POLITECNICO DI TORINO

Master's in Mechatronic Engineering

Master's Thesis

Artificial-intelligence-based control of soft robots to throw objects

Supervisors

Prof. Alessandro Rizzo

Prof. Egidio Falotico

Candidate

Costanza Comitini

July 2022

Abstract

Soft-Robotics is a new field research field, born to overcome the limitations of traditional rigid robots, soft-robots in particular can ensure a safer machine-human interaction and have the flexibility to maneuver in unpredictable and challenging circumstances. Because of their structure and the materials used in their construction, they are characterized by nonlinear behavior, thus resulting hyperedundant and underactuated. This suggests that the relationship between the task space and the actuations space is challenging to define and difficult to govern. The objective of this work was to employ artificial intelligence to train a soft robot to throw objects in the direction of a predetermined target, through the completion of a circular trajectory by the robot, in order to exploit the moment of inertia. The initial step in achieving this goal was the employment of a soft-robot simulator named Elastica. Since the launch had to be carried out through a circular trajectory of the robot, a preliminary analysis was done to determine how to actuate the robot to carry out this movement. The actuations have been defined through the use of two parameters. After, two neural networks were used to hcorrelate the actuations with the target, the first one predicts the two parameters that define the actuations required for the robot to move at a specific speed and on a specific curve, and the second predicts when to the object should be released in order to hit the target. For each of the two neural networks has been performed a model selection in order to determine the best combination of hyperparameters that allow to increase their performance. Finally, some tests have been carried out and their results reported, both for the simulator case and for the robot case through the implementation of the models obtained before. In particular, the distance between the desired target and the actual landing position has been saved, concluding that machine learning techniques can teach to a soft-robot how to throw an object.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction to Soft Robotics

Robots employed in manufacturing process are typically rigid-robots, able to compute fast, strong, precise and periodic position control tasks in assembly lines, also thanks to the materials that they are made of, rigid materials, such as alloys and metals [1]. Rigid robots have as disadvantages, the unsafe interaction between human and machine due to their often heavy structure and hard materials, the inability to work in unstructured and unpredictable environments and the absence of flexibility, for this reason the soft-robots were created. Soft robotics is a growing, new field that focuses on mechanical qualities and on the integration of materials, structures, and software, in order to compensate for shortcomings of rigid robots.

Bio-inspiration has been the key to this new category of robots, researchers looking to animal world and considering the capability to move in complex and unpredictable environments developed soft robots similar to worms and leeches [2], insect larvae [3] and molluscs as octopus and jellyfish [4]. The main differences between these two categories of robots are the materials they are made of, flexible, stretchable materials with reversible and variable properties for soft-robots hard materials with invariable properties for rigid-robots, in addition, the characteristic of the actuators, generally integrated and distributed throughout the structure for sof-robots while at least one actuator for each joint for rigid-robots [5].

The key features of soft robots are as follows:

- The capacity to bend continuously along their axis due to high deformable materials

- The capability to interact and advance in environments where the movement is limited, finding good applications in human assistance, in rescue situations and medical field
- Safety, always thank to the soft materials that also have usually lower weight respect to alloys or rigid materials, being in this way better for human interaction
- Difficult to control since characterized by numerous underactuated and compliant degrees of freedom (DoFs) and due to deformable materials.

In literature is possible to find different types of actuators for soft robots and not only one, below have been reported some of them:

- Fluidic actuation, where thanks to the increase and decrease of volume, extension, contraction, bending and twisting are obtained in the robot's movement. Can be distinguished by the type of fluid used:
  - Pneumatic actuation, where the air can be insert in the chamber to expand or extracted from the chamber to contract.
  - Hydraulic actuation, where liquid are used in substitution of air, making the response of the robot faster but at the same time increasing the weight of the robot and creating unwanted situations with liquid leaking.

  Through the inspiration of biological muscles, researchers employed pneumatic artificial muscles (PAM) such as McKibben artificial muscles composed of a flexible membrane able to contract, essentially works as follow, when the membrane is inflated, it expands radially and contracts axially, when the membrane is deflated, it reduces its section and increases its axial length. The application of McKibben PAM opened the doors to a variation numerous systems, like the Pleated PAM [6] or the use of vacuum [7]. Another approach for fluidic actuation, is to use variations of the thickness of wall chambers in order to direct the bending of the robot with the application of the same pressure. Or is possible to coil inextensible cables to restrict the radial deformation and increasing the axial one.
- Tendon-driven actuation where through the pulling and the stretching of tendons and cables the desired bending of the robot is obtained. Often the implementation

of tendon-driven actuation is mixed with other types of actuation for example,in relation with pressurized chambers, like the KSI tentacle manipulator [8], continuum robot presented by Pritts [9], the Air-Octor manipulator [10] and the OCTARM continuum robot [11]. In the Table 1.1 are reported some exaples of soft-robotic manipulators.

TABLE 1.1: SUMMARY OF THE DESIGN AND CONTROL OF SOME EXAMPLES OF SOFT ROBOTS

Manipulator	*Year Publication*	*Application*	*Desgin*	*Actuation Type*	*Actuators*	*Functionalities*	*Control*
KSI tentacle	1995	EndoscopyAgriculture Nuclear	Modular	Hybrid	Bellow-shaped penumatic bladder Radially arranged cables	Various stiffness Elongation Shortening	Model based
Artificial muscle cotniuum robot	2004	NSS	Modular	Intrinsic	Pneumatic chambers	Omnidirectional bending Shortening	-
Air-Octor	2005	Search and rescue	Modular	Hybrid	One pneumatic chamber Radially placed cables	Various stiffness Omnidirecctional bending	-
Oct Arm	2005-2008	Industrial Military	Modular	Intrinsic	Radially placed pneumatics chambers	Elongation Shortening Omnidirectional bending Elongation	Model based Hybrid
Octopus	2010-2016	Marine	Single body	Extrinsic	Cable	Elongation Bending	Model based Learning based
BHA	2011-2016	Industrial	Modular	Intrinsic	Bellow shaped pneumatic chambers	Omnidirectional bending Elongation Shortening	Learning based
Shrinkable, soft manipulator	2014	NSS	Modular	Hybrid	Latex chamber Radially place cables Braided fabric	Various stiffness Omnidirectional bending Elongation Shortening	-
STIFF-FLOP	2014-2016	Surgery	Modular	Intrinsic or Hybrid	Radially placed pneumatics chambers Ganular jamming or Radially cables	Varipus stiffness Omnidirectional benidng	-
Multisegment soft fluidic actuator	2015	NSS	Modular	Intrinsic	Anisotropic pneumatic chambers	Elongation Omnidirectional bending	Model based

- Stimuli-responsive smart material (SRM) actuation which as been proposed by the material science community offering other options to soft-robots designers. Using these types of actuations, robots with small weight are implemented. Some of them are:

  - Shape Memory Alloys (SMA)which through changes in temperature modify their structure and liberate the stored elastic energy. They find good applications in cases where compactness is necessary such as, in mobile hinges on foldable systems [12]
  - Shape Memory Polymers (SMP) [13],similarly, take advantage of thermal, light changes or chemical compounds to release energy. Respect to Shape Memory Alloys, they present lower density and stiffness, consequently, the

energy that they will release during the transitions will be limited. Through prestressed material allow to develop thermo-activated synthetic muscle fascicles, another possible approach is using also possible to use thermodynamic effects to activate thermo-active actuators.

– Electro-Active Polymers (EAP), where through the implementation of an electrical field the movement of ionic elements is created and generate forces. Since the forces generated are low and the response of the system becomes slower with the increase of the scales these types of actuation find application in small/micro scale systems. Examples of implementation are micro-manipulators [14], aquatic micro-walker [15] or actuators for vertebra [16].

1.2 State of the art of Soft Robot Controllers

Soft robots were born from the need to make up for the shortcomings or problems of rigid robots, as mentioned above, they can be very useful for different applications, situations in which a safe interaction with humans is necessary or with the achievement of impossible points and places for classic rigid robots, but to meet these needs an appropriate control of the soft robots is essential. This is not a simple goal, as the characteristics and properties of soft robots make the job even more complicated, more in detail, (1) the ability of elastic deformation through like bending, contraction or torsion, leads to a infinite degrees-of-freedom (DoF) motions while in rigid robots, the movement can be defined by three translations and three rotations; (2) materials with which soft robots are made leads to nonlinear characteristics and behavior limiting high frequency control; (3) the extensive range of design and actuation techniques, brings tho the need of different controllers having robots with unique properties. Essentially is possible to divide controllers by the model approach:

- model-based controllers based on analytical models and in necessary the knowledge of the parameters of the model.

- model-free controller where typically machine learning is used.

It is also possible to divide the controller by the assumptions that are made;

- kinematic controllers where the assumption of steady-state conditions is required.

- dynamic controllers when the above assumption is not possible.

- *Model-based kinematic controllers*, are the most used and adopted one, models can be implemented assuming g a steady-state assumption leading to a low-dimensional state space representation. If the soft robot is uniform and symmetric, external forces effects are not taken into account and torsional effects are limited, is possible to define the three-dimensional (3D) configuration space through three variable, called, constant curvature approximation (CC) [17]. More complex methods cane be implemented, such as, piecew ise constant curvature (PCC) where every single CC section is stitched together in case of multisection soft robots[18];variable constant curvature approximation (VCC) in which the curvature of the segment varies with own radius; beam theroy which is a simplification of the linear theory of elasticity and through large-deflections dynamics and axial extensibility to obtain accurate setpoint tracking [19]; Cosserat rod theory to model complex nonlinear soft robot manipulators [20]. These others methods even if are efficient are computational complex. Another disadvantage of these controllers is due to the need of feedback that a big amount of sensory data is mandatory.

- *Model-free kinematic controllers*, the advantage of this controller is that the knowledge of the parameters od the model is not necessary, while the disadvantage is that is not possible to guarantee converge of the controller since is not present a parameterized model. To learn the inverse kinematic, neural networks are implemented. Some of the proposed approaches are:

  - goal babbling [21], efficient for high redundant to elaborate samples from the task space to actuation space
  - distal supervised learning [22], to invert the learning network used for the forward kinematic model. Nonetheless, in this method the stochasticity in not taken into account end an error correction is not applied.
  - fuzzy logic controllers [23], where through local approximation and interpolation functions can be define the kinematic Jacobian.
  - local mapping [24],through which the inverse kinematic problem is solved considering it as a differential inverse kinematic problem and with which to reduce the stochastic effects.

- *Model-based dynamic controllers*, since to formulate the dynamic model is necessary the kinematic model which as already mentioned is a challenging task consequentially will lead to more uncertainty. In literature in possible to find the following methods for model-based dynamic controllers:
  - closed-loop task space dynamic controller [25], in which the kinematic has been determined through the constant curvature model and the related dynamic model defined through Euler–Lagrangian form, in which should be also considered the elastic potential energy respect to rigid robots.
  - sliding node controller [26], always only in simulation, and just for closed-loop configuration space control.
- *Model-free dynamic controllers*, which is still a new area of research. In the paper [27], has been proposed machine learning to mitigate the dynamic uncertainties, where is present feedback component, for tracking control strategy for uncertain nonlinear systems and a feedforward component implemented through neural networks. Another approach was proposed with the use of reinforcement learning, and then to reach a certain point the problem has been modeled like a Hidden Markov Model [28]. One of the most recent model-free dynamic controller was composed by a recurrent neural network (RNN) to learn the forward dynamic model and by trajectory optimization to elaborate the model that has been found [29].

1.3 Proposed approach

The objective of this thesis is to understand how to teach a soft-robot to throw objects to a specific target with taking advantage of the moment of inertia of the robot itself and through the use of machine learning. In order to create and study the model of this soft-robot, has been adopted a soft-robot simulator, called Elastica. Before different methodologies where tested on the simulator to find the feasible one and then implemented on the real soft-robot, the I-Support robot. The main difference between the simualtor and the real robot is that in the simulator the soft-manipulator is defined as an isotropic soft arm and the actuations are two torques, one oriented along the direction of the normal with respect to the robot axis and one along the binormal.

The proposed approach can be described with the following steps:

1. Create the dataset, since the proposed approach is a data-driven approach. To produce the data different throws have been simulated during the motion of the robot along the curve, and especially not for only one curve with a certain velocity of the robot, but for a set of these variables.

2. Implementation of a neural network (NN) to approximate the correlation between the actuation space and the task space, more precisely, two neural networks were used to solve the problem. The proposed model was an inverse model, in fact considering the landing coordinates of the launched object the neural network should predict the actuation set necessary to reach that target

3. Definition and the tuning of all the parameters and hyper-parameters that characterize the neural network, due to this, several neural network have been created and considered.

4. Training of all these neural networks defined by different parameters and hyper-parameters and comparison between them considering as an error the distance between the actual landing position and the desired position, ie the target, obviously choosing those with the lowest average error.

5. Throwing experiments. Considering the case of the simulator, 100 throws have been simulated with randoms targets. While in the case of the I-support 77 throws have been realized with 10 different targets.

1.4 Thesis organization

This is a brief overview of the chapters' content of this work of thesis:

Chapter 1 provides an overview of different types of soft robots developed during the time by the researchers, their disparate applications, and a wide range of methods and strategies adopted to control them. Also provides a short description of the proposed approach of this thesis and its purpose.

Chapter 2 provides an overview of machine learning fundamentals, some of the different types of neural networks and their parameters, hyper-parameters and their functions within the network.

Chapter 3 provides the description and the functioning of the simulator used, Elastica, the theory behind this simulator; the description of the soft robot used, the experimental setups, the instrumentation and methodology used to extrapolate the data from the soft robot.

Chapter 4 provides all the results obtained by simulating the launches and by throwing objects with the real robot.

Chapter 5 presents the conclusions of this work also some considerations over the proposed approach.

Chapter 2

Control models for throwing with Soft Robots

The vast majority of the formulas in this chapter come from the free Deep Learning book published in 2019 By Michael Nielsen [30]

2.1 Introduction to the controller

The aim of this work is to find a suitable controller for the soft-robot to throw an object making a curved trajectory to take advantage of the moment of inertia. In this work a model-free controller is proposed in order to generalize the solution for different soft-robots and make it feasible not only for the one used here. Has been proposed a solution with two neural networks to find the relation between actuations of the soft-robot and the landing position of the launched object, more precisely a first neural network to define the relation between the two parameters that define actuations and the landing positions and a second neural network to understand at which instant the throw should be executed in order to reach the desired target, landing position. The controller in question is a closed-loop model-free controller, since a feedback is present, and an Artificial Neural Networks (ANN) type is adopted for both networks. While can be used other types of neural networks according to different problems, for instance Recurrent Neural Network (RNN), applied for text processing, image tagger, sentiment analysis or translation. Another type is the Convolutional Neural Network (CNN), finding application in image processing, speech recognition, machine translation or computer vision. Talking again about the type

used in this work, ANNs, are famous due to the fact that are able to learn any nonlinear function, as a result, these networks are commonly referred to as Universal Function Approximators, and through their use, was possible to discover the mapping between the actuation space and the object's landing positions thanks to the first NN, while the instant of time at which to throw the object is known thanks to the second NN, both trained with the provided dataset. After this a target is chosen, the actuations necessary to reach that target are predicted by ANNs and finally those actuations are applied to the soft-robot and results are analyzed.

2.2 Neural Networks

Perceptrons are the key elements of a Neural Network, were invented in the 1950s and 1960s by Frank Rosenblatt based on earlier work by Warren McCulloch and Walter Pitts.

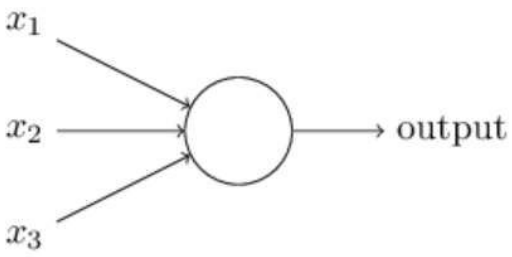


Figure 2.1: scheme of a perceptron/neuron

These elements have different binary inputs and provide a single binary output, defined as follow:

$$output = \begin{cases} 0 & if \quad \sum_j w_j x_j < threshold \\ 1 & if \quad \sum_j w_j x_j \geq threshold \end{cases} \tag{2.1}$$

essentially the output will be zero or one if the the weighted sum of all inputs $(x_1, x_2, ...,)$ is below above a threshold, the latter is then called bias, leading to the formula (2.2). Both weights and biases are parameters of the Neural Network

$$output = \begin{cases} 0 & if \quad \sum_j w_j x_j - b < 0 \\ 1 & if \quad \sum_j w_j x_j - b \geq 0 \end{cases} \tag{2.2}$$

where *weights* $(w_1, w_2, ...,)$ are indicating the relative importance of the inputs to the output and *bias* (b) is a metric that indicates how simple it is to induce the perceptron to produce a 1. Multiple perceptrons/neurons create a layer, and a Neural Network can be composed of a differnt number of those layers, can be distinguished three types of them:

- *input-layer* which is the first one of the NN and here the neurons are called input-neurons due to the fact that they receive inputs of the network
- *output-layer* which is the last layer fo the NN and contains the output neurons
- *hidden-layer* which can be more than one and is collocated between the input and the output layer.

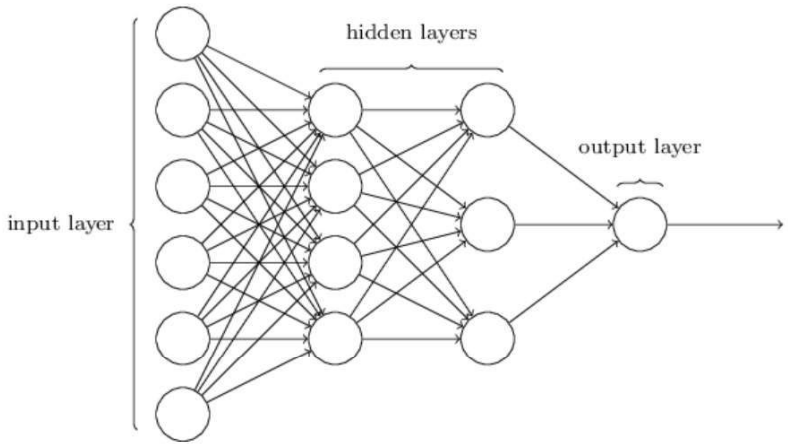


Figure 2.2: scheme of a neural network

All the neurons of a layer are connected with the next and the previous layer, taking this into account can be defined:

- a^l_j, called *activation* which is the output of the j^{th} neuron of l layer
- z^l_j, the input of the j^{th} neuron of l layer
- w^l_{jk}, the weight applied to the input of neuron j by the output of neuron k on the $(l-1)^{th}$ layer

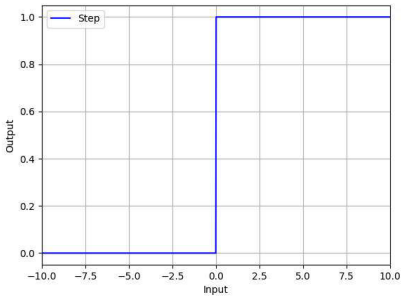
Go through different layers mean that more complex decisions are made, with this idea a multi-layer network may make sophisticated decisions. The network represented in

Figure (2.2) is also a feedforward neural network since outputs of a layer are inputs of the next one and informations are always propagated fed forward in fact there are not feedback loops. Talking now about hyperparameters, able to control the previously mentioned parameter *weight* and *bias*, one of those is the *activation function*. Activation function characterize each neuron affecting the output by a specific and selected function.

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{(l-1)} + b_j^l\right) \tag{2.3}$$

Some types of activation functions implemented for NN are listed below:

- ***Step or binary function***

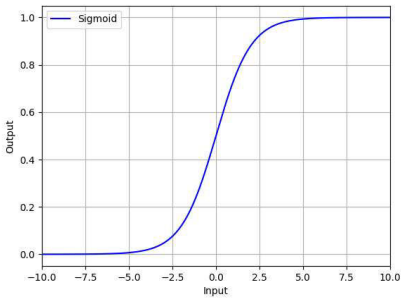


$$\sigma(x) = f(z) = \begin{cases} 0 & if \quad z < 0 \\ 1 & if \quad z \geq 0 \end{cases} \tag{2.4}$$

Figure 2.3: Step Function

The function is not differentiable in 0 and can only provide a binary output.

- ***Sigmoid function***

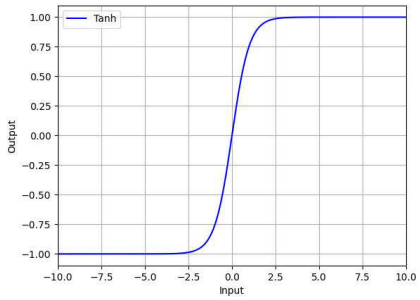


$$\sigma(x) = \sigma(z) = \frac{1}{1 - e^{-z}} \tag{2.5}$$

Figure 2.4: Sigmoid Function

Had been introduced in order to produce as output a value between 0 and 1 and solve the problem of the step function.

- ***Tangent or hyperbolic tangent function***

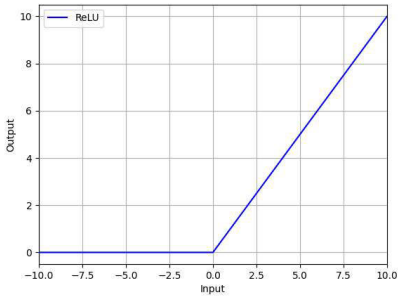


$$\sigma(x) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.6}$$

Figure 2.5: Hyperbolic Tangent Function

This almost always works better than the sigmoid because the mean of the values is 0 and not 0.5, but is not good for the output-layer due to the fact that makes more sense have a binary classification.

- ***Rectified linear unit (ReLU) function***

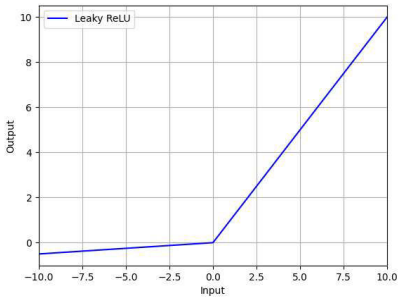


$$\sigma(x) = f(z) = \begin{cases} 0 & if \quad z = 0 \\ z & if \quad z \geq 0 \end{cases} \tag{2.7}$$

Figure 2.6: ReLU Function

This is the most used for hidden-layers, has the advantage of not having any backpropagation errors unlike the sigmoid function and its derivative is 1 for $z > 0$ and 0 for $z < 0$.

- ***Leaky ReLU function***



$$\sigma(x) = f(z) = \begin{cases} az & if \quad z = 0 \\ z & if \quad z \geq 0 \end{cases} \tag{2.8}$$

Figure 2.7: Leaky ReLU Function

An improved version of the ReLU function, instead of defining the ReLU function as 0 for negative values of x, we define it as an extremely small linear component of x. Because our neurons have a modest negative slope, rather than not firing at all for huge gradients, they actually output some value, making the layer much more optimal.

As can be noticed, all the proposed activation functions are nonlinear, in fact linear functions in the hidden layer are ineffective, can be helpful in regression problems.

2.2.1 Backpropagation and Gradient Descent Algorithm

Backpropagation is an algorithm for calculating the cost function's gradient necessary to optimize the paramenters of the NN, wieghts and biases. First version of this algorithm was introduced in 1970 by Finnish master student Seppo Linnainmaa [31] [32], however, it wasn't completely appreciated until a well-known study from 1986 by David Rumelhart, Ronald Williams and Geoffrey Hinton [33]. The aim of backpropagation is to provide information about how adjusting the weights and biases affects the network's overall behavior, to do so we are interested on the partial derivative $\partial C/\partial w$, $\partial C/\partial b$ of the cost function C respect to weight and bias of the network. Cost function determines how well a Machine Learning model performs for a given set of data, furthermore it calculates the difference between predicted and expected values and provides it as a single real number.

Some types are Mean Error (ME), Mean Squared Error (MSE), Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The one used in this work is the Mean Squared Error, described in equation (2.9)

$$C = \frac{1}{2n} \sum_{x} ||y(x) - a^{L}(x)||^2 \tag{2.9}$$

where: n is the number of training examples; L is the number of layers; $y = y(x)$ is the desired output; and $a^L = a^L(x)$ is the vector of activations output obtained with input x.

Two assumptions must be made in order to apply the backpropagation method. The first assumption is that the cost function may be expressed as an average $C = \frac{1}{2n}\sum_x C_x$ of cost functions C for individual training samples, x. The second assumption necessary regarding cost is that it can be expressed as a function of the outputs of the neural network $C = C(a^L)$. Through these considerations, the formula will the one in equation (2.10)

$$C = \frac{1}{2}||y - a^L||^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2 \tag{2.10}$$

This establishes the basis for a minimization problem: the cost function is based on the network's weights and biases, and means that the parameters should be updated along the steepest descent direction, as determined by the partial derivative of the cost function for that parameter:

$$w_j \rightarrow w'_j = w\_\alpha\frac{\partial C}{\partial w_j}$$
$$b_j \rightarrow b'_j = w\_\alpha\frac{\partial C}{\partial b_j}$$

where α is defined as the *learning rate* and controls how big steps are taken on each iteration for updating parameters w and b.

To update those parameters is necessary to compute the partial derivatives of the cost function and the backpropagation algorithm is the method to do so. Four essential equations drive backpropagation. Using all equations together, we can compute both the error δ^l and the cost function's gradient. But first the error δ^l must be defined, which is the vector of errors relating to layer l coming from the sum over all neurons of layer l of the error δ_j^l of neuron j in layer l

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{2.11}$$

This variable is introduced due to the fact that in each neuron is added a small change Δz_j^l, dealing to a neuron's output like $\sigma(z_j^l + \Delta z_j^l)$ rather than $\sigma(z_j^l)$, this modification propagates through the network changing the overall cost by a quantity of $\frac{\partial C}{\partial z_j^l}\Delta z_j^l$. Now, with this consideration, the four essential equations are introduced:

1) ***equation for the output layer's error***

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{2.12}$$

where $\frac{\partial C}{\partial a_j^L}$ quantifies how fast C changes with respect to the j^{th} output activation, while $\sigma'(z_j^L)$ quantifies how fast σ changes with respect to z_j^L. However in backpropagation is used the matrix-based form of the equation (2.12), defined in equation (2.13)

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{2.13}$$

where $\nabla_a C$ is a vector of partial derivatives $\partial C / \partial a_j^L$, which in case of quadratic cost is $\nabla_a C = (a^L - y)$, in this way equation (2.13) will become

$$\delta^L = (a^L - y) \odot \sigma'(z^L) \tag{2.14}$$

2) ***euqation for error δ^l as function of error of next layer $\delta^{(l+1)}$***

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2.15}$$

Can be computed the error δ^l for any layer in the network by combining (2.12) and (2.15), before computing δ^L from (2.12), with this can be obtained δ^{L-1} from (2.15) for then compute δ^{L-2} and so forth.

3) ***equation describing the rate of change of the cost in relation to any network bias***

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.16}$$

which thanks to (2.12) and(2.15) can be rewritten as in equation 2.17

$$\frac{\partial C}{\partial b} = \delta \tag{2.17}$$

4) ***equation describing the rate of change of the cost in relation to any network weight***

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.18}$$

which can be rewritten as in equation (2.19)

$$\frac{\partial C}{\partial w} = a_{in} \delta_{out} \tag{2.19}$$

where a_{in} represents the activation of the neuron input to the weight w and a_{out} represents the error of the neuron output from the weight w. Noticing that if a_{in} is small, $\frac{\partial C}{\partial w}$ will tend to small value, which means that weights outputs produced by low-activation neurons take a long time to learn.

Taking into account all these equations, parameters are updated as follow

$$w_j \rightarrow w'_j = w_- \frac{\alpha}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$$

$$b_j \rightarrow b'_j = w_- - \frac{\alpha}{m} \sum_x \delta^{x,l}$$

and due to computational complexity, they are not updated at every step but at every m backward passes, and this method is called *stochastic gradient descent algorithm*.

2.2.2 Normalization

In neural network's learning is rare the use of raw data, in fact most of the time, data are prepared in order to make the network optimization process easier and increase the likelihood of favorable results. This can be done through normalization. The most implemented methods are:

- *Min-Max normalisation*:

$$x' = \frac{x - min(x)}{max(x) - min(x)}(u - l) + l \tag{2.20}$$

  where x is the original data, x' is the normalized data, $min(x)$ and $max(x)$ are respectively the maximum and the minimum value of the original data, finally u and l are the upper and lower value of the range of the normalized data, especially for $u = 1$ and $l = 0$ the data is scaled between 0 and 1

- *Standardization*:

$$x' = \frac{x - \mu(x)}{\sigma(x)} \tag{2.21}$$

  where x is the original data, x' is the normalized data, $\mu(x)$ is the mean of data x and $\sigma(x)$ is the standard deviation of x.

As seen in 2.2.1 through gradient descent we're aiming for the loss function surface's lowest value, and normalization helps since will lead to a more symmetric cost function, avoiding the situation to end up in a local optima.

Input normalization is essentially important for two reasons, the first one is that in a data set with numerous inputs, we'll almost always have different scales for each feature an this condition could lead to a stronger influence in the final conclusions for some of the inputs, with an imbalance caused by their initial measurement scales rather than the

intrinsic character of the data. The second reason is that rescaling the input within limited ranges results in even smaller weight values, which makes output of the network units near the saturation areas of the activation functions less likely.

Chapter 3

Experimental Setup

3.1 Soft Robotics Simulator - Elastica

Soft robots are typically hyper-redundant and underactuated since the controller must coordinate almost limitless degrees of freedom using a limited number of actuators, a useful tool for these situations is Elastica. Elastica is a free and open-source simulation environment used to solve soft mechanism problems and has been developed by the Gazzola Lab in 2018 at the University of Illinois at Urbana-Champaign [34] [35]. Elastica serves as a good testing ground for control methods of distributed mechanics thanks to the software interface it offers, which enables the user to access well-developed control libraries and quickly construct control tasks, actuation modalities, variables and physical environments. The physics engine of Elastica uses a mechanism based on Cosserat rods, thin three-dimensional continuum elements that may bend, twist, stretch, and shear at every cross-section.

3.1.1 Cosserat Rods

In some cases the structure can be thought as a 1-dimensional rod and its mathematical treatment can be much simplified when the length of such structures is significantly greater than the radius (L/r > 1). Such 1-dimensional, slender constructions are mathematically described by the Cosserat rod theory, which takes into account the effects of bending, twisting, stretching, and shearing. This makes it possible for Cosserat rods to explain the effects of all six of the rod's degrees of freedom at each cross-section. The key

assumptions used in Cosserat rod theory are:

- $L >> r$ allowing the approximation of dynamical behavior by averaging balance laws for each cross section
- uncompressibility
- linearly elasticity

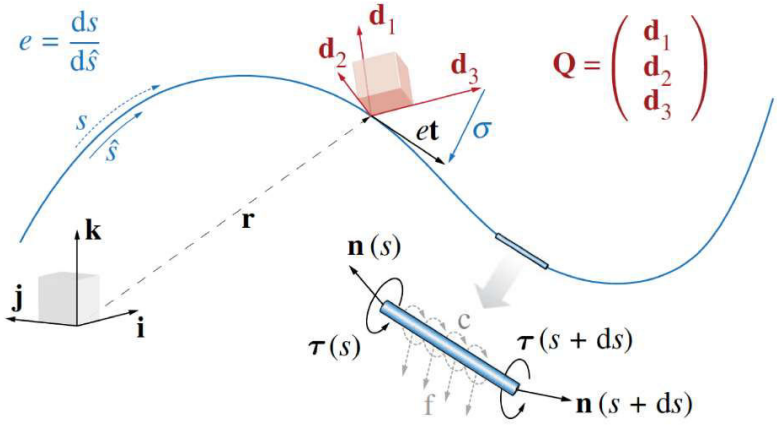


Figure 3.1: Cosserat rod - model

The conservation of linear and angular momentum throughout the rod is the equation that best describes its dynamics, but first is necessary to define some parameters:

- $r(s,t)$ which is the centerline and where $s \in [0, L]$ is the arch length of the rod and t is the time
- Laboratory frame, considering the generic vector **x**, will be $\bar{\boldsymbol{x}} = x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k}$
- Local frame, considering always generic vector will be $\boldsymbol{x} = x_1\boldsymbol{d_1} + x_2\boldsymbol{d_2} + x_3\boldsymbol{d_3}$
- $Q = \{d_1,\ d_2,\ d_3\}$ which is the rotation matrix of local and laboratory frame, then $x = Q\bar{x}$ and $\bar{x} = Q^T x$. Q changes constantly along the rod, the rod's curvature, $\boldsymbol{k}$ is represented by this change, which is described as $\partial_s d_j = \boldsymbol{k} \times \boldsymbol{d_j}$
- $\hat{s}$ which is the reference configuration
- s which is the deformed configuration

- $e = ds/d\hat{s}$ which is the stretch ratio
- $\bar{t}$ which is the unit tangent vector
- $\bar{r_s} = e\bar{t}$ which is the local orientation of the rod

Taking into account Q(s,t) and its evolution during time can be defined the translation velocity and angular velocity respectively $\bar{v} = \partial_t \bar{r}$ and $\partial_t d_j = \omega \times d_j$. Are also defined structural and material parameters:

- $\boldsymbol{A} = \frac{\hat{A}}{e}$ which is the cross-sectional area
- $\boldsymbol{B} = \frac{\hat{B}}{e^2}$ which is the bending stiffness matrix
- $\boldsymbol{S} = \frac{\hat{S}}{e}$ which is the shearing stiffness matrix
- $\boldsymbol{I} = \frac{\hat{I}}{e^2}$ which is the second area moment of inertia

Now, thanks to the clarifications made before, it's possible to express the conservation of the momentum, distinguishing two equations, one of the linear moment and the other one of the angular momentum, thus, the dynamics of the rod can be described.

Linear Momentum:

$$\rho A \cdot \partial_t^2 \bar{\boldsymbol{r}} = \partial_s \left(\frac{\boldsymbol{Q^T S \sigma}}{e} \right) + e\bar{\boldsymbol{f}} \tag{3.1}$$

Angular Momentum:

$$\frac{\rho \boldsymbol{I}}{e} \cdot \partial_t \omega = \partial_s \left(\frac{\boldsymbol{Bk}}{e^3} \right) + \frac{\boldsymbol{k} \times \boldsymbol{Bk}}{e^3} + \left(\boldsymbol{Q} \frac{\bar{\boldsymbol{r_s}}}{e} \times \boldsymbol{S\sigma} \right) + \left(\rho \boldsymbol{I} \cdot \frac{\boldsymbol{\omega}}{e} \right) \times \boldsymbol{\omega} + \frac{\rho \boldsymbol{I\omega}}{e^2} \cdot \partial_t e + ec \tag{3.2}$$

Thanks to the linear elastic assumption can be defined equations for torque curvature relations and load-strain relations, but first should be defined the bending stiffness and shearing stiffness, expressed through diagonal matrices 3x3.

Bending stiffness: $B = \begin{bmatrix} EI_1 & & \\ & EI_2 & \\ & & GI_3 \end{bmatrix}$

Shearing stiffness: $S = \begin{bmatrix} \alpha_c GA & & \\ & \alpha_c GA & \\ & & EA \end{bmatrix}$

where E is the elastic Young's modulus, G is the shear modulus I_i is the second area moment of inertia, A is the cross sectional area and α_c is equal to $4/3$ and is the constant for circular cross sections. Now can be defined the two equations:

Load strain:

$$\boldsymbol{n = S(\sigma - \sigma^0)} \tag{3.3}$$

Torque curvature:

$$\boldsymbol{\tau = B(k - k^0)} \tag{3.4}$$

where $\boldsymbol{\sigma^0}$ and $\boldsymbol{k^0}$ are reference curvatures.

Since the equations needed to simulate the dynamics and kinematics of the rods, as defined earlier, do not always have an analytical solution, numerical methods must be used. There are three steps to follow referring to the numerical method:

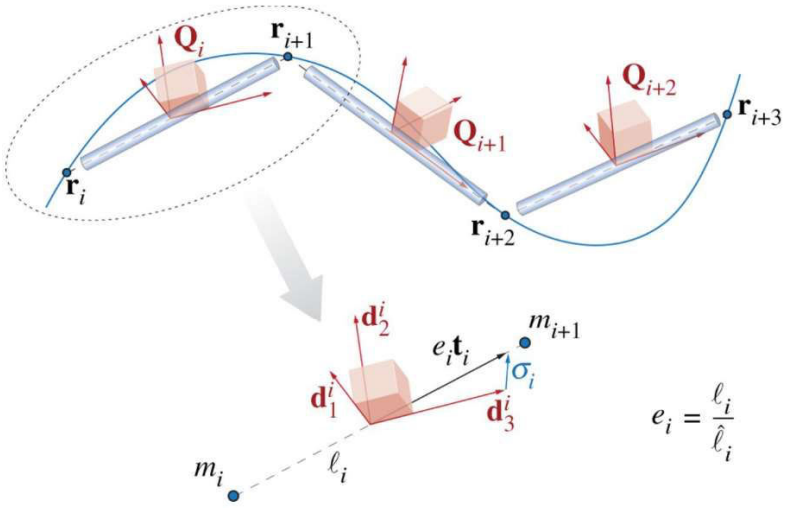


Figure 3.2: Cosserat rod - discretized model

1. *Spatial Discretization*

   First, it is required to define the various quantities related to the nodes and segments. At nodes, which are in total $(n + i)$, $i = [0, n]$:

   - $\boldsymbol{v_i} = \partial \boldsymbol{r_i}/\partial t$, velocity
   - m_i, pointwise mass
   - $\bar{\boldsymbol{f}}_{\boldsymbol{i}}$, external forces acting on the node

At every segments, which are n in total, $j = [0, n-i]$:

- $\boldsymbol{Q_j}(t)$, local reference frame, which specify how the segment is oriented
- $\boldsymbol{l_j} = \boldsymbol{r_{j+1}} - \boldsymbol{r_j}$, the edge
- $l_j = |\boldsymbol{l_j}|$, current length
- $\hat{l_j} = |\hat{\boldsymbol{l_j}}|$, reference length
- $e_j = l_j/\hat{l_j}$, stretch ratio
- $\boldsymbol{t_j} = \boldsymbol{l_j}/l_j$, unit tangent vector

Is given to each segment a set of dynamical parameters in addition to geometric ones:

- $\boldsymbol{\sigma_j} = \boldsymbol{Q_j}(e_j\bar{\boldsymbol{t_j}}) - \boldsymbol{d_{3,j}}$, shear strain vector
- ω_j, angular velocity
- $\hat{A_j}$, reference cross section area
- $\hat{\boldsymbol{J}}_{\boldsymbol{j}}$, mass second moment inertia
- $\hat{\boldsymbol{B}}_{\boldsymbol{j}}$, bend-twist matrix
- $\hat{\boldsymbol{S}}_{\boldsymbol{j}}$, shear-strain matrix
- $\boldsymbol{c_j}$, external couples acting on each segment

Should be also specified that at each interior node, an area/region is defined as $|\boldsymbol{D_i}| = (l_{i-1} + l_i)/2$, called Voronoi region, moreover, $\hat{\boldsymbol{D}}_{\boldsymbol{i}}$ is the Voronoi region at rest and $\boldsymbol{\epsilon_i = D_i/\hat{D}_i}$ is the dilation factor. Through these calrifications is possible to define curvature as $\hat{\boldsymbol{k}}_{\boldsymbol{i}} = \frac{log(\boldsymbol{Q_i Q_{i-1}^T})}{\hat{D}_i}$ and the bend-twist matrix as $\hat{\boldsymbol{B}}_{\boldsymbol{i}} = \frac{\hat{\boldsymbol{B}}_{\boldsymbol{i}} l_i + \hat{\boldsymbol{B}}_{\boldsymbol{i-1}} l_{i-1}}{2\hat{D}_i}$, both with $i = [1, n-1]$.

Finally, the equations of motion of the cosserat rod in spatially discretized form will be:

$$\partial_t \bar{\boldsymbol{r}}_{\boldsymbol{i}} = \bar{\boldsymbol{v}}_{\boldsymbol{i}} \qquad i = [0, n]$$

$$\partial_t \boldsymbol{d_{k,j}} = (\boldsymbol{Q_j^T}\boldsymbol{\omega_j}) \times \boldsymbol{d_{k,j}} \qquad j = [0, n-1], k = 1,2,3$$

$$m_i \cdot \partial_t \bar{v}_i = \Delta^h \left(\frac{Q_j^T \hat{S}_j \sigma_j}{e_j} \right) + \bar{\boldsymbol{F}} \qquad i = [0, n], j = [0, n-1]$$

$$\frac{\hat{\boldsymbol{J}}_j}{e_j}\cdot\partial_t\boldsymbol{\omega}_j = \Delta^h\left(\frac{\hat{\boldsymbol{B}}_i\hat{\boldsymbol{k}}_i}{\xi_i^3}\right)+A^h\left(\frac{\boldsymbol{k}_i\times\hat{\boldsymbol{B}}_i\hat{\boldsymbol{k}}_i}{\xi_i^3}\hat{D}_i\right)+(\boldsymbol{Q}_j\bar{\boldsymbol{t}}_j\times\hat{\boldsymbol{S}}_j\boldsymbol{\omega}_j)+\left(\hat{\boldsymbol{J}}_j\cdot\frac{\omega_j}{j}\times\boldsymbol{\omega}_j\right)+\frac{\hat{J}_{jj}}{e_j^2}\cdot\partial_t e_j+C_j$$
$$j=[0,n-1], i=[1,n-1]$$

2. *Time Discretization*

   It can be used any reliable time stepping approach to help the system evolve using a variety of various time stepping techniques. A simplistic, second-order Verlet scheme is the approach preferred.

3. *Boundary conditions and interaction forces*

   It's necessary to define initial conditions regarding position and velocity

   $\boldsymbol{r_i}(t=0)=\boldsymbol{r_i^0}$ and $\boldsymbol{v_i}(t=0)=\boldsymbol{v_i^0}$

   and also a fixed point boundary condition $\boldsymbol{r_0}(t)=\boldsymbol{r_0}$.

   The objective is to incorporate various physical effects that will enable modeling of how the system interacts with a complicated environment.

3.1.2 Elastica parameters and dataset creation

The method used to choose the most suitable soft robot models that Elastica can simulate is outlined in this section. The essential steps are:

1. Dataset creation
2. Neural networks training
3. Comparison of the trained Neural networks

The following geometrical and structural simulator parameters for the rod are taken into account:

- Young's modulus of the arm equal to $E = 1 \cdot 10^7\ Pa$
- Dissipation constant of the arm: 10
- Density: $1000\ kg/m^3$
- Poisson ratio: 0.5

- The origin of the reference system, where is attached the arm $O = \{0, 0, 0\}$
- The arm's length is 1 m and its circular section has radius equal to $0.05\,m$. Note that L»r as the assumption necessary for Cosserat rod theory is respected.
- The arm is directed toward the negative part of the z-axis
- Muscle torque scaling factor for normal/binormal directions: 7
- Number of control points: 3
- Arm's number of elements: 40
- Simulation's time step: $2.5 \cdot 10^{-4}\ s$
- Length of a simulation: 75000 timesteps

Following this methodology, data-set was generated applying different actuations to the arm. Since for each control point it is necessary to express the normal and bi-normal components of the torque, the input at the simulator is represented by an array of six elements. These torques are defined as described in equations (3.5):

$$\begin{cases} n = A \cdot \cos(\omega t) & \text{normal component} \\ b = A \cdot \sin(\omega t) & \text{binormal component} \end{cases} \tag{3.5}$$

where A and ω are the two parameters used to modify the actuations in order to explore the entire workspace of the arm. For parameter A, many functions have been examined in order to give the arm the most fluid movement possible and to improve the stabilization of the trajectory in a circle, which are shown in equation (3.6), (3.7), (3.8), (3.9) and (3.10):

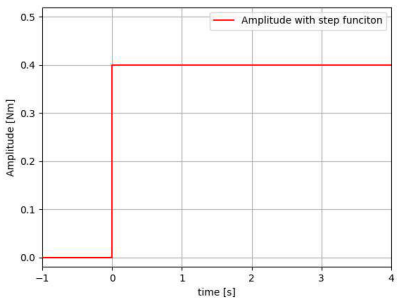


Figure 3.3: Amplitude through step function

$$A = \begin{cases} 0 & if \quad t < 0 \\ A^* & if \quad t \geqslant 0 \end{cases} \tag{3.6}$$

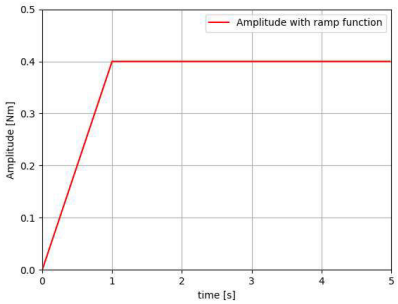


Figure 3.4: Amplitude through ramp function

$$A = \begin{cases} (k \cdot t) & if \quad t \leqslant t^* \\ A^* & if \quad t > t^* \end{cases} \tag{3.7}$$

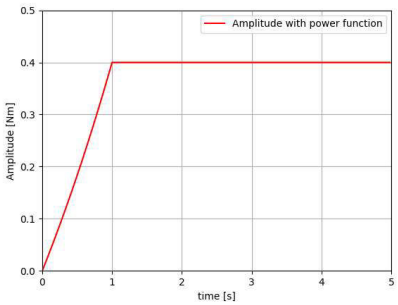


Figure 3.5: Amplitude through power function

$$A = \begin{cases} (k^t - 1) & if \quad t \leqslant t^* \\ A^* & if \quad t > t^* \end{cases} \tag{3.8}$$

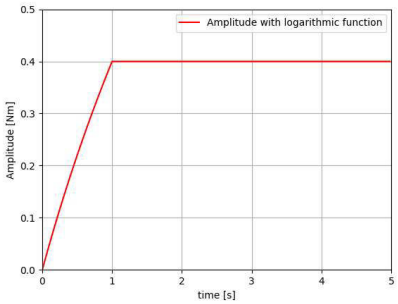


Figure 3.6: Amplitude through logarithmic function

$$A = \begin{cases} \log_{10}(k \cdot t + 1) & if \quad t \leqslant t^* \\ A^* & if \quad t > t^* \end{cases} \tag{3.9}$$

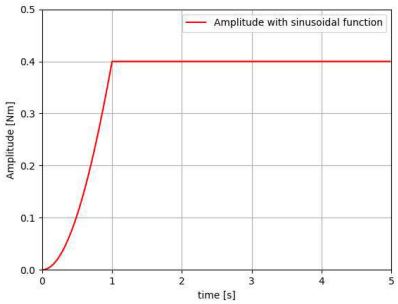


Figure 3.7: Amplitude through sinusoidal function

$$A = \begin{cases} \sin\left(kt - \frac{\pi}{2}\right) + 1 & if \quad t \leqslant t^* \\ A^* & if \quad t > t^* \end{cases} \tag{3.10}$$

where A^* is the value of amplitude at regime, t^* is the time instant for which the system is at regime, chosen equal to $2\,s$, and k is a dependent value according to the type of function used, t^* and A^*. The case in which the amplitude is obtained through a sinusoidal function is the one taken into account because it accompanies the movement better and guarantees a continuous and fluid motion avoiding abrupt movements of the robots. In order to obtain a vast workspace has been considered as value of *amplitude* a range from $0.25Nm$ to $0.5Nm$ with a step of 0.05, while as *omega* a range from $5rad/s$ to $10rad/s$ with a step of 0.2, so producing a set of 156 distinct actuations. The ball, which is connected at the end-effector, is launched to the floor that is $1.5m$ from the fixed end of the arm. The equations listed in the Table 3.1 can be used to determine where the ball will land given the end-effector position and velocity, respectively $\boldsymbol{x} = \{x_E, y_E, z_E\}$ and $\boldsymbol{v} = \{v_{xE}, y_{vE}, z_{vE}\}$

Table 3.1: Equations for landing positions

Quantity	***Equations***
Landing time [s]	$t_l = \frac{v_{zE} + \sqrt{(v_{zE})^2 - 2g(h_{floor} - z_E)}}{g}$
x-landing-coordinate [m]	$x_l = v_{xE} t_l + x_E$
y-landing-coordinate [m]	$y_l = v_{yE} t_l + y_E$
z-landing-coordinate [m]	$z_l = (v_{zE} t_l + z_E) - \frac{1}{2} g t_l^2$

As written before, 156 different actuations were implemented, and for each of these 100 points were collected, leading to a dataset of 15600 total positions, velocities and accelerations. In figure 3.10 is depicted the entire Soft-robot data set in 3D space while in figure 3.9 in the x-y plane.

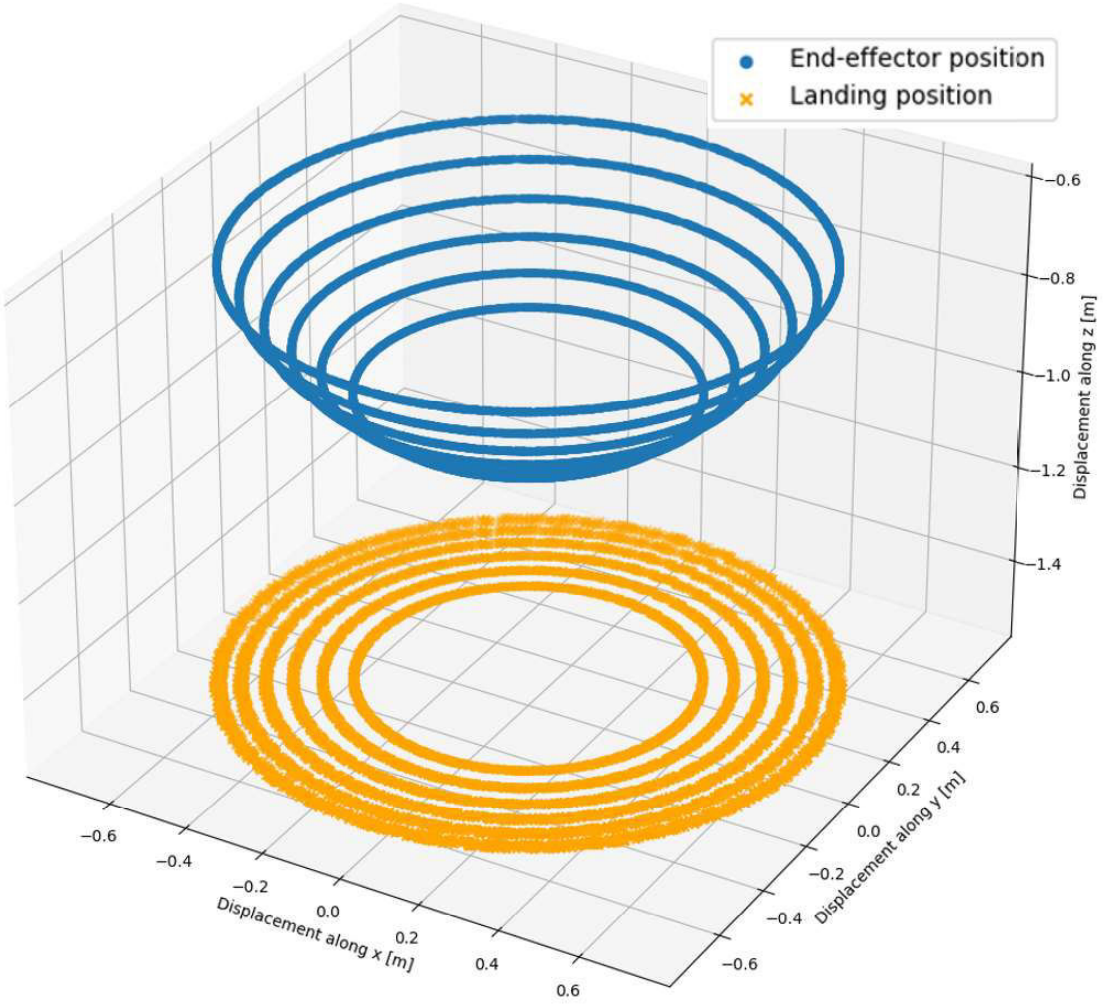


Figure 3.8: Soft-robot dataset in 3D

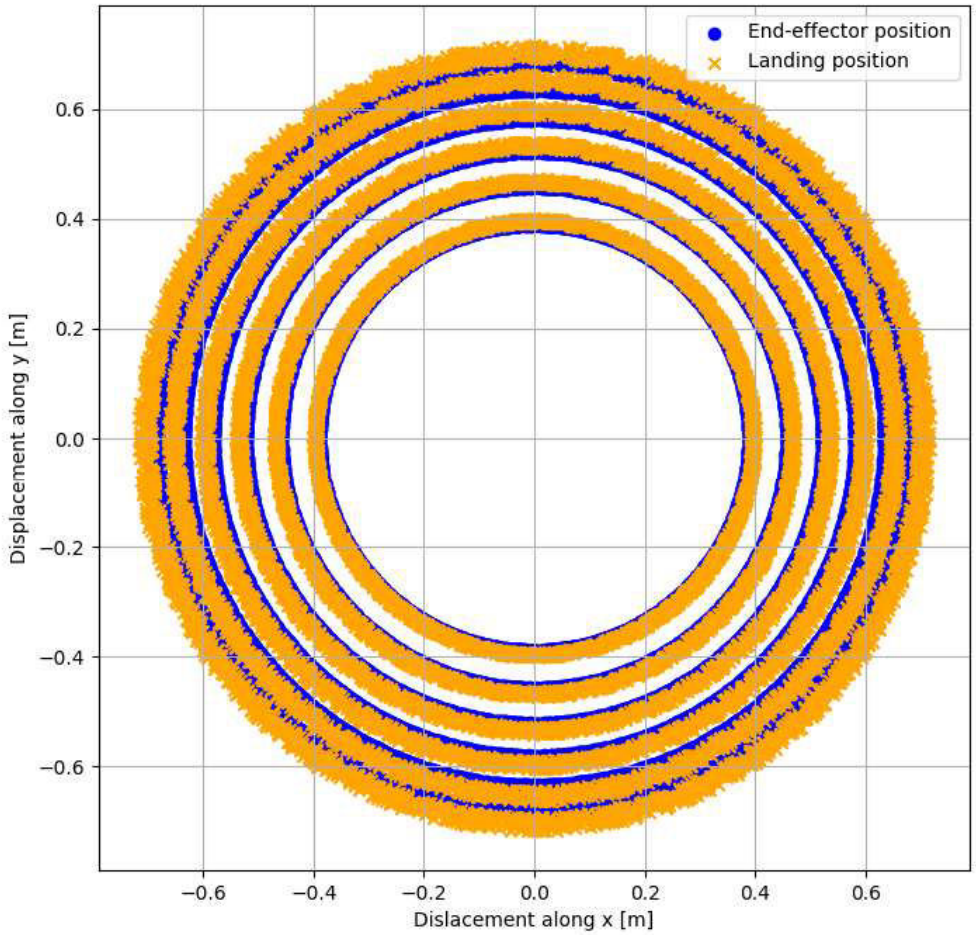


Figure 3.9: Soft-robot dataset in x-y plane

3.2 Description of the Soft Robot used

Even though only the proximal module is actuated in this work, the soft manipulator used in the experiments and depicted in Figure 3.10 is a modular robot with three couples of McKibben actuators per module. These actuators are placed at angle of 120° around its circumference and is held together by a set of perforated plastic rings. By the use of external bellow shape, the radial deformation of the internal chambers are restrained,

allowing in this way a better elongation of the module of the arm. In order to limit the weight of the arm, a robot with two different modules was used, characterized by a total length of $409 \pm 1mm$ and by a weight of $230.5 \pm 0.1g$. The soft-robot is also composed of a gripper of $27.5 \pm 0.1g$, attached to the end of the distal module, consisting of two fingers, which with an input pressure close and hold the object.

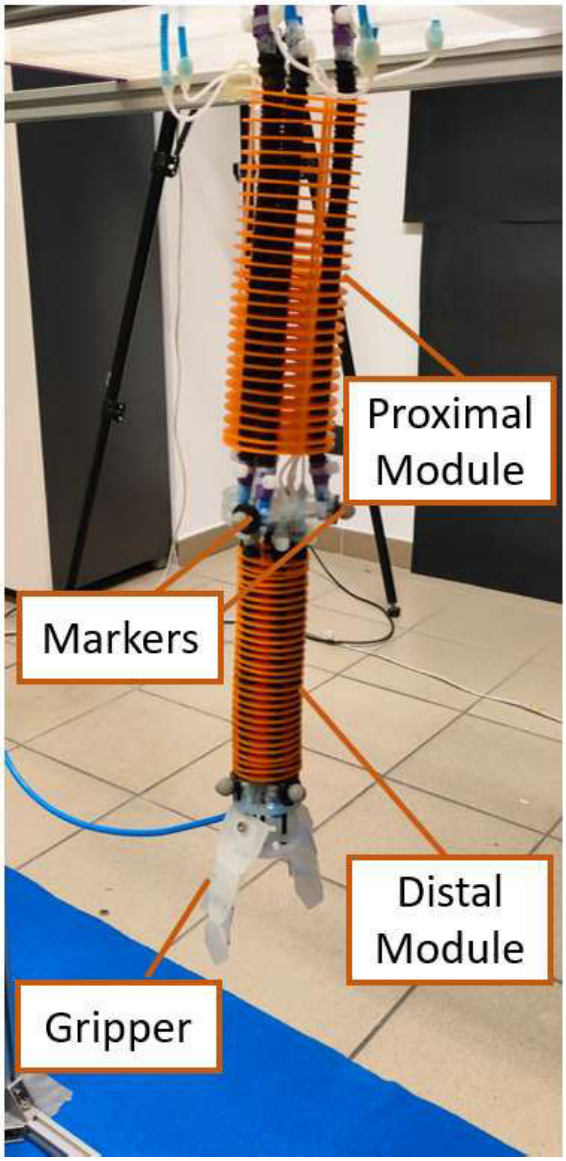


Figure 3.10: I-Support Robot

The robot is actuated through the actuation box which is supplied by three different components, as shown in figure 3.11:

- Air compressor
- Power supply
- Input data, generated by the computer thanks to MATLAB, then transmitted to the ArduinoDue, which controls the actuation box.

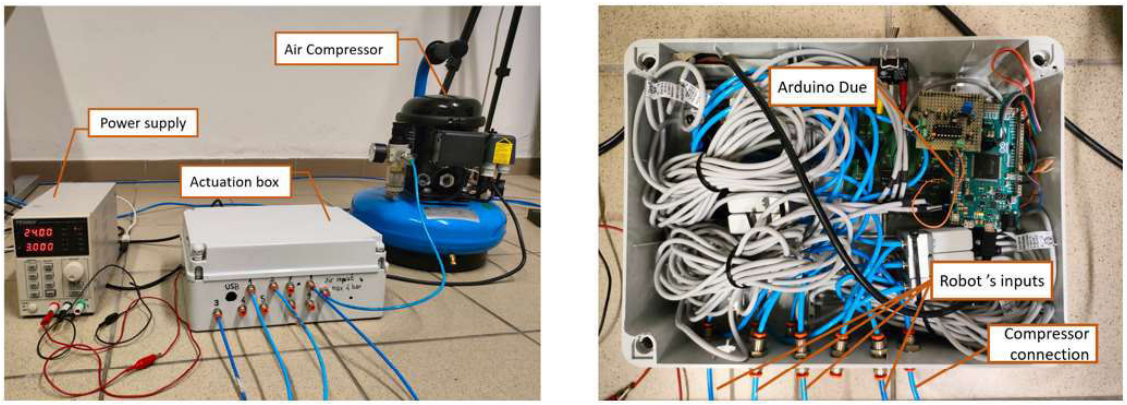


Figure 3.11: Actuation of the robot

The actuation box is composed of nine pressure valves (CAMOZZI K8P-0-E522-0), the control unit made up of an ArduinoDue, and other custom electronics. Even if there are present nine valves, only four were used, three to control McKibben actuators and one to control the gripper.

3.2.1 Gripper delay

The gripper plays a crucial role in the throwing task, in fact the timing of its opening has been studied. Since the opening of the gripper takes longer to occur in a real robot while in the simulator is immediate, it is necessary to implement this gripper delay in the neural network. The gripper was characterized employing fifty cycles of closing and opening, and the duration from the time the pressure command is sent and the moment the gripper fingers are $60mm$ (width of the thrown object) apart was timed. The results obtained are shown in figure 3.13 and figure 3.12, also a closing delay has been noted and reported, even if we are not interested in it since it has no effect on the launch because the picking of the object is not implemented and the object is placed manually inside the gripper.

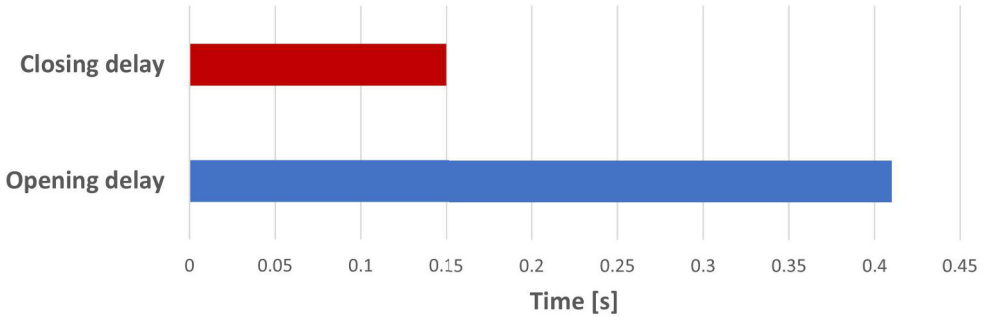


Figure 3.12: Gripper characterization

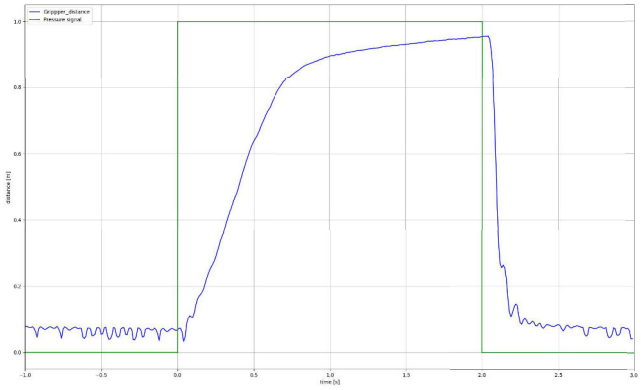


Figure 3.13: Signal and distance trend

3.3 Vicon

The Motion capture (Mocap) approach was used to reconstruct the robot's motion and continuously record its position over time, which is applied in many different fields (engineering, film-making, video game, medical etc.), in this case has been implemented through the use of Vicon system. There are numerous methods for motion capture:

- *Optical-Passive* where infrared cameras are used to track retroreflective markers. It is the industry's most widely used and adaptable method.
- *Optical-Active* where LED markers emit light that is detected by specialized cameras.

- *Inertial* where cameras are not necessary. The object wears inertial sensors, also referred to as IMUs, and the data from the sensors is wirelessly transferred to a computer.

- *Video/Markerless* where instead of using markers, this method uses software to monitor the individuals' movements.

The method used by the Vicon sytem is the optical-passive since is the most precise, adaptable, and popular type. After, the positions coordinates are obtained, is possible though process to obtain the kinematic variables of motion, trajectories, velocities and the accelerations of the robot. However there are some disadvantages in this Mocap, it is expensive in time-consuming for data-post processing, it easily confuses or does not reveal markers if they are too close to each other finally it is necessary to have a special room for this system, as any external infrared source can alter the accuracy of the cameras. In figure 3.14 is possible to see the set-up of the room, even if only partially. The system was composed of a total of 6 cameras equipped with a strobe that generates infrared light, which the markers reflect and return to the camera's lens, all the cameras are obviously pointing on the soft-robot, which is supported by a frame, the latter has the disadvantage of being able to obscure the markers in some cases creating in this way occlusions.

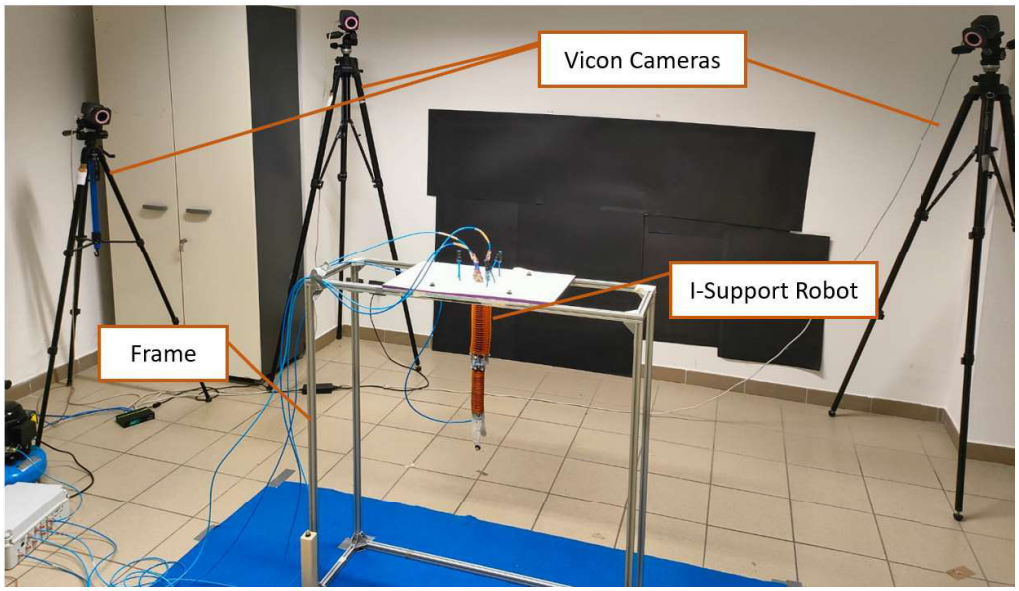


Figure 3.14: Signal and distance trend

3.4 I-Support model description

The essential steps to create the model of the I-support are:

- The creation of the dataset
- The training of the neural networks
- The comparison between trained neural networks

One of the most crucial phases is the creation of the dataset, any mistake made in this phase will affect the whole work. The goal is to create a dataset large enough to inspect the entire workspace of the robot and to be able to train the networks efficiently. For each trajectory performed by the robot, 200 points were recorded after 5 seconds from start of the movement and to obtain a dataset large enough, 117 different trajectories were analyzed. Since the trajectories are dependent by the actuations, various actuations where considered with different ranges for the parameters that characterized them. Since the chambers are placed at angle of 120° one from the other all the actuations are out of phase by one third of the period between two consecutive chambers. Three cases with different actuations were analyzed to implement a circular trajecotry:

- *case 1:*

$$Actuation = \begin{cases} A \cdot \sin(\omega t) & if \quad \sin(\omega t) \geqslant 0 \\ 0 & if \quad \sin(\omega t) < 0 \end{cases}$$

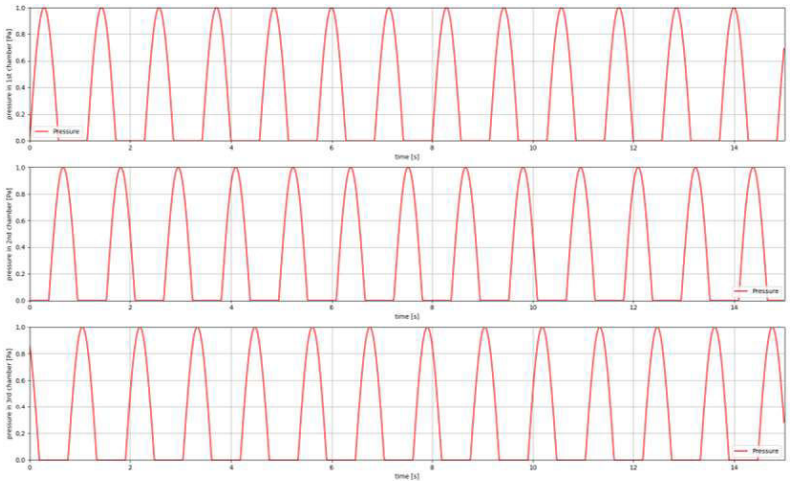


Figure 3.15: Pressure trend in each chamber in case 1

- *case 2:*

  $$Actuation = |A \cdot \sin(\omega t)|$$

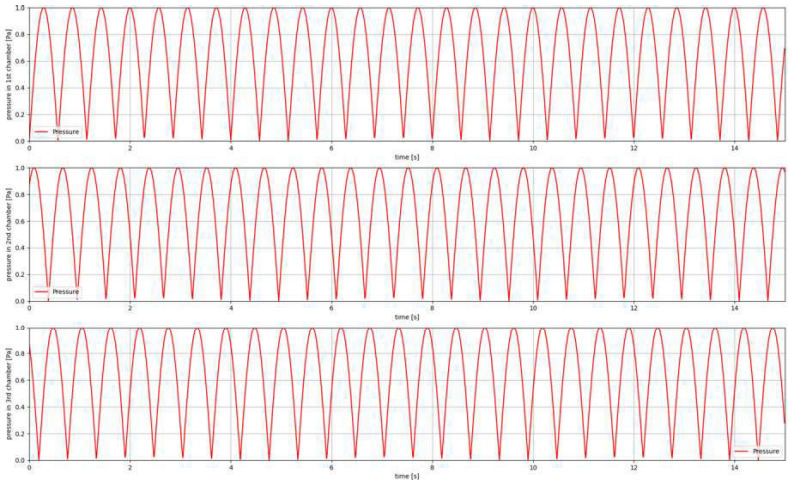


Figure 3.16: Pressure trend in each chamber in case 2

- *case 3:*

  $$Actuation = A \cdot \sin(\omega t) + A$$

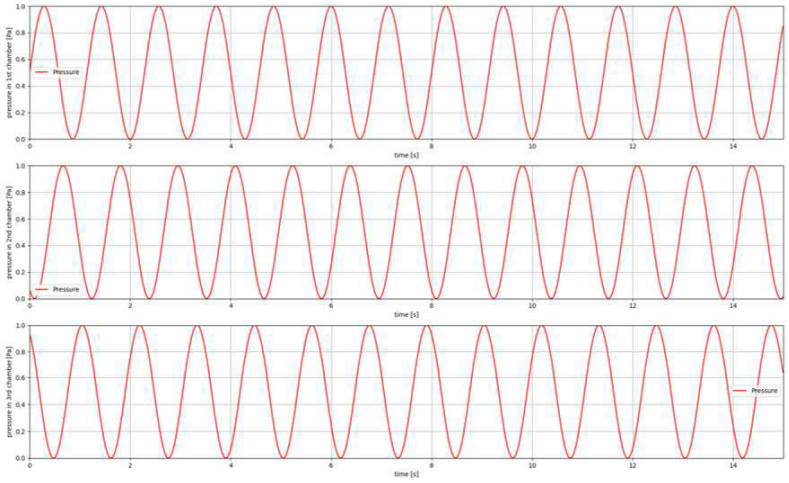


Figure 3.17: Pressure trend in each chamber in case 3

The actuations considered the best and therefore implemented in this work are those of case 3, since the pressure differential between the chambers in the first two cases was too small to permit the robot to move widely and in a circular trajectory, which can be seen in

Figures 3.18 and 3.19. As in the case of the simulator, the dataset has been obtained by considering several values for the two parameters A (amplitude) and ω (omega), which describe the actuations, in particular, for the amplitude, a range from 0.1 to 0.5 with steps of 0.05 was considered, while for omega, a range from 4 to 7 with stages of 0.25 was considered. In this way has been created a data-set of 23400 different end-effector positions and velocities, however, since the objective of this work is to teach a robot how to throw the relationships shown in table 3.1 in the previous chapter 3.1.2 are used in order to save landing coordinates and the actuation set used to reach it, obtaining in this way a dataset of landing positions, in figure 3.20 is reported the graph of positions of the end-effector during the trajectory and the positions that the object reaches if launched in those same points, but only for one actuation, in which $A = 0.4bar$ and $\omega = 5.25rad/s$.

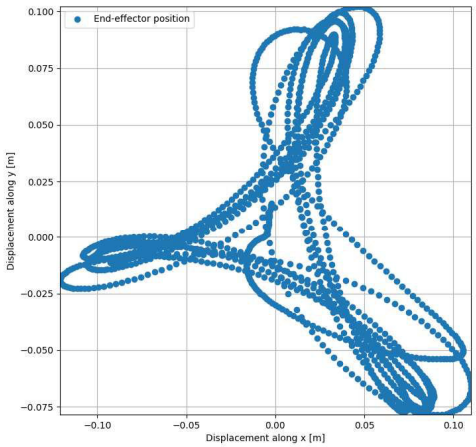


Figure 3.18: End-effector positions in case 1

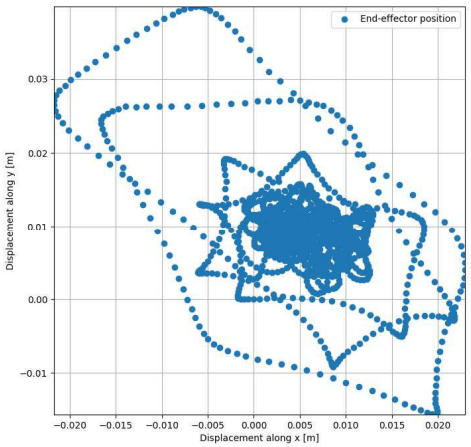


Figure 3.19: End-effector positions in case 2

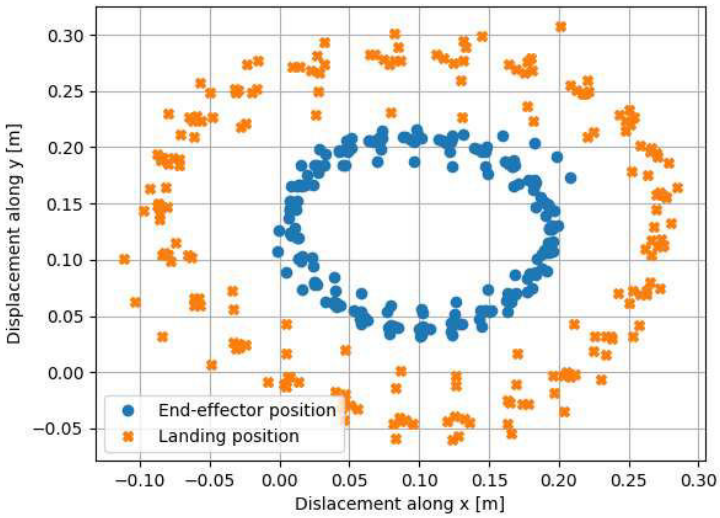


Figure 3.20: Landing positions for $A = 0.4bar$ and $\omega = 5.25rad/s$

In figures 3.21 and 3.22 are reported positions of the end-effector and of landing positions for all different actuations, namely, the entire data-set.

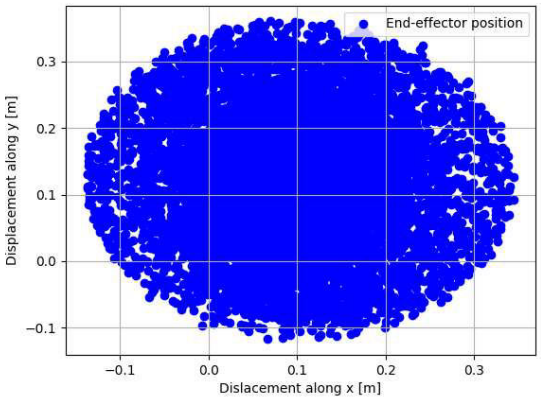


Figure 3.21: End-effector positions

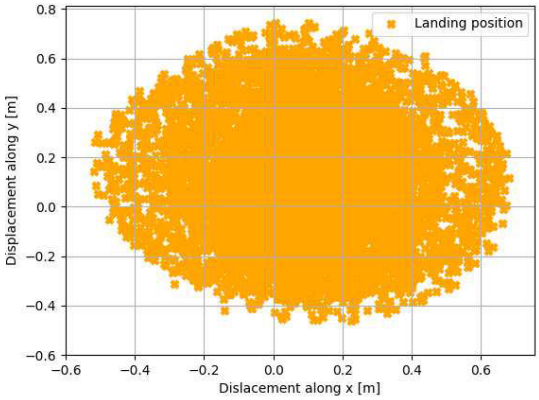


Figure 3.22: Landing positions

Finally to better notice the expansion of the workspace of the soft-robot is reported a graphic with both landing positions and end-effector positions, in figure 3.23

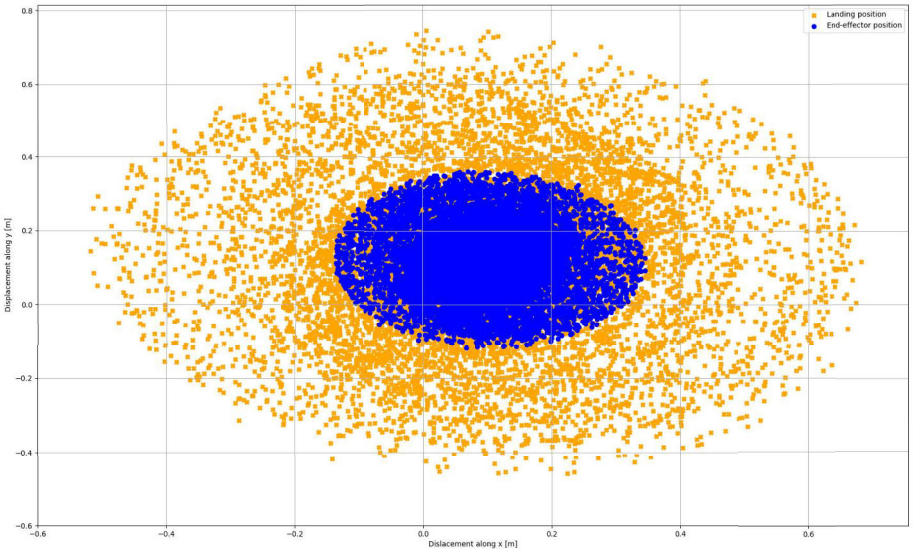


Figure 3.23: entire workspace of I-support on x-y plane

Chapter 4

Results

This chapter shows the models used and the results obtained. Specifically, paragraph 4.1 shows the models implemented and analyzed in the case of the simulator and in the case of the I-support, while in paragraph 4.2 the results of the model realized through the use of the Simulator are reported, and finally in paragraph 4.3 the results of the model implemented with the I-support and the launches carried out.

4.1 Generation of Models

Starting from the datasets generated and described in chapter 3 one model is implemented for the case of Simulator and one and one for the case with the I-support. For both models different neural networks to learn the relationship between the actuation space and the task space were realize and trained in order to find the most suitable.

4.1.1 Elastica model

As previously written, the goal is to ensure that the robot given a target is able to choose the necessary actions to achieve it. By analyzing the problem in its entirety, two sub-problems have been identified, the first is to understand with which angular velocity and at what amplitude the robot should rotate, to identify the trajectory that robot should follow, the second sub-problem is to understand at which instant the gripper should open to make land the object as close to the target. The first sub-problem has been solved thanks to the implementation of a neural network composed of one hidden-layer, an input

layer of 2 neurons and an output layer of 2 nuerons, since it receives as inputs the two coordinates of the target (x and y components), omitting the z component since it is the same for all the targets and therefore not influencing, while as output produces the two parameters, A and ω that characterize the actuations needed. By doing this, we are able to know which actuations should be used to reach the desired target, thus leading us to understand at which moment should be opened the gripper in order to perform the launch and therefore to the second sub-problem. The outputs of the first neural network are sent to Elastica which starts the motion, during the motion, the second network is constantly fed with the outputs of the first NN and with the coordinates of position of the end-effector for the three previous time instants, meaning that has an input layer of 11 neurons, and predicts the coordinates of the landing position (x and y components) of the object if would be thrown at the current time instant. The prediction of the second NN is then compared with the desired target and if the difference between these two distances is below a certain decided threshold the throw will be performed and the object released otherwise will continue the simulation until the end, meaning that in the case this threshold is not reached the object will not be thrown. The overall approach is described in Figure 4.1

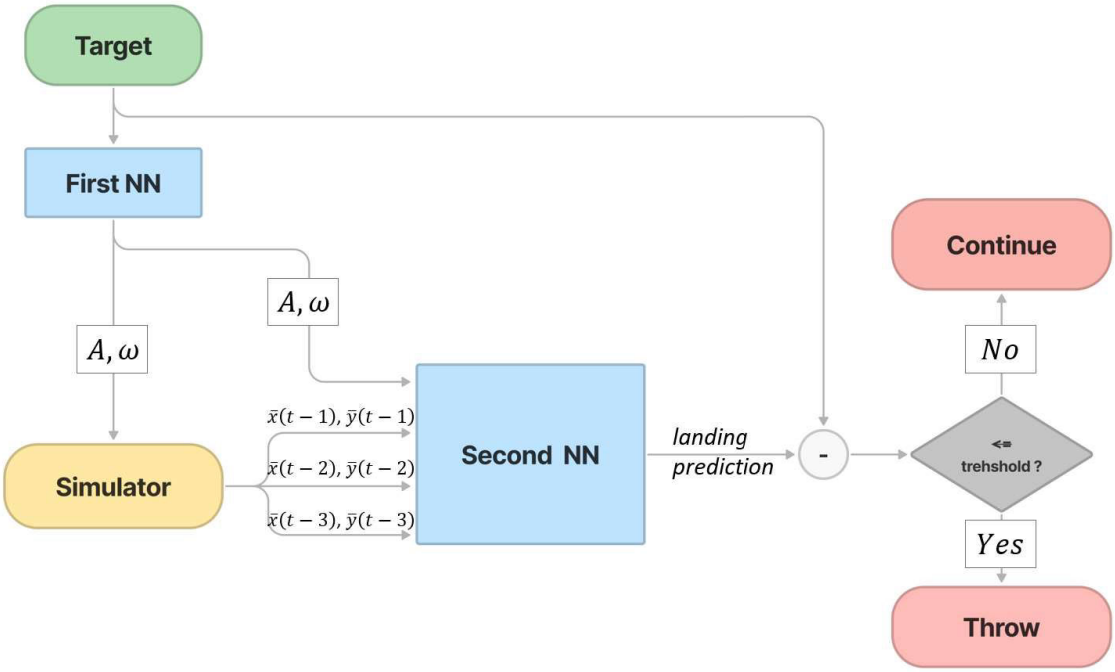


Figure 4.1: Entire strategy in case of Simulator

Optimization of the hyperparameters of a neural netwrok is a challenging problem with a sometimes empirical solution that, in any event, typically necessitates comparing the best model versions, as in this work. In both neural netwroks, First NN and Second NN, default values have been considered for partition between training, test set, learning rate, number of epochs, loss function and optimizer, while have been changed the batch's size, number of units of the hidden layer and their activation function, everything summarized in Table 4.1.

TABLE 4.1: HYPER-PARAMETERS ELASTICA

Hyper-parameters	***Batch's size***	***# Units***	***Activation function***
Changed	32	32	ReLU
	64	64	Sigmoid
		128	Tanh
		256	
Fixed	***Optimizer***	***# Epochs***	***Normalization***
	Adam	1000	Z-score (SS)
	Loss function	***Training set***	***Learning rate***
	MSE	90%	0.0001

After the neural networks have been trained, as written before, the results of the first NN are compared taking into account the average error, which is the difference between the prediction (A and ω) of the NN and the actual values of A and ω necessary to reach the desired target. The neural network with batch's size 64, number of neurons 128 and Sigmoid function as activation has been found to be the best, that is, the one with the lowest average error, as can be see in Figure 4.2.

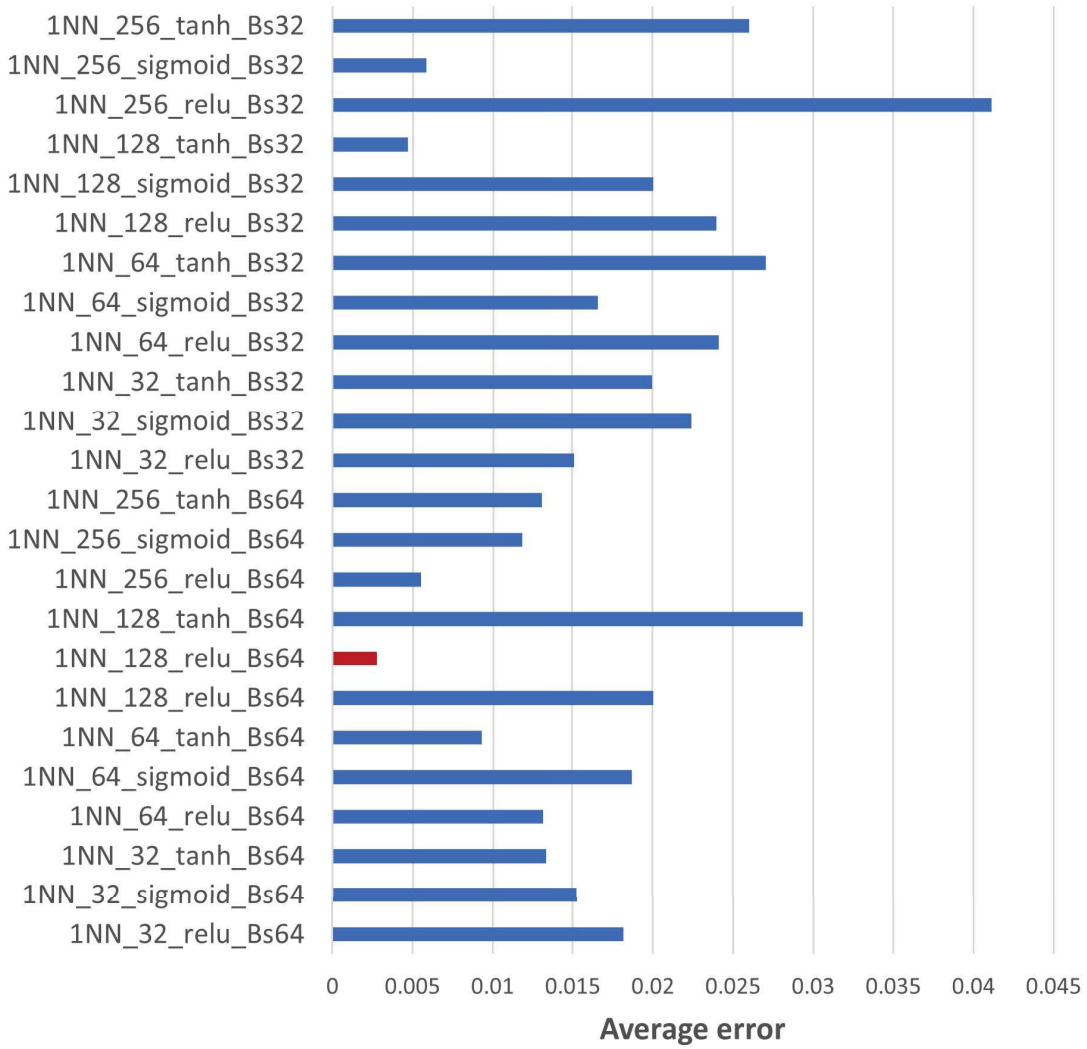


Figure 4.2: Results of First NN in Elastica model

While, considering the second NN, the trained neural networks are compared evaluating the average error as the Euclidean distance between the predicted landing position and the desired landing position, and has been found that the most efficient is the NN with batch's size 64, number of neurons 256 and hyperbolic tangent function, Tanh as activation. Results are reported in Figure 4.3.

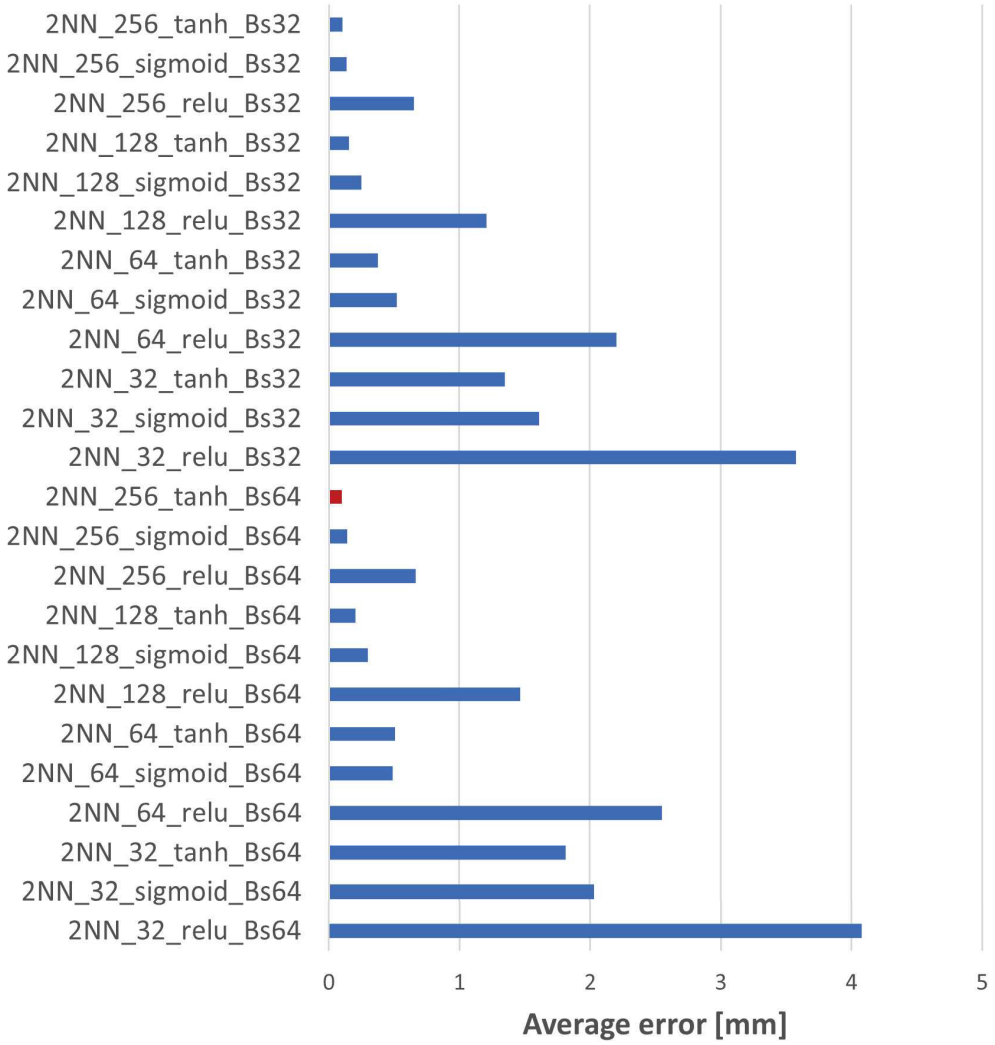


Figure 4.3: Results of Second NN in Elastica model

4.1.2 I-support model

The model for the case of the I-support is almost equal to the Elastica, it only has some differences. Also in this case the position of the end-effector of the soft-robot is constantly mapped thaks to the presence of the Vicon cameras, in fact without these cameras the strategy would not work. The main difference with the Elastica model is that in this case, inside the second neural network, the opening delay of the gripper is implemented, in fact in the model previously discussed this delay is not presence due to

the fact that the gripper opens immediately. Then now, the I-support robot through the use of the Vicon system provides the coordinates of position of the end-effector for the three previous time instants to the second neural network, and the latter provides the prediction of the landing position if the gripper will open at the next eighth instant of time, since the gripper takes 0.41 s to open which working at a frequency of $20Hz$ corresponds to 8 instants of time from when it receives the signal under pressure. The block diagram of the model is reported in Figure 4.4

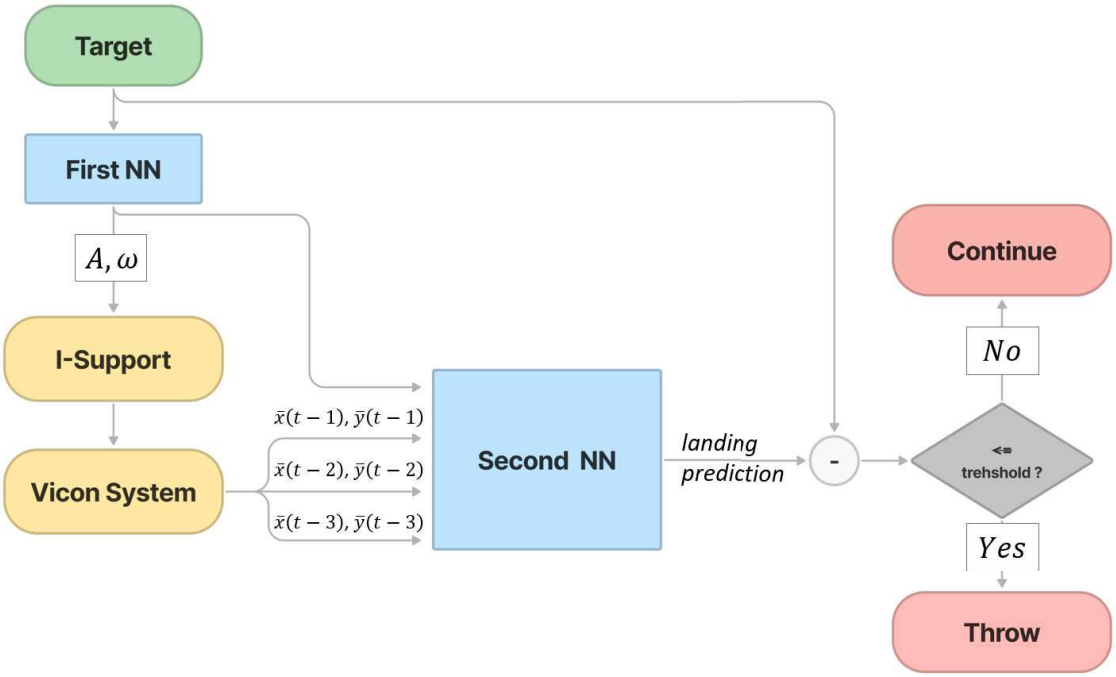


Figure 4.4: Entire strategy in case of I-support

As in section 4.1.1 both neural networks are trained with a different tuning of the hyper-parameters, compared to the previous training another value for the batch's size is considered there are two differences, one is that also neural networks with batch's size 16 are trained and the other is that the normalization is not applyied only on inputs of NNs but as well on the output but both additions are only for the second neural network, while for the first neural network the hyper-parameters remain unchanged. The hyper-parameters considered are summarized in Table 4.2

TABLE 4.2: HYPER-PARAMETERS I-SUPPORT ROBOT

Hyper-parameters	***Batch's size***	***# Units***	***Activation function***
Changed	16	32	ReLU
	32	64	Sigmoid
	64	128	Tanh
		256	
Fixed	***Optimizer***	***# Epochs***	***Normalization***
	Adam	1000	Z-score (SS)
	Loss function	***Training set***	***Learning rate***
	MSE	90%	0.0001

Considering the first NN, the trained neural netwroks are evaluated as before, taking into account the average error between the predicted values of A and ω and the actual values. The results of these trained neural networks are roport in Figure 4.5 and it turned out that the one with the best predictions is the one with batch's size equal to 32, number of neurons 256 and Relu as activation function.

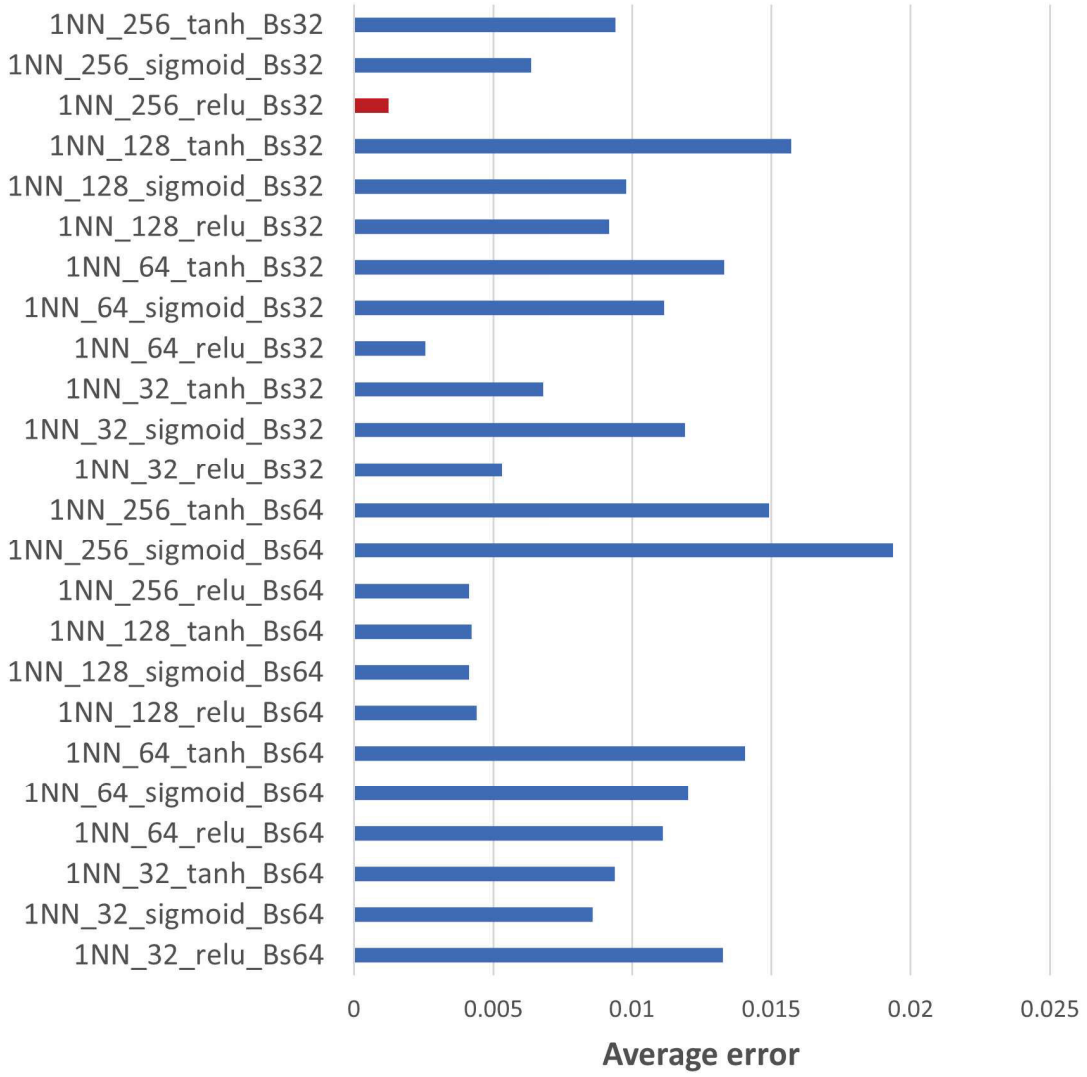


Figure 4.5: Results of First NN in I-support model

While for the second neural network, the evaluation of the trained neural networks are exactly as in section 4.1.1. The neural network with lower average error on the prediction is the one with batch's size equal 32, 512 neurons and Relu function as can be seen in Figure 4.6 .

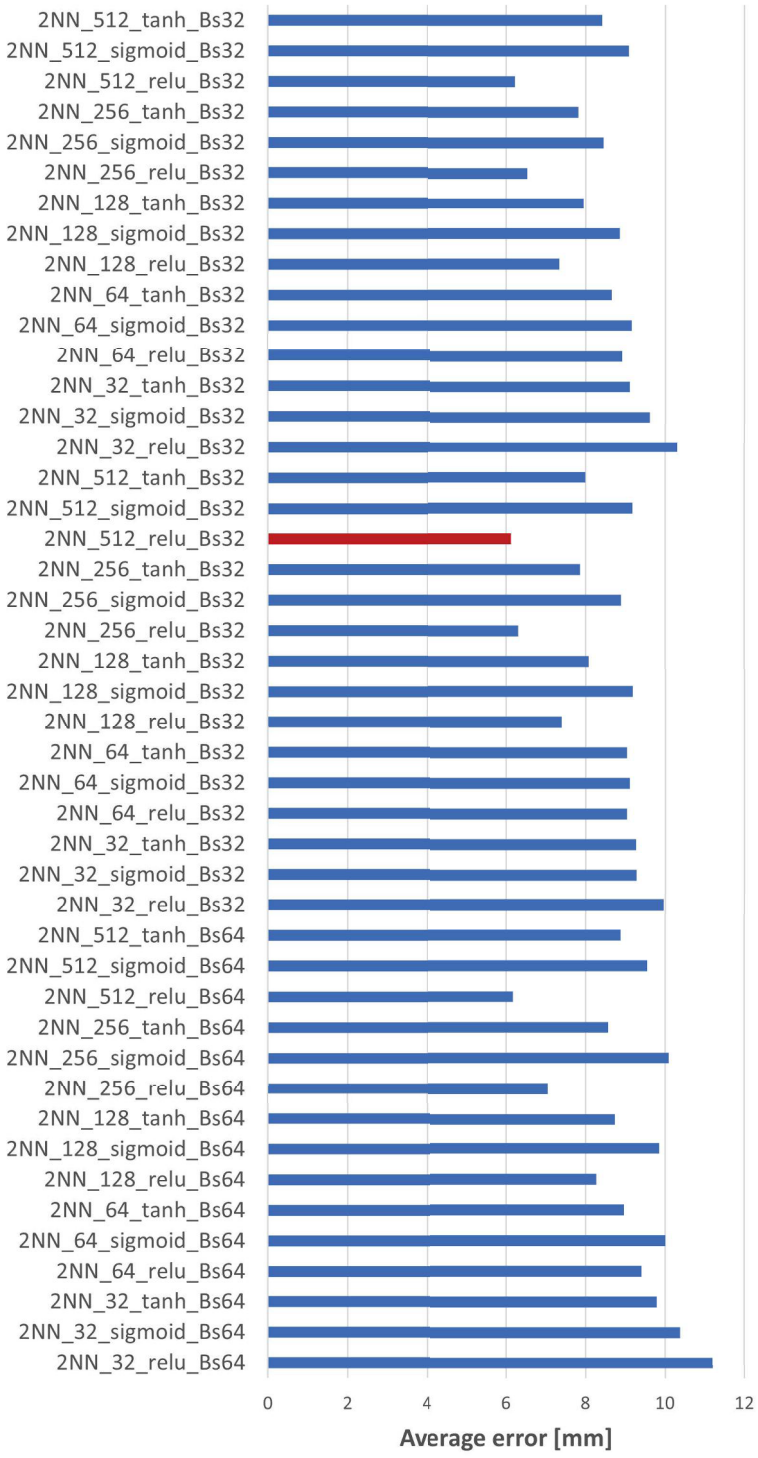


Figure 4.6: Results of Second NN in I-support model

4.2 Elastica Results

The last part of the Elastica model, as specified previously, consists of a decision block through which is decide whether or not to realize the throwing. More specifically, once the prediction on the landing position of the object is produced by the second neural network, it is compared with the desired target and if it is lower than the decided threshold, in this case $0.04\,m$, then the launch will be carried out, otherwise it will continue the simulation until its end. For the procedure one hundred random targets were defined, in particular, each x component and y component of the target were chosen randomly in a range between $-0.7\,m$ and $0.7\,m$. With a total of 100 targets, 84 of them were reached, meaning that launches have been carried out while for he remaining 16 targets no launch was performed since the predictions produced by the second neural network was never close enough to the desired landing position. In Figure 4.7 are shown the 100 targets, the predictions made of each target by the NN and the actual landing position of the object.

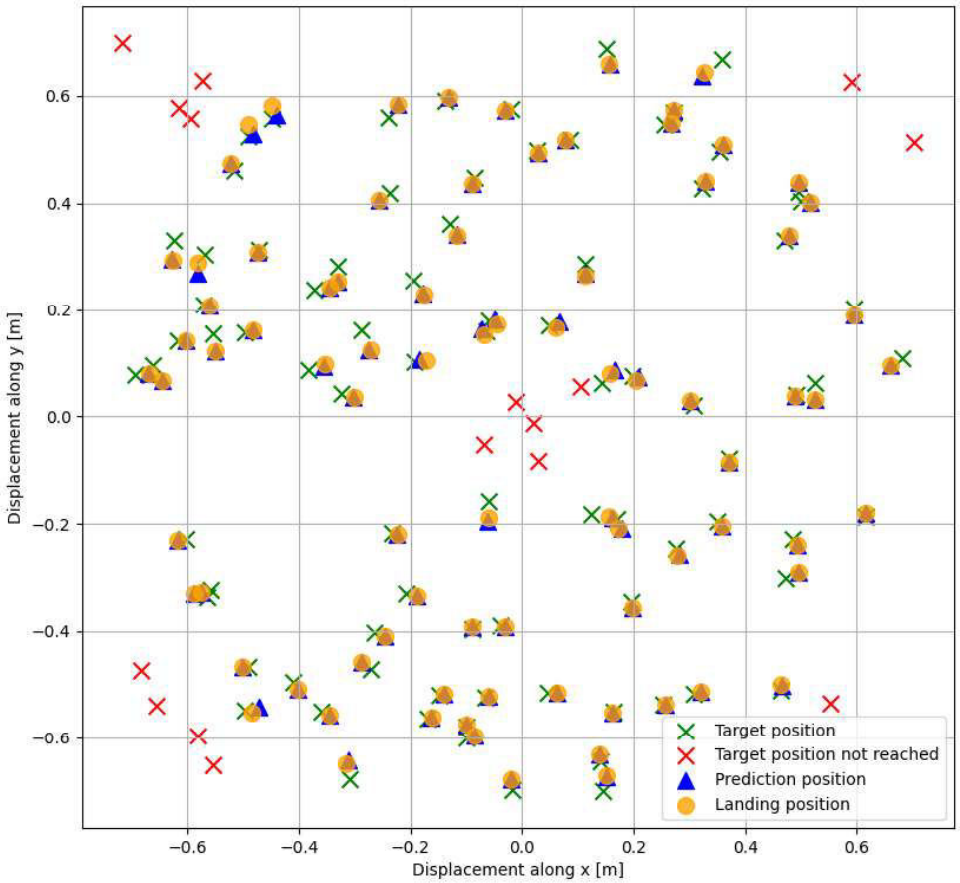


Figure 4.7: Throws in Elastica

The difference between the desired position and the actual landing position has been computed for each target, and an average of them evaluated, leading to ad average error equal to 0.0175 m

In Figure 4.8 can be seen all the targets respect to the entire Workspace with all the positions reached by the object once has been thrown obtained for the creation of the dataset.

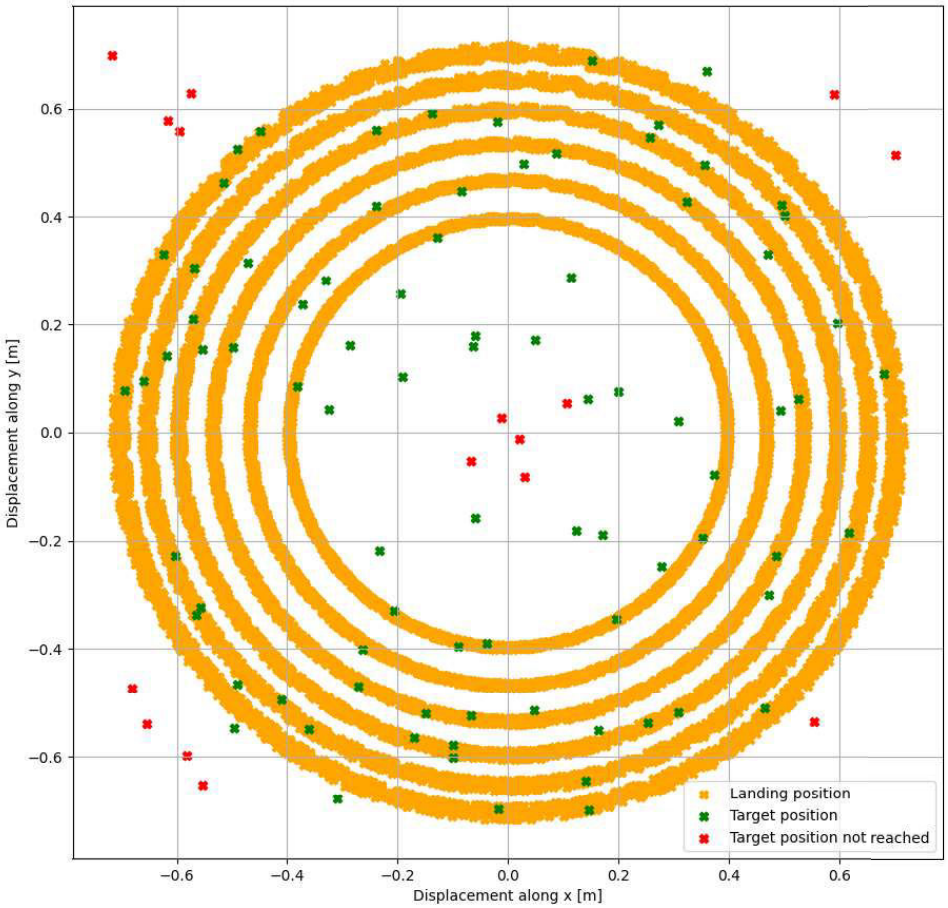


Figure 4.8: Targets and workspace in Elastica

4.3 I-Support Results

In the I-Support model, the decision block, in which the distance between the prediction of the landing position produced by the second neural network and the desired position is evaluated and compared. This distance is compared with a threshold, also in this case equal to $0.04m$ as for the Elastica model. For the experiments, 10 targets were selected, the procedure consisted in positioning a square box with a side equal to $14\,cm$ on the target position and carrying out 8 launch trials for each of them, actually in the

case of one target, since unreachable, only 5 trials were carried out, thus obtaining a total of 77 launches. Should be specified that respect to the Elastica model, here in the second NN, the height of the floor is not anymore equal to $-1\,m$ but is $-0.9\,m$ since the heiht of the box is $10\,cm$. The targets used are visible in the Figure 4.9

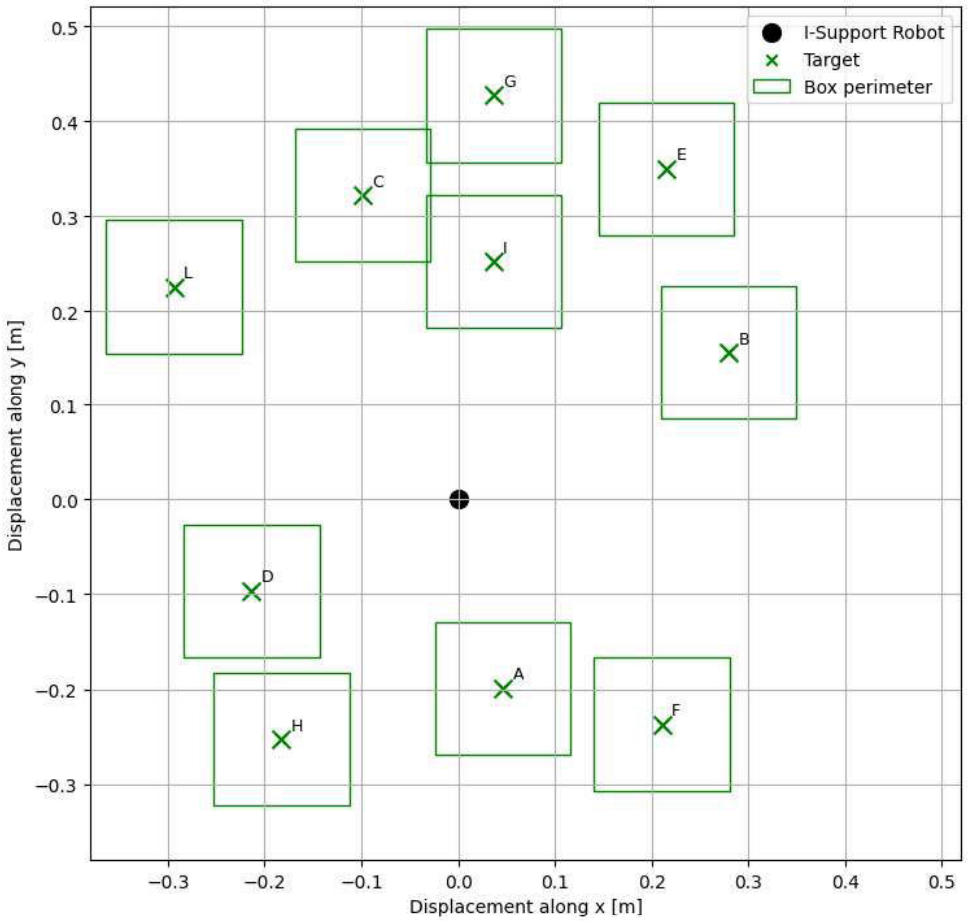


Figure 4.9: Targets for the I-Support

For each trial executed, a video has been recorded and the trajectories performed by the end-effector and by the object have been collected. Mapping of the object's position was possible, placing two markers on the object, which were then detected by the Vicon System, not creating any problem, in fact, as mentioned in section 3.3, the markers

placed on the objected were too close to the markers placed on the gripper, this forced the repetition of several trials, since the system was not able to recognize and save the trajectory of the object. The results obtained are visible in Figure 4.9 and in Table 4.3 where are reported the result for each trial and the distance between the effective landing position and the target position.

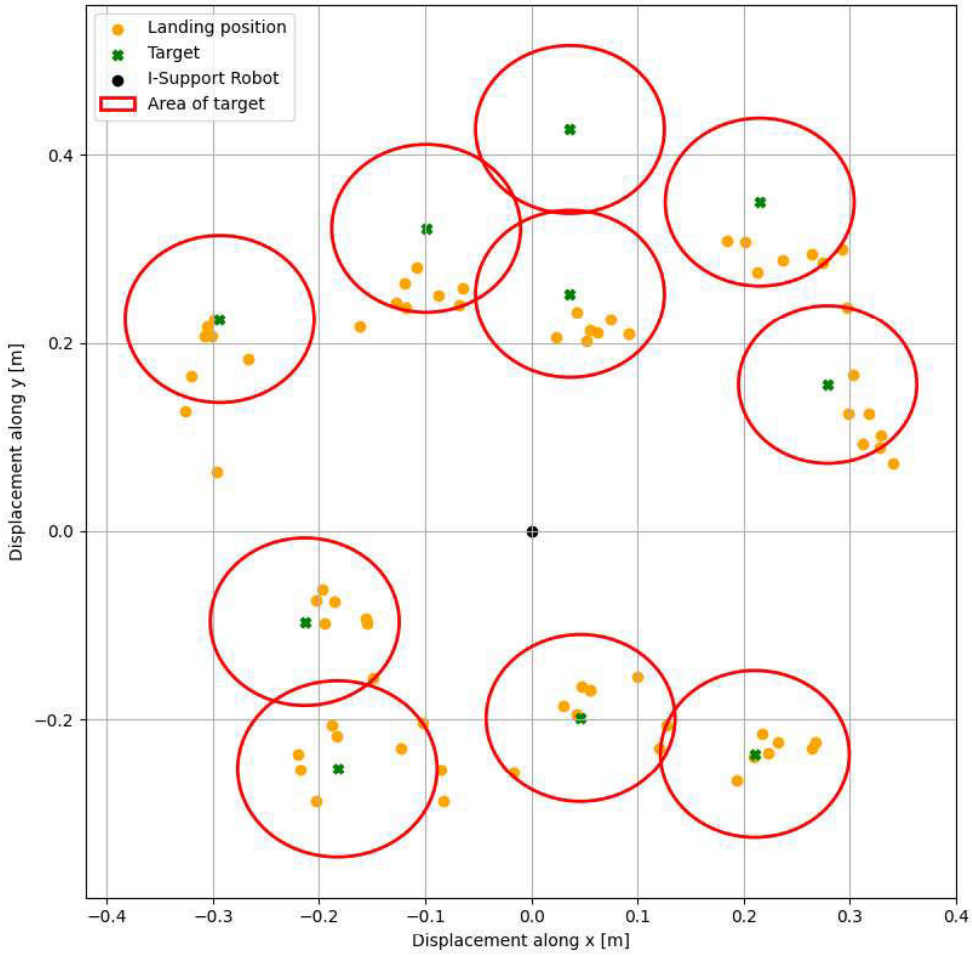


Figure 4.10: Targets and throws for the I-Support

TABLE 4.3: EXPERIMENT RESULTS SUMMARIZED

Target [m]	***outcome***				***average distance***
	# IN	**# OUT**	**# edge**	**# not launched**	***from the target [m]***
A=(0.046, -0.199)	5	-	3	-	0.045
B=(0.279, 0.155)	5	1	1	1	0.064
C=(-0.099, 0.322)	4	1	3	-	0.079
D=(-0.213, -0.096)	6	1	1	-	0.057
E=(0.215, 0.35)	5	1	2	-	0.082
F=(0.211, -0.237)	7	-	1	-	0.037
G=(0.036, 0.427)	-	-	-	5	-
H=(-0.183, -0.253)	6	-	2	-	0.058
I=(0.036, 0.252)	8	-	-	-	0.050
L=(-0.294, 0.226)	6	2	-	-	0.056

Finally in Figure 4.11 and 4.12 are represented for completeness the trajectories of two trials, trial 4 of target E and trial 5 always of target E, with landing inside the box and outside respectively.

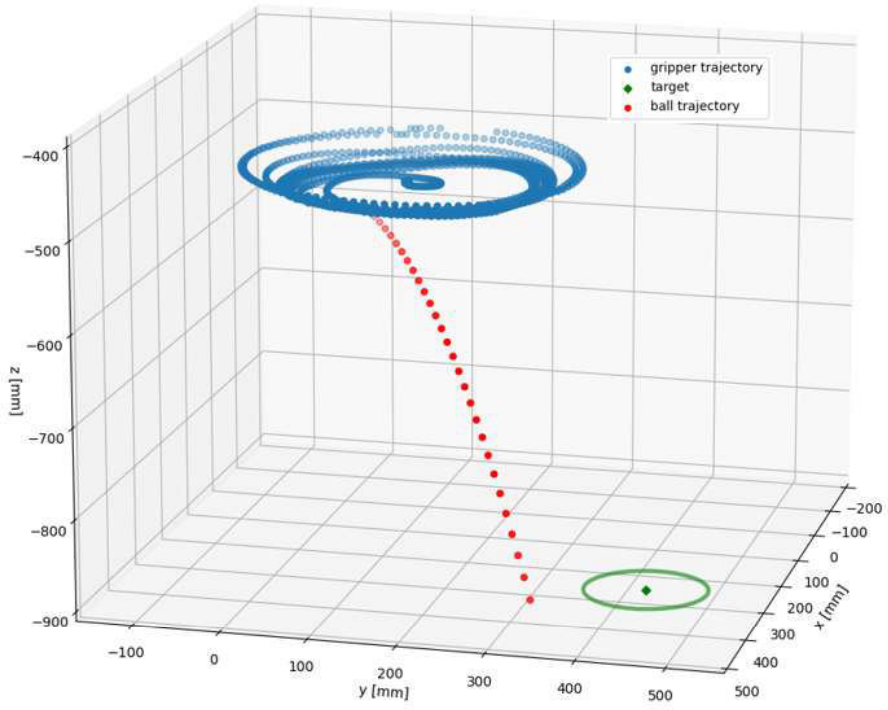


Figure 4.11: Trajectories of trial 4 target E

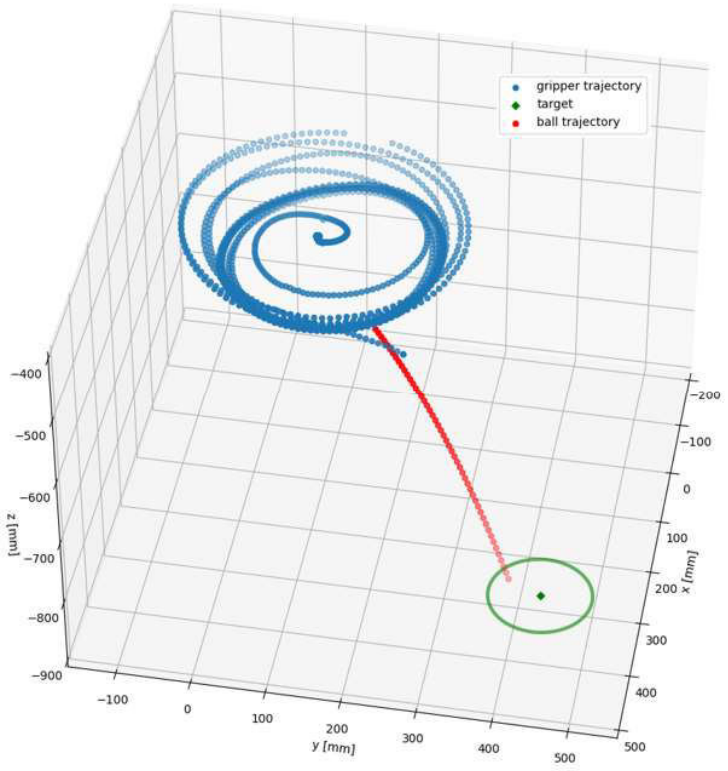


Figure 4.12: Trajectories of trial 5 target E

Chapter 5

Conclusions

This thesis aimed to use artificial intelligence to train a soft robot to throw objects and a closed loop controller was created with this purpose in mind. This data-driven methodology has been used on two separate platforms: the ideal Elastica soft robot simulator and the real I-Support robot, a soft manipulator. The results obtained are quite satisfactory despite the simplicity of the neural networks implemented since they are both characterized by only one hidden layer. More in detail in the model created for Elastica, the first neural network has an uncertainty of 0.0013 as regards the parameter A and of 0.0042 as regards the ω parameter, and they are much lower than the order of magnitude of these two parameters, which is -1 for A and 0 for ω. While the second neural network has an uncertainty of $0.09\,mm$ that is almost irrelevant respect to the workspace of the robot, which can be contained approximately in a circle with radius equal to $70\,cm$. In the model created for the I-support the accuracy registered in the first neural network is 0.0001 and 0.001 considering respectively parameter A and ω, while in the second neural network the accuracy is equal to $6.25\,mm$ always having a workspace of the robot almost equal to an are with radius $70\,cm$ since the furthest points reached by the object on landing are at this distance from the I-support. In this case the accuracy is lower maybe due to non linear condition of the robot and a dataset not well distributed, as visible from the generated workspace, since there is a high concentration of points in the more internal region. The strategy consists in giving a target and through a sort of cascade NN architecture and with a feedback from the Vicon system, the soft robot learns which actuations are necessary and which is the release point to make the object land in the desired position. Regarding the Elastica model, one hundred of throws

have been simulated and resulted with a success of the 84% and this percentage would increase if only targets present in the analyzed workspace were considered. While in I-support-model, seventy-seven throws have been performed and the success rate was of the 67.5%. The variability in the throws can be addressed at different causes such as the gripper, since the delay of the gripper was not always the same, the influence of the drag on the object that is neglected in the simulated throw or also to the initial position of the manipulator, as the soft-robot after each trial did not return to its previous position but remained marginally deformed. In conclusion, machine learning techniques can teach to a soft robot how to throw an object.

Bibliography

[1] Sami Haddadin, Lars Johannsmeier, and Fernando Díaz Ledezma. "Tactile Robots as a Central Embodiment of the Tactile Internet". In: *Proceedings of the IEEE* 107.2 (Feb. 2019). Conference Name: Proceedings of the IEEE, pp. 471–487. ISSN: 1558-2256. DOI: 10.1109/JPROC.2018.2879870.

[2] Sangok Seok et al. "Peristaltic locomotion with antagonistic actuators in soft robotics". eng. In: Book Title: 2010 IEEE International Conference on Robotics and Automation ISSN: 1050-4729. IEEE, 2010, pp. 1228–1233. ISBN: 978-1-4244-5038-1. DOI: 10.1109/ROBOT.2010.5509542. URL: https://ieeexplore.ieee.org/document/5509542 (visited on 06/05/2022).

[3] Huai-Ti Lin, Gary G. Leisk, and Barry Trimmer. "GoQBot: a caterpillar-inspired soft-bodied rolling robot". en. In: *Bioinspiration &amp$\mathsemicolon$ Biomimetics* 6.2 (Apr. 2011). Publisher: IOP Publishing, p. 026007. ISSN: 1748-3190. DOI: 10.1088/1748-3182/6/2/026007. URL: https://doi.org/10.1088/1748-3182/6/2/026007 (visited on 06/05/2022).

[4] Yusei Sakuhara, Hiroaki Shimizu, and Kazuyuki Ito. "Climbing Soft Robot Inspired by Octopus". In: *2020 IEEE 10th International Conference on Intelligent Systems (IS)*. ISSN: 1541-1672. Aug. 2020, pp. 463–468. DOI: 10.1109/IS48319.2020.9199967.

[5] Siegfried Bauer et al. "25th Anniversary Article: A Soft Future: From Robots and Sensor Skin to Energy Harvesters". In: *Advanced materials (Deerfield Beach, Fla.)* 26 (Jan. 2014). DOI: 10.1002/adma.201303349.

[6] Frank Daerden and Dirk Lefeber. "The Concept and Design of Pleated Pneumatic Artificial Muscles". eng. In: *International journal of fluid power* 2.3 (2001). Publisher: Taylor & Francis, pp. 41–50. ISSN: 1439-9776. DOI: 10.1080/14399776.2001.10781119.

[7] Dian Yang et al. "Linear Actuators: Buckling Pneumatic Linear Actuators Inspired by Muscle (Adv. Mater. Technol. 3/2016)". In: *Advanced Materials Technologies* 1.3 (2016). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/admt.201670016. ISSN: 2365-709X. DOI: 10.1002/admt.201670016. URL: http://onlinelibrary.wiley.com/doi/abs/10.1002/admt.201670016 (visited on 06/05/2022).

[8] G. Immega and K. Antonelli. "The KSI tentacle manipulator". In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 3. ISSN: 1050-4729. May 1995, 3149–3154 vol.3. DOI: 10.1109/ROBOT.1995.525733.

[9] M.B. Pritts and C.D. Rahn. "Design of an artificial muscle continuum robot". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. Vol. 5. ISSN: 1050-4729. Apr. 2004, 4742–4746 Vol.5. DOI: 10.1109/ROBOT.2004.1302467.

[10] W. McMahan, B.A. Jones, and I.D. Walker. "Design and implementation of a multi-section continuum robot: Air-Octor". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Aug. 2005, pp. 2578–2585. DOI: 10.1109/IROS.2005.1545487.

[11] W. McMahan et al. "Field trials and testing of the OctArm continuum manipulator". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* ISSN: 1050-4729. May 2006, pp. 2336–2341. DOI: 10.1109/ROBOT.2006.1642051.

[12] Amir Firouzeh et al. "Sensor and actuator integrated low-profile robotic origami". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Nov. 2013, pp. 4937–4944. DOI: 10.1109/IROS.2013.6697069.

[13] Yang Yang and Yonghua Chen. "Novel design and 3D printing of variable stiffness robotic fingers based on shape memory polymer". In: *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. ISSN: 2155-1782. June 2016, pp. 195–200. DOI: 10.1109/BIOROB.2016.7523621.

[14] Edwin W. H. Jager, Olle Inganas, and Ingemar Lundstrom. "Microrobots for micrometer-size objects in aqueous media: Potential tools for single-cell manipulation". English. In: *Science* 288.5475 (June 2000). Num Pages: 4 Place: Washington, United States Publisher: The American Association for the Advancement of Science, pp. 2335–8. ISSN: 00368075. URL: http://www.proquest.com/docview/213571879/abstract/2D89E13C69F844E3PQ/1 (visited on 06/05/2022).

[15] Gu Han Kwon et al. "Biomimetic Soft Multifunctional Miniature Aquabots". In: *Small* 4.12 (2008). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smll.200800315, pp. 2148–2153. ISSN: 1613-6829. DOI: 10.1002/smll.200800315. URL: http://onlinelibrary.wiley.com/doi/abs/10.1002/smll.200800315 (visited on 06/05/2022).

[16] Hyouk Ryeol Choi et al. "Biomimetic soft actuator: design, modeling, control, and applications". In: *IEEE/ASME Transactions on Mechatronics* 10.5 (Oct. 2005). Conference Name: IEEE/ASME Transactions on Mechatronics, pp. 581–593. ISSN: 1941-014X. DOI: 10.1109/TMECH.2005.856108.

[17] Matthias Rolf and Jochen J. Steil. "Constant curvature continuum kinematics as fast approximate model for the Bionic Handling Assistant". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Oct. 2012, pp. 3440–3446. DOI: 10.1109/IROS.2012.6385596.

[18] B.A. Jones and I.D. Walker. "Kinematics for multisection continuum robots". In: *IEEE Transactions on Robotics* 22.1 (Feb. 2006). Conference Name: IEEE Transactions on Robotics, pp. 43–55. ISSN: 1941-0468. DOI: 10.1109/TRO.2005.861458.

[19] I.A. Gravagne, C.D. Rahn, and I.D. Walker. "Large deflection dynamics and control for planar continuum robots". In: *IEEE/ASME Transactions on Mechatronics* 8.2 (June 2003). Conference Name: IEEE/ASME Transactions on Mechatronics, pp. 299–307. ISSN: 1941-014X. DOI: 10.1109/TMECH.2003.812829.

[20] Deepak Trivedi, Amir Lotfi, and Christopher D. Rahn. "Geometrically exact dynamic models for soft robotic manipulators". In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Oct. 2007, pp. 1497–1502. DOI: 10.1109/IROS.2007.4399446.

[21] Matthias Rolf and Jochen J. Steil. "Efficient Exploratory Learning of Inverse Kinematics on a Bionic Elephant Trunk". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.6 (June 2014). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 1147–1160. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2013.2287890.

[22] A. Melingui et al. "Qualitative approach for inverse kinematic modeling of a Compact Bionic Handling Assistant trunk". In: *2014 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. July 2014, pp. 754–761. DOI: 10.1109/IJCNN.2014.6889947.

[23] Peng Qi et al. "Kinematic Control of Continuum Manipulators Using a Fuzzy-Model-Based Approach". In: *IEEE Transactions on Industrial Electronics* 63.8 (Aug. 2016). Conference Name: IEEE Transactions on Industrial Electronics, pp. 5022–5035. ISSN: 1557-9948. DOI: 10.1109/TIE.2016.2554078.

[24] Thomas George Thuruthel et al. "Learning Global Inverse Statics Solution for a Redundant Soft Robot". In: Jan. 2016, pp. 303–310. DOI: 10.5220/0005979403030310.

[25] Apoorva Kapadia and Ian D. Walker. "Task-space control of extensible continuum manipulators". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. ISSN: 2153-0866. Sept. 2011, pp. 1087–1092. DOI: 10.1109/IROS.2011.6094873.

[26] Enver Tatlicioglu. "APOORVA D. KAPADIA IAN D. WALKER DARREN M. DAWSON". en. In: *RECENT ADVANCES in SIGNAL PROCESSING* (), p. 8.

[27] Andrew D. Marchese, Russ Tedrake, and Daniela Rus. "Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 1050-4729. May 2015, pp. 2528–2535. DOI: 10.1109/ICRA.2015.7139538.

[28] Yaakov Engel, Peter Szabó, and Dmitry Volkinshtein. "Learning to Control an Octopus Arm with Gaussian Process Temporal Difference Methods." In: vol. 18. Jan. 2005.

[29] David Braganza et al. "A Neural Network Controller for Continuum Robots". In: *IEEE Transactions on Robotics* 23.6 (Dec. 2007). Conference Name: IEEE Transactions on Robotics, pp. 1270–1277. ISSN: 1941-0468. DOI: 10.1109/TRO.2007.906248.

[30] Michael A. Nielsen. "Neural Networks and Deep Learning". en. In: (2015). Publisher: Determination Press. URL: http://neuralnetworksanddeeplearning.com (visited on 06/13/2022).

[31] siddarth2947. *[D] Jurgen Schmidhuber on Seppo Linnainmaa, inventor of back-propagation in 1970*. Reddit Post. Dec. 2019. URL: www.reddit.com/r/MachineLearning/comments/e5vzun/d_jurgen_schmidhuber_on_seppo_linnainmaa_inventor/ (visited on 06/12/2022).

[32] Seppo Linnainmaa. "Taylor expansion of the accumulated rounding error". en. In: *BIT Numerical Mathematics* 16.2 (June 1976), pp. 146–160. ISSN: 1572-9125. DOI: 10.1007/BF01931367. URL: https://doi.org/10.1007/BF01931367 (visited on 06/12/2022).

[33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986). Number: 6088 Publisher: Nature Publishing Group, pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (visited on 06/12/2022).

[34] Noel Naughton et al. "Elastica: A Compliant Mechanics Environment for Soft Robotic Control". In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021). Conference Name: IEEE Robotics and Automation Letters, pp. 3389–3396. ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3063698.

[35] *Forward and inverse problems in the mechanics of soft filaments*. en. DOI: 10.1098/rsos.171628. URL: https://royalsocietypublishing.org/doi/epdf/10.1098/rsos.171628 (visited on 06/26/2022).