Automatically-Tuned Model Predictive Control for an Underwater Soft Robot

W. David Null, William Edwards, Dohun Jeong, Teodor Tchalakov, James Menezes, Kris Hauser, and Y Z

***Abstract*—Soft robots have desirable qualities for use in underwater environments thanks to their inherent compliance and lack of need for exposed hardware. Nevertheless, these advantages come at the cost of considerable control challenges. Data-driven model predictive control (MPC) is an approach that has shown promise in controlling soft robots. However, manually tuning the many hyperparameters in the learned dynamics model and the optimizer can be extremely tedious. In this work, we explore using data-driven MPC to control an underwater soft robot, and employ the AutoMPC method to automatically tune the hyperparameters and generate the controller. In the process, we extend AutoMPC's capabilities to handle multi-task tuning and we add a barrier cost function to enforce actuator constraints. Our experiments show that the AutoMPC controller reaches targets with significantly higher accuracy and reliability than state-of-the-art baselines both in- and out-of-distribution of the training data.**

***Index Terms*—Machine learning for robot control, modeling, control, and learning for soft robots.**

I. Introduction

SOFT robots are the subject of an active research field due to their potential to comply safely to disturbances, and to conform their shapes to objects and obstacles, in analogy to biological systems such as octopus arms [1]. Soft robots are especially appealing candidates for underwater operations thanks to the inherent compliance of their actuation mechanisms [2]. However, modeling and controlling soft robots to achieve precision tasks has remained a significant challenge. Because soft actuators are nonlinear, have infinite degrees of freedom (DoF), and exhibit complex dynamic behavior such as oscillation and hysteresis, analytical models can be incomplete or too computationally intensive for real time operations. Data-driven approaches, such as neural networks, have been widely explored to model soft robot kinematics and dynamics [3], [4]. A promising approach to controlling soft robots utilizes these models in conjunction with model predictive control (MPC) and has achieved high positioning accuracy in soft robot control in prior work [5], [6], [7]. However, it is highly sensitive to the hyperparameters of the learned model and optimizer, such as the number and type of hidden units in a neural network model, cost function, and the optimization horizon. Tuning them manually is both time-consuming and may pose safety risks for the robot.

To address this challenge, we adopt the AutoMPC [8] approach to automatically tune the MPC controller using an offline

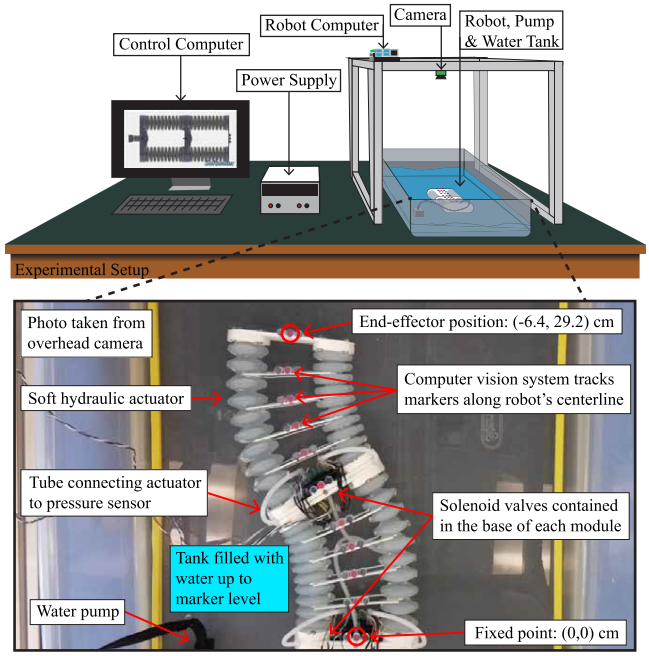

Fig. 1. Experimental setup of the 2D underwater soft robot. The sample image shown is taken from the overhead camera used by our controllers to track the color-based fiducials along the robot's centerline in real-time.

Manuscript received 22 May 2023; accepted 4 November 2023. Date of publication 16 November 2023; date of current version 29 November 2023. This letter was recommended for publication by Associate Editor F. Renda and Editor C. Laschi upon evaluation of the reviewers' comments. The work of William Edwards and Dohun Jeong was supported by NSF under Grant #IIS-2002492. *(Corresponding authors: Y Z; Kris Hauser.)*

W. David Null is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, and also with the Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: null2@illinois.edu).

William Edwards is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: wre2@illinois.edu).

Dohun Jeong, Teodor Tchalakov, and James Menezes are with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: dohunj2@illinois.edu; ttcha2@illinois.edu; jamesdm4@illinois.edu).

Kris Hauser is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, and also with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: kkhauser@illinois.edu).

Y Z is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, also with the Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, also with the Department of Nuclear, Plasma, and Radiological Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, and also with the Department of Nuclear Engineering and Radiological Sciences, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: yzyz@umich.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3333662, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3333662

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

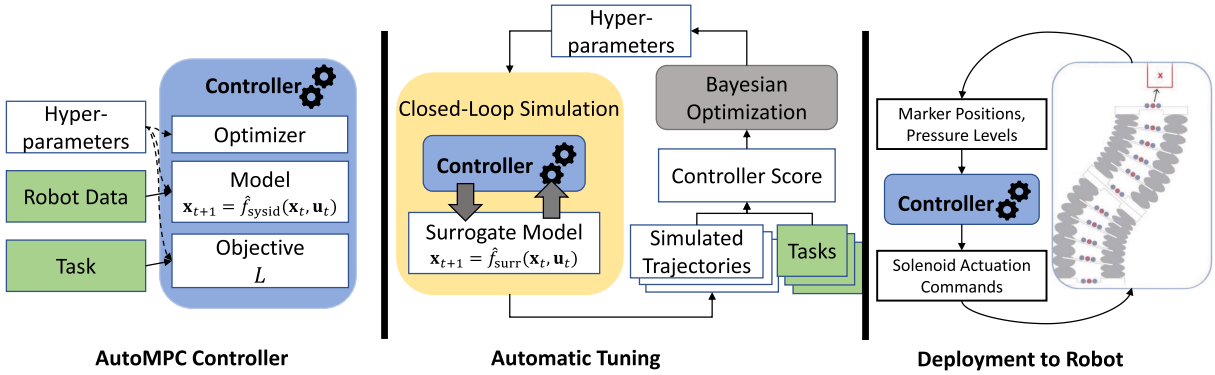

Fig. 2. AutoMPC controller diagram shown on the left, tuning process in the center, and deployment loop on the right.

data set, and apply it to a two-module, hydraulically-actuated underwater soft robot (Fig. 1). To better handle characteristics of the experimental system, this letter introduces several novel extensions of AutoMPC, namely to handle multi-task tuning and to enforce robot state constraints.

Experiments measure the performance of the AutoMPC controller in comparison to four baseline controllers, including open-loop learned inverse kinematics, learned visual servoing, offline reinforcement learning (RL), and RL on learned simulation models, with all models learned using the same data set. The auto-tuned MPC controller outperforms the baseline controllers on target reaching tasks and trajectory tracking tasks both in- and out-of-distribution of the training data.

II. Background

Analytical models such as piecewise constant curvature (PCC) [9] and Cosserat rod theory [10] have been employed for soft robot control in underwater environments [11], [12], [13]. These idealized models inevitably introduce errors, some of which can be mitigated using feedback control techniques such as visual servoing [14] and MPC [7]. However, most models are ill-equipped to handle real-world phenomena such as non-uniform material density, imperfect actuator fabrication, external forces such as gravity and friction, and the contact mechanics of interaction with the environment. Although recent work has made progress on modeling these effects using polynomial curvature models on a single underwater soft robotic tentacle [15], it remains unclear if this technique can be applied to pressure-driven multi-link soft robots. Finite element methods (FEM) provide more expressiveness for simulating complex mechanics [16], [17], but high computational costs limit their use for real-time control, and identifying model parameters is challenging. Reduced order FEM paired with feedback has been applied to control soft robots in real time [18], though this comes at the cost of simplifying the model.

Data-driven methods show promise in meeting these challenges because of their ability to capture nuanced characteristics of dynamics that are difficult to model analytically [3], [4], [10]. Feed-forward (FF) and recurrent neural networks (RNN) have been used extensively to model the forward and inverse kinematics, and dynamics of soft robots [5], [6], [19], [20], [21]. In many of these cases, training data is collected offline through a random actuation method known as motor babbling, though some works investigate more efficient ways to explore a robot's task and configuration space [22], and others use analytical or numerical models to generate training data in simulation [23], [24]. Other works use reinforcement learning to directly learn control policies that perform well in tasks like target reaching, and can be robust to system disturbances [25]. However, they typically require a large amount of online training data [24].

Data-driven MPC has shown promising results when applied to pneumatically actuated soft robots [5], [6], [20], [26]. Huang et al. demonstrate a ball-catching soft robot which uses an MPC controller with a dynamics model learned by linear regression [26]. Other works use deep neural networks (DNNs) to learn the dynamics of soft robots and develop MPC controllers based on linearized dynamics models [5], [6]. A major challenge of these approaches is tuning the many hyperparameters that come with data-driven MPC. Each cited work used manual tuning, which can be tedious and potentially cause instability and damage to the robot.

To address the challenges of manual MPC tuning, several works have explored automatic tuning of MPC parameters, including the model parameters [8], [27], [28], [29], the optimization objective [8], [29], [30], and control horizon [8], [29]. Methods explored in recent works include Bayesian optimization [8], [28], reinforcement learning [31], and differentiable MPC [32]. In this work, we make use of AutoMPC [8] (illustrated in Fig. 2) an open-source Python library for automatic tuning of data-driven MPC. In contrast to other tuning approaches, AutoMPC tunes the hyperparameters of a variety of dynamics models and planning algorithms, runs fully offline evaluating performance on a learned "surrogate" dynamics model, and is extensible to support custom task specifications. The previous version of AutoMPC could only tune a single task and had limited support for state constraints. This letter extends AutoMPC to incorporate multi-task tuning and a logarithmic barrier function to handle state constraints in unconstrained optimizers, like the iLQR algorithm being used in our application.

III. Underwater Soft Robot

A. Robot Description

Our experimental platform (Fig. 1) is a planar underwater soft robot, consisting of two modules connected end-to-end, each containing a parallel set of soft hydraulic actuators. The robot can achieve a variety of shapes and configurations by adjusting the pressure in each actuator.

To control the robot, a pump drives water through a network of solenoid valves into each actuator independently. To

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

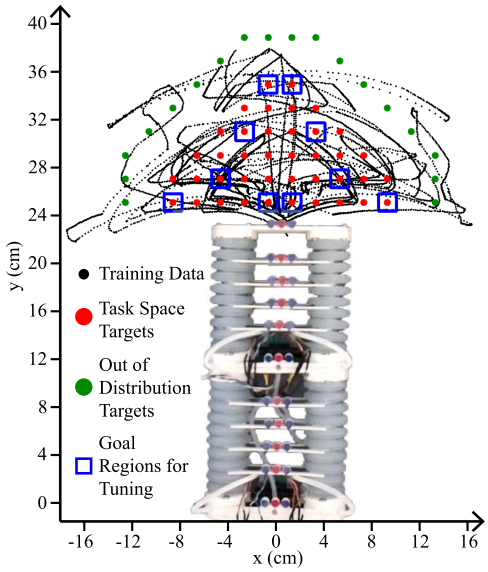

Fig. 3. End-effector trajectories, evaluation targets, and the goal regions used in tuning overlaid on the robot's Cartesian workspace. The robot is pictured in the fully depressurized home position for reference.

depressurize, the elasticity of the actuators forces water out of a separate set of solenoid valves. The operating range of each actuator is 97 kPa to 118 kPa. Pressure levels above 118 kPa risk damaging the actuator. Feedback is provided by pressure sensors for each actuator and by a computer vision system which tracks color-based fiducials arrayed along the centerline of the robot. For reference, the resolution of the vision system is approximately 0.5 mm, and the length of the robot is 240 mm when fully depressurized and 380 mm when fully pressurized. Considering that the robot motion is relatively slow, with a typical end-effector speed of under 1 cm/s, we chose the control frequency and data sampling rate to be 2 Hz. We define the state space of the robot as $\mathbf{x} = (\mathbf{x}_{\text{marker}}, \mathbf{x}_{\text{pressure}})$ consisting of eleven 2D Cartesian marker locations and four analog pressure readings, and its control $\mathbf{u}$ consisting of 8 binary solenoid inputs, controlling the intake and outtake valves for each actuator. For more details on the robot, see [33].

B. Offline Data Collection

Offline training data was collected by randomly actuating the solenoid valves and deliberately navigating the robot to various regions of the configuration space. The planned trajectories explored interesting areas of the state space such as pressure extremes and interesting shapes such as S-curves. Trajectories sampled at 2 Hz constitute the training set for both the AutoMPC and baseline controllers. The data set was lightly postprocessed to remove perception errors and ensure a consistent time step. Fig. 3 shows the coverage of the 11,358 data samples.

IV. AutoMPC

We design a controller for the underwater soft robot using AutoMPC [8], a software library which automates the design of data-driven MPC. The inputs to AutoMPC are a data set of discrete-time trajectories $\mathcal{D}$ and a task $\tau = (J, \mathbf{u}_{\min}, \mathbf{u}_{\max}, \mathbf{x}_{\min}, \mathbf{x}_{\max})$, where J is a performance metric which assigns numerical scores to trajectories, and $(\mathbf{u}_{\min}, \ldots, \mathbf{x}_{\max})$ give bounds for the controls and states. The output of AutoMPC is a controller which consists of a system ID model $\hat{f}_{\text{sysid}}$, an objective function L, and an optimizer which generates controls by solving a problem of the form

$$\begin{aligned} &\min_{\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}} L(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}) \\ &\text{s.t. } \mathbf{x}_{i+1} = \hat{f}_{\text{sysid}}(\mathbf{x}_i, \mathbf{u}_i) \\ &\text{and } \mathbf{u}_{\min} \le \mathbf{u}_t \le \mathbf{u}_{\max} \text{ and } \mathbf{x}_{\min} \le \mathbf{x}_t \le \mathbf{x}_{\max}, \end{aligned} \tag{1}$$

where $\mathbf{x}_{t:t+H}$ denotes $\mathbf{x}_t, \ldots, \mathbf{x}_{t+H}$, $\mathbf{u}_{t:t+H}$ denotes $\mathbf{u}_t, \ldots, \mathbf{u}_{t+H}$, and H gives the optimization horizon. AutoMPC uses Bayesian optimization to solve the challenging design problem of selecting a model, objective function, and optimizer which effectively solve the task τ. We refer to the search space of possible controller settings as the *configuration space* $\mathcal{H}$, and refer to a particular controller setting as a *configuration* $h \in \mathcal{H}$.

To handle multiple tasks during tuning, this letter introduces the notion of a *task transformer*, Θ, which given a specific task τ, generates the MPC objective function $\Theta(\tau, h) \to L_{\tau,h}$. Although it may seem reasonable to set L to be the same as the performance metric J, this has challenges in practice. In a target reaching task where J is designed as a sparse metric such that the robot is only rewarded for reaching the goal, MPC cannot directly optimize for J because the cost landscape is flat almost everywhere. Therefore, the task transformer creates more favorable objectives for online optimization while J is only used for offline hyperparameter tuning.

To evaluate a configuration, AutoMPC performs closed-loop simulation of the resultant controller. Since tuning is performed offline, a surrogate dynamics model $\hat{f}_{\text{surr}}$ is learned and used as a simulator. The simulated trajectory is evaluated by the task performance metric J to produce the controller score. To help address the sim-to-real problem, $\hat{f}_{\text{surr}}$ is distinct from $\hat{f}_{\text{sysid}}$ with independently selected model hyperparameters.

A. Multi-Task Tuning

Previously, AutoMPC has been used to tune controller performance for only a single task. In this work, we extend AutoMPC to synthesize and tune controllers which can generalize across multiple tasks. Consider a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$, each with an associated performance metric J_{τ_i}. The AutoMPC controller can be easily executed for different tasks by using the task transformer Θ to generate a task-specific objective function. A subset of the tasks $\mathcal{T}_{\text{train}} \subset \mathcal{T}$ are selected for tuning. To evaluate a candidate controller, AutoMPC runs simulations for every task $\tau \in \mathcal{T}_{\text{train}}$ in parallel, scoring each simulation with the associated performance metric J_τ. The scores are then aggregated into an overall controller score, which can be done using one of several methods, such as median, worst-case, or specified percentile, though in this letter we simply take the mean.

B. AutoMPC Set-Up for Underwater Robot

We apply AutoMPC to the underwater soft robot, using the state and control space defined in Section III-A. Since the controls are binary, the control limits are $\mathbf{u}_{\min} = \mathbf{0}$ and $\mathbf{u}_{\max} = \mathbf{1}$. The pressure levels are upper-bounded at 118 kPa to ensure that the actuators stay below their burst pressure, and lower-bounded

at 97 kPa since the actuators cannot achieve negative differential pressure in deployment. The end-effector point $\mathbf{x}_{\text{ee}} \in \mathbb{R}^2$ is represented by the marker at the top of the robot. The control frequency was chosen to be 2 Hz to match the sampling rate of the training.

We define a set of 40 tasks $\{\tau_1, \ldots, \tau_{40}\}$ each of which is defined by a goal region $\mathcal{G}_i \subset \mathbb{R}^2$ for the end-effector position. Each of the 40 tasks corresponds to a 2x2 cm^2 goal region centered at a target point as illustrated in Fig. 3. The size of the goal region was selected to maximize tuning performance based on observed precision of the robot. Of these, ten tasks representative of the task space are selected for $\mathcal{T}_{\text{train}}$. To test generalization, we also consider a set of 18 *out-of-distribution* tasks (also shown in Fig. 3) which are at least 4 cm outside the task space of 40 tasks. The associated performance metric for each task is the number of seconds for which the end-effector point is not in the goal region:

$$J_\tau(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = \Delta t \sum_{j=1}^{T} \left(1 - \mathbb{1}_{\mathcal{G}_\tau}\left(\mathbf{x}_{j,ee}\right)\right) \tag{2}$$

where Δt gives the controller period, and $\mathbb{1}_{\mathcal{G}_\tau}$ gives the goal region indicator function. Experiments found that this sparse metric yielded better tuning results than other metrics such as those based on incremental or terminal distance to target.

AutoMPC provides several options to specify the form of the task transformer, surrogate model, system ID model, and optimizer. The task transformer $\Theta(\tau, h)$ generates an optimizer-friendly objective function of the form

$$\begin{aligned} L_{\tau,h}(\mathbf{x}_{t:t+H}, \mathbf{u}_{t:t+H-1}) = &\underbrace{\bar{\mathbf{x}}_{t+H}^T \mathbf{F} \bar{\mathbf{x}}_{t+H}}_{\text{(a)}} + \underbrace{\mathbf{s}^T L_B(\mathbf{x}_{t+H})}_{\text{(b)}} \\ &+ \sum_{i=t}^{t+H-1} \left(\underbrace{\bar{\mathbf{x}}_i^T \mathbf{Q} \bar{\mathbf{x}}_i}_{\text{(c)}} + \underbrace{\mathbf{u}_i^T \mathbf{R} \mathbf{u}_i}_{\text{(d)}} + \underbrace{\mathbf{s}^T L_B(\mathbf{x}_i)}_{\text{(e)}} \right) \end{aligned} \tag{3}$$

where $\bar{\mathbf{x}}_i = \mathbf{x}_{i,ee} - \mathbf{x}_{\text{ee}}^d$, (a) and (c) give a quadratic penalty on the deviation between the end-effector point at time i and its desired position $\mathbf{x}_{\text{ee}}^d$, computed as the geometric center of task-specific goal region $\mathcal{G}_\tau$, (d) gives a quadratic penalty on actuation, and (b) and (e) give a logarithmic barrier for the pressure levels, scaled by $\mathbf{s}$ (see Section IV-D for details). Although the physical cost to actuating solenoid valves is low, the tunable coefficients in (d) allow the tuner to discourage undesirable behaviors, such as those that may lead to model instability.

To choose the surrogate model, we used AutoMPC to perform automatic selection and tuning from among all available modeling algorithms provided by the AutoMPC package, namely ARX, MLP, SINDy, and Koopman operators [8]. Tuned over 250 iterations, candidate models were evaluated based on prediction accuracy on a rolling 10 s horizon over approximately 8 minutes of recorded robot data set aside from $\mathcal{D}$. The selected surrogate model was a 1-layer multi-layer perceptron (MLP) with 101 hidden neurons.

We restricted the system ID model class to MLP due to its accuracy in surrogate tuning. The system ID MLP candidates were trained on the same data set as the surrogate model. For the optimizer, we chose iterative LQR (iLQR) [34] due to its speed and compatibility with MLP. Altogether, the controller's tunable hyperparameters in $\mathcal{H}$ include number of layers, layer sizes, learning rate, and activation function for the MLP, the optimization horizon, controller frequency, and number of iterations used by iLQR, and the coefficients $\mathbf{Q}$, $\mathbf{F}$, $\mathbf{R}$, and $\mathbf{s}$ in the objective.

C. Binary Control Outputs

To avoid the need for an additional low-level controller, we chose to have AutoMPC directly control the state of the solenoids rather than the internal pressure values. Since the solenoids require a discrete binary control signal but all AutoMPC optimizers are geared toward continuous control, we add a simple post-processing step which rounds the controller output to the nearest valid control option, either 0 (valve closed) or 1 (valve opened). Note that the rounding strategy should be embedded in the AutoMPC tuning loop rather than applied post-hoc, because via tuning AutoMPC can find controllers that perform well in this mode rather than simply hoping that rounding will work.

D. Barrier Cost Function

As discussed in Section IV-B, we introduce state constraints on the pressure levels to prevent the controller from trying to reach unattainable or dangerous pressure levels. Since our chosen optimization algorithm, iLQR, does not natively support state constraints, we introduce a logarithmic barrier function to enforce the constraints. These take the form

$$\mathbf{s}^T L_B(\mathbf{x}) = \sum_{i \in \mathcal{B}} -s_i \left[\log\left(\mathbf{x}_i - \mathbf{x}_{min,i}\right) + \log\left(\mathbf{x}_{max,i} - \mathbf{x}_i\right)\right] \tag{4}$$

where $\mathcal{B}$ is a set containing the indices of state dimensions with finite bounds. The optimal solution to this augmented control problem converges to that of the original problem only when the scale factor approaches zero. Thus, adding the logarithmic barrier may perturb the output of iLQR.

Several solutions have been proposed to address this issue, such as replacing the barrier function with its second order Taylor approximation [35], or introducing an outer-loop in the iLQR algorithm to decrease the scale until convergence [36]. The former does not strictly enforce the barriers, while the latter assumes that a control sequence for a strictly feasible trajectory is available. We choose to simply include the barrier scales as tunable hyperparameters for each constrained dimension ($s_i > 0$). When the initial guess trajectory is infeasible, due to perturbations or modeling errors, we relax the constraints to be ϵ-close to the nominal trajectory ($\epsilon = 10^{-10}$). Our experiments find that AutoMPC tunes the scales to be small enough to find a solution close to the original constrained optimization problem, but large enough to prevent constraint violation in the training tasks.

E. Tuning Results

We ran 200 iterations of tuning on 27 hyperparameters (eight hours on a commercial desktop CPU). Although the hyperparameter space is large, we note that the Bayesian optimizer used by AutoMPC [37] has been demonstrated to successfully scale to as many as 110 hyperparameters [38]. Fig. 4 reports the performance of the best-known controller over the course of the tuning process. We report the mean and standard deviation of performance across tasks in both the training set (Fig. 4(a))

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

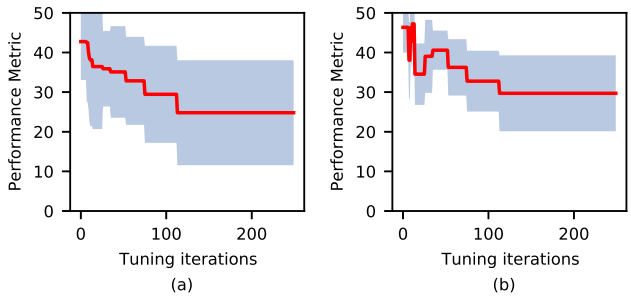

Fig. 4. Tuning curve showing the mean (solid line) and standard deviation (shaded region) of training (a) and testing (b) task performance metrics described in (2).

and testing set (Fig. 4(b)). We observe that AutoMPC improves consistently in controller performance across training and testing tasks over the first 100 tuning iterations, however, does not continue to improve over the next 100. Note that while the standard deviation of performance across tasks remains high, this is to be expected since the time needed to reach the goal region depends greatly on the distance between the goal and initial position.

The best configuration at the end of the tuning process had a single hidden layer MLP system ID model with 140 hidden units, tanh nonlinear layer, and an MPC horizon of 5, with iLQR being run every 4 control steps.

V. Results

A. Baseline Controllers

Four separate controllers, described in the following sections, are developed as baseline comparisons for AutoMPC. The same data set $\mathcal{D}$ is used to train the learned components of these controllers. Control frequencies are 2 Hz unless otherwise stated.

1) Open Loop Inverse Kinematics: This baseline controller uses a learned inverse kinematics model to predict the actuator pressure values needed to produce a desired end-effector position: $\mathbf{x}_P^d = \hat{g}(\mathbf{x}_{\text{ee}}^d)$. The model is a deep neural network with 4 hidden layers and 11,316 trainable parameters. Training and validation data were partitioned from $\mathcal{D}$. The control algorithm first generates a vector of desired pressures $\mathbf{x}_P^d \in \mathbb{R}^4$ from a desired end-effector goal $\mathbf{x}_{\text{ee}}^d \in \mathbb{R}^2$, and then drives each actuator to the desired pressure simultaneously. There is no feedback to the controller regarding the true end-effector position and the target pressures are calculated only once.

2) Visual Servo: This baseline controller incorporates feedback of the observed end-effector position to minimize the distance between $\mathbf{x}_{\text{ee}}$ and $\mathbf{x}_{\text{ee}}^d$ and correct for steady state error in the open loop controller. A manually tuned PI control loop running at 0.4 Hz updates the x and y coordinates of an adjusted target $\mathbf{x}_{\text{adj}}$ where $\mathbf{x}_{\text{adj}} = \mathbf{x}_{\text{ee}}^d$ at $t = 0$. The adjusted target is calculated as follows, where $\mathbf{K}_P$ and $\mathbf{K}_I$ are the proportional and integral gains respectively, $\mathbf{e} = \mathbf{x}_{\text{ee}}^d - \mathbf{x}_{\text{ee}}$ is the distance error, $\mathbf{e}_{\text{int}}$ is the integration error, and $\mathbf{e}_{\text{sat}}$ is the saturation limit of the integration error and $\mathbf{K}_P, \mathbf{K}_I \in \mathbb{R}^{2\times 2}$ and $\mathbf{x}_{\text{adj}}, \mathbf{e}, \mathbf{e}_{\text{int}}, \mathbf{e}_{\text{sat}} \in \mathbb{R}^2$:

$$\mathbf{e}_{\text{int}}(t) = \text{clamp}\left(\int_0^{t-1} \mathbf{e}(t)\, dt, -\mathbf{e}_{\text{sat}}, \mathbf{e}_{\text{sat}}\right)$$
$$\mathbf{x}_{\text{adj}} = \mathbf{x}_{\text{ee}}^d + \mathbf{K}_P\, \mathbf{e}(t) + \mathbf{K}_I\, \mathbf{e}_{\text{int}}(t) \quad (5)$$

New target pressures are calculated at each step using the adjusted target and the learned inverse kinematics model from the open loop controller: $\mathbf{x}_P^d = \hat{g}(\mathbf{x}_{\text{adj}})$. These new target pressure commands are carried out by a low level control loop running at 2 Hz.

3) Conservative Q-Learning: Conservative Q-Learning (CQL) [39] is a popular offline model-free reinforcement learning algorithm that has demonstrated good performance on the D4RL benchmark [40]. We constructed our data set by replicating $\mathcal{D}$ 10 times for each of the goal positions $\mathbf{x}_{\text{ee}}^d$ and augmenting the state with the current goal position. Next, we assigned the negative square error distance between the end-effector and the goal position ($\|\mathbf{x}_{\text{ee}}^d - \mathbf{x}_{\text{ee}}\|_2^2$) as the reward function. Finally, we used a continuous action space and rounded the outputs of the policy to obtain discrete controls. Given this multi-task reinforcement learning data set, we trained a CQL implementation from the D3RLPy library [41] and performed a hyperparameter grid search using the same parameter set as the original letter [39] to select the maximum reward policy through rollout simulations on the surrogate dynamics model $\hat{f}_{\text{surr}}$ which was used during the AutoMPC tuning process.

4) Trust Region Policy Optimization: Recently, deep RL was applied to control the end-effector position of a two-module pneumatically-actuated soft robotic manipulator capable of 3D motion [42]. The proposed method used a forward dynamics model, learned with an LSTM-based neural network, to train a control policy using trust region policy optimization (TRPO) which was evaluated in simulation and on the real robot through tracking three trajectories.

Here, we apply this approach for the task of position targeting. The forward dynamics,

$$\mathbf{x}_{i+1} = \hat{f}_{\text{fd}}(\mathbf{x}_i, \mathbf{x}_{i-1}, \mathbf{x}_{i-2}, \mathbf{x}_{i-3}, \mathbf{u}_i, \mathbf{u}_{i-1}, \mathbf{u}_{i-2}, \mathbf{u}_{i-3})$$

are learned using a recurrent neural network with a single hidden layer of 128 LSTM units, and is trained using data from $\mathcal{D}$ with a mean squared error loss function. The resulting model achieves an average end-effector distance error of 1.183 mm over the training data and 1.184 mm over the testing data with 25th and 75th percentiles of error being [0.545, 1.488] mm over the training data and [0.545, 1.550] mm over the testing data.

Next, we formulate our Markov Decision Process (MDP) in a similar fashion to [42]. We describe our states by $s_t = (\mathbf{x}_{\text{ee}}^d(t), \mathbf{e}(t), \mathbf{x}_{\text{ee}}(t), \mathbf{x}_{\text{mid}}(t)) \in \mathbb{R}^8$ where $\mathbf{x}_{\text{ee}}^d$ is the target position, $\mathbf{e}$ is the error between the end-effector and the target, $\mathbf{x}_{\text{ee}}$ is the current end-effector position, and $\mathbf{x}_{\text{mid}}$ is the current position of the midpoint of the robot. An action, given by $a_t \in \{0, 1, 2 \ldots, 255\}$, corresponds to a configuration of the eight binary solenoid valves. The reward function used is the same as [42], $r_t = -\|\mathbf{x}_{\text{ee}}^d - \mathbf{x}_{\text{ee}}\|_2$. The control policy was parameterized by a deep neural network with a single hidden layer with 64 neurons and Tanh activation function, and was trained using TRPO with parameters $\gamma = 0.99$ and Max KL divergence of 0.005. For each batch of training, the forward dynamics model would simulate the robot attempting to reach a goal position from the starting depressurized position over 100 timesteps. The targets used in each batch of training were the same ten targets used in $\mathcal{T}_{\text{train}}$ for tuning AutoMPC. Thus there were 1,000 steps computed per batch, and 5,000,000 training steps in total over 5,000 epochs. This trained control policy was

TABLE I
GOAL REACHING FOR DIFFERENT CONTROLLERS

Controller	Task Space Set: 40 targets, 50 seconds per run				Out-Of-Distribution Set: 18 targets, 70 seconds per run			
	Targets Reached		x_{ee} Error (cm)	Time Outside $\mathcal{G}$ (s)	Targets Reached		x_{ee} Error (cm)	Time Outside $\mathcal{G}$ (s)
	At All	At End	Mean (Std)	Mean (Std)	At All	At End	Mean (Std)	Mean (Std)
Open Loop IK	14	7	2.407 (1.432)	43.862 (10.449)	0	0	4.713 (2.295)	70.0 (0.0)
Visual Servo	27	15	1.382 (0.763)	40.647 (12.839)	6	4	2.844 (3.877)	64.063 (13.963)
CQL	14	4	2.765 (1.724)	42.488 (13.633)	2	0	5.826 (4.452)	68.372 (4.854)
TRPO	35	2	4.259 (2.058)	41.931 (5.304)	5	1	2.740 (1.595)	65.61 (8.025)
AutoMPC	40	39	0.296 (0.249)	19.113 (8.343)	10	7	1.496 (1.604)	55.318 (18.723)

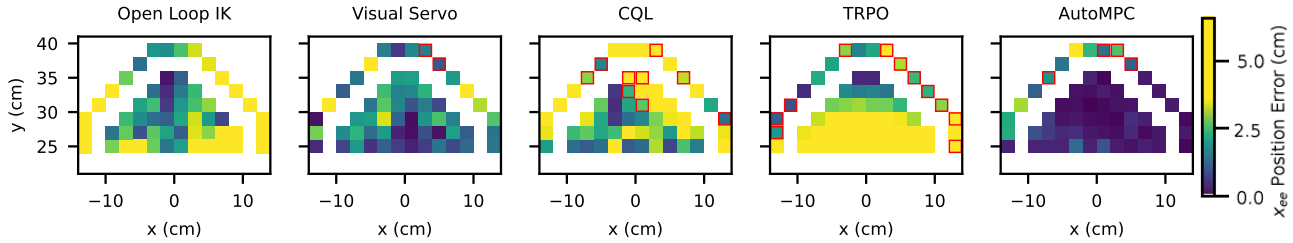

Fig. 5. 2D Histograms representing the final distance error at the end of each experiment for AutoMPC and baseline controllers on the chosen task space and out-of-distribution targets. Red outlines represent runs where an acutator's pressure limit was exceeded and the robot emergency-stopped. In these cases, the last recorded distance to the goal is taken as the final error and reported through the color of the underlying square.

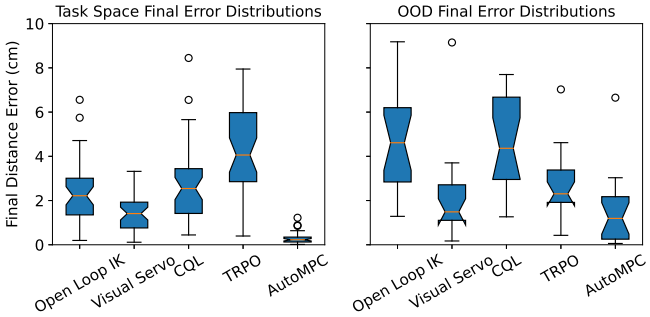

Fig. 6. Final controller error distributions over task space and OOD targets.

tested in closed loop in simulation and on the underwater soft robot itself.

B. Experimental Results

We test the controllers on the 40 goal positions in the task space. In each run, the soft robot starts from a fully depressurized home position with retracted actuators and is given 50 s to reach a specified target. We measure the distance error $\|\mathbf{x}_{ee} - \mathbf{x}_{ee}^{d}\|_2$, and the time the end-effector spent outside the goal region $\mathcal{G}$. When testing out-of-distribution testing tasks, the robot was given 70 s to reach the goals to give it ample time to reach the farther goal positions.

Table I reports the number of targets reached, mean final end-effector position error, and mean time spent outside of $\mathcal{G}$ (i.e., (2)). Fig. 5 plots the final end-effector position error for all five controllers. Open Loop IK performs poorly in certain regions, most likely because the mapping from end-effector positions to pressure configurations is ambiguous. With the addition of closed loop feedback, Visual Servo is able to improve performance in locations where Open Loop IK had high error. CQL is the only controller which exceeds actuator pressure limits in the task space. We presume that the augmented data set used to train it is not large enough to allow for convergence to an adequate policy. TRPO performed better in simulation on

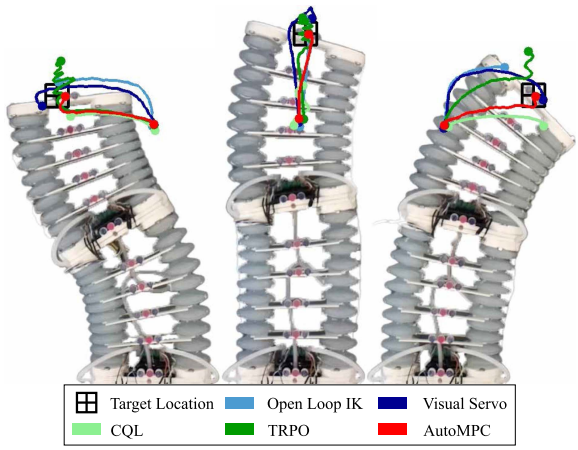

Fig. 7. Experiment trajectories from all controllers for selected targets: left is $(-9, 27)$ cm, middle is $(1, 33)$ cm, and right is $(9, 27)$ cm. The underlying photograph shows the robot's final state from the AutoMPC controller. The paths taken by AutoMPC and the baseline controllers are overlayed. Best viewed in color.

the LSTM-based model $\hat{f}_{\text{fd}}$ achieving a 0.886 cm end-effector distance error, compared to 4.259 cm error on the real robot. Among the task space targets, TRPO initially performs well, passing through the goal region in 35 out of 40 cases, but as the experiment continues the trajectory tends to drift in the +y direction. This drifting is less prevalent in the OOD trajectories because these targets are at the furthest points of the robot's workspace and many experiments are stopped prematurely due to overpressurization errors.

In contrast, AutoMPC consistently achieves high accuracy in reaching the target, outperforming the four baselines for 35 of the 40 task space goals and 8 of the 18 out-of-distribution goals. It also significantly outperforms baselines on other metrics, reaching the most targets during and by the end of the experiment for both task space (39/40) and out-of-distribution goals (7/18), lower mean distance error, and lowest time outside $\mathcal{G}$.

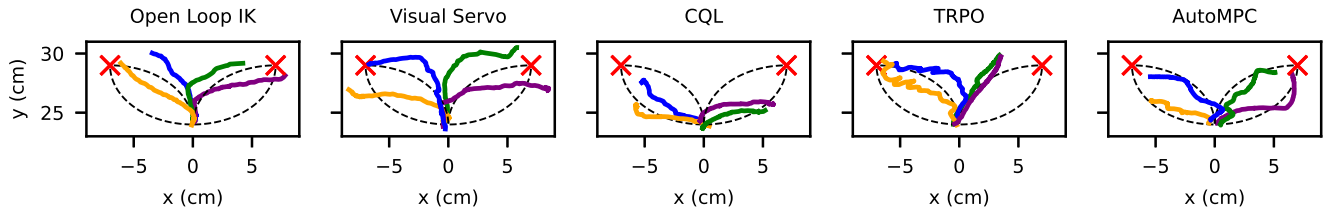

Fig. 8. Trajectory tracking results. The four desired trajectories are shown in black dashed lines, actual trajectories in color: orange-bottom left, blue-top left, purple-bottom right, green-top right.

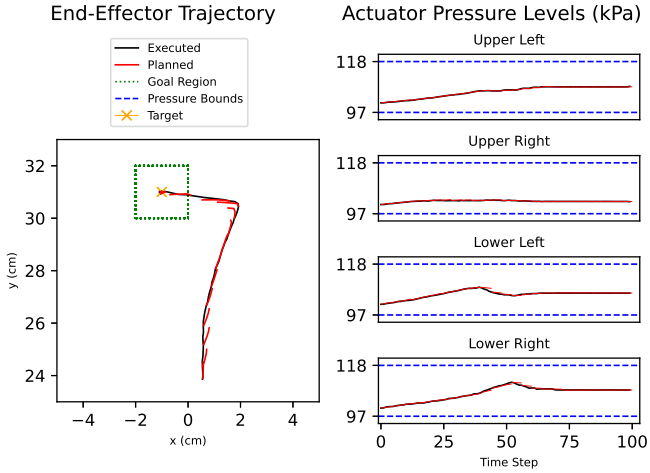

Fig. 9. Left: End-effector trajectory converging to target executed by AutoMPC with planned trajectories overlaid. Right: executed and planned pressure levels of all four actuators, indicating high prediction accuracy.

We further evaluate the controllers on a simple trajectory tracking task. As shown in Fig. 8, we specify four different trajectories beginning at the home position (0,24) and terminating at two different task space targets, $(-7, 29)$ and (7,29) with a rate of 1.9 mm per second. For each controller, we implement trajectory following by sequentially updating the target point according to the given trajectory at every timestep. Fig. 8 illustrates the paths taken by each of the controllers. The AutoMPC controller has the lowest average trajectory tracking absolute error at 1.39 cm compared to 2.87 cm for open loop IK, 1.74 cm for visual servo, 2.23 cm for CQL, and 1.77 cm for TRPO.

C. Discussion

Fig. 7 shows representative trajectories for the five controllers on three goals. Open Loop IK tends to undershoot the target, while Visual Servo tends to overshoot. CQL tends to finish below the target, whereas TRPO begins by taking a fairly direct path to the target, but drifts in the +y direction as mentioned. This is due to the control policy learning to exploit small inaccuracies in the LSTM model during training to hold the end-effector stationary. In the real-world, these small adjustments push the end-effector into unexplored regions above the target further worsening the drift. These regions tend to be unexplored because the policy learns in simulation how to not overshoot the target. AutoMPC generally takes the most direct path and stays on target throughout the experiment. Fig. 7 also shows the robot's final shape after the AutoMPC controller run, which is not symmetrical for the left and right targets. Tracked trajectories in Fig. 8 are also observed to be asymmetrical.

Fig. 9 visualizes a representative run of the AutoMPC controller's planned and executed trajectories for an example target reaching task. We observe that the robot largely follows the controller's internal system ID model, demonstrating low modeling error and successful execution of the trajectories. Tuning and testing performances of the AutoMPC controller are observably similar within the task space and degrade on out-of-distribution tasks. As marked by the red boxes in Fig. 5, most controllers occasionally reach pressure limits, triggering run termination. AutoMPC met limits on four out-of-distribution targets despite our barrier costs due to worsening model error as dynamics predictions $\hat{f}_{\text{sysid}}$ became less accurate as the robot left the task space. In these cases, we observe that AutoMPC always optimized trajectories that would not have exceeded pressure limits if $\hat{f}_{\text{sysid}}$ were accurate, but pressures still tended upwards under the true dynamics. Possible approaches to handle out-of-distribution dynamics errors include adaptive control to learn the error online, and robust control to penalize regions where the model is uncertain. A potential disadvantage of the AutoMPC controller is increased computation time due to the use of trajectory optimization. With the tuned hyperparameters its control loop is limited to approximately 5 Hz, which is faster than the robot's solenoid control frequency but 4–5x slower than the baselines.

VI. Conclusion

In this work, we control an underwater soft robot by tuning a data-driven MPC controller using the AutoMPC approach. To apply AutoMPC to our robot, we implement multitask tuning to achieve generalization across a selected task space and a barrier cost function to enforce state space constraints. Trained on an offline data set, we demonstrated that AutoMPC can outperform a variety of baseline controllers and can generalize relatively well to out-of-distribution tasks. Areas of future work include applying our methods to larger robot arms with more modules, exploring more complex tasks like grasping, and considering the impact of data collection strategies and tuning task selection on performance.

References

[1] J. Hughes, U. Culha, F. Giardina, F. Guenther, A. Rosendo, and F. Iida, "Soft manipulators and grippers: A review," *Front. Robot. AI*, vol. 3, 2016, Art. no. 69.
[2] Q. Tan et al., "Underwater crawling robot with hydraulic soft actuators," *Front. Robot. AI*, vol. 8, 2021, Art. no. 688697.
[3] D. Kim et al., "Review of machine learning methods in soft robotics," *PLoS One*, vol. 16, no. 2, 2021, Art. no. e0246102.

[4] K. Chin, T. Hellebrekers, and C. Majidi, "Machine learning for soft robotic sensing and control," *Adv. Intell. Syst.*, vol. 2, no. 6, 2020, Art. no. 1900171.

[5] M. T. Gillespie, C. M. Best, E. C. Townsend, D. Wingate, and M. D. Killpack, "Learning nonlinear dynamic models of soft robots for model predictive control with neural networks," in *Proc. IEEE Int. Conf. Soft Robot.*, 2018, pp. 39–45.

[6] P. Hyatt, D. Wingate, and M. D. Killpack, "Model-based control of soft actuators using learned non-linear discrete-time models," *Front. Robot. AI*, vol. 6, 2019, Art. no. 22.

[7] C. M. Best, M. T. Gillespie, P. Hyatt, L. Rupert, V. Sherrod, and M. D. Killpack, "A new soft robot control method: Using model predictive control for a pneumatically actuated humanoid," *IEEE Robot. Automat. Mag.*, vol. 23, no. 3, pp. 75–84, Sep. 2016.

[8] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 7379–7385.

[9] R. J. Webster and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *Int. J. Robot. Res.*, vol. 29, no. 13, pp. 1661–1683, 2010.

[10] P. Schegg and C. Duriez, "Review on generic methods for mechanical modeling, simulation and control of soft robots," *PLoS One*, vol. 17, no. 1, 2022, Art. no. e0251059.

[11] F. Xu, H. Wang, K. W. S. Au, W. Chen, and Y. Miao, "Underwater dynamic modeling for a cable-driven soft robot arm," *IEEE/ASME Trans. Mechatronics*, vol. 23, no. 6, pp. 2726–2738, Dec. 2018.

[12] Z. Gong, J. Cheng, K. Hu, T. Wang, and L. Wen, "An inverse kinematics method of a soft robotic arm with three-dimensional locomotion for underwater manipulation," in *Proc. IEEE Int. Conf. Soft Robot.*, 2018, pp. 516–521.

[13] F. Renda, F. Giorgio-Serchi, F. Boyer, C. Laschi, J. Dias, and L. Seneviratne, "A unified multi-soft-body dynamic model for underwater soft robots," *Int. J. Robot. Res.*, vol. 37, no. 6, pp. 648–666, 2018.

[14] F. Xu, H. Wang, W. Chen, and J. Wang, "Adaptive visual servoing control for an underwater soft robot," *Assem. Automat.*, vol. 38, no. 5, pp. 669–677, 2018.

[15] F. Stella, N. Obayashi, C. D. Santina, and J. Hughes, "An experimental validation of the polynomial curvature model: Identification and optimal control of a soft underwater tentacle," *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, pp. 11410–11417, Oct. 2022.

[16] K. Wandke and Y. Z., "MOOSE-Based finite element hyperelastic modeling for soft robot simulations," *IEEE Access*, vol. 9, pp. 139627–139635, 2021.

[17] G. Runge, M. Wiese, L. Gunther, and A. Raatz, "A framework for the kinematic modeling of soft material robots combining finite element analysis and piecewise constant curvature kinematics," in *Proc. IEEE 3rd Int. Conf. Control Automat. Robot.*, 2017, pp. 7–14.

[18] R. K. Katzschmann et al., "Dynamically closed-loop controlled soft robotic arm using a reduced order finite element model with state observer," in *Proc. 2nd IEEE Int. Conf. Soft Robot.*, 2019, pp. 717–724.

[19] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Learning dynamic models for open loop predictive control of soft robotic manipulators," *Bioinspiration Biomimetics*, vol. 12, no. 6, 2017, Art. no. 066003.

[20] P. Hyatt and M. D. Killpack, "Real-time nonlinear model predictive control of robots using a graphics processing unit," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 1468–1475, Apr. 2020.

[21] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Trans. Robot.*, vol. 35, no. 1, pp. 124–134, Feb. 2019.

[22] M. Rolf and J. J. Steil, "Efficient exploratory learning of inverse kinematics on a bionic elephant trunk," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 6, pp. 1147–1160, Jun. 2014.

[23] C. C. Johnson, T. Quackenbush, T. Sorensen, D. Wingate, and M. D. Killpack, "Using first principles for deep learning and model-based control of soft robots," *Front. Robot. AI*, vol. 8, 2021, Art. no. 654398.

[24] G. Li, J. Shintake, and M. Hayashibe, "Deep reinforcement learning framework for underwater locomotion of soft robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 12033–12039.

[25] X. You et al., "Model-free control for soft manipulators based on reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 2909–2915.

[26] Y. Huang, M. Hofer, and R. D'Andrea, "Offset-free model predictive control: A ball catching application with a spherical soft robotic arm," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 563–570.

[27] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomiin, "Goal-driven dynamics learning via Bayesian optimization," in *Proc. Conf. Decis. Control*, 2017, pp. 5168–5173.

[28] D. Piga, M. Forgione, S. Formentin, and A. Bemporad, "Performance-oriented model learning for data-driven MPC design," *IEEE Control Syst. Lett.*, vol. 3, no. 3, pp. 577–582, Jul. 2019.

[29] M. Forgione, D. Piga, and A. Bemporad, "Efficient calibration of embedded MPC," *IFAC-Papers OnLine*, vol. 53, no. 2, pp. 5189–5194, 2020.

[30] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic LQR tuning based on Gaussian process global optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 270–277.

[31] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Optimization of the model predictive control meta-parameters through reinforcement learning," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106211.

[32] S. Cheng, M. Kim, L. Song, Z. Wu, S. Wang, and N. Hovakimyan, "DiffTune: Auto-tuning through auto-differentiation," Sep. 2022, *arXiv:2209.10021*.

[33] W. D. Null, J. Menezes, and Y. Z., "Development of a modular and submersible soft robotic arm and corresponding learned kinematics models," in *Proc. IEEE Int. Conf. Soft Robot.*, 2023, pp. 1–6.

[34] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.

[35] J. Chen, W. Zhan, and M. Tomizuka, "Constrained iterative LQR for on-road autonomous driving motion planning," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst.*, 2017, pp. 1–7.

[36] J. Chen, W. Zhan, and M. Tomizuka, "Autonomous driving motion planning with constrained iterative LQR," *IEEE Trans. Intell. Veh.*, vol. 4, no. 2, pp. 244–254, Jun. 2019.

[37] M. Lindauer et al., "SMAC3: A versatile Bayesian optimization package for hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 23, no. 1, pp. 2475–2483, 2022.

[38] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970.

[39] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1179–1191.

[40] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4RL: Datasets for deep data-driven reinforcement learning," Apr. 2020, *arXiv:2004.07219*.

[41] T. Seno and M. Imai, "d3rlpy: An offline deep reinforcement learning library," *J. Mach. Learn. Res.*, vol. 23, no. 315, pp. 1–20, Nov. 2022. [Online]. Available: http://jmlr.org/papers/v23/22-0017.html

[42] A. Centurelli, L. Arleo, A. Rizzo, S. Tolu, C. Laschi, and E. Falotico, "Closed-loop dynamic control of a soft manipulator using deep reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 4741–4748, Apr. 2022.