Open-loop model-free dynamic control of a soft manipulator for tracking tasks

Open-loop Model-free Dynamic Control of a Soft Manipulator for Tracking Tasks

Andrea Centurelli, Alessandro Rizzo, Silvia Tolu, Egidio Falotico

***Abstract*— This paper presents a novel controller for soft arms to be employed in dynamic tracking tasks. Creating dynamic controllers for continuum soft robots has been among the most important objectives for this field. This is because relying on the steady-state assumption of static controllers greatly limits the capabilities of these robotic platforms, whose advantageous compliance and flexibility is paid with dynamics that are highly non-linear and hard to model. For this reason, a data-driven solution based on long-short term memory networks is introduced. The methodology is then tested both on simulated and real continuum robots. The results show that the controller allows to accurately follow trajectories in the task-space with an average error lower than 4mm.**

I. INTRODUCTION

The hard limitations that rigid robots show in tasks that include the interaction with humans has led to the surge of the field of soft robots: this is due to their soft and lightweight bodies which generally do not pose any harm to users. Despite both traditional and data-driven control paradigms have been explored by researchers in the past years, the design of controllers for this type of manipulator is still an open topic in this fast-evolving community.

The models for these flexible robots are most commonly obtained through constant curvature approaches [1]. The evolution of the aforementioned technique, which exploits a decomposition of the manipulator's body structure in sections, is the piecewise constant-curvature models, that have been used for designing soft robot controllers [2] [3]. Yet another deeply explored and well-performing modelling method involves the usage of Cosserat rod theory [4], [5], [6].

The complexity of designing robust control systems for soft manipulators lays in the nonlinearity and mutability of their forward models; this has led to an interest in exploring the capabilities of data-driven controllers which can be categorized either as model-based or model-free. A data-driven model-based approach relying on the Koopman operator has been presented in [7] to implement model predictive control on a soft robotic arm in static conditions. The first fully model-free implementation based on neural networks used a feed-forward neural network to approximate the inverse static model of a non-redundant soft robot [8]. This was further extended in [9][10] to account for redundancies inspired by the work proposed in [11]. An example of a model-free data-driven dynamic controller is given by [12], where a sequential quadratic programming optimization is applied to generate actuators trajectories. Since this method is too computationally expensive to be applied online, this is used to generate a dataset for a feed-forward neural network which is then employed as a control policy in point-reaching tasks. A comprehensive review on the control of soft manipulators can be found here [13].

In this work we aim to provide an easily applicable method that accomplishes excellent open-loop dynamic tracking performances without the need for an explicit parametrization which is often complex to achieve for certain types of soft robots. These include, but are not limited to, elastic and pneumatically actuated continuum manipulators whose dynamics are sensitive to construction uncertainties and working conditions and are hence highly nonlinear and time-variant. Data-driven methods offer a far simpler solution which, as will be shown in this paper, attain results that are just as accurate as their model-based counterparts. For the reasons above we present a data-driven method embedded in an open-loop controller capable of trajectory tracking for soft manipulators. The controller relies on the learning of the inverse static model. This is then used as a baseline controller for trajectory tracking tasks. Data gathered from this task are used for learning the inverse dynamics model using a long-short term memory (LSTM) network. This is then used as an open-loop controller in tracking tasks. We implement this on the I-Support robot, a soft manipulator pneumatically actuated, and on a simulated soft robot. We demonstrate that, in both cases, the proposed controller can achieve precise and timely tracking of trajectories whose points span in the workspace of the two robots.

II. Experimental setup

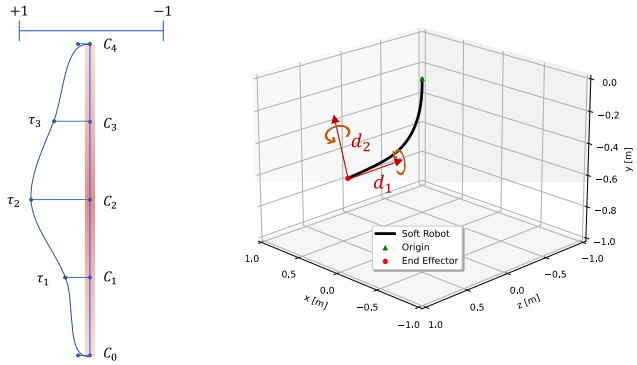
The control strategy described in this paper was tested first on a simulator called Elastica [14][15], and then on

This work was partially supported by the European Union's Horizon 2020 FET-Open program under grant agreement no. 863212 (PROBOSCIS project).

Andrea Centurelli and Egidio Falotico (corresponding author) are with the BioRobotics Institute, Scuola Superiore Sant'Anna, Pontedera, Italy and with the Department of Excellence in Robotics and AI, Scuola Superiore Sant'Anna, Pisa, Italy (email:{andrea.centurelli,egidio.falotico}@santannapisa.it).

Alessandro Rizzo is with the Department of Electronics and Telecommunications, Polytechnic of Turin, 10129 Turin, Italy (email:alessandro.rizzo@polito.it).

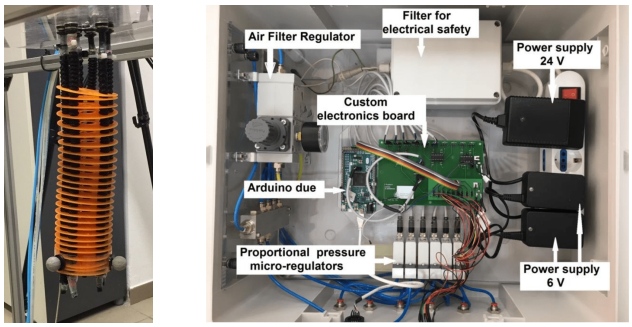
Silvia Tolu is with the Department of Electrical Engineering (Automation and Control), Technical University of Denmark, 2800 Kgs. Lyngby, Denmark (email:stolu@elektro.dtu.dk).



(a) Spline interpolation of the torque magnitudes along the rod's body

(b) Simulator plot. The normal d_1 and binormal d_2 are relative to the rod's body

Fig. 1: PyElastica simulated soft robot



(a) I-Support robot

(b) Actuation box

Fig. 2: Experimental setup

a pneumatically actuated soft robotic platform named I-Support [16].

A. Elastica

Elastica is a recently developed open-source software for the simulation of soft and slender structures. Based on Cosserat rods, it accounts for objects bend, twist, stretch, and shear. For the experiments, we used the Python version of Elastica (PyElastica), following the example given by the authors of [17] who used this library to simulate soft robots, allowing a quick interface with the simulator through an add-on library which includes an OpenAI gym environment.

The soft robot simulated is composed of a single module (a Cosserat rod), fixed to a ceiling and actuated through the application of internal torques continually distributed along the module body. These were defined by interpolating the torque magnitudes of three equispaced points on the arm through a spline, as shown in Fig. 1a. The direction of the torques is defined with the structure of the arm; in our case, two different directions were selected: the normal direction d_1, perpendicular to the body, and the binormal direction d_2 perpendicular to both d_1 and the body. This particular setting does not allow the module to twist, as there is always null torque in the orthonormal direction.

The inputs to the simulator are hence three torque magnitudes in the range $[-1, +1]$ for each direction of movement, for a total of 6 actuations, whereas its length is 1 m. A lightweight representation of the simulated arm can be seen in Fig. 1b, where the directions of the torques are shown.

B. I-Support robot

The manipulator, as seen in Fig.2a, is made of only one module, composed of three pairs of McKibben-based actuators and three cables which are alternately displaced at an angle of 60° along a circle of radius 3 cm, and kept together with a set of perforated plastic rings. The total length of the manipulator is $L_0 \approx 20$ cm while the total weight is 160 g. In this work, we only use pneumatic actuation to validate our controller. The pneumatic chambers are McKibben actuators, a type of pneumatic artificial muscles (PAMs), each composed of an external chamber, consisting of a polyester braided sheath, and an internal chamber, consisting of a latex balloon. The insertion of the internal chamber in the external one allows it to expand only longitudinally and not radially, producing an elongation of the actuator whenever pressure is applied to it. The robot is controlled through an actuation box (Fig.2b) made up of 3 proportional pressure micro-regulators which map a voltage in the range of 0-10V to a pressure in the range of 0-3 bar, and the control unit composed of an Arduino Due, a DAC, and an amplifier to manage the pressure micro-regulators.

To input the actuations to the manipulator, the Arduino is interfaced with MATLAB through serial communication. The inputs are given from MATLAB in the form of digits from 0 to 255, which are then converted to Volts (from 0-5V) to be fed to the pressure regulators that hold a range of 0-1.2bar. The setup also includes a VICON motion tracking system, composed of 8 infrared cameras that get the spatial position of the 3 markers, seen in Fig.2a, so that the end-effector position can be calculated as the centroid of the triangle that they form.

III. Methodology

The first step towards obtaining the controller is to collect a static dataset through pseudorandom motor babbling, which will then serve as training data for an artificial neural network (ANN) that approximates the inverse static (IS) model of the robot. Once the IS model is obtained, trajectories are generated by randomly sampling points in the task-space of the manipulator and then interpolating them with a spline. These trajectories in the task-space are mapped into the actuation-space using the IS network just trained.

After that, actuation values are given as input to the robot at a frequency that does not allow a steady-state assumption (10Hz) while recording the corresponding

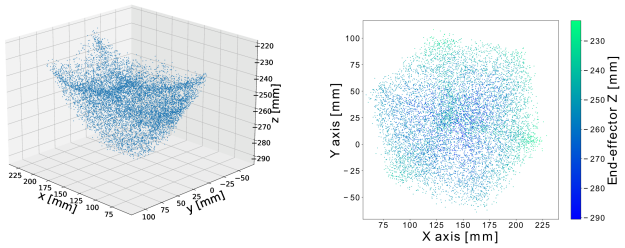
task-space positions. The trajectories outlined by the end-effector end up being significantly different from the desired ones by virtue of the non-zero velocity of the end-effector while moving from one point of the path to the next one. For this reason, the dataset collected at 10Hz is used in a second training of a neural network. The result is an open-loop dynamic controller that achieves very good precision in trajectory tracking tasks.

A. Static Dataset Collection

The collection of a training dataset is the initial step towards obtaining the final controller. To achieve a good exploration of the task space, we used pseudo-random actuations after each of which a pause of two seconds was required to allow the soft manipulator to stop oscillating; once the oscillations come to an end, the position of the end-effector is recorded with the corresponding actuation in the dataset.
The dataset is comprised of 4000 entries, which makes the process to obtain it slightly longer than two hours; scattering all the end-effector positions reached during the babbling on the I-Support produces Figure 3.

B. Inverse Static Model

The only difference in the methodologies used for the simulator and the real manipulator lies in the procedure we used to obtain the inverse statics (IS) of the platform at hand. This is due to the fact that the static model of a robot can be directly inverted only if the forward function that maps the actuation-space $\boldsymbol{\tau} \subset \mathbb{R}^n$ to the task-space $\boldsymbol{x} \subset \mathbb{R}^m$ is injective, meaning $n \leq m$. For this reason, the IS model of the I-Support is easily obtainable with a simple artificial neural network, whereas the same cannot be said for the simulated arm, which has an actuation-space of dimension 6 and a task-space of dimension 3. To solve the inversion problem for this latter case, there are a few possibilities that were explored in the literature: the first would be to include the orientation in the task-space variables, hence increasing the task-space dimension to 6 and allowing direct inversion. Another option [10] could be to invert the approximated Jacobian J that maps discrete intervals in the actuation-space to discrete intervals in the task-space $\Delta\boldsymbol{x} \approx J(\tau_0)\Delta\boldsymbol{\tau}$, which have to be small enough to make the linearization possible. We



(a) 3D (b) 2D with heatmap

Fig. 3: Task Space of the I-Support robot

opted for an approach based on reinforcement learning (RL), which eliminates the problem on a non-injective forward static model by learning the IS in an iterative way.

C. Reinforcement learning in PyElastica

The idea of using RL to learn the inverse statics of a soft manipulator has already been explored in the literature [18] [19]. In this paper, we used TRPO (Trust Region Policy Optimization [20]) a state-of-the-art algorithm that belongs to the branch of online learning. The novelty of this algorithm is the presence of a trust region that avoids updates of the policy network that might cause a fall in the gradient ascent. The optimization can be seen as the maximization of the difference between the expected sum of discounted rewards with the new policy, and the old policy, as follows:

$$\underset{\pi'}{\text{maximize}} \quad \eta\left(\pi'\right) - \eta(\pi)$$
$$\text{with} \quad \eta(\pi) - \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

limited by a constraint, the trust region, which consists in the Kullback-Leibler divergence between the new and the old policy:

$$\text{s.t.} \underset{\tau\sim d^{\pi}}{E}\left[D_{KL}\left(\pi'\|\pi\right)[s]\right] \leq \delta$$

with τ being the trajectory of the RL agent (the succession of states, actions, and rewards) sampled using the old policy, and delta is a hyperparameter of the algorithm, which decides the conservativeness of the policy update.
The RL agent is given as observation the target position $\boldsymbol{x}_{tar}$ to be reached, and produces as action an actuation which brings the end-effector in a position $\boldsymbol{x}_{ee}$. The scope of the agent is to maximize a reward that is simply the opposite of the distance between the desired target and the actual end-effector position as: $r = -\|\boldsymbol{x}_{tar} - \boldsymbol{x}_{ee}\|_2$. After 100 000 epochs of training the rewards stop increasing and the resulting inverse static network achieves an average error of 1.2 cm.

D. Feedforward neural network on I-Support

Once the dataset is collected, it becomes training data for the IS network, which is a simple artificial neural network with the three end-effector coordinates $\boldsymbol{x}_{ee} \in \mathbb{R}^3$ as inputs, and the three actuations $\boldsymbol{x}_{tar} \in \mathbb{R}^3$ as outputs. The ANN is composed of a hidden layer of 64 neurons with a hyperbolic tangent (tanh) activation function and an output layer with linear activation. Trying different network architectures by modifying hyperparameters, such as the activation functions, the number of neurons, and the gradient descent type, has proven to make a very small difference; this is to be expected considering the relative simplicity of the IS model of this manipulator.

Two strategies that increased the accuracy of the network were instead the optimization of the batch size fed

to the network during the training and the expansion and normalization of the dataset. The batch size that proved to be most effective without extending the training time excessively is 64.
Linearizing the IS model for very small segments of the task-space proved to be an acceptable simplification in case the actuation deltas between two consecutive inputs. We used $\Delta\boldsymbol{\tau} < 0.1 \cdot \max(\boldsymbol{\tau})$. After having expanded the dataset to a length of 8000, we normalized the inputs to the network (i.e. the end-effector positions) using a standard scaler.
The training was done on 80% of the dataset, while the remaining 20% was dedicated to the validation and test set. After 500 epochs of training using Adam gradient descent with a mean square error (MSE) loss function, the error on each actuation was:

$$|\tau^i - \hat{\tau}_i| \approx 1.5\% \pm 1\% \qquad \text{with } i = 1, 2, 3$$

where the percentage is over the total actuation span.

E. Controller

We generate a dataset of dynamic trajectories, that are used to train the open-loop controller. To do this, $n = 5$ points are randomly sampled from the task-space of the manipulator; these and the resting position of the end-effector are then interpolated to create the trajectories. The interpolation is performed by using a cubic spline and the resulting trajectory is saved only if it satisfies the task-space boundaries as the interpolation might fall outside. The dataset $\mathbf{x} \in \mathbf{T}_{\mathbf{x}}^{tar}$ is made of 240 trajectories, composed either of 60 or 90 points to ensure that it comprises a sufficiently large diversity of step sizes $\Delta\mathbf{x}$. Then, the IS network previously trained is used to map the trajectories from the task-space to the actuation-space, generating another dataset $\mathbf{T}_\tau$ that is used as inputs to the soft manipulator controlled at a frequency of 10Hz.
The entries in the actuations dataset $[\tau_1, \tau_2, \tau_3] \in \mathbf{T}_\tau$ are used as target values for the training of the controller network, which receive as inputs the positions obtained feeding the actuation values to the manipulator ($\mathbf{x} \in \mathbf{T}_{\mathbf{x}}^{ee}$), as the overview in Fig. 4 shows.
The controller, being open-loop, coincides with an approximation of the manipulator inverse dynamic (ID) model.
Considering that an assumption over the degree of dependency of the dynamics with the past is very dubious with this type of robot, we approached the problem of choosing the inputs to the model in an empirical way; more precisely, the key parameter to be chosen is the length of the horizon over the past end-effector positions. The inputs have the form:

$$[\mathbf{x}_{t+1}, \mathbf{x}_t, ..., \mathbf{x}_{t-T}]$$

with $\mathbf{x}_{t+1}$ being the next target position, $\mathbf{x}_t, ..., \mathbf{x}_{t-T}$ being the past end-effector positions inside the horizon length T.
We tried two different types of neural networks: an ANN like the one used for the inverse statics approximation, and an LSTM network. In fact, the choice of architecture is not obvious: LSTMs should fare better in the task at hand considering the predisposition of said networks to predict outputs belonging to a time series; at the same time, considering the limited length of the horizon, ANNs present a limited number of parameters to optimize. To compare the two architectures, we used the same number of epochs (N=1000), the same batch size (b=128), and the same activation function (*tanh*). The dataset $\mathbf{T}_{\mathbf{x}}^{ee}$ is divided in windows of the same length as the horizon length T, to produce inputs of dimensions $[\sim 170\,000, T+2, 3]$ for the LSTM, and of dimension $[\sim 170\,000, (T+2)\cdot 3]$ for the ANN. This means that the horizon length is implemented in the LSTM by returning its hidden state only at timestep $T+2$, whereas in the ANN the $T+2$ inputs are fed at once. The dataset is then split into distinct training and testing datasets, the latter including 15 trajectories that will not be seen during training.
The index used to evaluate the performance of the networks is the average of the errors $\mu(\Delta\mathbf{e})$ and their standard deviation $\sigma(\Delta\mathbf{e})$ where:

$$\Delta\mathbf{e} = \{\Delta e_t^i \ : \ \Delta e_t^i = |\hat{\tau}_t^i - \tau_t^i| \ \ i = 1, 2, 3; \\ t = 1, ..., |\text{dataset}|\}$$

The results of the model selection show that the LSTM outperforms the ANN for any horizon length T, achieving the lowest error of just above 1% of the total actuation range for T=2 (i.e input=$[\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{x}_{t-1}, \mathbf{x}_{t-2}]$).
The choice for the model to be used as a controller coincided with the best performing network.

IV. Results

The way the trained controller is deployed is shown in Figure 5, where the test trajectory is either one of the generic trajectories in the test dataset, not seen during training.

A. Results on the simulator

The trained controller achieves very low tracking errors, with an average error on the test trajectories of 3.3 mm, which is an ample improvement with respect to the errors given by the inverse statics that are almost 10 times bigger, averaging at around 3.1 cm.
Figure 6 shows the performances of the dynamic controller (left) next to the inverse static one (right) for two generic trajectories. The refinement of the execution is immediately evident when looking at the error, which is calculated as the euclidean norm between the desired and reached trajectory at each time instant.

B. Results on the I-Support

To test the dynamic controller, the LSTM network was first transferred to MATLAB, from which the soft manipulator is controlled.

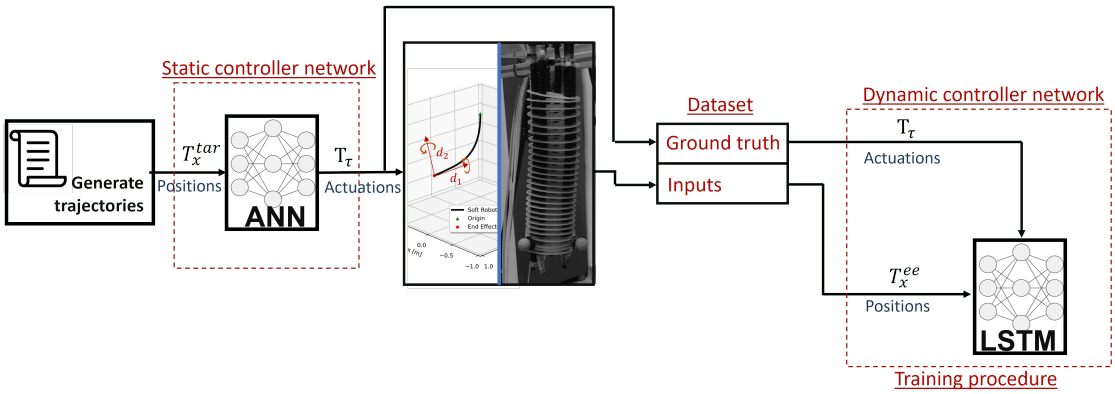


Fig. 4: Open-loop controller training overview

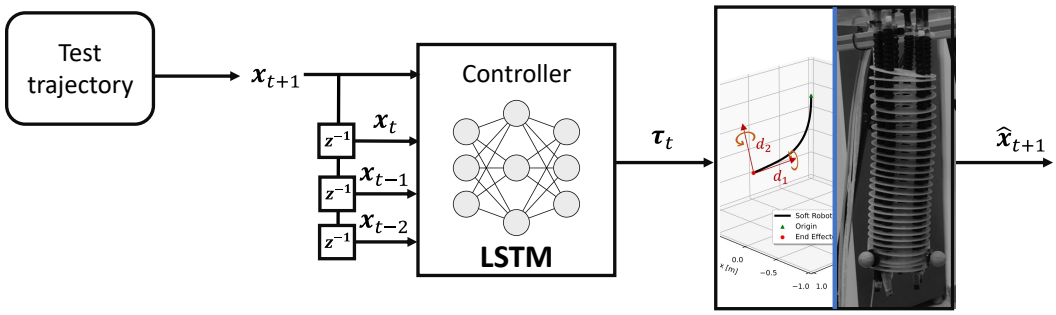


Fig. 5: Open-loop configuration of the controller when applied to the manipulator

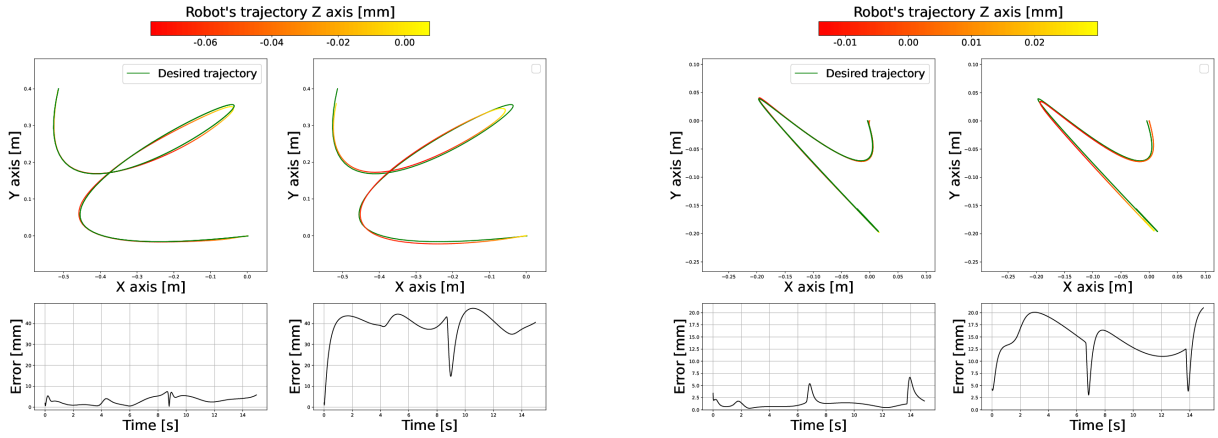


Fig. 6: Trajectories in PyElastica, with the dynamic (left) and the static controller (right)

The test on these had the IS producing an average error $\mu(\Delta\mathbf{X}_{IS}) \approx 10.6mm$, with $\Delta\mathbf{X}_{IS}$ being:

$$\Delta\mathbf{X}_{IS} = \{\Delta\mathbf{x}_t^i \ : \ \Delta\mathbf{x}_t^i = \|\hat{\mathbf{x}}_{t,IS}^i - \mathbf{x}_t^i\|_2 \quad i = 1,2,3; \\ t = 1,...,|\text{trajectory}|; \ \mathbf{x}_t^i \in \mathbf{T}_x^{tar(TEST)}\}$$

whereas the dynamic controller has an average error, calculated as above, of $\mu(\Delta\mathbf{X}_{ID}) \approx 3.2mm$.

To be more precise, the average absolute errors in three dimensions x, y, z, are shown in Table I. In Figure 7 we show four generic trajectories followed by the I-Support's end-effector and the error in the cartesian space. The average velocity for these are between 4.5cm/s and 3.5cm/s.

TABLE I: Open-loop errors on x,y,z, generic trajectory

	IS	Dynamic controller
$\mu(\Delta x)$	6.8 mm	2.6 mm
$\mu(\Delta y)$	6.5 mm	1.8 mm
$\mu(\Delta z)$	1.1 mm	1.1 mm

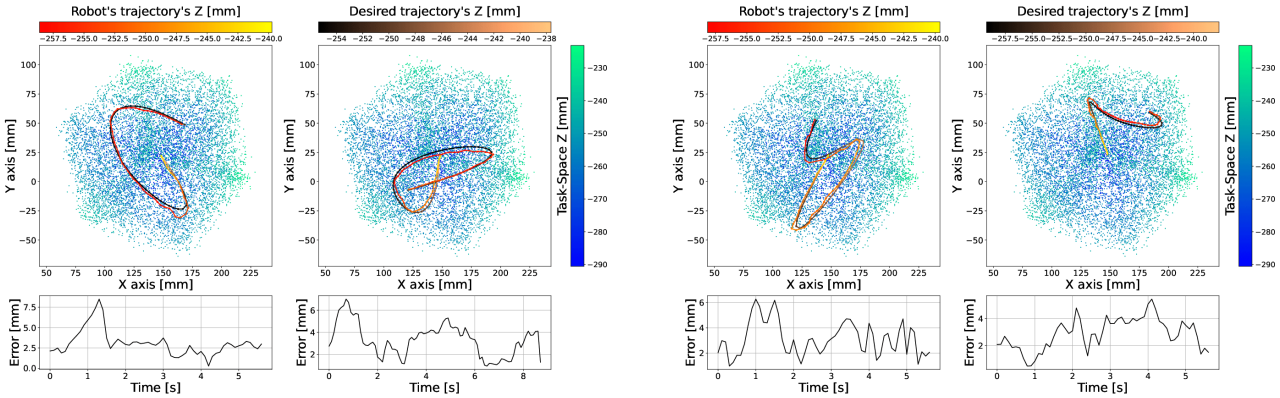


Fig. 7: Trajectory examples on the I-Support robot

V. Conclusions

The result of this paper is a controller that can track trajectories dynamically with marginal errors. The biggest advantage of the method discussed is that it does not rely on a parametrization of either the static or the dynamic model of the platform to be used, which makes it applicable to any soft robot.

Further work in the future will include the development of a closed-loop version of the controller, as well as considering orientation in complex 3D movements which involve also twist of the soft robotic arm.

References

[1] I. A. Gravagne, C. D. Rahn, and I. D. Walker, "Large deflection dynamics and control for planar continuum robots," *IEEE/ASME transactions on mechatronics*, vol. 8, no. 2, pp. 299–307, 2003.

[2] R. K. Katzschmann, C. Della Santina, Y. Toshimitsu, A. Bicchi, and D. Rus, "Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model," in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 454–461, IEEE, 2019.

[3] C. Della Santina and D. Rus, "Control oriented modeling of soft robots: the polynomial curvature case," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 290–298, 2019.

[4] S. Grazioso, G. Di Gironimo, and B. Siciliano, "A geometrically exact model for soft continuum robots: The finite element deformation space formulation," *Soft Robotics*, vol. 6, no. 6, pp. 790–811, 2019.

[5] J. Till and D. C. Rucker, "Elastic stability of cosserat rods and parallel continuum robots," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 718–733, 2017.

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

[7] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using koopman operator theory," *IEEE Transactions on Robotics*, 2020.

[8] M. Giorelli, F. Renda, G. Ferri, and C. Laschi, "A feed-forward neural network learning the inverse kinetics of a soft cable-driven manipulator moving in three-dimensional space," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5033–5039, IEEE, 2013.

[9] T. Thuruthel, E. Falotico, M. Cianchetti, F. Renda, and C. Laschi, "Learning global inverse statics solution for a redundant soft robot," in *ICINCO 2016 - 13th International Conference on Informatics in Control, Automation and Robotics, Doctoral Consortium*, vol. 2, pp. 303–310, 2016.

[10] T. George Thuruthel, E. Falotico, M. Manti, A. Pratesi, M. Cianchetti, and C. Laschi, "Learning closed loop kinematic controllers for continuum manipulators in unstructured environments," *Soft robotics*, vol. 4, no. 3, pp. 285–296, 2017.

[11] L. Vannucci, E. Falotico, N. Di Lecce, P. Dario, and C. Laschi, "Integrating feedback and predictive control in a bio-inspired model of visual pursuit implemented on a humanoid robot," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9222, pp. 256–267, 2015.

[12] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 124–134, 2018.

[13] T. George Thuruthel, Y. Ansari, E. Falotico, and C. Laschi, "Control strategies for soft robotic manipulators: A survey," *Soft robotics*, vol. 5, no. 2, pp. 149–163, 2018.

[14] M. Gazzola, L. Dudte, A. McCormick, and L. Mahadevan, "Forward and inverse problems in the mechanics of soft filaments," *Royal Society open science*, vol. 5, no. 6, p. 171628, 2018.

[15] X. Zhang, F. K. Chan, T. Parthasarathy, and M. Gazzola, "Modeling and simulation of complex dynamic musculoskeletal architectures," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[16] M. Manti, A. Pratesi, E. Falotico, M. Cianchetti, and C. Laschi, "Soft assistive robot for personal care of elderly people," in *2016 6th IEEE international conference on biomedical robotics and biomechatronics (BioRob)*, pp. 833–838, IEEE, 2016.

[17] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: A compliant mechanics environment for soft robotic control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3389–3396, 2021.

[18] Y. Ansari, E. Falotico, Y. Mollard, B. Busch, M. Cianchetti, and C. Laschi, "A multiagent reinforcement learning approach for inverse kinematics of high dimensional manipulators with precision positioning," in *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, vol. 2016-July, pp. 457–463, 2016.

[19] Y. Ansari, M. Manti, E. Falotico, M. Cianchetti, and C. Laschi, "Multiobjective optimization for stiffness and position control in a soft robot arm module," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 108–115, 2017.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.