rarely satisfies the gradient-stopping criterion. By contrast, linear MPC completes each optimization in 22 ms ($\pm$ 1.5 ms). All experiments were conducted on an AMD Ryzen 7 1800X running Ubuntu 24.04 with a real-time kernel.

Because trajectory optimization is initialized from the current state estimate, we evaluated the tip-position estimation produced by the fusion network architecture in Sec.II using the data-collection routine described in Sec. IV-A. We observed a mean tip position estimation error of 1.25 cm ($\pm$ 1.21 cm) and a maximum error of 16 cm and when removing 0.5% of outliers the maximum error reduces to 7.2 cm. Fig. 8 shows the state estimation error over a segment of a recorded trajectory.

VI. CONCLUSION

This paper presents an approach to solving an closed loop optimal-control problem for tracking the end point of DLOs. We adopt a floating-base kinematic-chain model and introduced a novel perspective on a data-driven approximation of the hybrid dynamics to formulate a dynamical system for the DLO. In doing so, we eliminate restrictive assumptions made in [10]—for example, the requirement of a globally stable fixed point—and make our method more transferable to new scenarios.

We demonstrate that a simple MLP provides a sufficiently accurate approximation of the hybrid dynamics. Future work could integrate the more model-rich hybrid-dynamics approach from [11] and compare its performance and runtime against our purely data-driven approximation. We have implemented and compared several solvers—simple gradient descent, linear MPC, and DDP—in both simulation and on real hardware. Gradient descent does not converge quickly enough for closed-loop control, while linear MPC and DDP successfully solve the optimal-control problem online. All solvers track simple 2D geometric shapes effectively, although larger errors and suboptimal shape fidelity appear in hardware experiments. Future studies should compare open- vs. closed-loop tracking behavior and assess how state-estimation quality impacts overall performance.

Looking ahead, we plan to accelerate and enhance our existing solvers, incorporate environmental and hardware constraints more tightly, and explore constraint-friendly methods such as sequential quadratic programming (SQP).

REFERENCES

[1] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey," The International Journal of Robotics Research, vol. 37, no. 7, pp. 688–716, Jun. 2018, doi: https://doi.org/10.1177/0278364918779698.

[2] V. E. Arriola-Rios, P. Guler, F. Ficuciello, D. Kragic, B. Siciliano, and J. L. Wyatt, "Modeling of Deformable Objects for Robotic Manipulation: A Tutorial and Review," Frontiers in Robotics and AI, vol. 7, Sep. 2020, doi: https://doi.org/10.3389/frobt.2020.00082.

[3] S. Duenser, J. M. Bern, R. Poranne and S. Coros, "Interactive Robotic Manipulation of Elastic Objects," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 3476-3481, doi: 10.1109/IROS.2018.8594291.

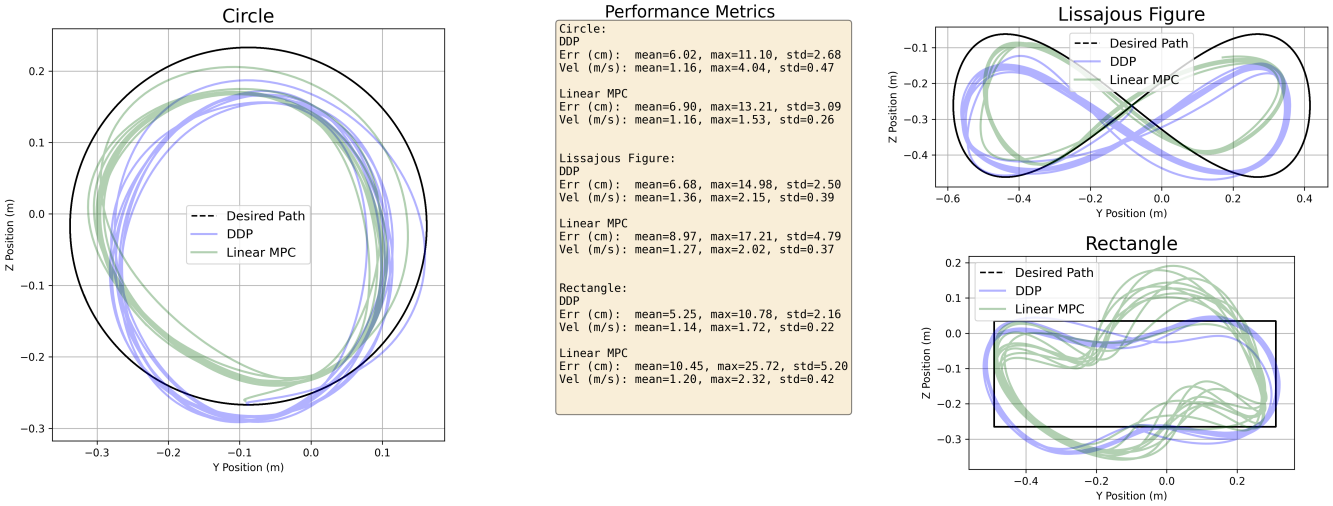

Fig. 7. Hardware: Tracking of different shapes in y-z-axis. A visual record of the experiments can be found in the supplementary video.

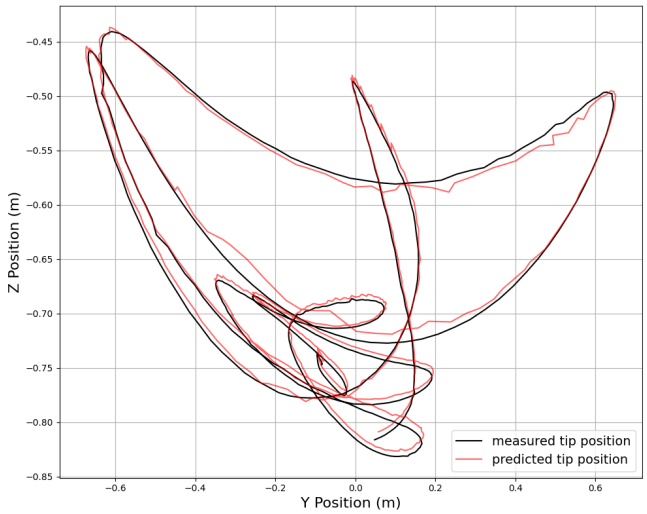

Fig. 8. Compare measured vs. predicted tip position by the DLO state estimation

[4] S. Zimmermann, R. Poranne and S. Coros, "Dynamic Manipulation of Deformable Objects With Implicit Integration," in IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 4209-4216, April 2021, doi: 10.1109/LRA.2021.3066969.

[5] S. Mamedov, A. R. Geist, J. Swevers and S. Trimpe, "Pseudo-rigid body networks: learning interpretable deformable object dynamics from partial observations," 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 2024, pp. 9542-9548, doi: 10.1109/IROS58592.2024.10802305.

[6] M. K. Klankers and J. J. Steil, "Modeling of Deformable Linear Objects Under Incomplete State Information," 2025 IEEE International Conference on Robotics and Automation (ICRA), Atlanta, GA, USA, 2025, pp. 10965-10972, doi: 10.1109/ICRA55743.2025.11127734.

[7] C. Hinze, M. Zürn, M. Wnuk, A. Lechler and A. Verl, "Nonlinear Trajectory Control for Deformable Linear Objects based on Physics Simulation," IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society, Singapore, 2020, pp. 310-316, doi: 10.1109/IECON43393.2020.9254923.

[8] M. Zürn, M. Wnuk, C. Hinze, A. Lechler, A. Verl and W. Xu, "Kinematic Trajectory Following Control For Constrained Deformable Linear Objects," 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), Lyon, France, 2021, pp. 1701-1707, doi: 10.1109/CASE49439.2021.9551613.

[9] S. Tiburzio, T. Coleman, D. Feliu-Talegon, and D. Santina, "Controlling Deformable Objects with Non-negligible Dynamics: a Shape-Regulation Approach to End-Point Positioning," arXiv.org, 2024. https://arxiv.org/abs/2402.16114 (accessed Sep. 14, 2025).

[10] J. A. Preiss, D. Millard, T. Yao and G. S. Sukhatme, "Tracking Fast Trajectories with a Deformable Object using a Learned Model," 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 1351-1357, doi: 10.1109/ICRA46639.2022.9812189.

[11] S. Mamedov, A. R. Geist, R. Viljoen, S. Trimpe and J. Swevers, "Learning Deformable Linear Object Dynamics From a Single Trajectory," in IEEE Robotics and Automation Letters, vol. 10, no. 7, pp. 7635-7642, July 2025, doi: 10.1109/LRA.2025.3577421.

[12] R. Featherstone, "Rigid body dynamics algorithms," Boston, MA: Springer US, 2008.

[13] J. Lloyd and V. Hayward, "Real-time trajectory generation using blend functions," in Proceedings. 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, USA, 1991, pp. 784,785,786,787,788,789, doi: 10.1109/ROBOT.1991.131682.

[14] J. Koenemann et al., "Whole-body model-predictive control applied to the HRP-2 humanoid," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 3346-3351, doi: 10.1109/IROS.2015.7353843.

[15] "Homepage," Franka.de, 2016. https://franka.de/de/ (accessed Feb. 16, 2026).

[16] J. Bradbury et al., JAX: composable transformations of Python+NumPy programs. 2018.

[17] J. Heek et al., Flax: A neural network library and ecosystem for JAX. 2024.

[18] jax2onnx, (2025), GitHub, https://github.com/enpasos/jax2onnx

[19] CppAD, (2025), GitHub, https://github.com/coin-or/CppAD

[20] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," in IEEE Transactions on Automatic Control, vol. 37, no. 3, pp. 332-341, March 1992, doi: 10.1109/9.119632.

[21] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: a parametric active-set algorithm for quadratic programming," Mathematical Programming Computation, vol. 6, no. 4, pp. 327–363, Apr. 2014, doi: https://doi.org/10.1007/s12532-014-0071-1.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[23] K. Cho et al., 'Learning phrase representations using RNN encoder–decoder for statistical machine translation', in Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1724–1734.