Closed-Loop Dynamic Control of a Soft Manipulator Using Deep Reinforcement Learning

Andrea Centurelli, Luca Arleo, Alessandro Rizzo, *Senior Member, IEEE*, Silvia Tolu, *Member, IEEE*, Cecilia Laschi, *Senior Member, IEEE*, and Egidio Falotico, *Member, IEEE*

***Abstract*—The focus of the research community in the soft robotic field has been on developing innovative materials, but the design of control strategies applicable to these robotic platforms is still an open challenge. This is due to their highly nonlinear dynamics which is difficult to model and the degree of stochasticity they often incorporate. Data-driven controllers based on neural networks have recently been explored as a viable solution to be employed for these manipulators. This letter presents a neural network-based closed-loop controller, trained by a deep reinforcement learning algorithm called Trust Region Policy Optimization (TRPO). The training takes place in simulation, using an approximation of the robot forward dynamic model obtained with a Long-short Term Memory (LSTM) network. The trained controller allows following different paths executed with different velocities in the workspace of the robot. The results demonstrate that the controller is effective in normal working conditions and with a payload attached to the end-effector of the manipulator.**

***Index Terms*—Manipulator dynamics, robot control, robot learning, soft robotics.**

I. Introduction

THE development of control strategies for soft manipulators is still an open challenge since these platforms exhibit infinite degree-of-freedom and highly non-linear dynamics. [1].

The most commonly used strategy to obtain a model for these robots has been a constant curvature approach based on Bernoulli-Euler mechanics alongside an improvement of it that leverages the assumption of a piecewise constant-curvature [2]. These models have been used for the development of controllers [3], [4], [5] that started to explore the possibility to make soft manipulators interact with the environment. Effective application in real-world scenarios are limited for soft manipulators also because of deflection they exhibit under loading conditions as in the case of a payload attached to the manipulator.

Recently, as an alternative approach, data-driven techniques have been applied to address the challenge of modeling soft continuum manipulators. An early example of neural network-based control is shown in [6], where a continuous nonlinear feedback is coupled with a neural network-based feedforward to for the dynamic uncertainties of the robot. This approach is an application of the methodology described in [7], where the variables to be controlled belong to the configuration space of the manipulator. Only a few years later, a model-free static controller was developed in [8] using a feed-forward neural network to learn the inverse kinematics of a non-redundant manipulator. This approach was later improved and expanded in [9] and [10] to account for redundancies. Another notable example is presented in [11], which focuses on a linear model predictive control (MPC) for a continuum robot using a system identification based on the Koopman operator to derive the model of the robot. The same authors later improved their work in [11] by switching to a nonlinear model predictive control (NMPC) with and without linearized feedback. Another hybrid modeling approach, combining machine learning methods with an existing first-principles model, is presented in [12] to improve overall performance for a sampling-based non-linear model predictive controller. An extensive survey on the control strategies for soft manipulators is presented in [13].

Concerning reinforcement learning (RL) applications to soft robots, there are still only a few examples to cite, among which [14] where the authors developed a positional open-loop control of a pneumatically actuated continuum robot using Deep-Q-Network (DQN), achieving accurate point-to-point positioning in quasi-static conditions. The same authors later improved their work in [15] by using the same methodology to provide feedback, producing a closed-loop controller that relies on a parametric model based on Cosserat rod theory that is, once again, developed for quasi-static conditions. The preliminary work in [16] based on RL was then extended in [17] where a multi-objective optimization is applied on a soft robot with the same design principles as the ones used in this letter to achieve both position and stiffness control of the manipulator. Recently, in [18] different RL algorithms were used for the control of a simulated rod-based soft robot. This work presented a comparison of control performances for these algorithms, but

Manuscript received August 25, 2021; accepted January 3, 2022. Date of publication January 31, 2022; date of current version March 2, 2022. This letter was recommended for publication by Associate Editor M. Khoramshahi and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported by the European Union's Horizon 2020 FET-Open program under Grant 863212 through PROBOSCIS Project. *(Corresponding author: Egidio Falotico.)*

Andrea Centurelli, Luca Arleo, and Egidio Falotico are with the BioRobotics Institute, Scuola Superiore Sant'Anna, Pontedera 56025, Italy, and also with the Department of Excellence in Robotics and AI, Scuola Superiore Sant'Anna, Pisa 56127, Italy (e-mail: andrea.centurelli@santannapisa.it; luca.arleo@santannapisa.it; egidio.falotico@santannapisa.it).

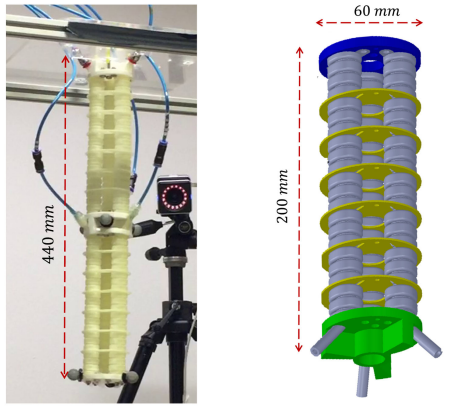
Alessandro Rizzo is with the Department of Electronics and Telecommunications, Politecnico di Torino, Turin 10129, Italy (e-mail: alessandro.rizzo@polito.it).

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

Cecilia Laschi is with the Department of Mechanical Engineering, National University of Singapore, Singapore 117575 (e-mail: mpeclc@nus.edu.sg).

the proposed approach is not applicable to a real robot due to the huge number of trials needed for the learning.

A complete overview of machine learning methods for soft robots is provided in [19]. Although the aforementioned are brilliantly thought implementations, they lack the capability to function outside the quasi-static working conditions or do not consider payloads attached to the manipulator. With respect to controllers dealing with a payload attached to the manipulator, in [20] a partially supervised methodology was implemented on a pneumatically actuated soft robot: a neural network is trained on data gathered through offline trajectory optimization. This dynamic controller is used only for point-reaching tasks. Recent approaches present also controllers for soft manipulators under variable loading conditions [21], [22], but they rely on a static model of the robot or can track low dynamics trajectories. In this work, we present, for the first time, a closed-loop dynamic controller based on RL for tracking tasks that can deal with different payloads attached to the end effector of the manipulator. The training takes place in simulation, relying on an approximation of the robot forward dynamic model obtained with a Long-short Term Memory (LSTM) network. The letter is organized as follows: in Section II we present the soft arm used in this work and the experimental setup; in Section III we present the neural network used for the approximation of the forward dynamic model; in Section IV we present the controller based on the Trust Region Policy Optimization algorithm; in Section V we show the results in tracking tasks with and without payload attached to the soft robot; we conclude in Section VI by discussing the results and presenting future research directions.



(a) Experimental setup (b) CAD of a module

Fig. 1. AM-I-Support robot. Refer to [23] for further details.

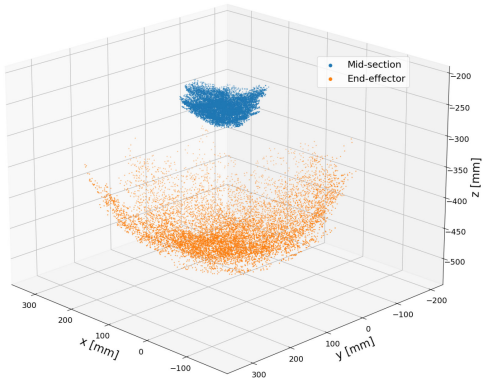


Fig. 2. AM-I-Support's task space.

II. Experimental Setup

The methodology developed was tested on a pneumatically actuated soft continuum robot. The positional coordinates needed to train the networks are taken using a VICON motion capture system with 8 infrared cameras; the position of a point is the centroid of the triangle defined by 3 markers attached to the cylindrical body of the robot. The inputs to the manipulator are given through MATLAB, which interfaces with an Arduino Due. The latter sends a digital signal that, after being converted to analog with a simple DAC, controls as many electronic pressure regulators as the robot actuators.

A. Am-I-Support

The validation platform named AM-I-Support [23], shown in Fig. 1, is given by the evolution of the previously developed I-Support [24], [25]. This manipulator is composed of two identical modules, called proximal and distal, connected to each other through nuts and bolts to allow complex movements and span a larger task space. Like in its previous version, each AM-I-Support module is actuated only through its three pneumatic chambers, even if its design allows more degrees of actuation through cables pulled by three DC motors. This robot is fabricated following a *Design for Additive Manufacturing* (DfAM) approach, with the material used to 3D-print it being:

- a thermoplastic polyurethane with Shore A hardness equal to 80 (TPU 80 A LF), for the pneumatic actuator chambers and the rings placed along the chambers,
- polylactic acid for the terminals.

This choice of materials allows large elongations and deformations thanks to its tensile module of 20–25 MPa, which ensures a maximum actuation pressure of $400\,\mathrm{KPa}$. Actuating all the three chambers of a module at once with said maximum pressure, the maximum elongation $\Delta L_{\max} \approx 47\% L_0$ is achieved, with $L_0 \approx 220\,\mathrm{mm}$ being the resting length of both modules. On the other hand, actuating only one chamber to its limit leaving the other two idle results in the module maximum bending angle of $\theta_{\max} \approx 137^\circ$ (w.r.t. the axis orthogonal to the base). These two extremes in the robot performance are only achievable while assuring the steady-state hypothesis, whereas in a dynamic task they are hardly reachable because of the delay generated by the intrinsic material properties of the manipulator and by the response of the low-level control of the pressure valves.

The task space of this manipulator (Fig. 2) is included in a cube of size $50 \times 50 \times 17\,\mathrm{cm}$. In the figure both the end-effector and mid-section task spaces are shown, respectively in orange and blue; as it is to be expected, the movements of the proximal module (at the top) are reduced due to the weights of the distal one (at the bottom), but they nevertheless contribute to the robot dexterity and its capability to achieve complex configurations.

Lastly, the manipulator has a terminal on its tip that allows it to be interfaced with a gripper. For the experiments, the terminal was instead directly interfaced with the weights for the sake of the implementation simplicity.

III. Forward Dynamic Model Approximation

A. Preliminaries

This work relies on an abstraction of the controller training from any physical parametrization of the forward model of the manipulator to be controlled. This feature allows the methodology to be applied to any soft robot, whereas almost any type of approximated physical model would make it applicable only to specific manipulators. A possible solution for the achievement of this important property can be found in Recurrent Neural Networks (RNN), such as LSTM networks [26]. LSTM networks have been proven to fare better than simpler feedforward networks in the prediction of data belonging to time series.

Considering that by virtue of the theory of manipulators dynamics the control inputs are functions of the manipulator's task space variables of their derivative:

$$\boldsymbol{\tau} = f\left(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}\right)$$

Where $\boldsymbol{\tau}$ is the actuation vector, the robot inputs, and $\mathbf{x}$ is the three-dimensional position vector. Discretizing acceleration and velocity with a time-step δt leads the control inputs to be functions of the present and past task space variables as follows:

$$\boldsymbol{\tau}_t = f\left(\mathbf{x}_{t-2}, \mathbf{x}_{t-1}, \mathbf{x}_t, \delta t\right) \tag{1}$$

Considering that the δt can be taken as constant in case a fixed control frequency is chosen, the only dependence is on the current task space variable and the ones belonging to the previous 2 timesteps.

The sampling of the dataset to train the network is done by pseudo-random motor babbling at the frequency of 10 Hz, where the next set of actuations is obtained by adding a randomly sampled $\Delta\boldsymbol{\tau}$ that spaces between $\pm 20\%$ of the total actuation range. The maximum and minimum values of $\Delta\boldsymbol{\tau}$ have to be chosen rather carefully: variations in actuation that are too high would saturate either the physical capabilities of the robot or the response of the electro-valves, further enhancing the non-linearity of the system; moreover, extreme transitions such as $|\Delta\boldsymbol{\tau}| > 15\%$ are seldom or never used in real applications. On the other hand, a network trained on deltas that are too close to 0 would make the outcome of the erratic action choices of the reinforcement learning agent impossible to predict, hence adding a degree of stochasticity to the environment that could prevent the convergence of the training. The positional coordinates sampled through a VICON tracking system are relative to both end-effector and mid-section. A total of 10 000 samples, which can be seen scattered throughout the task space in Fig. 2, are gathered in less than 16 minutes. To ensure the proper functioning also in the situation in which the manipulator is moving weights, a total of three of the aforementioned datasets are taken: one without weight, one carrying a weight of 115 g, and the last one carrying 165 g.

TABLE I
Prediction Errors of the Forward Model

Horizon length	Error IQR [mm]		
	First model	Second model	
	ANN	LSTM	ANN
1	[10.2 26.1]	[10.5 26.1]	[10.6 26.4]
2	[4.2 9.6]	[3.7 7.3]	[3.6 8.4]
3	[1.2 3.1]	[0.8 2.1]	[1.0 2.4]
4	[1.2 3.0]	[0.7 1.8]	[1.00 2.3]

B. Forward Dynamic Model

Starting from the aforementioned hypotheses, a model selection is carried out based on the horizon length T that defines how many past positional coordinates the input should encompass. The mapping that represents the forward dynamic model is hence given by:

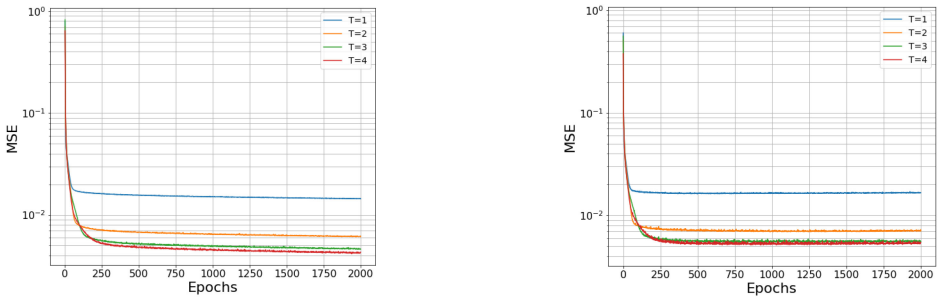
$$\left(w, \boldsymbol{\tau}_t, \mathbf{x}_t, \ldots, \mathbf{x}_{t-T}\right) \rightarrow \mathbf{x}_{t+1} \tag{2}$$

where w is the weight with which the dataset was taken. Both the task spaces and actuation spaces of the robot are initially normalized in the range $[-1, +1]$. The weight input is instead normalized between 0 and 1, with 1 coinciding with the maximum weight of $165\,\mathrm{g}$. These inputs and outputs are used to train a simple feedforward neural network whose hyperparameters were optimized by verifying the performance with different neuron numbers (32, 64, 128, or 256), number of layers (1 or 2), and activation functions (ReLu, tanh, or sigmoid). Then, another model selection on the horizon length is carried out using the fixed hyperparameters found with the aforementioned optimization: the network has two layers of 64 neurons, with layers activations being sigmoid and the output activation being linear. To prevent overfitting, the dataset is first split between training set (75%), validation set (10%), and test set (15%). Then, during training an early stop strategy with patience of 100 steps is employed alongside a learning rate (LR) step decay of 70% and patience of 50 steps to ensure the validation loss is non-increasing. The results shown in Table I justify the assumption made earlier in (1): an horizon length of $T = 1$ is not sufficient to approximate the dynamics of the system, whereas increasing it to $T = 2$ gives a much better prediction performance which is further improved with a horizon of $T = 3$, and plateaued for longer sequences.

A subsequent and wider model selection was carried out by adding to the inputs in (2) the actuation values for the past T timesteps, resulting in:

$$\left(w, \boldsymbol{\tau}_t, \ldots, \boldsymbol{\tau}_{t-T}, \mathbf{x}_t, \ldots, \mathbf{x}_{t-T}\right) \rightarrow \mathbf{x}_{t+1} \tag{3}$$

These inputs lend themselves to be used for the training of an LSTM network as the information belonging to each timestep (including the weight) can be used as input to an LSTM cell. A heuristic hyperparameter optimization is carried out for this type of network too, this time to find the most suitable number of neurons (in a range between 32 and 256), activation, and recurrent activation functions (ReLu, sigmoid, and tanh). The performance increase is significant, as can be seen from



(a) Loss

(b) Validation loss

Fig. 3. Training results without early stop. In the legend T indicates the horizon length for the position and actuation inputs. Early stop and learning rate decay prevent the training loss to further decrease while the validation loss plateaus.

the interquartile ranges (IQR) of the errors in Table I (the two numbers represent the 25$^{\text{th}}$ and the 75$^{\text{th}}$ percentiles of the error). The optimal horizon length is once again $T = 3$. For this model, adding further past information improves the prediction power of a minuscule amount. Only the errors relative to the end-effector prediction are shown, although the LSTM network also predicts the mid-section positions, with errors having a 99% confidence interval of 0.79; 0.84 mm in the best configuration. Fig. 3 shows the MSE loss (3 a) and MSE validation loss (3 b) without the aforementioned precautionary measures. The application of early stop and LR decay prevents this, hence guaranteeing uniform prediction errors through the task spaces of the manipulators.

To ensure that a recurrent neural network really provides an advantage with respect to its feedforward counterpart, the results of the best performing LSTM are compared to the ones given by the best performing feedforward neural network, which is obtained with a further hyperparameter search carried out in the same fashion described earlier in this chapter. The comparison shows that the LSTM produces estimates with errors that are 20% lower than the ones of the feedforward neural network trained with the same dataset.

IV. Learning Environment

Once the approximated dynamics of the manipulator are achieved, it is used as an environment for the training of the control policy.

A. Elements of Reinforcement Learning

The application of reinforcement learning is built upon a mathematical framework to describe the interaction between an agent and an environment; this framework is known as Markov Decision Process (MDP) and it consists of the 5-tuple

$$(S, A, P_a, R_a, \gamma)$$

which for the problem at hand is defined as follows:

- S is the *state space*, where each state s is given by a vector containing the weight w being carried, the next target position $\mathbf{x}_{t+1}^{tar} \in \mathbb{R}^3$, the current tracking error $\mathbf{e}_t = \mathbf{x}_t^{tar} - \mathbf{x}_t^{ee} \in \mathbb{R}^3$ (where the superscript *ee* stands for *end-effector*), and the current positional coordinates of the manipulator's points captured by the VICON system: both end-effector and mid-section positions $\mathbf{x}_t^{ee,mid} \in \mathbb{R}^6$. The state is hence defined as:

$$s_t = \left[w, \mathbf{x}_{t+1}^{tar}, \mathbf{e}_t, \mathbf{x}_t^{ee,mid}\right] \in \mathbb{R}^{13}.$$

- A is the *action space*; the actions a produced by the agent coincide with the six actuations given to the pneumatic chambers of the manipulator to be controlled.

$$a_t = \boldsymbol{\tau}_t \in \mathbb{R}^6 \quad 0 \leq a_t \leq 350\,\text{kPa}$$

- $P_a(s, s')$ is the probability that action a takes the agent from state s to state s'.
- $R_a(s, s')$ is the *reward function* that maps the transition from a state s to a successive state s' to the immediate expected reward r, a scalar value that quantifies how well the agent is performing the objective task. The reward function for the tracking task is simply given by the opposite of the cartesian tracking error in the 3D space:

$$r_t = -\|\mathbf{x}_t^{tar} - \mathbf{x}_t^{ee}\|_2$$

  More elaborate reward functions could be engineered to reduce the training time. The application of simple variations, such as giving additional rewards in case the end-effector stays within a radius of 10 mm and 5 mm from the target, or punishments whenever the current error becomes greater than the previous one, were tried only to verify slight to no improvements. For this reason, the simplest reward possible has been kept as the standard.
- γ is the discount factor that is responsible for how much weight is given to distant future rewards.

B. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) [27] is one of the best performing on-policy algorithms in the field of Deep Reinforcement Learning. As the name suggests, the focal point of the algorithm is the presence of a *trust region* that prevents excessively aggressive updates of the policy network parameters which, if accepted carelessly, could lead to a drop in the gradient ascent that would nullify the training progress done until the misstep. The objective function to be maximized is relative to

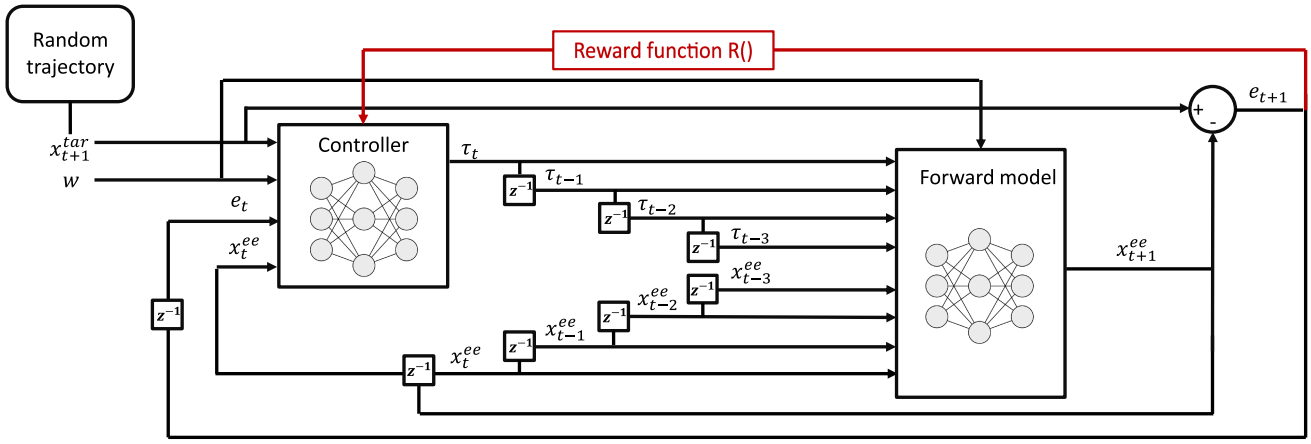


Fig. 4. Training Overview. The approximated forward model acting as an environment receives as inputs the current and past positions and actuations and predicts the next position. The controller acts as the RL agent, receiving the randomly generated target position, the weight carried, the current error, and the current position, and predicting the next optimal actuation to maximize the reward. z^{-1} represents a time delay operator.

the so-called *relative policy performance identity:*

$$\underset{\pi'}{\text{maximize}} \ \eta(\pi') - \eta(\pi) \tag{4}$$

where π is the current policy, π' is the new policy, and the operator $\eta(\cdot)$ is the expected sum of discounted rewards when the action, and hence the agent's trajectory τ, is sampled from the argument policy:

$$\eta(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$$

with γ being the discount factor. The constraint applied to the maximization in (4) is where the trust region concept comes to light:

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

where $D_{KL}(\cdot\|\cdot)$ is the Kullback-Leibler (KL) divergence between two policies, given the state s sampled from d_π, the *discounted future state distribution*.

TABLE II
TRPO HYPERPARAMETERS

Hyperparameter	Value
γ	0.99
Max KL	0.005
λ	0.98
Timesteps per batch	1024
Number of training steps	3 000 000

TABLE III
CONTROLLER NETWORK HYPERPARAMETERS

Number of hidden layers	2
Neurons per hidden layer	64
Hidden layers' activation	tanh
Output activation	linear
Total number of parameters	4995

C. Training Overview

An intuitive overview of the training process is shown in Fig. 4. As it can be seen, the environment is given by the approximated direct dynamic model of the manipulator previously obtained with the LSTM network; this predicts the next end-effector and mid-section position, given as input the weight applied to the end-effector, current action $a_t \equiv \boldsymbol{\tau}_t$ coming from the agent, alongside the previous actuations and positional coordinates shown in (3). The agent on the other hand receives the LSTM predictions as observations, as well as the tracking error, the next target position and, once again, the weight the manipulator is carrying. This next target position is taken from a randomly generated path obtained through a simple interpolation of a set of 5 randomly sampled points of the task space and the end-effector resting position. This secures that every subset of the task space is explored thoroughly; to ensure that the interpolation produces a path whose every point belongs to the task space, trajectories that have points outside of it are discarded.

Furthermore, a random normalized weight sampled uniformly in [0,1] with a resolution of 0.1 (equivalent to a weight of $\approx$ 17 g) is applied at every episode. The agent policy network, whose hyperparameters are reported in Table III, is thus trained to become the closed-loop controller to be applied to the manipulator. The controller, although receiving only positional data, is trained while performing trajectories at various velocities that are randomized by the interpolation process, in an environment that contains the dynamics of the system.

The training was carried out on a GPU (an NVIDIA GeForce GTX 1050 Ti) with the parameters shown in Table II for the gradient descents of the policy and the state-value function networks, while 8 CPUs (Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz and 16 GB of RAM) were used to parallelize as many agents on each of them. The communication overhead between the various agents and the GPU, and the inefficiency of running a recurrent neural network on a CPU are overshadowed by the advantages of parallelization. The optimized training procedure takes about 40 minutes to complete.

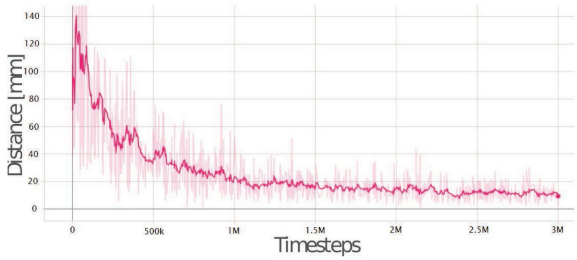


Fig. 5. Tracking error during the training.

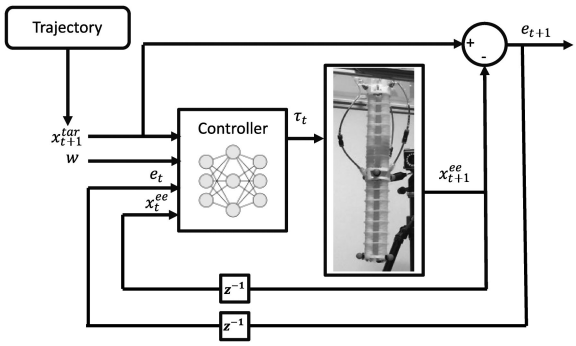


Fig. 6. Overview of the controlled system. z^{-1} represents a time delay operator.

The rolling average of the tracking error seen in example Fig. 5 plateaus at around $8.5\,\mathrm{mm}$ for the AM-I-Support robot; further training does not contribute substantially to the performance.

V. Results

The tests made can be divided into two couples: firstly we tested the generalization of the controller without weight on three benchmark trajectories; secondly, the controller was implemented on the same trajectory with different weights. Both test methodologies were first carried out in-silico, and then on the real manipulator (whose controller is shown in Fig. 6). The actuation frequency, and hence also the sampling frequency of the VICON system, is the same ($10\,\mathrm{Hz}$) as the one used to collect the datasets on which the forward model was trained.

A. Dynamic Tracking Without Weight Disturbances

Three benchmark trajectories were selected for this test section: a circumference of radius $150\,\mathrm{mm}$, a lemniscate of width $300\,\mathrm{mm}$, and a wavy circumference of radius $150\,\mathrm{mm}$, all performed twice. The choice of these three specific shapes is purely demonstrative: the test in-silico is carried out to prove that the controller is able to generalize on trajectory shapes that it did not explore during training. Importantly, the points of the circle are equidistant from each other, whereas the points of the other two benchmarks accumulate around the bends of the shapes. This provides good examples of both constant and variable velocity trajectories.

1) Test In-Silico: The training proved to generalize well both at constant and at variable velocity, with results coherent with the rewards of the training final episodes (as shown in Table IV).

TABLE IV
Errors [mm] in Simulation

	Error IQR [mm]		
	Δx	Δy	Δz
Lemniscate	[1.1 3.0]	[1.2 3.8]	[1.6 5.8]
Circumference	[1.3 3.2]	[1.1 3.4]	[2.3 6.3]
Wavy Circle	[1.2 3.8]	[0.8 3.1]	[2.7 7.9]

TABLE V
Errors [mm] on the Robot

	Error IQR [mm]		
	Δx	Δy	Δz
Lemniscate	[3.9 8.9]	[6.2 13.0]	[1.5 6.0]
Circumference	[2.4 9.0]	[3.8 8.8]	[1.9 12.6]
Wavy Circle	[3.5 11.9]	[4.2 17.3]	[2.4 8.6]

The time to complete the circumferences on the AM-I-Support is of 17 seconds whereas it takes 12 and 20 seconds to make the lemniscate and the wavy circumference, respectively. The velocities that the end-effector achieves are between 85 mm/s and 170 mm/s.

2) Test on the Manipulator: The second set of tests, the most significant ones, is carried out on the real manipulators. The trials are performed by first transferring the trained neural network from TensorFlow to MATLAB, which can easily interface with both the Arduino Due and the VICON cameras. The performance of the controller (shown in Table V and in Fig. 7) decreases only slightly when it is placed in the real environment, proving that the approximation provided by the LSTM network is precise enough to guarantee a valid environment substitute to the manipulators. The plot shows the cartesian errors during the three tests, whose averages were $12.4\,\mathrm{mm}$ for the lemniscate, $15.5\,\mathrm{mm}$ for the wavy circumference, and $11.8\,\mathrm{mm}$ for the circumference

B. Dynamic Tracking With Weight Disturbances

To test the tracking under the disturbance of weights attached from the end-effector it was necessary to choose a benchmark trajectory that fits in the reduced task space that the manipulator end-effector covers when carrying an object; in fact, when the maximum weight of $165\,\mathrm{g}$ is applied, the task space ranges for the x and y coordinate shrink of approximately 25% of the no-load scenario, whereas the range for z reduces of 50%. For this reason, the benchmark trajectory is a circumference with a radius of $10\,\mathrm{cm}$. The tests are once again run first on the approximated environment and then on the manipulator, in the no-load condition, with the two weights that the forward model is trained on ($115\,\mathrm{g}$ and $165\,\mathrm{g}$), and on two extra sets of different weights ($50\,\mathrm{g}$ and $130\,\mathrm{g}$) for validation purposes. The weights during both the trials, in-silico and on the manipulator, are known a priori by the controller. It was not in the scope of this letter to estimate the carried weight online, although future developments will include it. Having an online, reliable estimation of the weight will be fundamental for the controller to

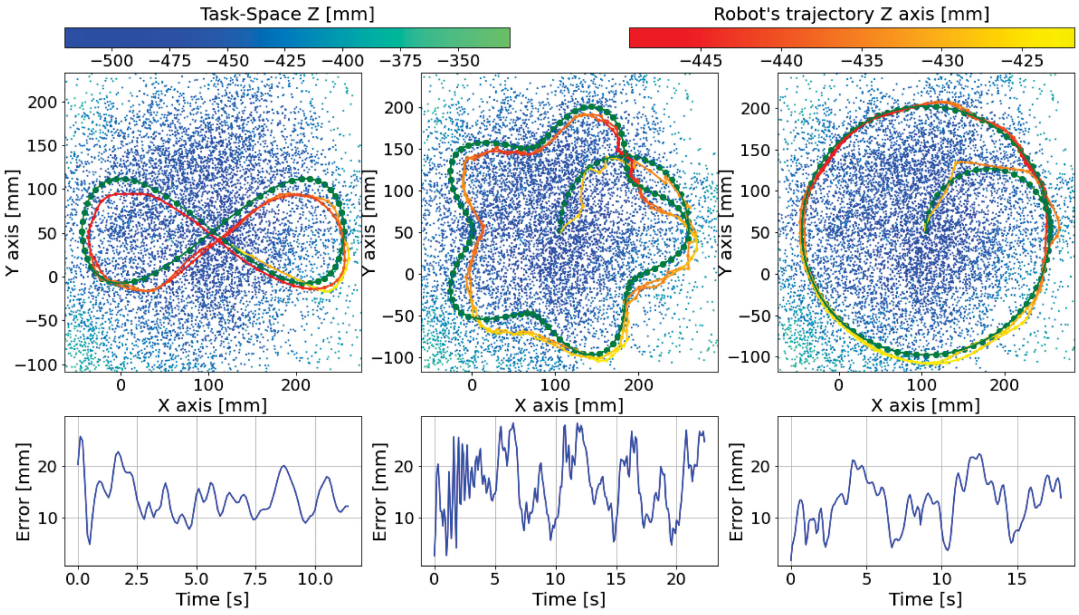


Fig. 7. Closed-loop trajectories and cartesian error on the AM-I-Support, without weights. The scattered green points indicate the target positions. The trajectory starts in the robot's resting position ($z \approx -420\,\mathrm{mm}$), and then move to a plane with $z = -440\,\mathrm{mm}$. During training, no planar trajectory was given as input, further proving the controller generalization capability.

TABLE VI
ERRORS [MM] IN SIMULATION (WITH WEIGHTS)

	Error IQR [mm]		
Weight [g]	Δx	Δy	Δz
50⋆	[2.0 7.4]	[1.1 7.4]	[1.3 4.3]
115	[1.5 7.6]	[2. 7.8]	[1.4 3.7]
130⋆	[1.3 7.3]	[2.2 7.8]	[1.9 4.8]
165	[2.4 9.6]	[3.4 11.6]	[2.3 6.9]

[0](⋆) weight not seen during training

TABLE VII
ERRORS [MM] ON THE ROBOT (WITH WEIGHTS)

	Error IQR [mm]		
Weight [g]	Δx	Δy	Δz
50⋆	[2.7 9.4]	[3.5 9.9]	[6.4 15.0]
115	[1.4 8.7]	[2.4 10.1]	[9.4 13.7]
130⋆	[2.4 12.3]	[4.4 12.6]	[11.1 13.7]
165	[3.5 10.2]	[2.6 10.8]	[8.1 11.3]

[0](⋆) weight not seen during training

avoid a drop in performance. This could be either treated from the software point of view via the implementation of online system identification techniques (possibly neural network-based), or from the hardware point of view by equipping the base of the manipulator with a scale that recognizes the extra weight added.

1) Test In-Silico: As said when explaining the training structure, the controller during training sees weights going from $0\,\mathrm{g}$ to $165\,\mathrm{g}$ with a resolution of $16.5\,\mathrm{g}$. The results in Table VI show tracking errors that are comparable with the ones obtained without weight in the previous section. The performance decreases

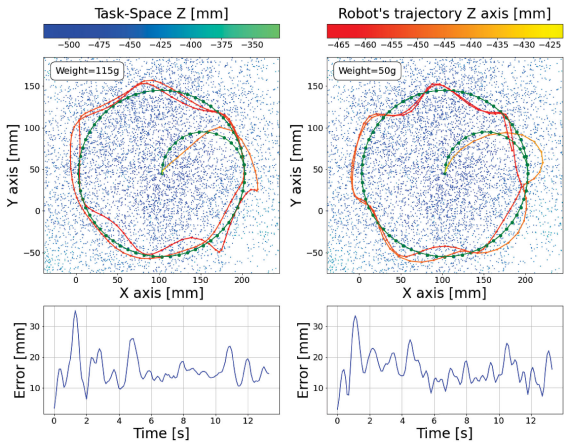


Fig. 8. Closed-loop trajectories on the AM-I-Support, with weights. The green target trajectory has an initial height that varies with the weight, and a final height of $z = -460\,\mathrm{mm}$.

slightly when increasing the weight but is still satisfactory.The fact that the errors for the weights unseen during the forward model training are in line with the others is a first proof that LSTM (i.e. the environment) generalizes well over different disturbances.

2) Test on the Manipulator: The results for the tracking on the manipulator with the weights are shown in Table VII and in Fig. 8. The average cartesian errors for the trial with $115\,\mathrm{g}$ and $50\,\mathrm{g}$ are respectively $15.2\,\mathrm{mm}$ and $15.8\,\mathrm{mm}$. Although the errors are slightly larger than for the undisturbed scenario, the controller still proves to be working well. It should be noticed that for the disturbed scenario there is a significant difference in terms of accuracy between the first and the second execution of the trial due likely to the impact of the payload on the initial movement of the robot.

VI. Conclusion

In this letter, we presented a learning-based closed-loop control strategy employable on any continuum and/or soft robot for dynamic tracking tasks. The two steps to achieve this entail two branches of artificial intelligence: firstly, supervised learning on recurrent neural networks is used to provide a simulated manipulator without the need for any parametric knowledge of its dynamics, and secondly, reinforcement learning that is based on this approximation is successfully implemented to create the control scheme. By relying on the learning of the forward dynamic model we demonstrate that it is possible to apply deep RL algorithms for the dynamic control of real soft manipulators. Furthermore, the proposed method is particularly well suited for manipulators that are either low-budget or whose assembly generates inherent variability of their physical properties. The results showed that the tracking accuracies with and without a payload are comparable, proving that the disturbance is successfully learned by the controller. It should be considered that, as mentioned, the workspace of the manipulator when an object is attached is lower. Our dynamic controller extends the capability of the one presented in [16] where only the proximal module of the robot was controlled (keeping the distal one passive). Moreover, considering other recent approaches that deal with payload attached to the soft manipulator and rely on a static model of the manipulator [22] or can track low dynamics trajectories [21], our controller guarantees faster and smoother tracking movements, keeping a comparable level of accuracy. Further development of this control scheme would go in the direction of continual learning, for example by advancing the training of the controller network directly on the robot. Moreover it would be advantageous to introduce an online estimation of the weight attached to the robot, as discussed earlier.

References

[1] C. Laschi, B. Mazzolai, and M. Cianchetti, "Soft robotics: Technologies and systems pushing the boundaries of robot abilities," *Sci. Robot.*, vol. 1, no. 1, 2016, Art. no. eaah3690.

[2] B. A. Jones and I. D. Walker, "Kinematics for multisection continuum robots," *IEEE Trans. Robot.*, vol. 22, no. 1, pp. 43–55, Feb. 2006.

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

[4] R. K. Katzschmann, C. D. 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 *Proc. 2nd IEEE Int. Conf. Soft Robot.*, 2019, pp. 454–461.

[5] C. D. Santina and D. Rus, "Control oriented modeling of soft robots: The polynomial curvature case," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 290–298, Apr. 2020.

[6] D. Braganza, D. M. Dawson, I. D. Walker, and N. Nath, "A neural network controller for continuum robots," *IEEE Trans. Robot.*, vol. 23, no. 6, pp. 1270–1277, Dec. 2007.

[7] B. Xian, D. M. Dawson, M. S. de Queiroz, and J. Chen, "A continuous asymptotic tracking control strategy for uncertain nonlinear systems," *IEEE Trans. Autom. Control*, vol. 49, no. 7, pp. 1206–1211, Jul. 2004.

[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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 5033–5039.

[9] T. Thuruthel, E. Falotico, M. Cianchetti, F. Renda, and C. Laschi, "Learning global inverse statics solution for a redundant soft robot," in *Proc. 13th Int. Conf. Informat. Control, Automat. Robot., Doctoral Consortium*, 2016, vol. 2, pp. 303–310.

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

[11] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using koopman operator theory," *IEEE Trans. Robot.*, vol. 37, no. 3, pp. 948–961, Jun. 2021.

[12] 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.

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

[14] S. Satheeshbabu, N. K. Uppalapati, G. Chowdhary, and G. Krishnan, "Open loop position control of soft continuum arm using deep reinforcement learning," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 5133–5139.

[15] S. Satheeshbabu, N. K. Uppalapati, T. Fu, and G. Krishnan, "Continuous control of a soft continuum arm using deep reinforcement learning," in *Proc. 3 rd IEEE Int. Conf. Soft Robot.*, 2020, pp. 497–503.

[16] 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 *Proc. IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatronics*, 2016, vol. 2016, pp. 457–463.

[17] 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 Robot. Automat. Lett.*, vol. 3, no. 1, pp. 108–115, Jan. 2018.

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

[19] D. Kim *et al.*, "Review of machine learning methods in soft robotics," *PloS one*, vol. 16, no. 2, 2021, Art. no. e0246102.

[20] 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.

[21] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Koopman-based control of a soft continuum manipulator under variable loading conditions," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 6852–6859, Oct. 2021.

[22] Q. Zhao, J. Lai, K. Huang, X. Hu, and H. K. Chu, "Shape estimation and control of a soft continuum robot under external payloads," *IEEE/ASME Trans. Mechatronics*, to be published, doi: 10.1109/TMECH.2021.3116970.

[23] L. Arleo, G. Stano, G. Percoco, and M. Cianchetti, "I-support soft arm for assistance tasks: A new manufacturing approach based on 3 d printing and characterization," *Prog. Additive Manuf.*, pp. 1–14, 2020.

[24] Y. Ansari, M. Manti, E. Falotico, Y. Mollard, M. Cianchetti, and C. Laschi, "Towards the development of a soft manipulator as an assistive robot for personal care of elderly people," *Int. J. Adv. Robotic Syst.*, vol. 14, no. 2, 2017, Art. no. 1729881416687132.

[25] M. Manti, A. Pratesi, E. Falotico, M. Cianchetti, and C. Laschi, "Soft assistive robot for personal care of elderly people," in *Proc. 6th IEEE Int. Conf. Biomed. Robot. Biomechatronics*, 2016, pp. 833–838.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.