university of groningen

faculty of science and engineering

A cerebellar-inspired architecture to control a soft robot with redundant Degrees of Freedom

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of
Prof. Dr. Herbert Jaeger (Artificial Intelligence, University of Groningen)
and
Dr. Yuji Kawai (Institute for Open and Transdisciplinary Research Initiatives, Osaka University)

Andreas Gassilloud (s4201523)

November 2, 2025

Contents

Acknowledgments

Many thanks to the Symbiotic Intelligent Systems Research Center of the University of Osaka, who provided the opportunity for this project in collaboration with the University of Groningen. Only possible with the guidance and practical help of members of those two organizations, I want to express my sincere thanks to each of them, and briefly list their contributions:

KAWAI Yuji , for enabling the project by providing the basis of this project in many ways, including both substance and organization, as the external supervisor for this project.

Herbert JAEGER, for enabling the project as the internal supervisor for this project.

PARK Jihoon, who assisted in many miscellaneous matters.

ISHIHARA Hisashi, for providing the robot Affetto, and allowing its use for experiments.

ATSUTA Hiroshi, for providing the Python code base that enables communication between computer and the robot Affetto, such as control of the robot, which was utilized and built upon for all experiments involving the robot.

Erhan OZTOP, for providing a 3D tracking camera used in experiments, as well as a code base for operating it.

Abstract

This thesis expands on previous work using a cerebellar-inspired control architecture for robots operating under kinematic parameter uncertainty. The system combines an inverse kinematics approximator with a correction component based on Echo State Networks (ESNs). This removes the need for precise kinematic parameters required by traditional approaches based on mathematical models, making the architecture well suited for robots where parameter estimation is impractical, such as soft robots with non-rigid actuators or those with redundant degrees of freedom (DoF). The cerebellar-inspired control architecture is validated on the soft robot ”Affetto,” focusing on its redundant DoF arm. Three correction strategies are tested: (1) adjusting the desired end-effector input to the approximator, (2) correcting its joint angle outputs, and (3) applying both corrections simultaneously. Two ESN based correction components are compared: Ensembles of standard leaky integrator ESNs and reBASICS-type ESN ensembles. Results show that correction pathways can either interfere or combine constructively, depending on the strength of the ESN base correction component. The reBASICS-type ensembles significantly outperform the standard-type ESN ensembles, attributed to lower degree of correlation between the modules of the ensemble.

1 Introduction

Commanding a robot requires understanding its kinematics, that is the relationship between joint angles, joint positions, and the resulting end-effector pose. When specifying a desired end-effector position, solving for the required joint angles involves the robot's inverse kinematics. This typically relies on mathematical models parameterized by the robot's exact kinematic structure. But even with precise models, solutions may be non-trivial. For example, for redundant manipulators where the available Degrees of Freedom (DoF) exceed the task constraints, the inverse kinematic problem has a theoretical infinite number of solutions (Burdick 1989 [1]). Soft robots, which operate joints with non-rigid mechanisms, may also be solved with geometrically exact models. However, those suffer from high computational and parameter tuning requirements (Renda et al. 2012 [2], Trivedi, Lofti, and Rahn 2008 [3]), because of the movement noise that results from the inherently imprecise actuation of soft robots.
Alternative approaches that use machine learning based modeling have been successful, such as an inverse kinematics approximator consisting of a Multi-layer Perceptron (MLP) (Thuruthel et al. 2016 [4], 2016 [5], 2017 [6]). These inverse kinematics approximator MLPs learn the relationship between the joint angles and the end-effector position, to output the joint angles for any desired end-effector position as input. The training data may be sampled through motor babbling: the random generation of joint angles and recording of associated end-effector positions. Training data achieved through motor babbling may be further curated through a quality function (Hlavac 2020 [7]), e.g. to select joint angle solutions that minimally change between neighboring end-effector points. Even so, such inverse kinematics approximators are susceptible to small changes in the system, such as when adding external loads or when changing the exact location of the end-effector, for example by attaching operative tools in surgical tasks. Further, they are limited by an inability to transfer knowledge across similar platforms (Kalidindi et al. 2019 [8]).
To address this, Kalidindi et al. (2019) [8] present a human-cerebellum inspired adaptive kinematic controller for soft robotic applications. It draws inspiration from Desmurget and Grafton (2000) [9], who observed control loops in humans' goal-directed movement, in which the cerebellum likely takes on a role of correcting erroneous movement. The cerebellar-inspired approach therefore adds a correction component consisting of an Echo State Network (ESN) (Jaeger 2001 [10]) to adjust the inverse kinematics approximator MLP, from here on referred to as the ESN component and the MLP controller, respectively. In this architecture, the MLP controller first learns to approximate the inverse kinematics based on samples that record joint angles and their associated end-effector positions. Then, the MLP controller is used to command the robot with the desired end-effector position given as input, and the to be commanded joint angles as output. Finally, the error between desired and achieved end-effector positions is used to train the ESN component that adjusts the MLP controller input, similar to how the cerebellum might

adjust erroneous movement.
This is opposed to other attempts at correcting the inverse kinematics approximator, such as an approach done by Pique et al. (2022) [11] that continually retrains an MLP controller to compensate for newly introduced outside disturbances, or an approach by Chen et al. (2025) [12] that attempts to account for hysteresis in soft robots by including pressure magnitude and directional change in the input of the MLP controller. The cerebellar-inspired architecture would instead be able to keep the MLP controller as is, and add corrections from the ESN component as necessary based on circumstances.
Kalidindi et al. (2019) [8] explore different inverse kinematics approximator variations, and then compare the performance to the cerebellar-inspired pipeline that adds the ESN component for correction. However, they don't explore different options for the ESN component itself, and don't report attempts to optimize it. Further, their experiments consist of simulations in which the MLP controller performs very well already without the added ESN component. As such, further work might explore the utility of this machine learning-based solution in a real robot.
In a follow-up work, Tan, Yu, and Ni (2022) [13] use simulations and a real robot with rigid actuators and redundant DoF in experiments. They conclude that the cerebellar-inspired architecture performs well compared to mathematical model-based solutions. They also compare two ESN variants for the ESN component, a leaky integrator ESN and a non-leaky ESN. Like Kalidindi et al. (2019) [8], Tan et al. (2022) [13] have a very well performing MLP controller, and adding correction from the ESN component leads to near perfect results for either ESN type. Given this, the potential of the cerebellar-inspired architecture is not entirely explored, and no strong conclusions are drawn about the type of ESN component.
Kalidindi et al. (2019) [8] base the cerebellar-inspired architecture mainly on the work of Yamazaki et al. (2005 [14], 2009 [15]), who explore the cerebellum and its role in movement. Yamazaki et al. (2007) [16] liken granular cell clusters that each contain ~ 100 neurons to ESNs, and suggest that ~ 1014 of those granular cell clusters may form an ensemble similar to an ESN ensemble in an eye blink conditioning task. However, both Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] use only a singular ESN. Especially in tasks where the correction to be done by the ESN component becomes more complicated, it might be useful to expand the ESN component to be an ensemble of ESNs, to leverage the increased representational capacity provided by an ensemble.
This is done by Kawai, Atsuta, and Asada (2024) [17] who utilize ”reBASICS”, a type of ESN ensemble (Kawai et al. (2023) [18]), for the ESN component in simulations. In those simulations, Kawai et al. (2024) [17] model errors in control, as would be expected to occur in a real robot. To combat this error, they expand on the cerebellar-inspired architecture of Kalidindi et al. (2019) [8] to additionally provide feedback to the angle output of the MLP controller, based on the difference between the commanded joint angles and the actually achieved joint angles.

1.1 Research Questions

For my graduation project, I combine the work of multiple members of the Symbiotic Intelligent Systems Research Center of Osaka University. I was provided access to the soft robot ”Affetto” (Ishihara and Asada 2015 [19]) to use for experiments. ”Affetto” is a child-like soft robot with air-pressure based pneumatic actuators, whose arm has 4 actuators. As such, the robot is an ideal candidate for a machine learning based inverse kinematics solution, as it is limited by the inherent imprecision of pneumatic actuators and has redundant DoF for movements of the arm. This allows testing of the cerebellar-inspired architecture in a setting it has the most supposed value for, to expand upon the previous work of Kalidindi et al. (2019) [8] and Tan et al. (2022) [13], and Kawai et al. (2024) [17].

> Q1. How will the cerebellar-inspired architecture perform for a soft robot with redundant DoF?

With the more difficult use case of a real soft robot with redundant DoF comes the expectation that the MLP controller that generates the inverse kinematics performs generally worse than in previous work. Therefore, a strong ESN correction component may be more important than in previous work. Because of this, another focus lies on optimizing the ESN component so that it is able to provide any correction necessary. For this, different ESN variations are compared for use in the ESN component: the leaky integrator ESN variations as implemented by Tan et al. (2022) [13], and the *reservoir of basal dynamics* (reBASICS) by my external supervisor, Yuji Kawai (Kawai et al. (2023) [18]).

> Q2. What ESN variation should the ESN component of the cerebellar-inspired architecture be made of?

Additionally, more options on the exact ESN component correction architecture are explored. Specifically, Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] use the ESN component to correct the input of the MLP controller, which consists of the desired end-effector positions. However, Kawai et al. (2024) [17] also modify the joint angles output of the MLP controller. As such, the performance of both correction pathways individually, as well as combined, will be compared to each other.

> Q3. Will correcting the *joint angles*-output of the MLP controller instead of just its *desired end-effector position*-input improve the cerebellar-inspired architecture?

1.2 Thesis Outline

First, the literature review discusses the grounding of the cerebellar-inspired architecture in its biological validity and engineering application. For this, the role of the Cerebellum in movement 2.1 is briefly discussed , as well as Echo State Networks 2.2, and their formalism 2.3, and how they are used in cerebellar-inspired architecture to control robots 2.4. Next, the practical application of the inverse kinematics approximator in robotics is discussed 2.5, how the ESN component is added for the full cerebellar-inspired architecture 2.6, and how it performs as a stand-alone compared to the full cerebellar-inspired architecture that includes correction from the ESN-component 2.7. Further, reBASICS is

presented 2.8. Finally, reBASICS performance as an ESN component in the cerebellar-inspired architecture as well as an expansion of the architecture pipeline with additional correction paths are discussed 2.9.
Concerning the methods, the robot ”Affetto” that will be used in the experiment, the robot’s joints, and the general behavior of its actuators are detailed 3.1. The training of the Inverse Kinematics Approximator MLP is specified 3.2. Further, the ESN component training and pipeline modification that will be attempted to address research question Nr°3 are described 3.3. Based on the conclusion of the literature review, the method section defines the ESNs that will be compared in the experiments to address research question Nr°2 3.4.
The experiment section describes the build-up of the experiment, such as the camera and tracking of the end-effector 4.1, followed by how data was gathered for the MLP controller 4.2. After an analysis on the gathered data 4.3, MLP controllers were trained and compared 4.4. Details related to the depth measurement analysis and pre-experiments to ensure the frames per second necessary for recording continual movements, and the algorithm to read in camera data are moved to the Appendix .1, .2, and .3. Next, a target movement was generated 4.6, and the MLP controller performance in tracing it was measured 4.7. This is followed by the inclusion of the ESN component correcting the MLP controller 4.8.
With this setup, the final experiments were conducted comparing the ESN component variations, as well as the feedback paths of adjusting the MLP controller’s *joint angles*-output, its *desired end-effector position*-input, or both simultaneously 5.1. To expand the interpretability of the results, the dynamics of the different ESN variations that were compared fo the ESN component are explored 5.2, followed by a deeper analysis of the feedback path to the MLP controller’s *joint angles*-output 5.3.
Final conclusions are drawn, with suggestions for future research 6.

2 Background Literature

2.1 The role of the Cerebellum - movement selection, movement correction, or event timing?

The exact involvement of the cerebellum in movement has been explored for a long time. An example of earlier work is Marr et al. (1969) [20], who argue that the role of the Granular cells and Purkinje cells is to encode associative memory, functionally acting like a look-up table, which determines what movement is executed based on the current activation pattern of the Purkinje cells.
Further work extended the role of the cerebellum beyond mere movement selection to include a continual influencing of movements under the Internal Models framework, formalized by Wolpert et al. (1995) [21]. The Internal Models framework includes, among other things, that the cerebellar structure corrects movement by anticipating the movement outcomes based on experience, and attempts to preemptively correct the movement. This is what is theorized to allow rapid and precise motor control even in the presence of sensory delays. This framework is still considered valid, as for example concluded in a review by Tanaka et al. (2020) [22].
The amount of granular cells in the human brain is estimated by Palay et al. (1974) [23] to be between 50–70 billion, with 50-100 granular cells forming a glomerulus. While the Internal Models framework as expanded upon by Wolpert et al. (1998) [24] and Imamizu et al. (2003) [25] theorizes that multiple segregated regions exist and are individually activated for different learned movements, how many glomeruli make up a module is not clearly established. Yamazaki et al. (2007) [16] estimate and model an ensemble of 1.012 glomeruli with 100 granular cells per glomerulus for an eye blink conditioning task.
Alternative theories to the Internal Models framework appear to be complementary rather than contradictory. For example, research on the cerebellum often explored the role of the cerebellum for event timing specifically, such as for skilled movement or eye blinking (Ivry et al. 2002 [26]). This is in parallel to research about the involvement of the cerebellum for movement correction, such as Contreras et al. (1997) [27]. A way to integrate those two functions is suggested by Yamazaki et al. (2012) [28], who propose that the cerebellum may be able to simultaneously encode the commands for both timing control and the gain needed to adjust movements. This suggests that while event timing is functionally different from error correction of movements, they both are aspects of movement prediction that are processed in the same cerebellar structure.

2.2 Echo State Networks

Echo state networks (ESNs) (Jaeger 2001 [10]) are a type of recurrent neural network, that is large, fixed, and randomly connected. Driving this ”reservoir” network with an input signal induces a nonlinear response signal in each neuron, from which a desired output signal is combined in any desired trainable linear combination (Jaeger 2007) [29].
ESNs were initially introduced for nonlinear signal processing and control tasks such as time-series prediction and pattern generation (Jaeger 2001 [10]; Jaeger et al. 2004 [30]). Subsequent work extended the reservoir computing framework to a range of dynamical systems and robotic control problems, demonstrating the capacity of recurrent reservoirs to model and generate complex temporal patterns (Maass et al. 2002 [31]; Sussillo Abbott 2009 [32]; Antonelo et al. 2007 [33]; Lukoševičius Jaeger 2009 [34]). This body of research established ESNs as a practical and computationally efficient approach for real-time adaptive control, while retaining interpretability in terms of dynamical system theory. Parallel research in computational neuroscience has explored recurrent network architectures with dynamical properties resembling those of biological circuits (Stanley et al. 1999 [35], Stanley 2001 [36], Kistler et al. 2002 [37]). While not part of the reservoir computing framework, these studies share conceptual similarities with ESNs in exploiting high-dimensional recurrent dynamics for temporal processing and motor control. Mussa et al. (2000) [38] suggest that such types of neural networks could eventually form the basis from which signals would be collected and exploited to control prosthetics.
There exist other reservoir methods, such as the independently and simultaneously developed Liquid State Machines by Maas et al. (2002) [31]. Liquid State Machines are functionally similar to ESNs, but were developed from a computational neuroscience background. As such, they include more complexity, such as biologically realistic models of spiking integrate-and-fire neurons. As this makes them more difficult to implement and fine-tune compared to ESNs, they are less widespread for engineering applications (Lukosevicius et al. (2009 [**?**]).

2.3 Echo State network formalism

As described by Lukosevicius (2012) [39], an ESN is described by the Euler discretized equation:

$$x(t) = (1-\alpha)x(t-1) + \alpha \tanh\left(W_{in}[1;u(t)] + W_{rec}x(t-1)\right) \tag{2.1}$$

Where the reservoir state $x(t)$ is updated based on both its previous state $x(t-1)$ modified by $(1-\alpha)$ as well as the new input $u(t)$ and a bias input of 1 scaled by the input weights W_{in}, and the recurrent weights $W_{rec}x(t-1)$, both of which are themselves scaled by a tanh function for nonlinearity and the leaking rate α. To extract an output, the reservoir state $x(t)$ is modified by the output weights $W_{out}(x(t))$.

2.4 Usage of Cerebellum-inspired ESN structures in robotics

An overview of how Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] interpret the granular cell clusters and its surrounding structure is shown in a reprint of ”Figure 1” 2.1 by Tan et al. (2022) [13]. This is based on the work of Yamazaki et al.(2007) [16], who theorize that functionally the roles of the granular layer can be regarded as a liquid state generator and the Purkinje cells as the readout neurons. The simplified theoretical structure is shown in Figure 2.1, which shows the core mechanism of the granular cells, where the mossy fibers act as input, the granular cells as the ESN reservoir, the parallel fibers as output, which together with the Purkinje cells effectively create the trainable output layer.

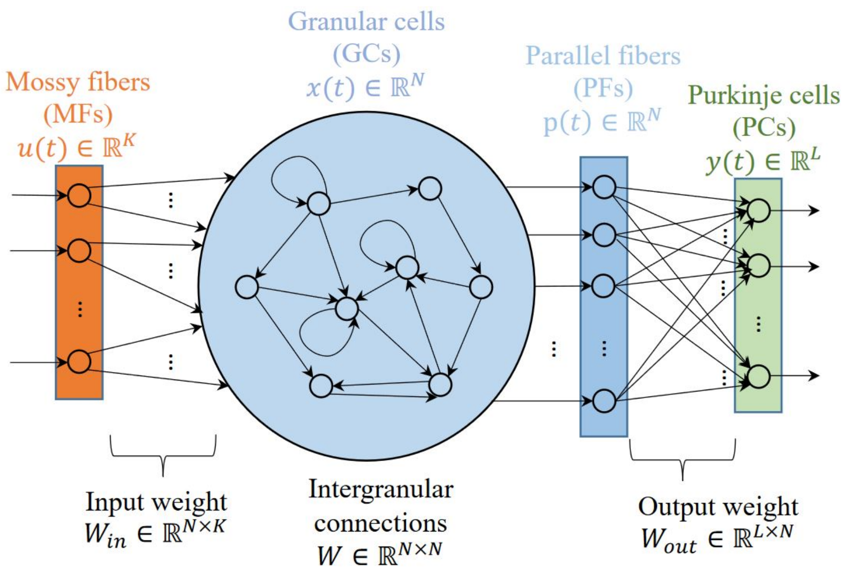


Figure 2.1: The Cerebellum inspired model. Reprint of ”Figure 1” from Tan et al. (2022) [13]

This structure is then assumed to refine the desired movement commanded from elsewhere in the brain. Those desired movement commands are modeled through an MLP, as described in the next section.

2.5 The Inverse Kinematics approximator MLP as the basis of the Cerebellum-inspired architecture in robotics

Both Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] train the ESN component to adjust inputs to an inverse kinematics approximator consisting of an MLP, based on the error in the desired end-effector coordinates y_d.

The MLP controller learns the approximate inverse kinematics of the robot by generalizing on samples of associated angles and end-effector coordinates. Combined with the correction from the ESN component, this forms the full cerebellum-inspired architecture. What information to include in the mapping on which this MLP controller is trained is a design choice, for which Kalidindi et al. (2019) [8] explore what they term an open-loop controller:

$$(y_d(t), \theta(t-1)) \to \theta(t) \tag{2.2}$$

where $y_d(t)$ is the next desired end effector point of the series y_d, and $\theta(t)$ is the previous angle configuration. They compare this to what they term a closed-loop controller:

$$(y_a(t-1), y_d(t), \theta(t-1)) \to \theta(t) \tag{2.3}$$

where they additionally include the previous actual effector position $y_a(t-1)$, which might contain information on hysteresis, the mechanical damping or momentum present due to the previous states. Tan et al. (2022) [13] use the exact same mapping as equation 2.3.

Both Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] gain the training data for this through motor babbling, in which they command random joint angles $\theta(t)$ to the robot and then record the resulting end-effector position $y_a(t)$.

Tan et al. (2022) [13] added a constraint in the random exploration where the next explored angle combination is limited to an angle change of absolute values between 3% to 10% of the actuators' maximum range. Tan et al. (2022) [13] further specify that they recorded 4000 samples to train an MLP with one hidden layer of size 40.

2.6 Adding the ESN layer to modify the MLP layer

This MLP controller may then be corrected by adding the ESN component, which is trained to adjust the desired position $y_d(t)$ input of the MLP by adding its own output $W_{\text{out}}(t-1)$ and creating $\overset{*}{y}_d(t)$.

$$\overset{*}{y}_d(t) = y_d(t) + W_{\text{out}}(t-1) \tag{2.4}$$

Where the ESN output $W_{\text{out}}(t-1)$ is trained using a supervised Hebbian learning rule [40] based on the error position $e_y(t)$ and reservoir state $r(t)$, with the error signal $e_y(t)$ calculated from the difference between the desired position $y_d(t)$ and the actual position $y_a(t)$. A visual example of this process is detailed in the Methods section 3.3.

$$e_y(t) = y_d(t) - y_a(t) \tag{2.5}$$

Kalidindi et al. (2019) [8] update the weights online, after each timestep t, as shown in equation 2.6. They use a learning rate of $\eta = 8$ to modify the Hebbian learning part $\left(r(t) \cdot \text{sign}\left(e_y(t)\right)\right)$, which utilizes a sign function to modify the error signal $e_y(t)$, possibly to reduce the impact of large error magnitudes. Further, they use a cubic regularization term $\lambda W_{\text{out}}(t)^3$, modified by $\lambda = 0.02$.

$$\Delta W_{\text{out}}(t) = \eta \left(r(t) \cdot \text{sign}\left(e_y(t) \right) \right) - \lambda W_{\text{out}}(t)^3 \tag{2.6}$$

Tan et al. (2022) [13] use a similar but simpler online update rule, and a learning rate of $\eta = 0.001$ that is applied when updating $W_{\text{out}}(t)$.

$$\Delta W_{\text{out}}(t) = \eta \left(e_y(t) \cdot r(t) \right) \tag{2.7}$$

Finally, both update W_{out} for the next timestep $(t+1)$.

$$W_{\text{out}}(t+1) = W_{\text{out}}(t) + \Delta W_{\text{out}}(t) \tag{2.8}$$

2.7 Performance of the MLP controller as a stand-alone, compared to adding correction from the ESN component

Kalidindi et al.'s (2019) [8] simulation experiments utilize a kinematic model of a 9-DoF, 3-module soft arm called the Bionic Handling Assistant [41] to trace a circle with the end-effector. The main finding is shown in a reprint of "Figure 2" and "Figure 3b", in Figure 2.2 and 2.3, with the ESN component labeled "Cerebnet". As a stand-alone experimental condition, the open-loop mapping (equation 2.2) and the closed-loop mapping (equation 2.3) MLPs perform quite well, with both being able to achieve near-perfect performances. However, they find a difference in the average performance per condition, caused by variability. The open-loop mapping has relatively high variability, the closed-loop mapping a lower average error caused by lower variability, and the experimental condition that additionally adds the correction by the ESN component performs near perfectly with very low variability.
In one of Tan et al. (2022) [13] experiments, they utilize a real Jaco^2 manipulator with 6-DoF to compare the MLP controller as a stand-alone, adding the correction from the simple ESN, or adding the correction from the leaky integrator ESN. The main finding is shown in a reprint of "Figure 18" and "Figure 20", in Figure 2.4 and 2.5. Here as well, the MLP controller has several perfect performances, but high variability. Adjusting the MLP controller with both the simple ESN and the leaky integrator ESN reduces the variability and leads to consistent near-perfect performance, with the leaky integrator ESN having marginally fewer outlier performances.

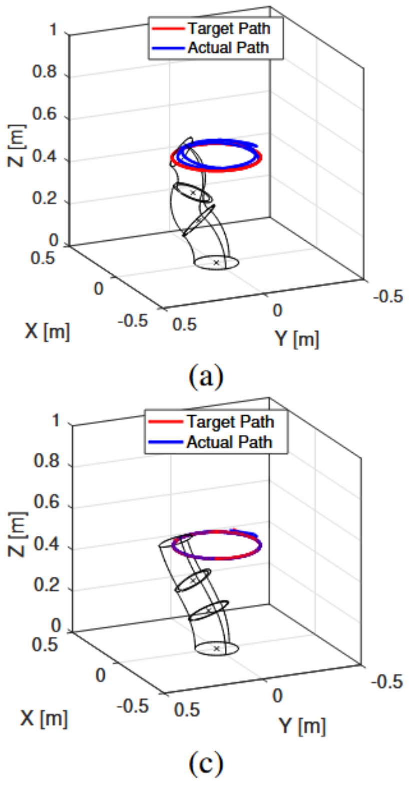


Figure 2.2: Simulation result examples between MLP controller ”open-loop” mapping (a), ”closed-loop” mapping (b), and with ”Cerebnet” as ESN component (c). Reprint of ”Figure 2” from Kalidindi et al. (2019) [8]

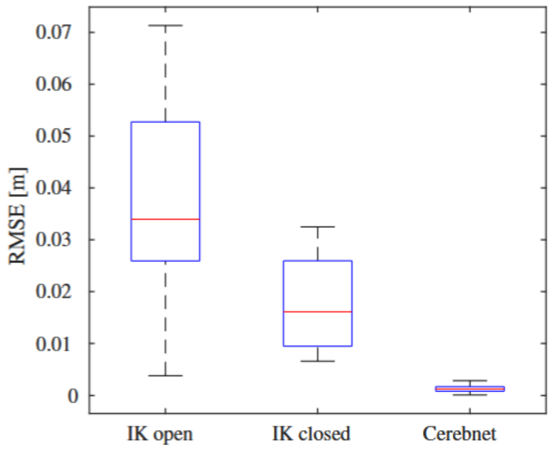


Figure 2.3: RMSE between MLP controller ”open-loop” mapping (IK open), ”closed-loop” mapping (IK closed), and with ”Cerebnet” as ESN component, in simulation experiments. Reprint of ”Figure 3a” from Kalidindi et al. (2019) [8]

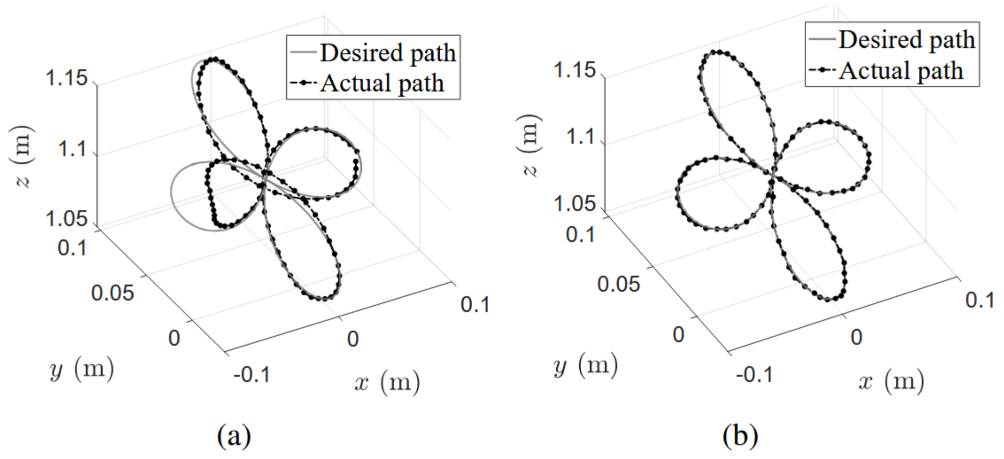


Figure 2.4: Rigid robot experiment results with example of (a) particularly badly performing ”MLP” without ESN correction, and (b) normal performing ”LIN-MFCL” for leaky integrator ESN as ESN component. Reprint of ”Figure 18” from Tan et al. (2022) [13]

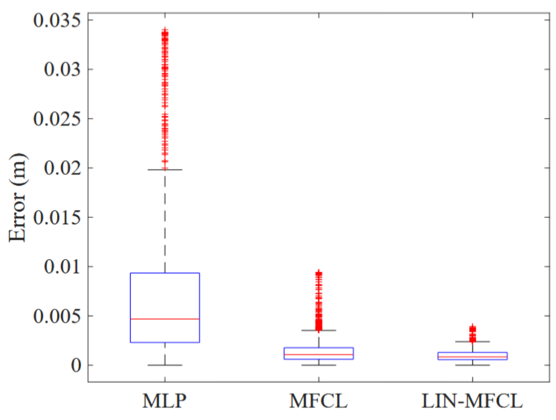


Figure 2.5: Error between MLP controller, ”MFCL” for non-leaky ESN as ESN component, ”LIN-MFCL” for leaky integrator ESN as ESN component, in rigid robot experiments. Reprint of ”Figure 20” from Tan et al. (2022) [13]

2.8 ESN variation reBASICS

The *reservoir of basal dynamics* (reBASICS) by Kawai et al. (2022) [42] is an ensemble of ESN-like modules with particular attributes. They are driven by an input consisting of a pulse before task start followed by zero input, based on the proposed models of neural populations in the cerebellum by Mauk and Buonomano (2004) [43]. Secondly, reBASICS consists of multiple ESN modules k, forming an ensemble, creating a neuronal organization similar to granular cells.
Third, in a trade-off for stability and orthogonality between module outputs, Kawai et al. (2023) [18] find the best parameters for reBASICS modules to be $N = 100$ neurons per module, with a scaling coefficient for the recurrent weights of $g = 1.2$, and a connection

probability $p = 0.1$, which they use for a task with time constant $\tau = 0.01s$. The input to drive the modules is a pulse with a magnitude of 5, for a duration of 51 ms, or 26 discrete time steps. The pulse occurs before the task starts and is followed by zero input. The high scaling coefficient of $g = 1.2$ leads to reservoir activity that gets initiated by the starting pulse input and persists despite the following zero input. The small size of $N = 100$ neurons per module ensures that the activity remains stable.

2.9 reBASICS as ESN component in robotics and expansion of cerebellar-inspired architecture

Kawai et al. (2024) [17] expand on the previous work of Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] by expanding the pipeline to include an additional feedback path to the MLP controller's joint angle output. Furthermore, they test reBASICS as the ESN component. The pipeline can be seen in a reprint of "Figure 1", Figure 2.6, which includes the onset signal activating the reBASICS ESN component as well as starting the trajectory generator, with its output of joint angles to be commanded modified by the ESN component (red arrow) as input into the MLP controller. The MLP controller then passes on the joint angles to be commanded to the proportional-derivative (PD) controller that calculates the torque to be commanded given the joint angle input. Because the PD controller in real robots is often inaccurate to some degree, Kawai et al. (2024) [17] simulate imperfect control, to which the ESN component adds a torque correction (blue arrow). If both feedback paths are applied together, first the Torque correction is trained, and then the position correction.

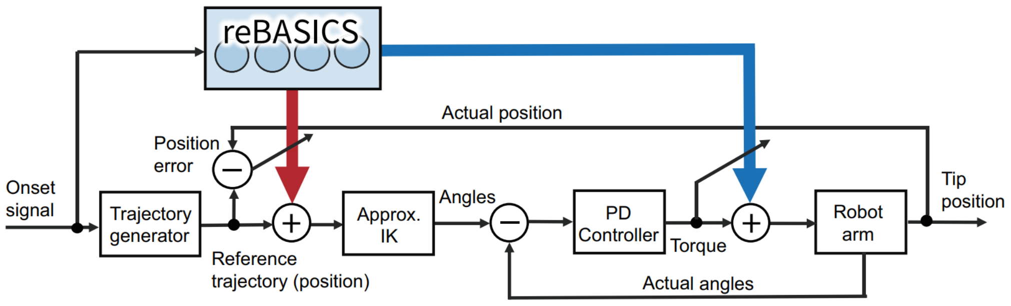


Figure 2.6: Pipeline of Control and Correction. Reprint of "Figure 1" from Kawai et al. (2024) [17]

In experiments simulating a 3 DoF robotic arm moving in a 2-dimensional plane, Kawai et al. (2024) [17] compare the individual feedback paths, as well as reBASICS to an ESN. The results are shown in a reprint of "Figure 5", Figure 2.8. Adding an ESN component significantly improved results, as the MLP controller without correction performed relatively worse than in the previous work of Kalidindi et al. (2019) [8] and Tan et al. (2022) [13], likely because the task was more difficult with the artificially added torque error. Furthermore, reBASICS performed better compared to a leaky integrator ESN that had mirroring hyperparameters, except for being made of 1000 neurons, with each neuron included in the output layer. All feedback paths performed well, with both feedback paths applied simultaneously performing best. This shows that corrections added at the stage of angles or torques are useful and that both feedback paths may be utilized together.

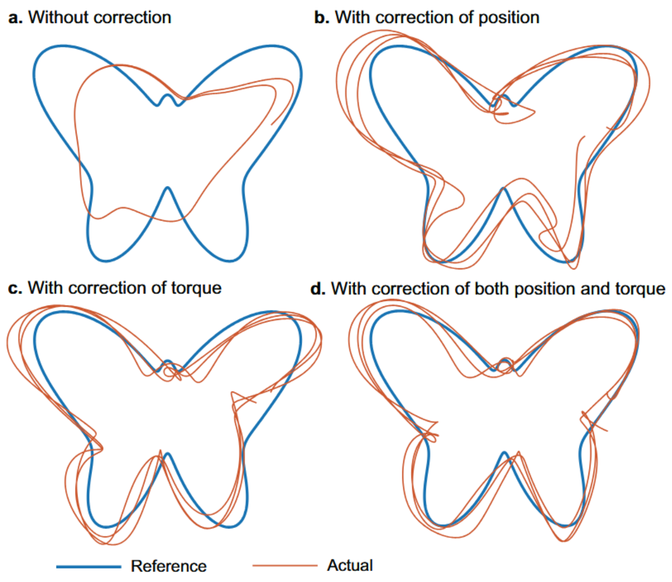


Figure 2.7: Error of MLP controller ”Without correction” (a), with ”Position” correction (b), ”Torque” correction (c), or ”Both” simultaneously (d), for reBASICS, in simulation experiments. Reprint of ”Figure 4” from Kawai et al. (2024) [17]

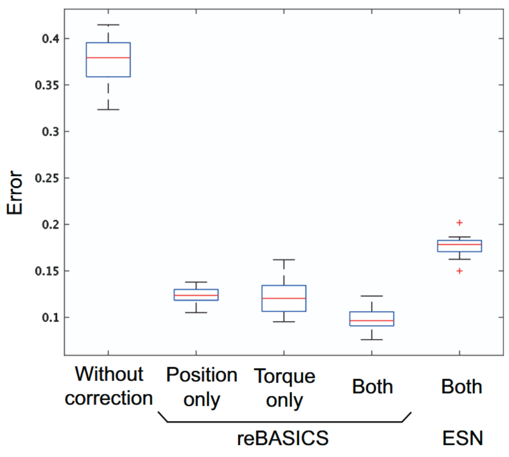


Figure 2.8: Error of MLP controller ”Without correction”, with ”Position” correction, ”Torque” correction, or ”Both” simultaneously, for reBASICS and leaky integrator ESN, in simulation experiments. Reprint of ”Figure 5” from Kawai et al. (2024) [17]

3 Methods

3.1 Details on the robot "AFFETTO"

The robot "Affetto" used in this thesis is a child-like soft robot as presented by Ishihara et al. (2015) [19]. This robot consists of a torso the size of a small child, with similar DoF compared to a human. For the experiments, one of the robot's arms is used. The arm consists of 4 joints. As shown in Figure 3.1, the utilized joints are labeled as "5", "13", "14", and "15". Using the 4 joints means that 4-DoF are available for the end effector to reach a position in the 3-dimensional task space. This enables a sufficiently challenging task, as any end effector position can be reached by multiple angle configurations.

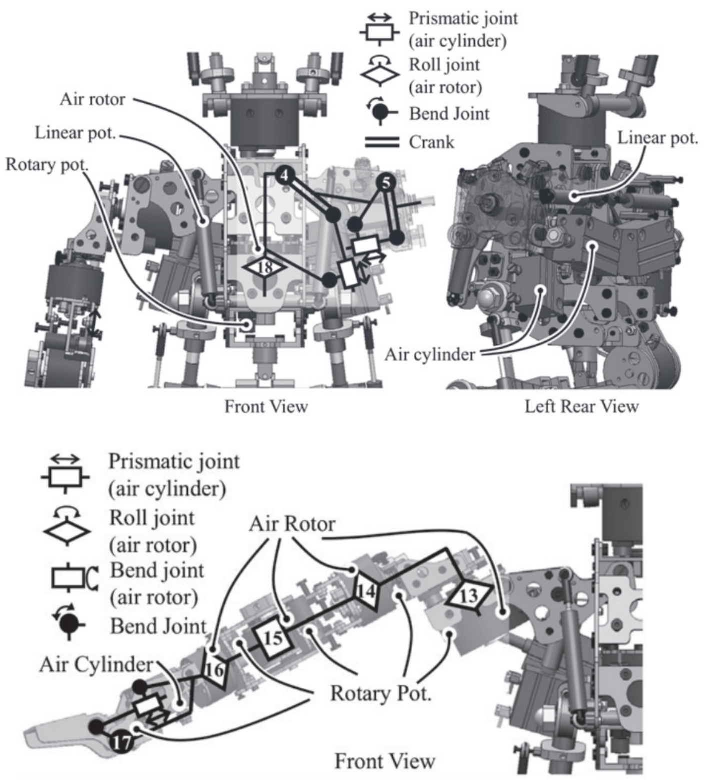
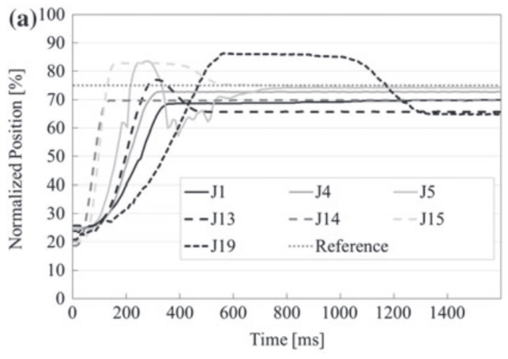


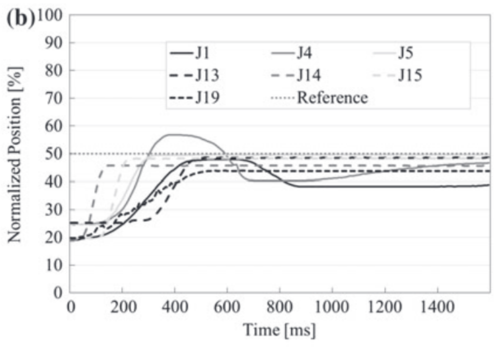
Figure 3.1: The arm of Affeto. Reprinted from Ishihara (2015) [19]

Motor angle commands are given to the robot at a rate of 30 commands per second (30Hz). There is no limitation on the angle commands given, except that the joint angles have to be within 5% and 95% of the possible motor extension for safety reasons, where 0% and 100% are the extreme ends achievable by the joints angle space given by construction. The time taken to achieve the position of the angle command is based on its hysteresis

(previous state). However, angle commands can still be issued and renewed at the rate of 30 commands per second, even if the motor cannot realistically reach the commanded angle before the next command is issued.
The robot's motors function pneumatically, through air pressure. While this offers the advantage of non-rigidity to external forces, their movements are less precise. Subfigure 3.2 shows the precision in movement of selected joints "J1", "J4", "J15", "J13", "J14", "J15", and "J19". The joints have a starting angle of 25%, followed by a continuous command of "75%", where the "75%" desired final position is placed as a visual aid labeled "Reference". Modeled after human joints, the joints have a unique range and placement, which makes the amount of mass to move, as well as the amount of distance to travel, differ. This makes the joints behave differently. Subfigure 3.2b shows the same with a starting angle of 25%, followed by a continuous command of "50%". As can be seen, the control is imperfect, generally achieving a joint angle within 10% of the desired angle.



(a) Changes from 25 to 75%



(b) Changes from 25 to 50%

Figure 3.2: Joint Angle changes after a new command, measured in Percentage of Total Tange. Reprint of "Figure 9" from Ishihara et al. (2015) [19]

3.2 Training the Inverse Kinematics Approximator MLP

The MLP controller was trained on an "open-loop" 2.2 type mapping, except that the previous angle configuration $\theta(t-1)$ was excluded. The main reason for this is that the MLP controller in previous work performed nearly perfectly without correction from the ESN component. This encourages creating a weaker inverse kinematics approximator to enable a meaningful comparison in pipeline variations. Further, if biological plausibility were to be taken into consideration, the control and proprioception of the robot at $30Hz$ implying feedback within $\sim 33ms$ would be unrealistic. Therefore, the mapping used consists only of end-effector position $y(t)$ and the related joint angles $\theta(t)$.

$$y_d(t) \rightarrow \theta(t) \tag{3.1}$$

To gather the data, uniform random angles between 5% and 95% of available range to all four actuators were commanded, and the resulting end-effector positions achieved were recorded. This method of random exploration is meant to satisfactorily explore the coordinates reachable by the end-effector. This may include the discovery of multiple joint angle configurations that result in the same end-effector position due to the redundant degrees of freedom. Further, each data point with a similar end-effector position may have a different random starting position before it, which leads to the data reflecting variability in hysteresis and its effect on the end-effector position. However, no quality filter for the training data is applied as is, for example, explored by Hlavac (2020) [7]. This means that sampled end-effector positions may not be uniform.

3.3 Training the two different ESN correction component feedback paths

Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] use the ESN component to adjust the input of the MLP controller. To visualize this process, an example from later experiments is presented in Figures 3.3 and 3.4. Figure 3.3 shows the desired or target end-effector coordinates in green. Those coordinates are given as input into the MLP layer, which provides the joint angles. The joint angles are commanded to the robot, which then achieve the end-effector coordinates in blue. The errors between the corresponding points in the movement are visualized as error lines.

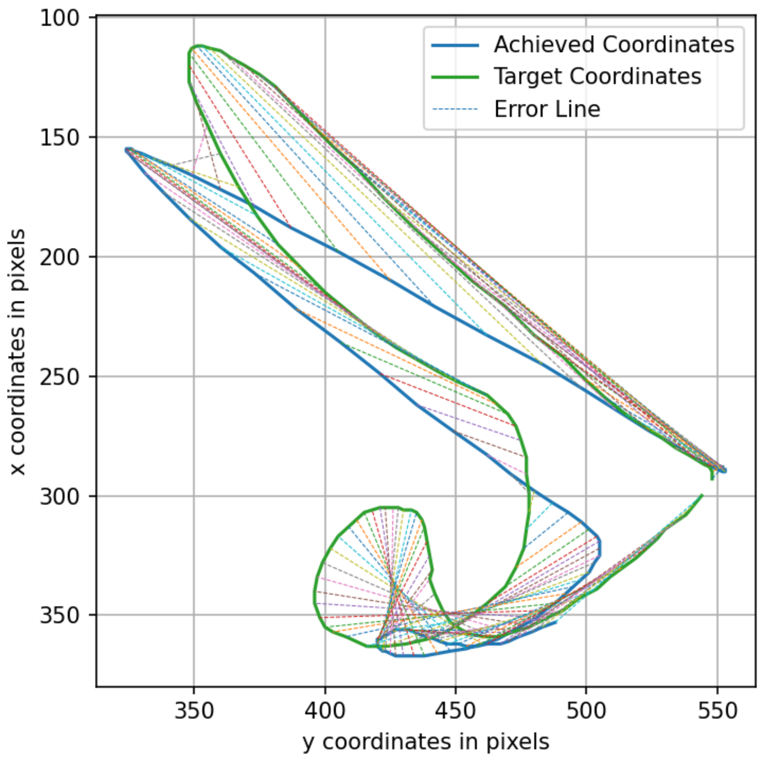


Figure 3.3: Target Coordinates (green) vs Achieved Coordinates (blue), with error visualized as lines connecting the respective points

Put formally, the ESN component is trained to adjust the target coordinates or desired position $y_d(t)$ input of the MLP by adding the ESN component's output $W_{\text{out}}(t)$ and creating the adapted coordinates $\overset{*}{y}_d(t)$.

$$\overset{*}{y}_d(t) = y_d(t) + W_{\text{out}}(t) \tag{3.2}$$

Where the ESN component output $W_{\text{out}}(t)$ is adapted through gradient descent using the error signal $e_y(t)$, which is calculated from the difference between the desired position $y_d(t)$ and the actual position $y_a(t)$.

$$e_y(t) = y_d(t) - y_a(t) \tag{3.3}$$

In effect, the ESN component is trained to reproduce the error, which is then used to adjust the input coordinates given to the MLP layer. Figure 3.4 visualizes how, using the error lines calculated from Figure 3.3, the target coordinates are adapted in the opposite direction of the previously occurred error. This can be repeated for multiple training cycles, where the adapted target coordinates are then fed into the MLP layer again, producing joint angles that when commanded to the robot may again produce errors, which leads to this cumulative adaptation process being repeated until performance stops improving.

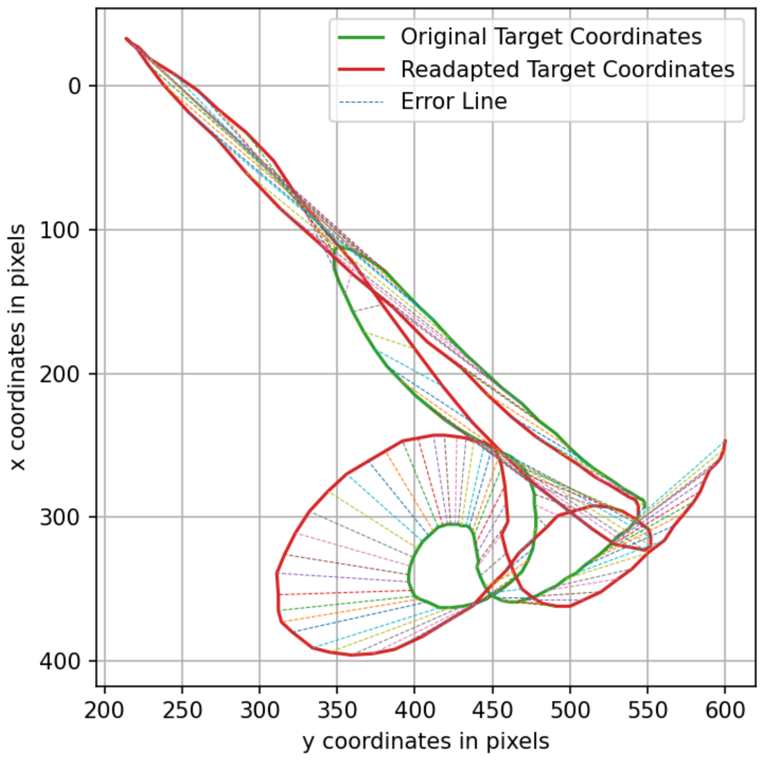


Figure 3.4: Target Coordinates (green) vs Achieved Coordinates (blue), with error visualized as lines connecting the respective points

Another vector to improve control of the robot is to adjust the MLP controller's joint angles output, to adjust for imprecise Torque calculations or effects caused by hysteresis.

Figure 3.5 shows details on the complete commanding structure of the robot, including a visualization of the two correction mechanisms which are shown with the input correction in *red* and the angle output in *blue*. This is similar to Figure 2.6 from Kawai et al. (2024) [17], except that the joint angle output is corrected directly based on the error in the achieved angle, instead of at the level of torque. The command structure includes the ESN correction component labeled as reBASICS, and the approximate Inverse Kinematics. Further, instead of first completely training the ”Angle feedback” path and then training the MLP controller’s ”MLP feedback” path in the ”Both” together experimental condition, as done by Kawai et al. (2024) [17], the two paths are trained continuously and simultaneously.

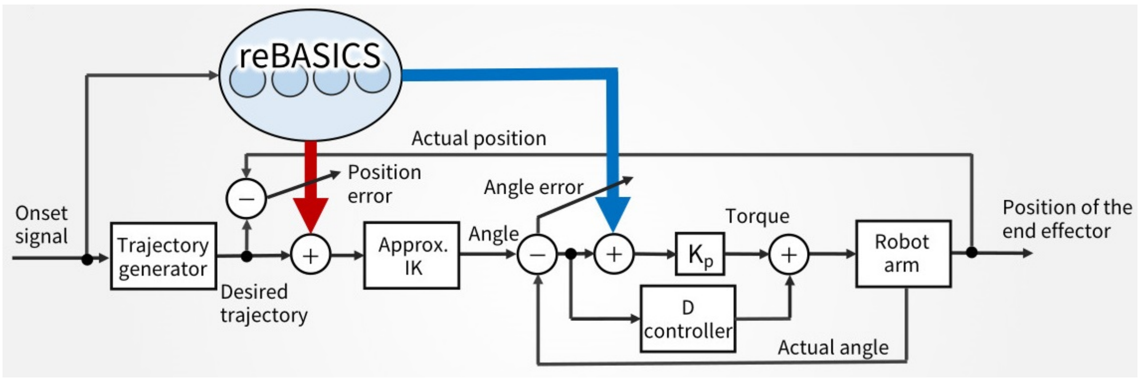


Figure 3.5: Control pipeline.
Reprint of lab internal presentation slide by Yuji Kawai

3.4 Hyperparameters ESN components

The ESNs used for this experiment are based on Tan et al. (2022) [13], and Kawai et al. (2023) [18]. While Tan et al. (2022) [13] use only a single ESN, to mirror conditions with the ensemble-like reBASICS, an ensemble of ESNs is used.
This makes the implementation for the ESN ensemble and reBASICS differ only in hyperparameters and input. The implementation is based on Lukosevicius (2012) [39], and utilizes the python code ”Simple ESN” (Chevallier 2020 [44]) and ”keras” (Cholet et al. [45]), as presented earlier in Equation 2.1
While often ESNs are used in combination with linear regression, the robot control pipeline is an online adaptive training task, which makes gradient descent more suitable. The output weights W^{Out} for both ensembles are updated for each module $k \in M$ after the completion of a full movement with time points t in T, using gradients accumulated across the entire sequence in a batch update. The error is calculated as a Mean Square Error (MSE). In preliminary exploratory experiments, a simple gradient update was found ineffective and consequently an Adam optimizer (Kingma, Diederik, and Jimmy 2015 [46]) is included.
To equal the comparison for the experiments, the same amount of total neurons per ensemble was used. The reBASICS configuration used by Kawai et al. (2023) [18] used 800 modules with 100 neurons each. To match the total amount of neurons, the ESN

ensemble was set to 80 modules with 1000 neurons each, which is arguably suboptimal. The hyperparameters of the two ensemble types are presented in table 3.1.

Parameter	ESN ensemble	reBASICS
Module amount M	80	800
Network size N	1000	100
Leaking rate α	0.5	0.5
Scaling coefficient g	0.9	1.2
Connection probability p	1	1
Recurrent Weight $W^{\text{Rec}}_{k,ij}$ initiation	$\mathcal{U}(-0.5,\ 0.5)$	$\mathcal{U}(-0.5,\ 0.5)$
Input $u(t)$	continual	pulse, then 0
Total readout neurons per ensemble	80	800

Table 3.1: Hyperparameters of the ESN ensemble and reBASICS used in experiments

Kalidindi et al. (2019) [8] drive their ESNs with $[\Delta y_d(t), \Delta\theta(t)]$, the desired end-effector position displacement $\Delta y_d(t)$ and change in joint angles $\Delta\theta(t)$, which Tan et al. (2022) [13] expand to include joint angles $\theta(t)$. All of those are dependent on the previous state, and therefore adjusted during the training of the cerebellar-inspired structure, as the ESN component adjusts the MLP controller. This means that the desired end-effector position displacement $\Delta y_d(t)$, change in joint angles $\Delta\theta(t)$, and joint angles $\theta(t)$ at time point t change between training cycles. As a consequence, the ESN component receives different inputs in each training cycle. This should result in different reservoir states $r(t)$ for each time point t, potentially invalidating the output weights that were trained based on the reservoir states $r(t)$ occurring during the previous training step. The reason this still works for Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] may be that their base-layer MLP is quite strong as a stand-alone, which leads to minimal error and the adjusted joint angles remaining similar across training cycles, resulting in only minor variations in reservoir states that don't invalidate previously learned output weights. In any case, their experimental results show that the learned output weights contribute valuable corrections. However, it seems likely that bigger corrections of the MLP controller will be necessary in experiments with the soft robot Affetto. Those bigger corrections would result in bigger changes of the joint angles $\theta(t)$ between training cycles. Using those as input into the ESN component might change reservoir states to the point of completely invalidating the output weights learned from the previous training cycle, leading to model collapse. Therefore, I made the decision to include two versions of the standard leaky integrator ESN ensemble, where one is driven with input $u(t) = [y_d(t)\,;\,\theta(t)]$ similar to previous work, and one is driven with the input $u(t) = y_d(t)$, containing only the unchanging desired coordinates $y_d(t)$. The input is normalized to be in range $u(t) \in (-1, 1)$, and preceded by a warm-up of "0" inputs for 8 discrete time steps with the output resulting from the warm-up discarded.
The input of reBASICS consists, like in previous work, of a pulse with a magnitude of 5, for a duration spanning 26 discrete time steps (858 ms). The pulse occurs before the task starts and is followed by zero input. 108 discrete time steps are discarded before the

output is utilized, to allow the reservoir to reach stable dynamics. The pulse input kick-starts the reservoir activity, with the scaling coefficient for recurrent weights of $g = 1.2$ keeping reservoir activity continuous in the face of the following zero inputs. However, modules' outputs may still converge to a fixed point. Therefore, modules were resampled if they displayed an output with a standard deviation of less than 0.05.

4 Experimental Setup and Preliminary Experiments

4.1 Camera Setup

The camera used for tracking is an Intel ”Realsense d435i”. This device is capable of 3-dimensional recording through stereovision from a stereo camera pair and an infrared projector. It was positioned in front of the robot ”Affetto” at a distance of approximately 1.5 meters. To create an end-effector that is detectable by the camera, a table-tennis ball was used to signify the end-effector's position. The ball is glued onto a plastic holder that could be stuck onto the robot's arm.

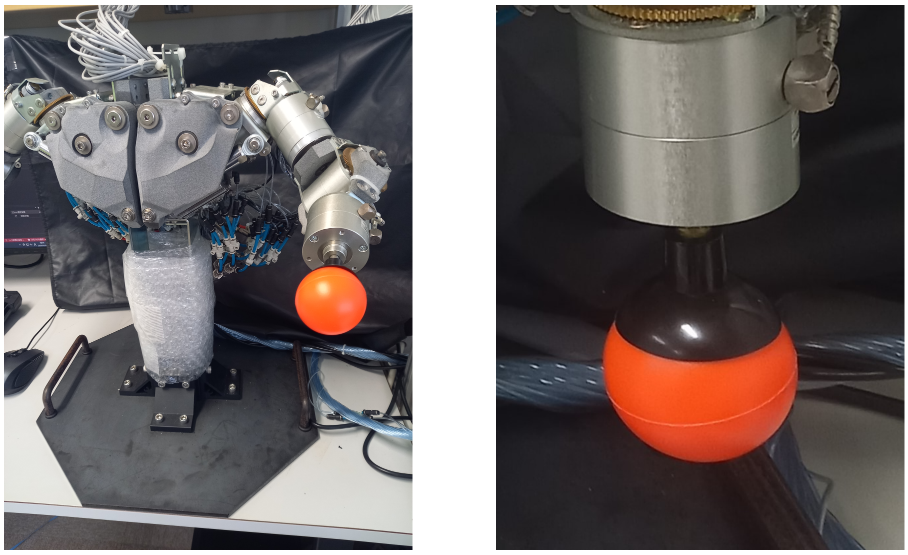

(a) Frontal view of the ball-shaped end-effector (b) Detail of the end-effector attachment

Figure 4.1: The end-effector of Affetto's arm

The camera was operated using its internal software, which is functionally connected to the open-source software ”ROS2” [47]. A python code that utilized this functionality was used, which included object detection of the end-effector, as well as a projection from horizontal and vertical pixel data, and depth measurement, into 3-dimensional real space measurements.

4.2 Gathering Data for the inverse kinematics approximator through Motor Babbling

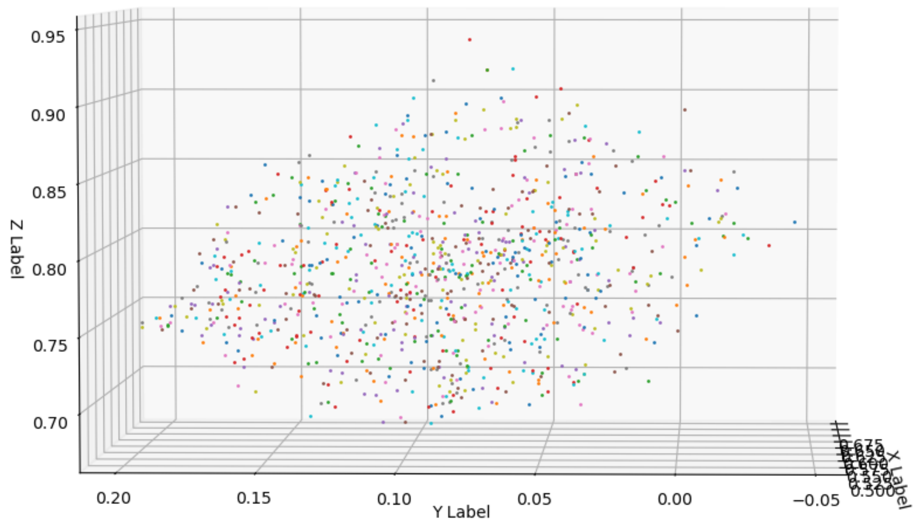
To gather kinematic data, the 4 motors of the robot's arm were used. To generate the mapping of $y(t) \rightarrow \theta(t)$ 3.1, uniform random angles between 5% and 95% of their respective available range were commanded to all four motors, and the resulting end-effector positions achieved were recorded.
When given a command for an angle configuration, Affetto may take up to 600 ms to reach its intended configuration, and then an additional second to "settle in" (Ishihara and Asada 2015 [19]). Therefore, the same command is given continuously for 2 seconds, effectively maintaining the same respective air pressure in the motors for that duration. The achieved end-effector position is then recorded and paired with the motor angle information as a data point.

Algorithm 1 Generating inverse kinematics data points

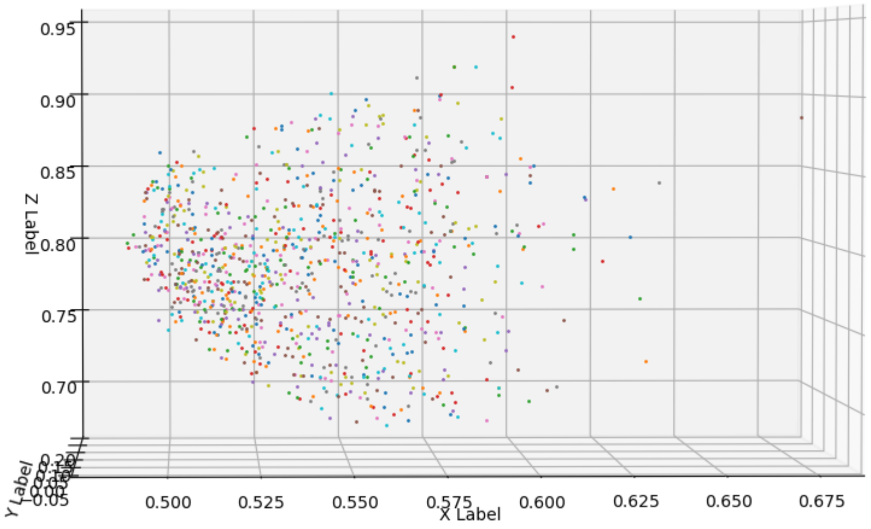
```
while gathering data points do
    if 100 seconds passed, (50 datapoints gathered) then
        Reset the robot and wait 5 seconds, to prevent any potential fatigue
    end if
    Joint Angles $\theta_{1-4}$ get assigned new random Angles $\theta_{1-4} \in [5\% \sim 95\%]$
    Wait 2 seconds
    Read in end-effector position coordinates $(x, y, z)$
    Save end-effector position $(x, y, z)$ and angles $\theta_{1-4}$ as data-point
end while
```

4.3 Analysis of data gathered for inverse kinematics approximator

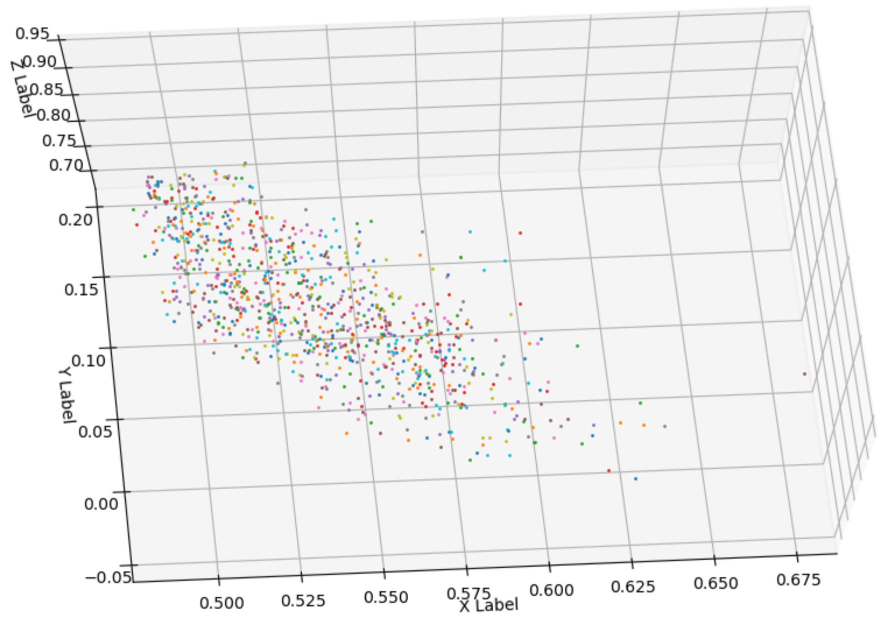
To get a sense of the arm's reach, 2000 data points were gathered initially. The distribution is visualized in Figure 4.2. The visualization uses the camera's internal software, which measures the distance between coordinates in meters, and labels the depth axis X, the horizontal axis Y, and the vertical axis Z. The area covered by the end-effector resembles an oval with a slight tilt towards the left. This is consistent with the expected area of reach for the left arm of the robot. However, within the X-axis encoding depth, we can visually spot a few unrealistic outliers, suggesting that the depth measurement may not be entirely reliable.



(a) Frontal view



(b) View onto the left side of the robot



(c) View from above and onto the left/frontal side of the robot

Figure 4.2: Visualization of 2000 data points (labeled with X-axis as depth, Y-axis as frontal horizontal, and Z-axis as vertical, in meters)

4.4 Training the MLP to generate inverse kinematics

The gathered data is now used to train an MLP for Inverse Kinematics. This used Keras [45]. The coordinates are given as inputs, and the MLP learns to output the angles associated with the respective coordinates. First, a model with a hidden layer of 20 nodes was trained which used the Adam optimizer, as it is an established standard and requires little to no learning rate tuning [48], and 'mean absolute error' for the loss function. The batch size is set to 100. To determine the appropriate number of epochs needed for successful training, the epoch size must be chosen so that training does not stop too early nor allow overtraining. To test this, the loss per training epoch are visualized in Figure 4.3. The loss is the MSE calculated as

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

with y_i being the true angle configuration and $\hat{y}_i$ the predicted angle configuration *percentage* of total available range. The figure shows that the model reaches peak performance around epoch 100, but also does not degrade afterward. Given this, in the next comparison between different model sizes, the epoch size for training for all models was set to 200 epochs.

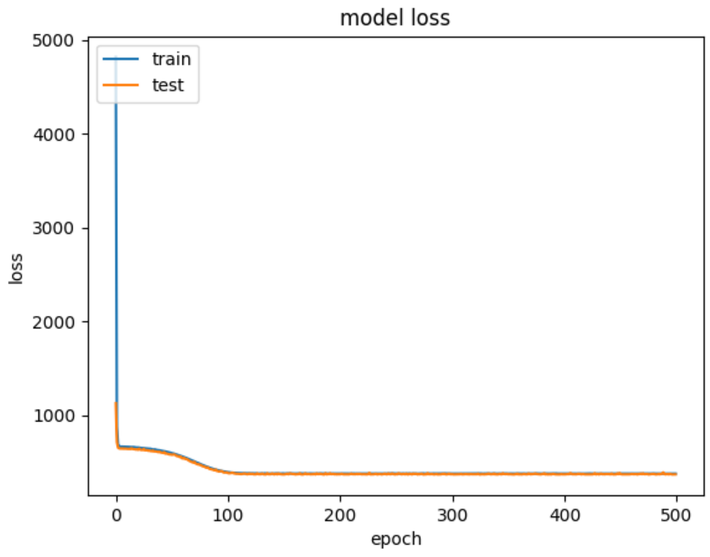


Figure 4.3: Training-loss and Validation-loss during training

Figure 4.4: Visualization of 20-node model training)

To ensure an optimal MLP size, differently sized MLPs were tested. The smallest MLP has 20 nodes in the hidden layer and was compared to MLPs with 30 nodes, 40 nodes, 60 nodes, and an MLP with two hidden layers of 20 and 10 nodes respectively. To gain confidence in the results, each model was trained 10 times in a 10-fold cross-validation for 200 epochs. It would have been proper to use early stopping, but this was not done because of a lack of rigor in the pre-experiments, where at face value overtraining didn't

appear to occur and 200 epochs were enough for all models to converge successfully. Further, no testing data was set aside as testing is done through commanding the robot using the MLP models.
The MLP models were applied to control the robot with each model of a different size commanding the robot's movement using an algorithm nearly identical to algorithm 1, but instead of random joint angles, the joint angles produced by the MLP models were commanded. The resulting coordinates are then compared to the goal coordinates. Figure 4.5 shows the average distance between the static achieved and desired points in centimeters, with the standard deviation represented as error bars. All models achieve an average error of around $8cm$ and do not differ significantly in performance. In conclusion, increasing model complexity does not improve performance.

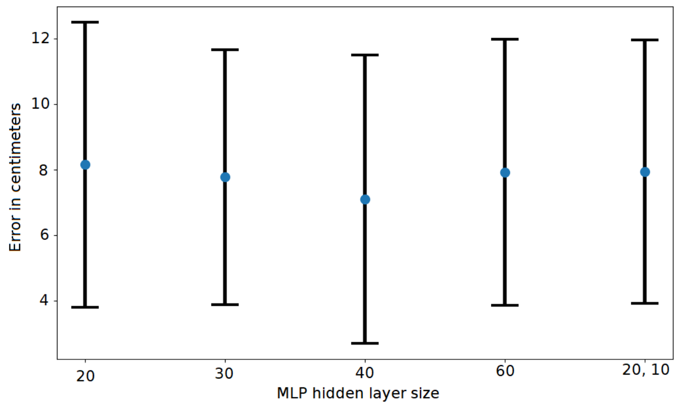


Figure 4.5: MLP model accuracy in commanding robot. STD and Distance from goal in cm of 10 MLP models per condition, of size 20, 30, 40, 60, and (20, 10)

Figure 4.6 shows a visualization of 50 data points from the training data in red, and the corresponding coordinates achieved by the output of an MLP of size 20. The corresponding target and achieved coordinates are connected with a line. The point visualization indicates that the model's joint angle output has a bias towards the center and possibly to the top-right. Both the average error of $\sim 8cm$ and the visualized performance clearly show that the MLP controller performance performs worse than the often near-perfect performance in previous work.

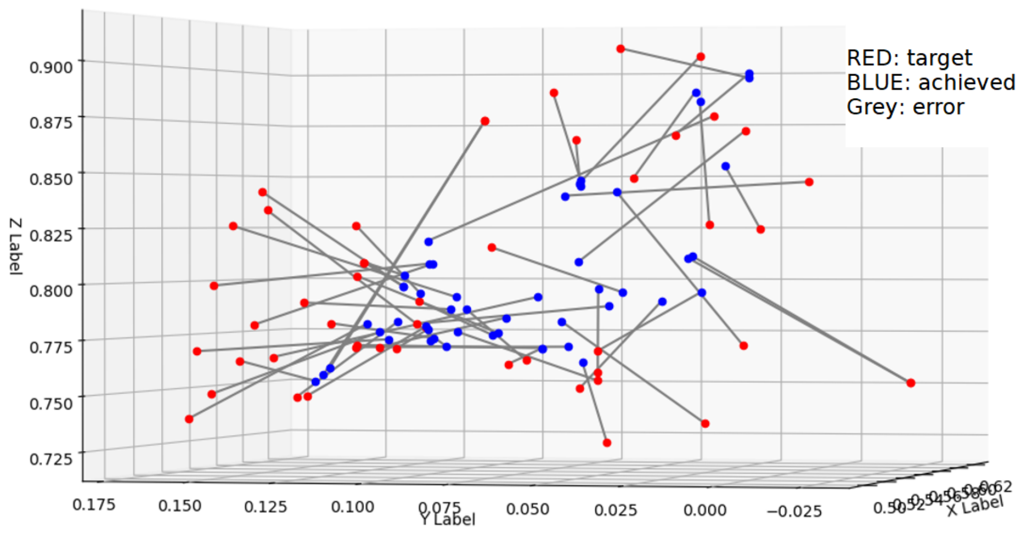


Figure 4.6: MLP model (size 20) accuracy in commanding robot.
Frontal view. 50 data points visualized.

4.5 Removing the depth spatial dimension because of unreliability

At that point in the pre-experiments it was discovered that the data from the depth spatial dimension was unreliable to the point that removing it is justified. For a detailed exploration of that see the Appendix .1. A reduction in position information only increases the difficulty in commanding the robot, with only two dimensions of position feedback available while still using all four motors. Therefore the premise of the project is not affected, which is to test the cerebellar-inspired architecture in a challenging use case.
A triangulation estimation between measurements of the camera internal software's data and the raw camera data shows that the distance between two pixels in the 2D data is roughly equivalent to $0.087cm = 0.87mm$. This estimation is based on the difference in maximum and minimum values of the horizontal and vertical real space coordinates (spanning $(\sim 27cm$ and $\sim 25cm)$ respectively) from the previous data, and triangulating them with the difference in maximum and minimum of the horizontal and vertical pixels from newly collected data (spanning $(\sim 304pixels$ and $\sim 294pixels)$ respectively).

$$27cm \approx 304pixels \rightarrow 0.089cm \approx 1pixel \text{ (vertical axis)}$$
$$25cm \approx 294pixels \rightarrow 0.085cm \approx 1pixel \text{ (horizontal axis)}$$

With the exclusion of the depth measurement, the MLPs had to be retrained using only the X-axis and Y-axis data. The exact same parameters as before were used with MSE loss. The results can be seen in Figure 4.7.
Additionally, as the previous comparison between MLPs showed no meaningful difference, additional smaller MLPs were included to observe at what size performance degrades. That's why additional MLPs with hidden layers of sizes 3, 5, and 10 were in-

cluded. Similar experimental conditions as before were used, a ten-fold cross-validation for each MLP size and data input form, but this time with early stopping if improvements did not happen for 20 training cycles. There is no significant performance drop when omitting the Z-axis depth measurement, confirming that the depth measurement is too unreliable to be included. Furthermore, there is no benefit in increasing the MLP size beyond a hidden layer of 10 neurons. As a result, for the next parts of the experiment, an MLP with a hidden layer size of 10 using only X- and Y-axis data was used.

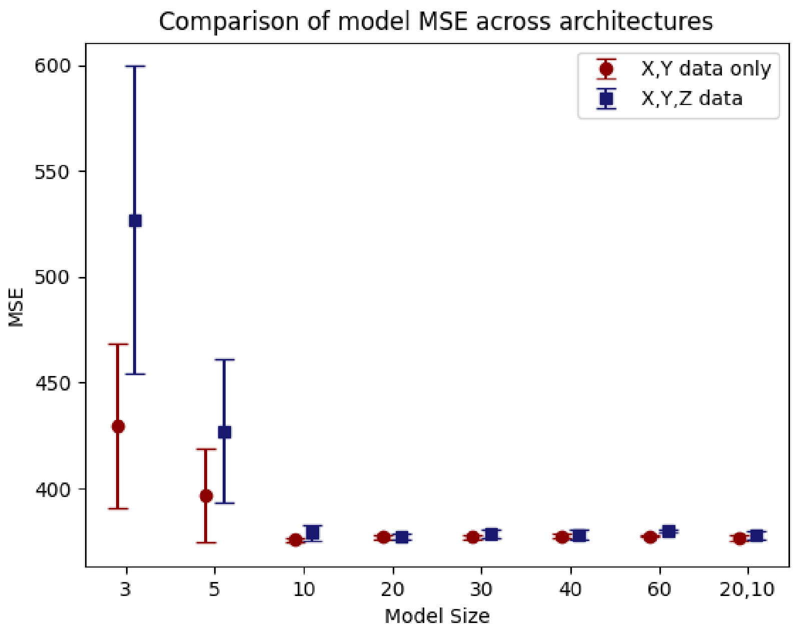


Figure 4.7: Total available joint angle range measured in *percentage* MSE and STD of 10 MLP models per condition, with a hidden layer size of 3, 5, 10, 20, 30, 40, 60, and (20, 10)

4.6 Creating a continuous movement target

The next step is to create a continuous target movement. The self-imposed criteria for generating a target movement are to ensure appropriate complexity and fully utilize the range of motion of all joints. Commanding unique sine waves for each joint allows smooth swinging between the two extremes of each joint's range of motion, which should result in an appropriate target movement.
The first joint (shoulder bending/lifting the arm) and the second joint (shoulder rotating the arm forward/backward) were set to complete a full sine curve off-phase to each other, and the third joint (rotating the elbow) and the fourth joint (bending the elbow) to complete two sine waves. The first two joints are set to move slower, as they affect the entire arm more heavily with their movements compared to the latter two joints. The sine curves are shown in Figure 4.8. The starting and ending positions are the same to allow a seamless repetition of the movement. Through trial and error, 4 seconds was found to be an

appropriate time to complete one loop. With the robot receiving 30 commands per second, 120 commands were generated for each joint's actuator. The shoulder lifting joint (joint 1) started fully lifted, the other joints at 50% of their range of motion.

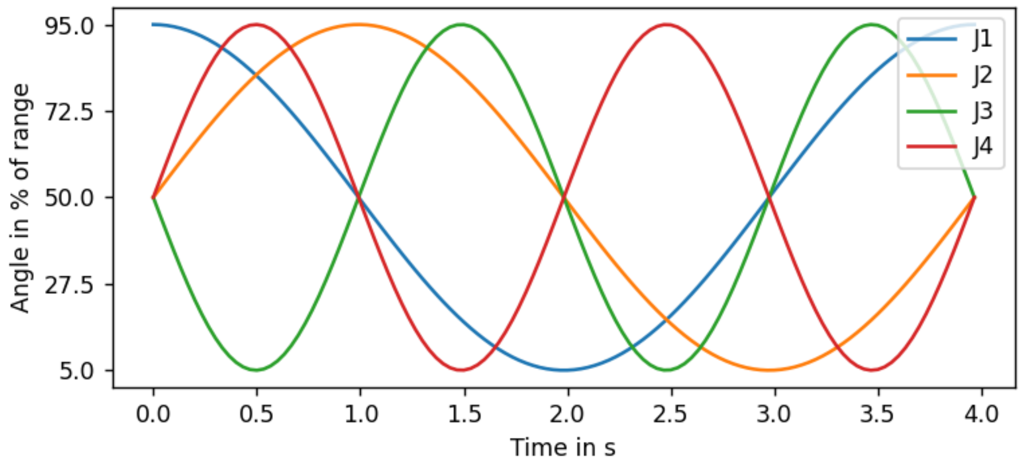
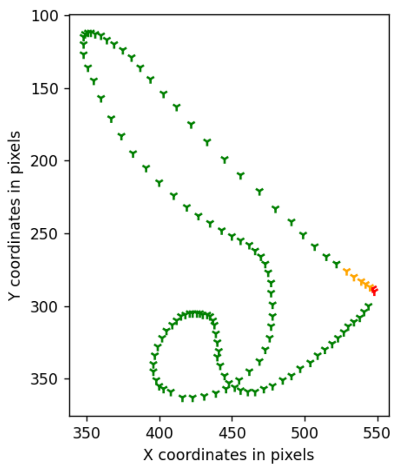
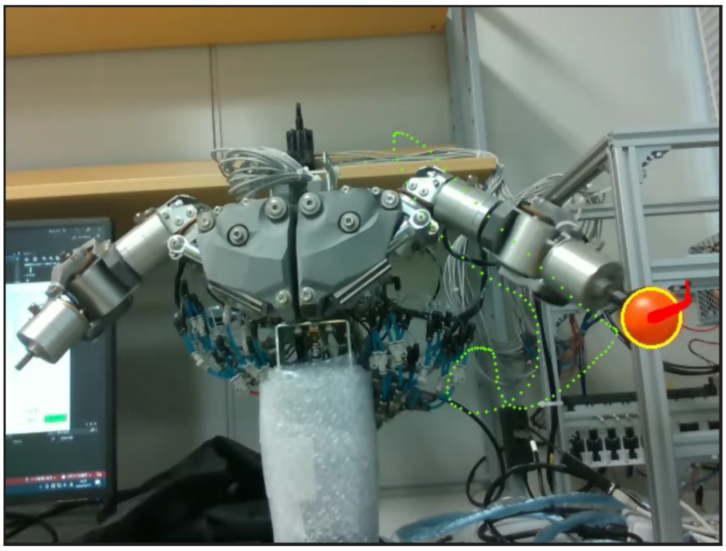


Figure 4.8: Generated sines to generate target movement

The robot executed five loops (20 seconds total), with the first four loops discarded as a warm-up, and the final loop's recorded end-effector positions (120 total) set as the target movement. Figure 4.9a visualizes the movement, with starting coordinates in red and orange. As points are recorded at $30Hz$, spacing distance between points indicates movement speed, with a higher spacing distance caused by a higher relative speed of the end-effector. Notably, the movement has two parts that are relatively fast in the upper straight movement areas, while the small circle at the bottom left is relatively slow.
Due to the inherent imprecision in the movement, the 5 completed cycles were not identical to each other, and the movement of one cycle does not end perfectly where it started. This can be seen by the distance between the starting and ending end-effector position in Figure 4.9a, with Figure 4.9b showing the movement plotted onto the camera data. To ensure a target movement that can be repeated seamlessly, the last 5 data points were manually changed, by incrementally increasing the distance from the previous data point by an additional 1 pixel on both axes towards the starting point.
Previous work by Kalidindi et al. (2019)[8] and Tan et al. (2022) [13] uses simulations or electric motors, for a singular repetition of a movement. However, to allow potential characteristics typical of the pneumatic actuators to take effect, such as motor hysteresis, the target movement was extended to 5 continuously repeated loops, which extends the movement to $120 * 5 = 600$ data points, or 20 seconds.
To get an estimation of inherent error caused by pneumatic inaccuracies, the sinusoidal joint angle commands from Figure 4.8 that generated the target movement were commanded 5 times continuously. This was done a few times, and the MSE was then calculated based on the difference between the resulting end-effector positions and the cleaned and extended target movement. Typical values would range between $MSE \approx [70 \sim 120]$, which is roughly equivalent to a standard deviation of $[8.3 \sim 11.8]$ pixels per data point, or $[0.72 \sim 1]$ centimeters.



(a) Target movement. Red visualizes the starting point, orange the immediately followed time steps



(b) Target movement mapped onto the camera data. Image taken from a video tracking the movement.

Figure 4.9: Target movement generation results

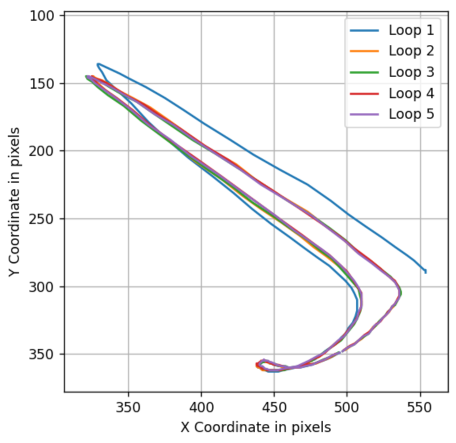
4.7 Using the MLP controller to trace the target movement

The target movement is first used to evaluate the MLP controller of the cerebellar-inspired architecture as a stand-alone. This means giving the target movement coordinates for all $5 * 120 = 600$ time steps to the MLP as input, and using the output consisting of the 4 joint angles for the 600 total time steps to command the robot.

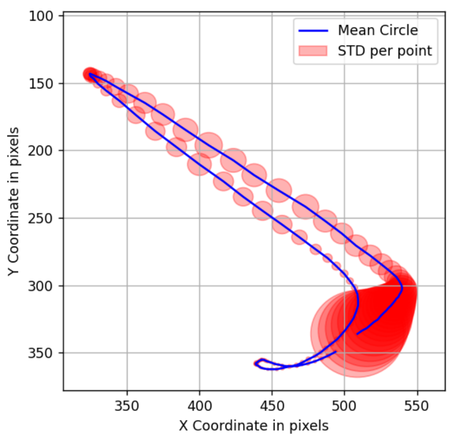
Figure 4.10 shows the resulting movement. Subfigure 4.10a plots the complete movement. The first loop starts differently, but the other four are similar. This difference is because the robot is allowed to settle on the commanded starting point before starting the movement. In the subsequent 4 loop repetitions, the ”starting points” are commanded for just $\sim 1/33$ of a second as part of the continual movement commands, and are influenced by hysteresis from the preceding loops.

Subfigures 4.10b and 4.10c show loop averages and standard deviations per point as red circles. Including the first loop skews the standard deviation of the initial points. However, excluding the first loop in Figure 4.10c shows that the movement is consistent, with slightly increased deviation during the faster parts in the upper area.

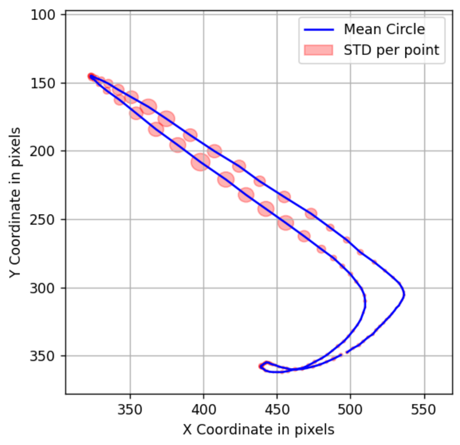
To observe the potential influence of repeated movements on the execution of the movement, the development of the MSE compared to the target movement is plotted as well. For this, the complete movement consisting of the 5 loops was repeated multiple times, with 5 second rests between complete movements, as would be done when training the ESN component. The result is shown in Figure 4.10e. A reduction in MSE can be seen after the first two movement completions, which suggests that there is a warm-up effect. As most data to train the MLP was collected in a warmed-up state of the robot, it makes sense that the output of the MLP is more accurate after a warm-up.



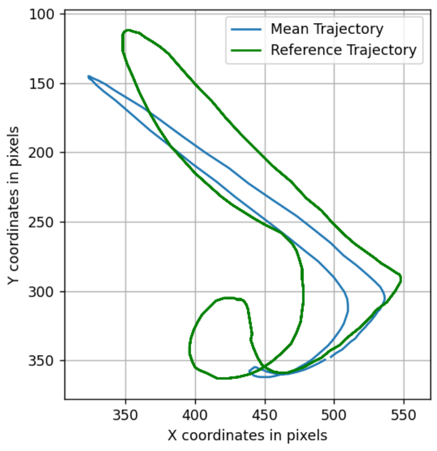
(a) All 5 loops together



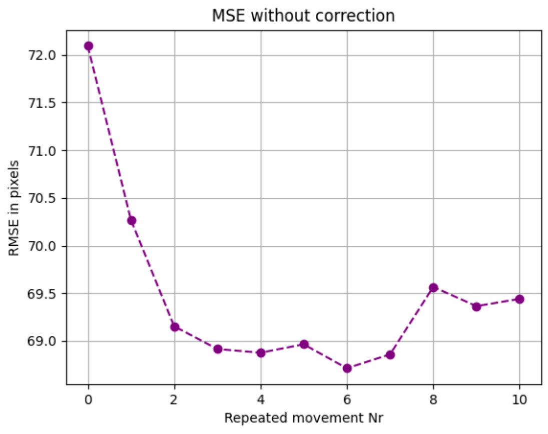
(b) 5 loop average with STD per point



(c) Excluding first loop, 4 loop average with STD per point



(d) Last 4 loop average (blue) plotted against reference trajectory (green)



(e) MSE of movement in relation to target movement, over repeated executions with ∼ 5 second break

Figure 4.10: MLP generated movement, 600 data points with 120 points per loop for 5 repeated continuous loops

4.8 Including the ESN-type correction layer

With the target movement established and the MLP basis performing reasonably well, the next step is to add the ESN component for correction. The algorithm for this is shown in Algorithm 2. To recap, the output of the ESN component $ESNoutput$ has the shape $(6,600)$, corresponding to the 6 neurons of the output layer for the 2 spatial dimensions and the 4 joint angle dimensions, over the 600 hundred time points t that make up a complete movement.
In preliminary experiments, a simple gradient descent was used. With that, learning often critically failed or was very slow. To overcome this, an Adam optimizer was implemented, and the ESN component was trained for multiple epochs for each completed movement, rather than only once.

Algorithm 2 Extended version: training the ESN component to correct a movement

```
Require: MLP base layer MLP component, ESN ensemble layer ESN component, target coordinates $y_d$. epoch length $E$
 1: while training not converged do
 2:     Get $ESNoutput$ from ESN component
 3:     Adjust $\hat{y}_d \leftarrow y_d - ESNoutput$
 4:     Get angles $\theta$ from MLP component mapping $\hat{y}_d \rightarrow \theta_d$
 5:     Adjust $\hat{\theta}_d \leftarrow \theta_d - ESNoutput$
 6:     Command robot with $\hat{\theta}_d$
 7:     Record achieved coordinates $y_a$ of completed movement
 8:     Record achieved joint angles $\theta_a$ of completed movement
 9:     for epoch = 1 to $E$ do
10:         Update Weights $W_{Out}$ of ESN component with Adam
11:     end for
12: end while
```

With Adam included in the algorithm 2, learning was successful, as shown in the next section. Through trial and error, an epoch total of $E = 50$ per completed movement was found to be suitable and was used for all following experiments. In effect, the high amount of epochs per training cycle led to the ESN output being similar to if training had been done with linear regression.

5 Final experiments results and Analysis

5.1 Final experiments

The final experiments focus on a comparison between feedback pathway options, namely the MLP controller input adjustment (MLP feedback), the MLP controller output adjustment (Angle feedback) and both simultaneously, followed by a comparison between ESN component variations reBASICS or the two standard leaky integrator ESN ensembles with differing input.
Due to limited time, a complete comparison of all feedback pathways for all ESN component alternatives was not done. Instead, first the feedback pathways were explored to determine the best one, and then all ESN component alternatives for that feedback pathway were compared. Since in preliminary exploratory experiments the ESN component using reBASICS seemingly performed best, it was used as the basis to determine the ideal feedback pathway. In a trade-off between cost of time and confidence in the results, training for each feedback path experimental condition was done 3 times, with a newly randomly initialized ESN component each time. Further, training is stopped after failing to improve on the lowest achieved MSE 5 consecutive times. The results can be seen in Figures 5.1, 5.2, and 5.3.

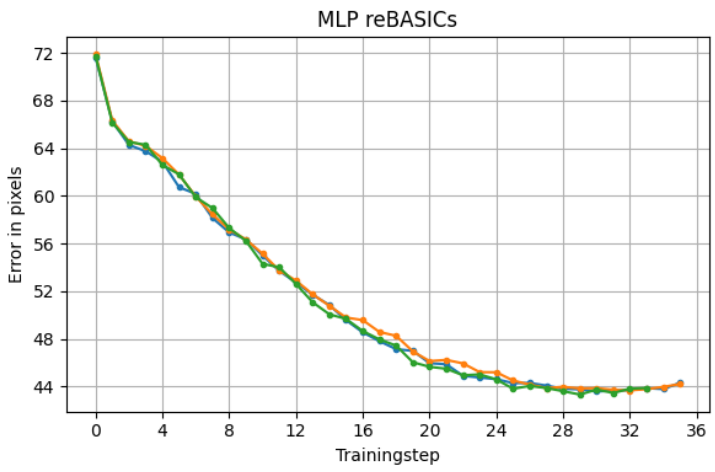


Figure 5.1: MSE performance tracked over training steps for reBASICS feedback to MLP

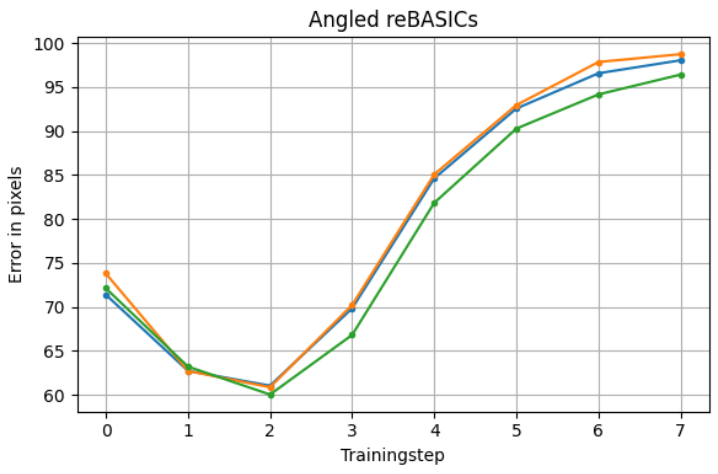


Figure 5.2: MSE performance tracked over training steps for reBASICS feedback to Angles

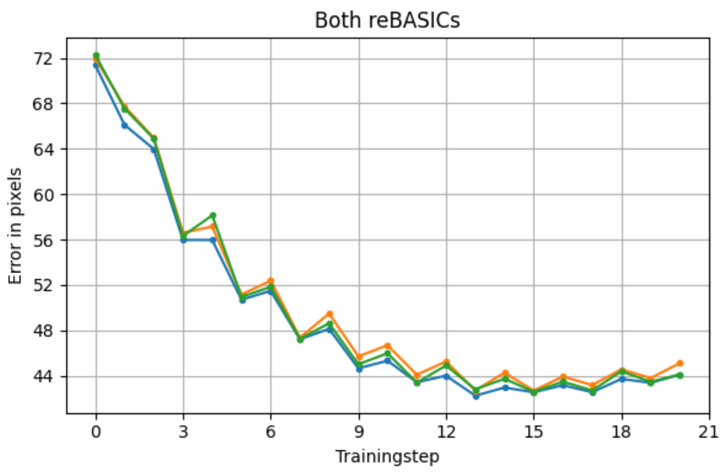


Figure 5.3: MSE performance tracked over training steps for reBASICS feedback to both MLP and Angles

Figure 5.1 shows that the ”MLP feedback” pathway consistently improves until it reaches an MSE of just below 1901, which approximates to a standard deviation (STD) per point of $\sqrt{1901} \approx 43.6pixels \approx 3.8cm$. ”Angle feedback”, as shown in Figure 5.2, reaches an MSE as low as 3604 ($\approx 5.22cm$ STD per point) within two training steps, but degrades afterwards. This is faster than the ”MLP feedback” which reaches an MSE below 3604 after 5 training steps.

This shows that corrections to the joint angles allow for better control. However, this type of correction based on the error between achieved and desired angles is limited by whether or not the angles are realistically achievable given previous joint angle positions. If error persists for a point in the movement, the ”Angle feedback” pathway continually increases the magnitude of the correction, which leads to a degradation of the model. This is likely because the commanded joint angles differ so extremely between neighboring points that

they interfere with each other. This may possibly be prevented by adding a regularization term that encourages neighboring points to have similar joint angles.
Giving feedback via ”Both” pathways, as seen in Figure 5.3 shows that they interact generally positively together, since an MSE as low as 1810 ($\approx 3.7cm$ STD per point) is reached much faster than in the other experimental conditions, after 13 training steps. However, despite the overall much faster improvement in the ”Both” condition, the interaction of the feedback mechanisms also produces a slight worsening after each improvement, in what appears to be an interference effect. It seems possible that this occurs because the angle correction corrects for MLP output of the previous training step but is applied to the new MLP output in the next training step. It seems possible that applying only one feedback per training step, and alternating the feedback paths, may get rid of the interference effect and possibly lessen the total amount of training steps necessary. However, this was not explored for this thesis.
Because using both feedback paths was much quicker and achieved a slightly better performance, the comparison between ESN component variations was done with the ”Both” feedback condition. The interference effect visible in Figure 5.3 was only discovered in post-experiment analysis, and therefore a possible benefit of using the ”MLP feedback” condition for comparing the ESN component variations was not considered. Both the standard ESN ensemble with continual input in the form of the desired coordinates y_d and with the input including the angles currently associated with those desired coordinates (y_d, θ) were newly initialized for three training attempts.
The ESN ensemble using the adaptive input (y_d, θ) displayed immediate divergence, with a near doubling in MSE after the first training step and getting continually worse. The movement became erratic and fast, and therefore this experimental condition was interrupted and discontinued. This immediate degradation is possibly because the change between training steps was so significant that the joint angles θ changed too dramatically, severely changing the input and therefore the dynamics of the reservoirs. This may be especially the case with the accelerated output layer training of 50 epochs per training step. It is possible that fine-tuning the feedback strength may resolve that problem, to reach a performance like in previous work (Kalidindi et al. 2019 [8], Tan et al. 2022 [13]). However, this was not further explored, and the ESN ensemble with this type of input was excluded from further analysis. Only the ESN ensemble with input y_d is used for detailed comparison, shown in Figure 5.4.

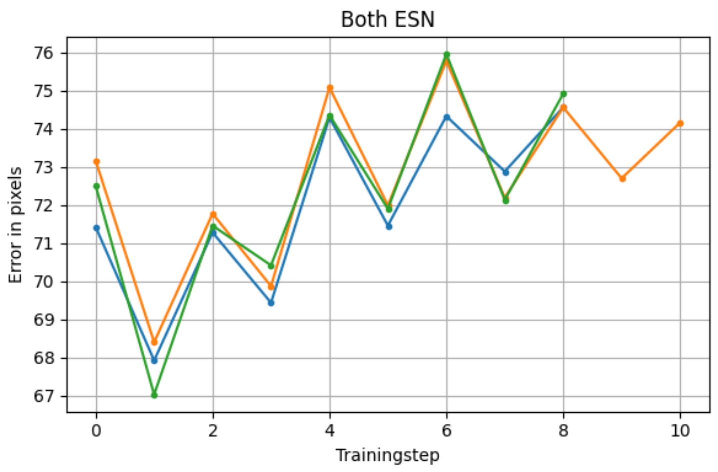
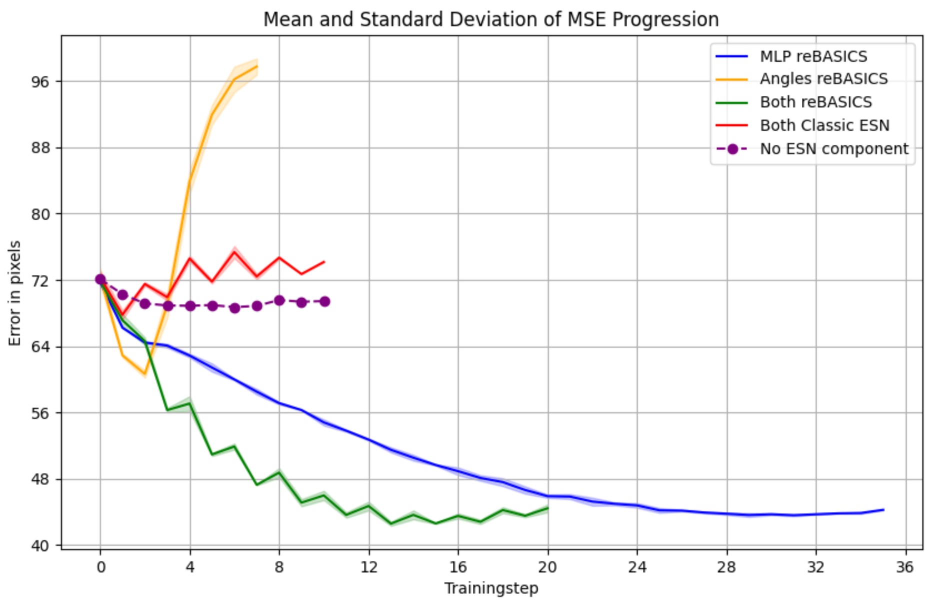
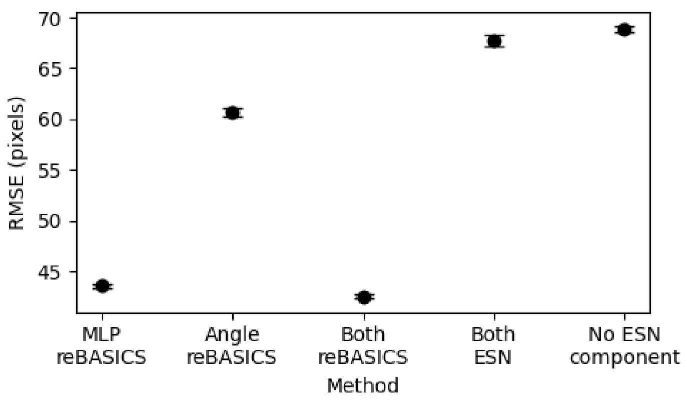


Figure 5.4: Standard ESN feedback to both MLP and Angles

Surprisingly, the training starts to degrade after the first training step with an improvement that does not exceed the improvements from a warm-up effect by much. There is an alternation in improvement and degradation, mirroring the training steps with alternating improvement and stagnation in the reBASICS ”Both” condition in figure 5.3. This opens up the possibility that the interference effect is relatively stronger than improvements from training. In the ”Angle feedback”-only condition for reBASICS, degradation also happens rather quickly. This suggests that the ESN ensemble is possibly just weaker and makes slower progress in training than reBASICS, but is hindered by the Angle feedback pathway’s fast degradation. It seems possible that degradation could be prevented from occurring if the ESN ensemble were to be trained with the ”MLP feedback”-only condition. This was only discovered in post-experiment analysis and is therefore not explored in this thesis.
Figure 5.5 shows the averages and barely visible STD of all experiments, as well as the minimum MSE per training cycle, averaged over the 3 training cycles per experimental condition. Further, the progression of MSE when repeatedly commanding the MLP controller’s output unadjusted by the ESN component is included to observe the warm-up effect.



(a) All ESN experimental conditions MSE average per training step, with STD added as shade, and including MSE progression of MLP controller with no ESN component correction

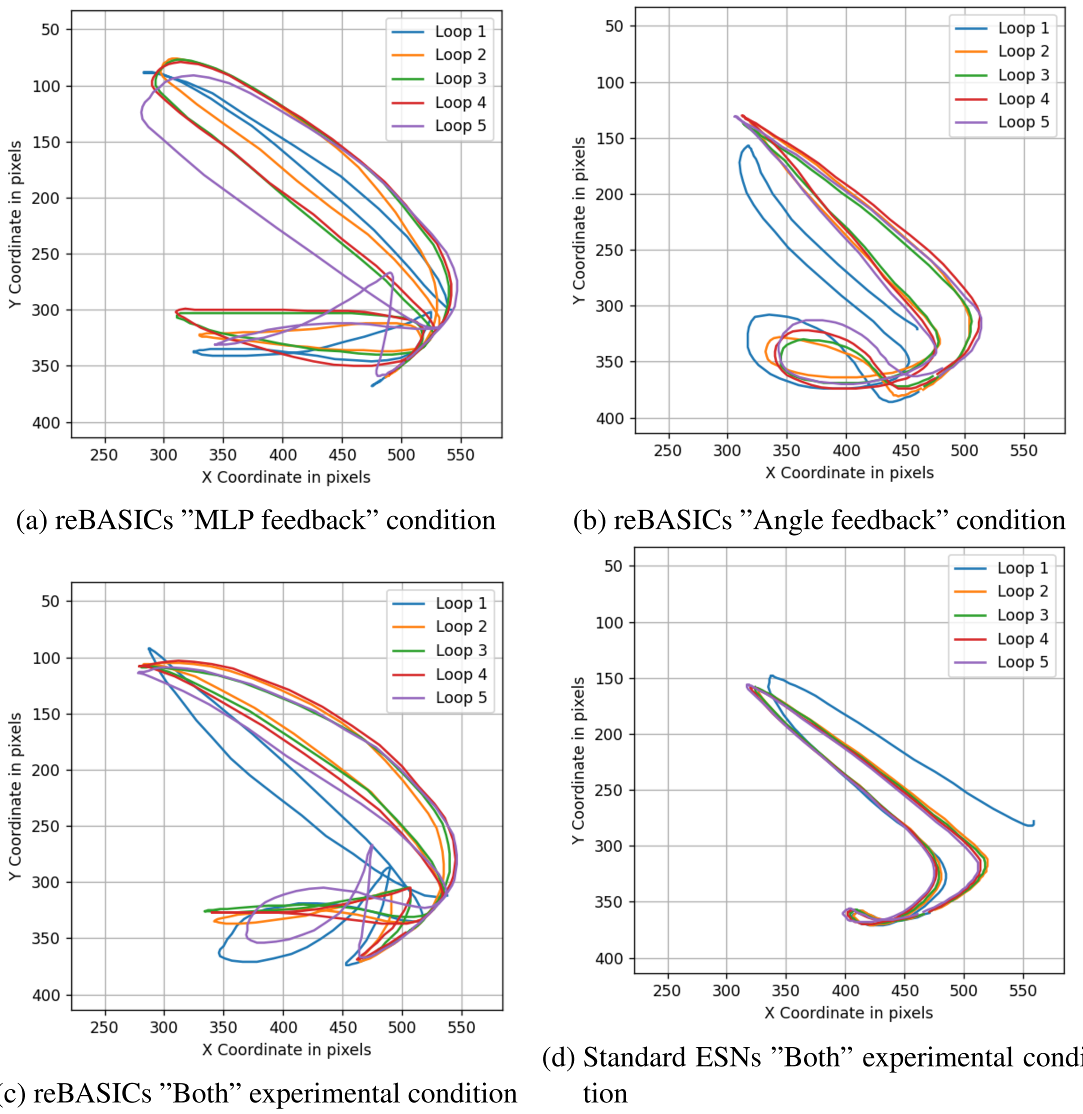


(b) Final RMSEs with STD

Figure 5.5: ESN performance and comparison with reBASICS and no ESN component

As can be seen in subfigure 5.5b the ESN component with a standard ESN ensemble barely outperforms the version where no ESN component is used, due to the immediate degradation. To check the significance in variation between conditions, the p-value between conditions was calculated. All conditions were significantly different compared to each other, with the largest *p*-value between conditions being 0.0042 between the reBASICS ”Both” and the reBASICS ”MLP feedback” condition.
It is remarkable how little variation there is between training cycles within the same feedback path experimental condition. This is likely the case because the ESN component is made out of an ensemble of ESNs, and ensembles contain the average of different initialization variations between individual ESNs. This leaves little variation between newly

generated ensembles. Given this, a single training cycle per condition might have been enough, and the time might have been spent instead on including more types of experimental condition, such as different epoch amounts E, or using the ESN ensemble for all feedback path conditions.
To further compare the experimental conditions, their trajectories are explored similarly to Figure 4.10. Because the training cycles within the same experimental condition are very similar, only the first of the three training cycles per experimental condition is shown.



(a) reBASICs ”MLP feedback” condition

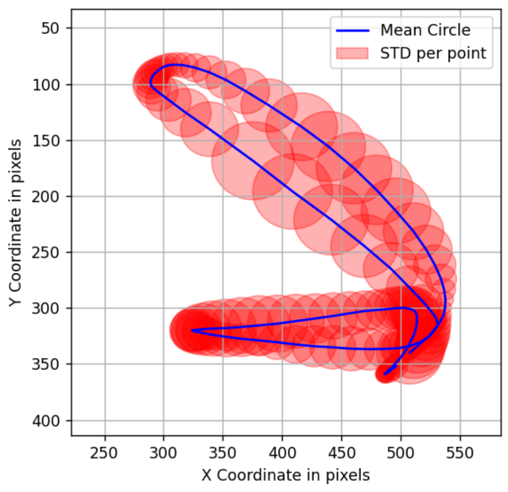
(b) reBASICs ”Angle feedback” condition

(c) reBASICs ”Both” experimental condition

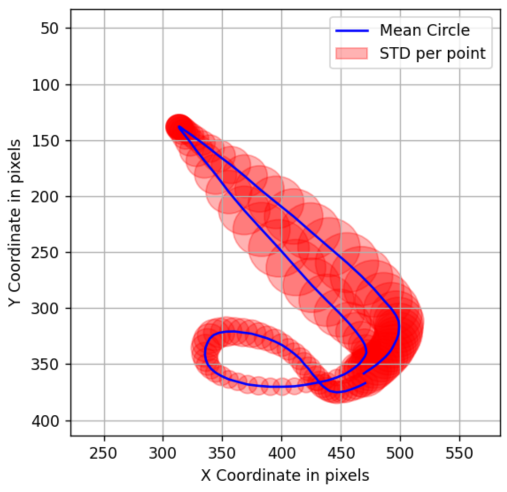
(d) Standard ESNs ”Both” experimental condition

Figure 5.6: Five loops with lowest MSE per experimental condition

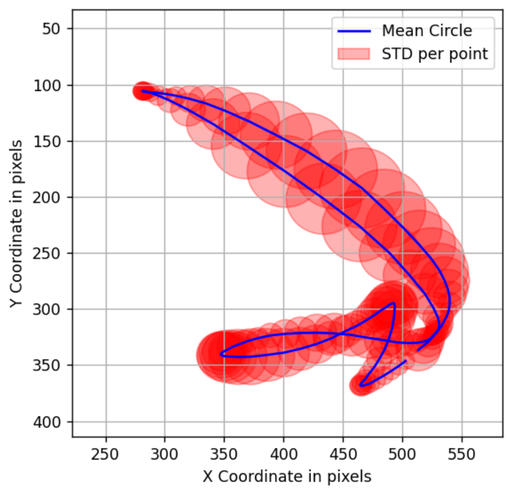
As can be seen in Figure 5.6, the ”Angle feedback” condition executes the small circle at the bottom left part of the movement much cleaner than the other conditions. This shows that there is a significant mismatch between the intended angles and the actual angles, and adding a correction produces a more accurate movement. Because only the first training step was successful, the ESN ensemble corrected movement is very similar to the MLP controller as a stand-alone shown in Figure 4.10.



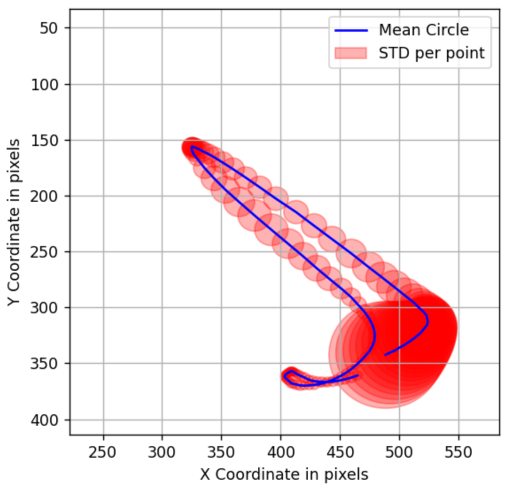
(a) reBASICs "MLP feedback" experimental condition



(b) reBASICs "Angle feedback" experimental condition



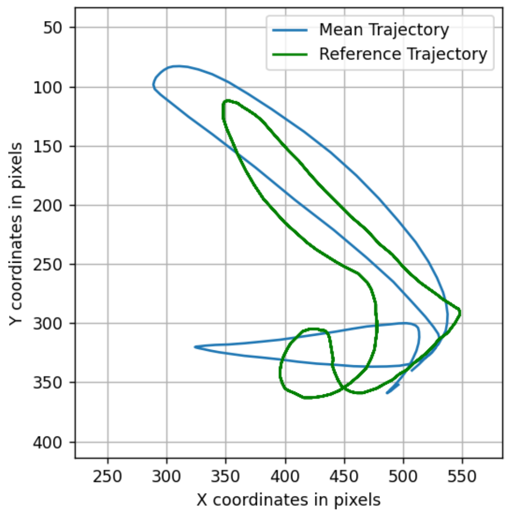
(c) reBASICs "Both" experimental condition



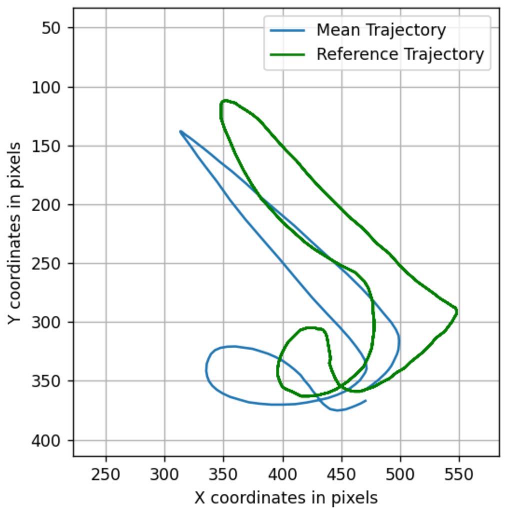
(d) Standard ESNs "Both" experimental condition

Figure 5.7: Mean and STD of the five loops with lowest MSE per experimental condition

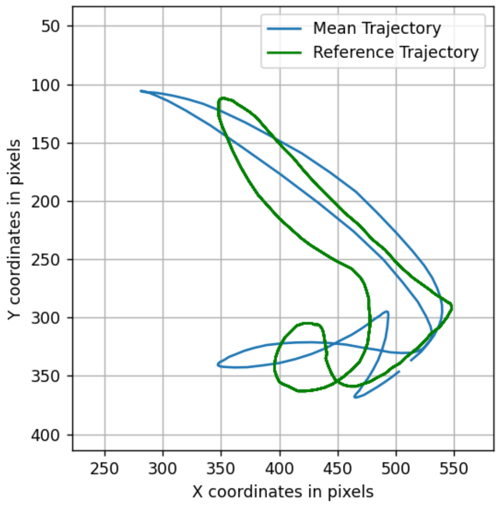
Figure 5.7 shows the averaging of the 5 loops per experimental condition. The faster parts during the two straight parts of the movement have higher variation between loops.



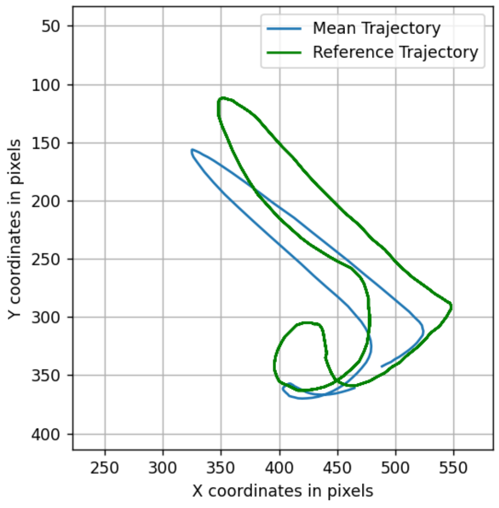
(a) reBASICs "MLP feedback" experimental condition



(b) reBASICs "Angle feedback" experimental condition



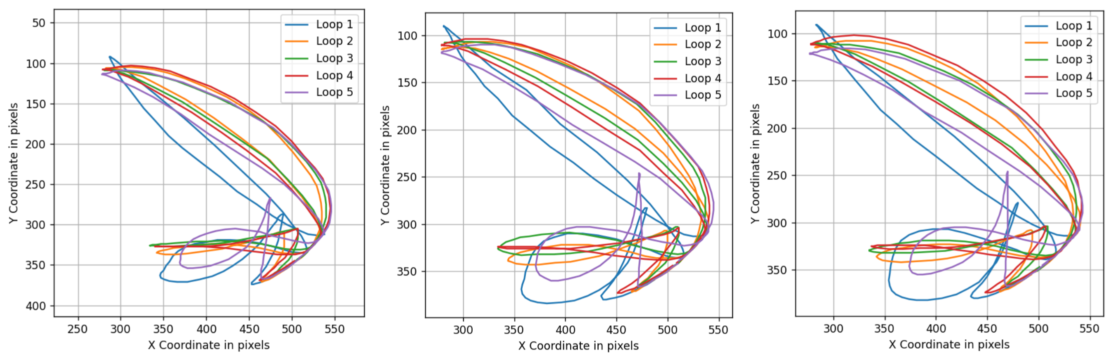
(c) reBASICs "Both" experimental condition



(d) Standard ESNs "Both" experimental condition

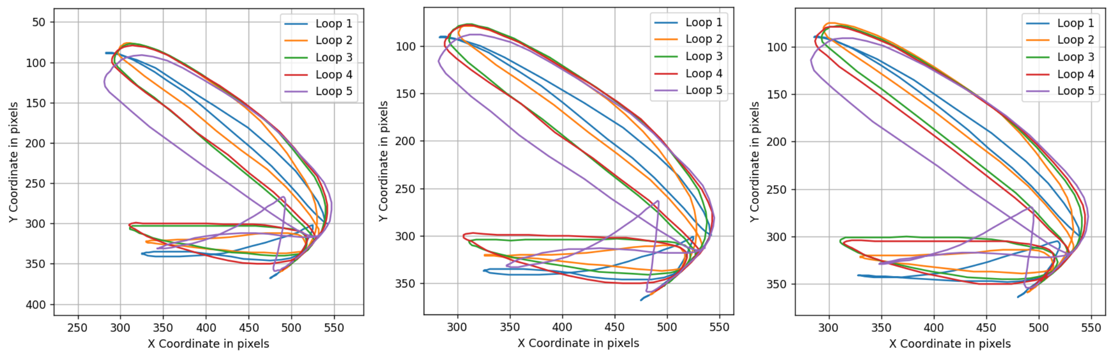
Figure 5.8: Average of the 5 loops, plotted against the target

Figure 5.8 shows the average of the 5 loops of the first of each 3 training cycles per experimental condition plotted against the target trajectory. It is surprising that despite movement between the reBASICS "MLP feedback" and the reBASICS "Both" experimental conditions being different, as displayed in Subfigure 5.8a and 5.8c, they end up with a nearly identical MSE. This could either be by chance, but it also seems plausible that the limiting factor for further improving the movement is the MLP controller, as it reaches its limits in providing useful inverse kinematics. Interestingly, the "MLP only" feedback path experimental condition manages to compensate for the errors in joint angles solely by exploiting the MLP controller's provided inverse kinematics. To further explore this, let's consider the individual loop trajectories for all 3 training cycles for the reBASICS "MLP feedback" and "Both" conditions, displayed in Figure 5.9 and 5.10.



(a) First training cycle (b) Second training cycle (c) Third training cycle

Figure 5.9: The 5 loops of each of the three training cycles' best performance for reBASICS "Both" condition



(a) First training cycle (b) Second training cycle (c) Third training cycle

Figure 5.10: The 5 loops of each of the three training cycles' best performance for reBASICS "MLP feedback" condition

As seen in both Figure 5.9 and 5.10, the similarity of trajectory within an experimental condition is very high , not just on average, but also on the level of individual loops. This is especially visible in the bottom part of the first loop in blue for the "Both" condition being unique compared to the rest, but also the final loop Nr 5 in purple displaying a distinct "Z" shape at the bottom, in both the "Both" and "MLP feedback" experimental conditions. This suggests that the type of correction that has to be done to achieve the target movement is different for each loop, possibly because the pneumatic actuators behave differently the more the movement progresses. While this is not as prominent in the calm movement produced by the MLP controller as a stand-alone (Figure 4.10) , the corrections of the ESN component add more complexity to the movement, possibly exaggerating effects that are characteristic of pneumatic actuators. Ultimately, the ESN component fails to perfectly provide the different correction needs over the entire movement. This suggests that either the ESN component might benefit from being stronger than both the reBASICS and the standard ESN ensemble versions used here, or that a better MLP controller is needed.

5.2 Analysis of reBASICS and ESN ensemble

The differences in performance between reBASICS and ESN ensemble probably lie in their different representational capacities. Functionally, the ESN component may be considered as an aggregate or reservoir of frequencies, that the output layer combines into the frequency combination that adds the desired correction for each time point t. The representational capacity of the ensemble is likely directly linked to both the amount of total output units, as well as the representational capacity of each individual ESN per ensemble, in terms of the variation of frequencies they provide. Arguably, the parameters of the standard ESN ensemble having 80 output neurons as opposed to the 800 output neurons of reBASICS leads to a difference in representational capacity, which causes the difference in performance observed here. The other factor might be a difference in the variation of frequencies and phases offered per ensemble.
As can be seen in comparing reBASICS reservoir output in Figure 5.11 and standard ESN ensemble reservoir output in Figure 5.12, the reBASICS type ensemble has much richer dynamics, while the standard ESN ensemble has very homogeneous output. This is likely the effect of the input to the standard ESN ensemble consisting of the target coordinates, a homogeneous input. This is especially visible in the output repeating 5 times, just like the input does for the 5 movement circles. In contrast, the reBASICS output does not depend on a continual input to drive the reservoir, after the initial pulse. This means that the dynamics in reBASICS depend primarily on the recurrent weights within the network, which are kept active by the high scaling factor $g = 1.2$.
This is further shown in Figures 5.14 and 5.15, which summarize the magnitudes of phase ranges for different frequencies present, through a Fast Fourier Transformation for each of the different ensembles output. The cumulative magnitude per frequency for all modules in the ensemble are displayed. The percentage of phases present in that frequency range is displayed through the percentage of phase encoding color present in each bar of the bar graph. For visualization purposes, the frequencies between 0 and 1 on the left in the figures are zoomed in compared to the frequencies between 1 and 15. As a Fast Fourier Transformation allows detection of frequencies and their phases only below half Hz of 30 present in the data, frequencies above 15 Hz are not detectable. The frequency and phase range in the reBASICS models have much wider variety compared to the standard ESN ensemble, which has a much more homogeneous frequency and phase distribution, with certain phases barely present for certain frequencies. Another way to assess this is with how close to orthogonality the outputs are to each other, by calculating the degree of correlation. The degree of correlation measure 5.1 consists of the inner product between all the output vectors, where $\mathbf{M}$ is the total amount of output vectors, $\mathbf{i}$ and $\mathbf{j}$ refer to different vector indices.

$$\text{Average degree of correlation} = \frac{1}{M(M-1)} \sum_{i \neq j} \left| \langle \mathbf{v}_i, \mathbf{v}_j \rangle \right| \tag{5.1}$$

Table 5.1 shows the degree of correlation for each model. The reBASICS model is much closer to orthogonality, whereas the standard ESN ensemble models have a very high degree of correlation. It is therefore reasonable to conclude that the relative performance difference between ESN component variations is not just the effect of a difference in amount of output neurons per ensemble, but also a difference in representational capacity contributed by each output neuron.

ESN component	**Model 1**	**Model 2**	**Model 3**
reBASIC models	0.6194	0.6201	0.6072
standard ESN ensemble models	0.9903	0.9909	0.9847

Table 5.1: Degree of Correlation values for different ESN components

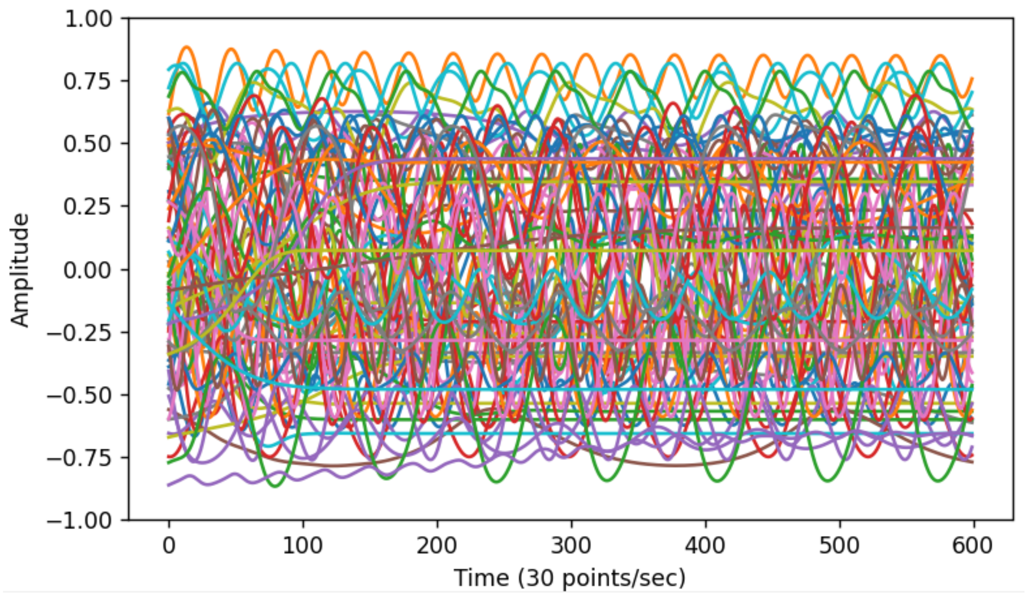


Figure 5.11: reBASICS 1 output visualization, 80 displayed of 800 total output neurons

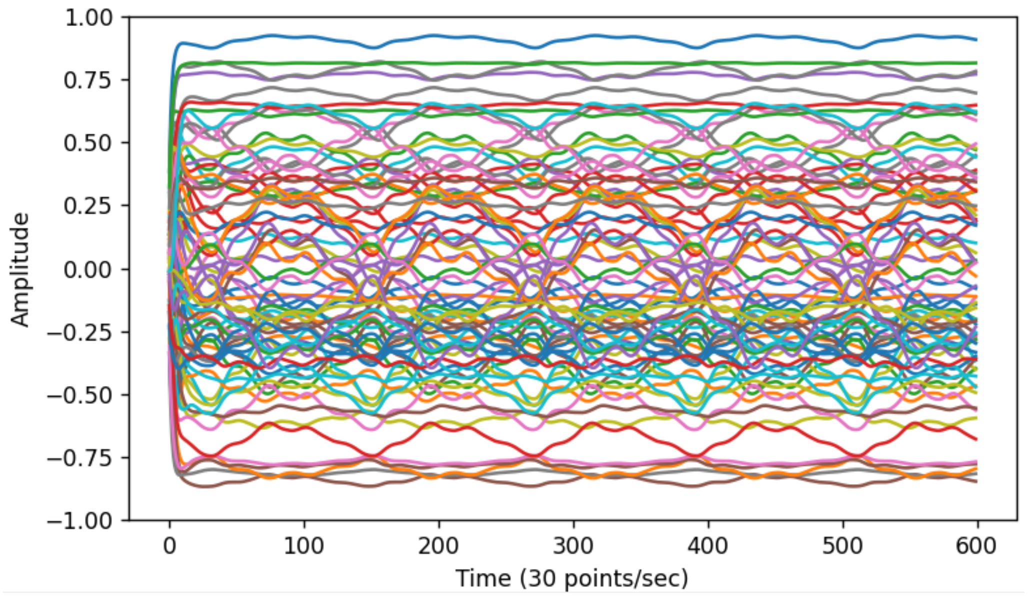


Figure 5.12: ESN ensemble 1 output visualization, all 80 output neurons displayed

´

Figure 5.13: reBASICS model 3

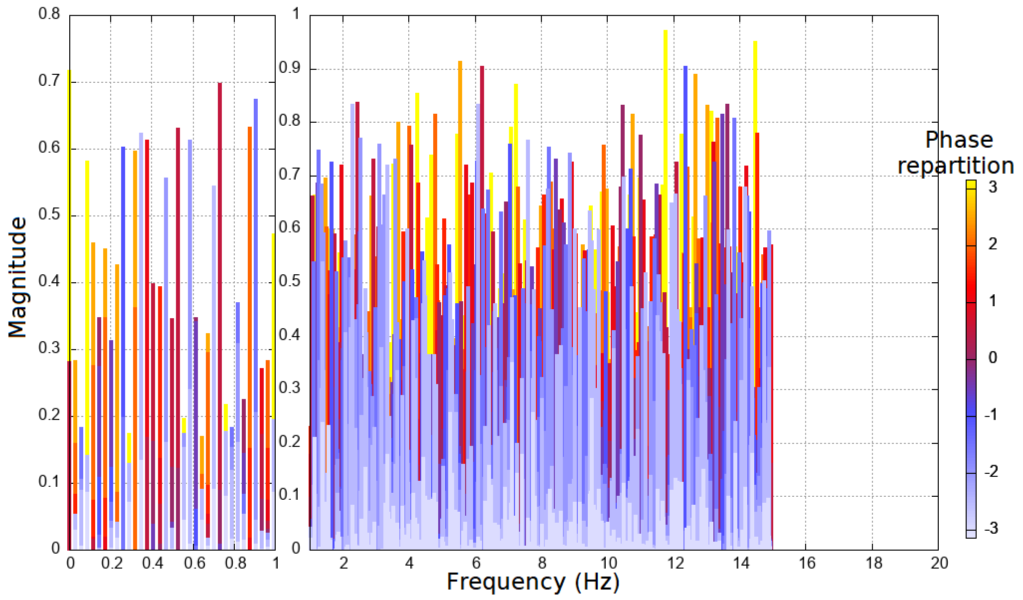


Figure 5.14: reBASICS 1 frequency analysis, with the coloring showing the phase repartition distribution per frequency

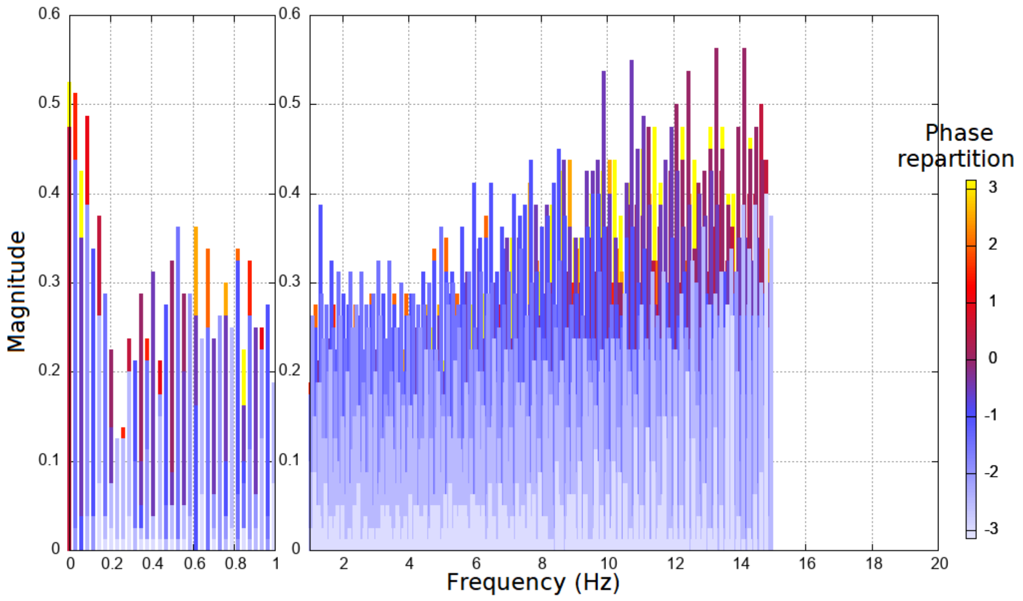


Figure 5.15: ESN ensemble 1 frequency analysis, with the coloring showing the phase repartition distribution

As the hyperparameters set include a connection probability of $p = 1$, which likely influenced average degree of correlation, additional ESN variations with different hyperparameters were generated and analyzed post-hoc. For this, for each ESN size 100 modules were created. As can be seen in Table 5.2, the classic ESN ensembles perform much better. The reBASIC versions perform much worse, but as hinted at by the high STD between ensembles, this is the because point convergence happened at a much higher rate despite the within module STD cutoff of 0.05.

ESN component	Neurons per module	Connection probability	Average Degree of Correlation within ensembles	STD between ensembles
reBASIC	100	0.1	0.871228	0.110195
ESN ensemble	100	0.1	0.725673	0.043037
ESN ensemble	200	0.1	0.724236	0.029034
ESN ensemble	400	0.1	0.718802	0.020360
ESN ensemble	800	0.1	0.718547	0.017152

Table 5.2: Hyperparameter search for Average Degree of Correlation values for different ESN component variations

An example of a point convergence by reBASICS is shown in Figure 5.16, that likely play a role in driving up the average degree of correlation. A typical ESN ensemble of the ESN with 100 neurons per module shown in Figure 5.17, that despite the much lower average degree of correlation still has the very present periodicity caused by the movement being repeated 5 times. This means that nonetheless it might perform worse, as key frequencies and phases might be missing.

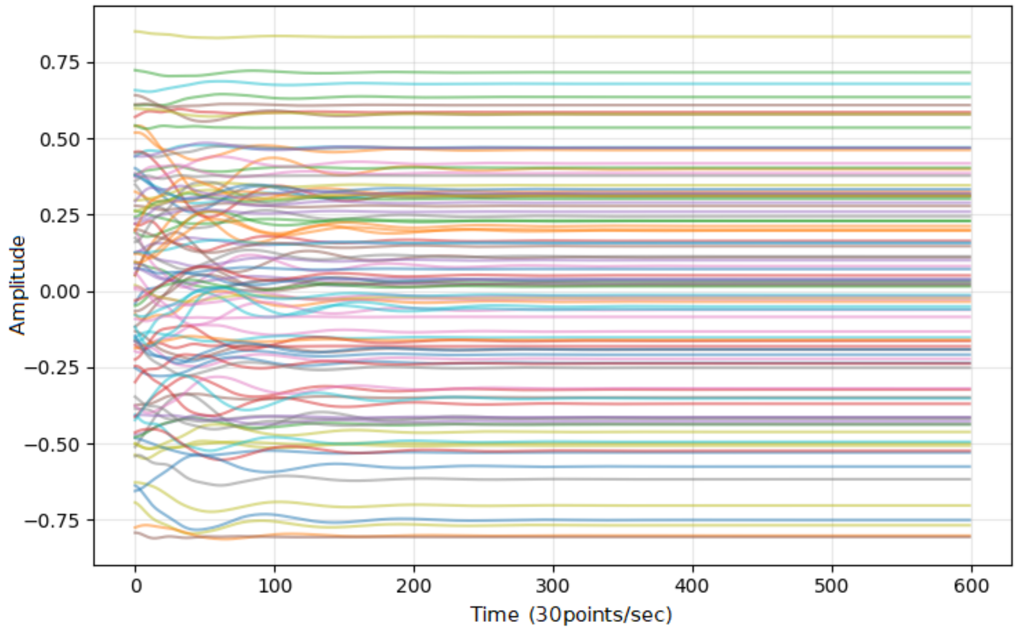


Figure 5.16: Point Convergence example of a reBASICS module with $p = 0.1$, output visualization

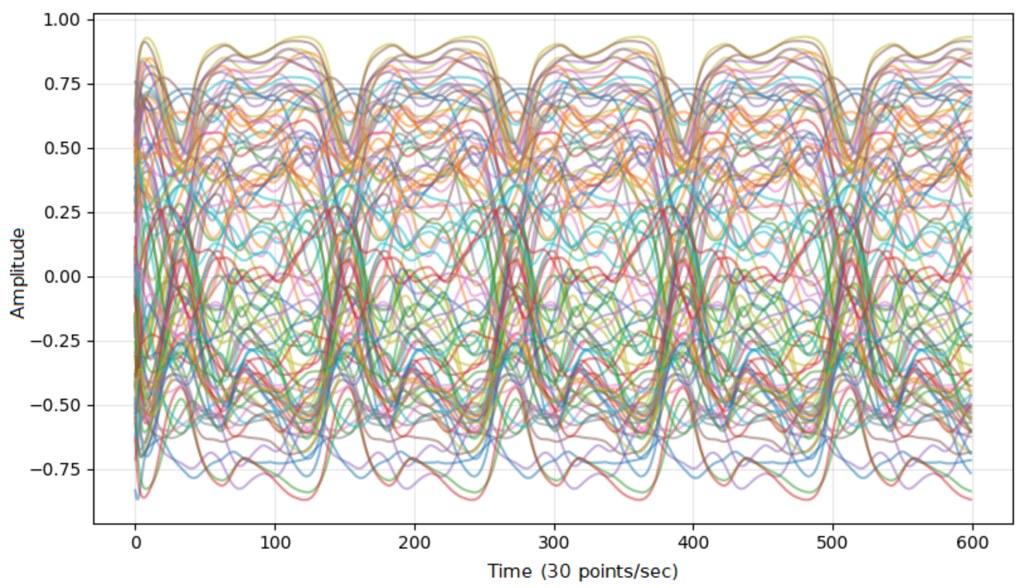


Figure 5.17: Example of 100 neuron ESN ensemble module with $p = 0.1$, output visualization

5.3 Analyzing the effect of the ”Angle feedback” path

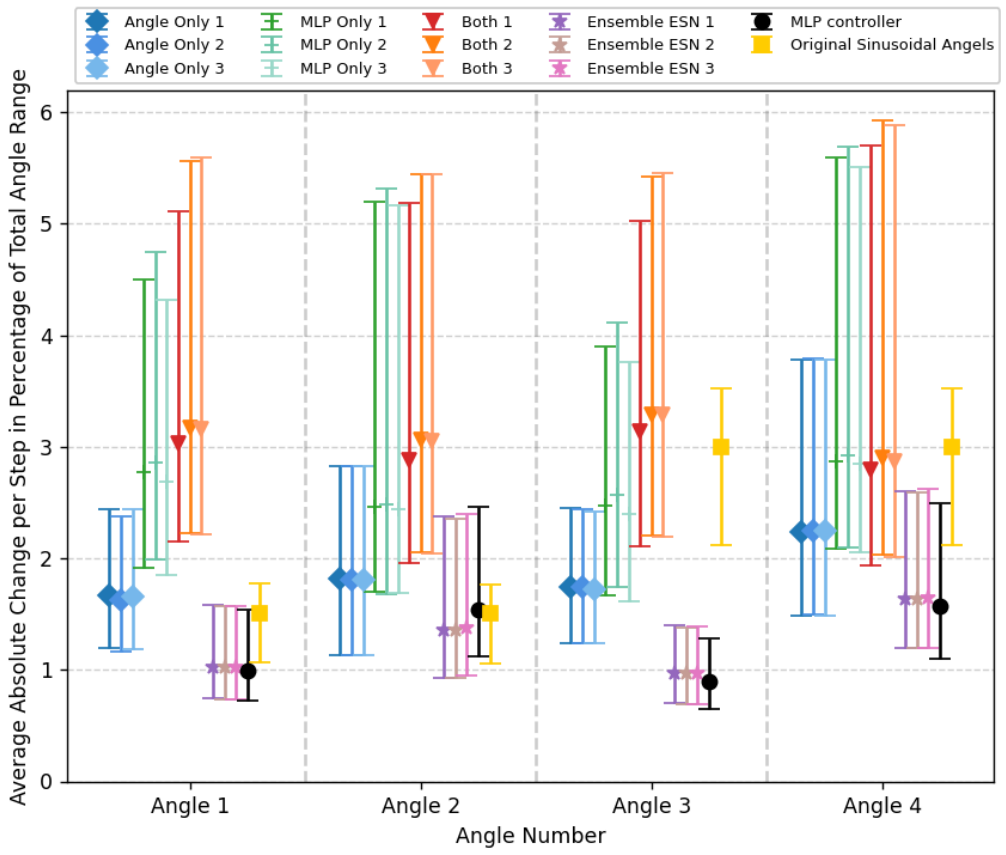
To examine the effect of the corrections to the MLP controller joint angle output, first the mean absolute change commanded to the angles, also termed mean absolute first-order difference (MAFD), averaged over all 600 steps of the complete movement are analyzed in Figure 5.18.

$$\text{MAFD} = \frac{1}{N}\sum_{t=1}^{N} |x_t - x_{t-1}| \tag{5.2}$$

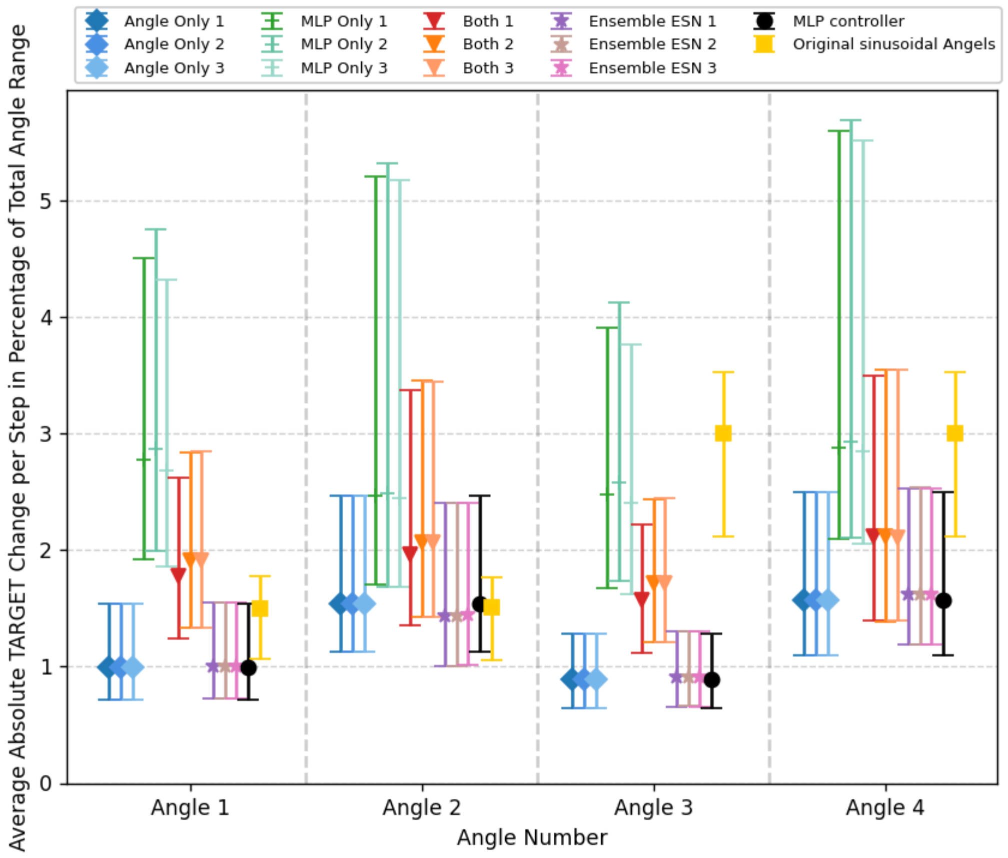
Subfigure 5.18a shows the actually commanded angles, and Subfigure 5.18b shows the joint angle output of the MLP controller if the ”Angle Feedback” correction were omitted from the fully trained models. All individual models are plotted, for each experimental condition and all 3 training cycles per condition. For comparison, the MLP controller without any ESN component correction is displayed in black, and the *original sinusoidal joint angle commands* that were originally created to produce the target movement and mark an example ”ideal solution” are included in yellow.

The standard Ensemble ESN models displayed in purple tones have nearly identical commanded angles as the MLP controller, because only the first training step was successful. The sinusoidal angle commands in yellow represent one possible ideal solution, that by nature of its smoothness uses the joint angles efficiently, leading to a low mean absolute first-order difference. In comparison, the MLP controller in black has less angle changes between the steps, which is reflected in its movement being more constricted (subfigure 5.8d). The other experimental conditions all have relatively more active angle commands, reflected in their more sweeping movement.

Noticeable in this comparison is the ”Both” condition, displayed in in red tones, with its difference between the real commanded angles (Subfigure 5.18a) as compared to the target angles not yet adjusted by the angle feedback path (Subfigure 5.18b). The stark difference shows that the joint angle adjustment path has to be exaggerated a lot to achieve the intended joint angles. Furthermore, the ”MLP feedback”-only path achieves a similar degree of exaggeration relying only on the Inverse Kinematics offered by the MLP controller.



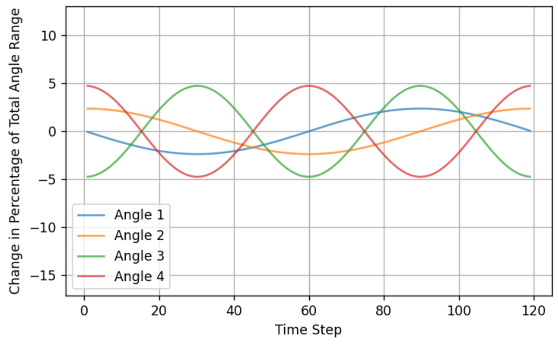
(a) Average Absolute Angle Change per Step, with Angle feedback path



(b) Average Absolute Angle Change per Step, with Angle feedback path adjustment removed, equating to Average Absolute TARGET Angle Change per Step

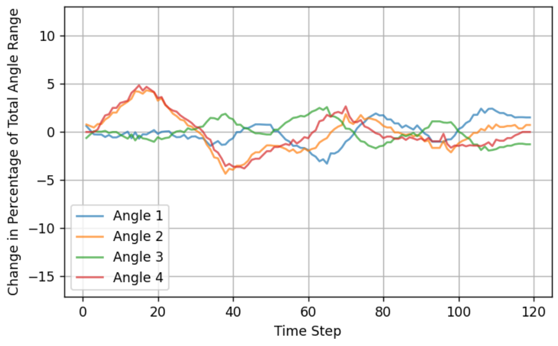
Figure 5.18: Average Absolute Angles and Asymmetric STD, *As Commanded* to the robot VS if the ”Angle path feedback” adjustment were omitted from the fully trained models

This exaggeration of commanded joint angles in order to achieve the intended angles or end-effector position is also visible in figure 5.19, where subfigures 5.19b and 5.19d show the joint angles that would be commanded if the corrections of the joint angles were omitted in the fully trained models of the conditions that include ”Angle feedback”. To the right, Subfigure 5.19c and 5.19e show them with the correction added as intended. This enables a direct comparison between the intended angles to be achieved, and the commanded angles necessary to achieve them, which indeed is done by exaggerating the commanded joint angles. Another characteristic is that the angle change drops to zero, which is a result of the commanded angles reaching the limit of the actuator range.
For comparison, both the ”MLP only feedback” condition and the sinusoidal joint angle commands that created the target movement are displayed in Subfigures 5.19f and 5.19a. While the actual movements between ”MLP only feedback” and ”Both feedback path” conditions are somewhat different (Subfigure 5.8a and 5.8c), the commanded angles are very similar, including the high magnitude of angle change between steps. Such commanded angle changes of up to 10% of total actuator range between time points are likely not physically achievable in $1/30$ of a second. Both conditions probably tend towards such a solution with a high rate of commanded change to increase the acceleration necessary to reach the desired end-effector position. However, the sinusoidal angles that generated the target movement (Subfigure 5.19a) show that smoother solutions exist, which the ESN correction component cannot correct the MLP controller into producing. This suggests that the MLP controller might be the limiting factor. This is further evidenced by both reBASICS ”MLP only feedback” and ”Both feedback path” conditions developing very similar solutions, as seen in Subfigure 5.19e and 5.19f. Another way to visualize this is given in Figure 5.20, which plots the fully commanded joint angles in percentages of actuator range for the entire movement, which shows very similar commanded joint angles.

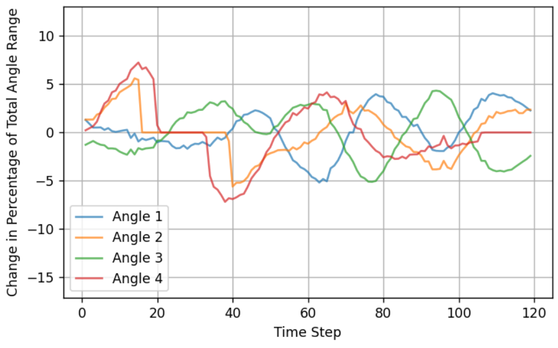


(a) Sinusoid Angles that created target movement

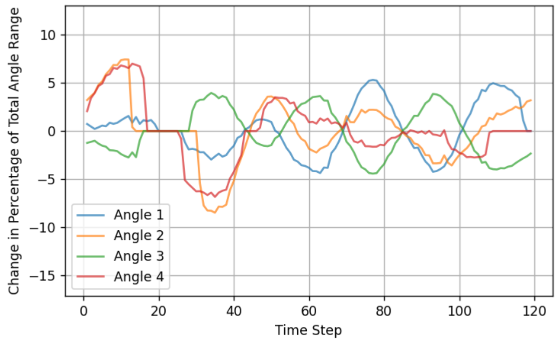
Column with no "Angle feedback" present *Column with "Angle feedback" present*



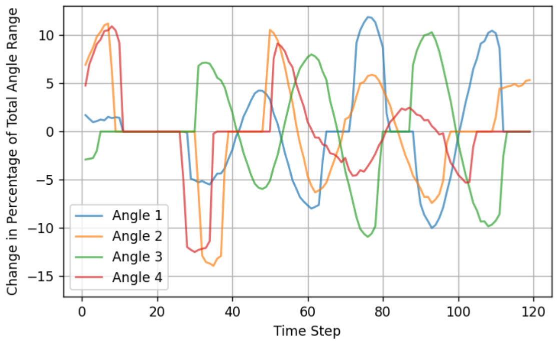
(b) "Angle Feedback" with angle correction omitted (uncorrected MLP controller)



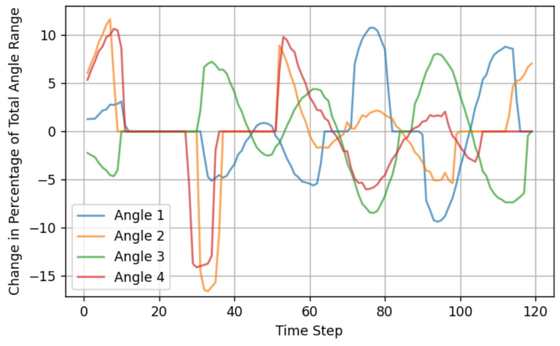
(c) "Angle Feedback" condition, as is



(d) reBASICS "Both" condition with angle correction omitted

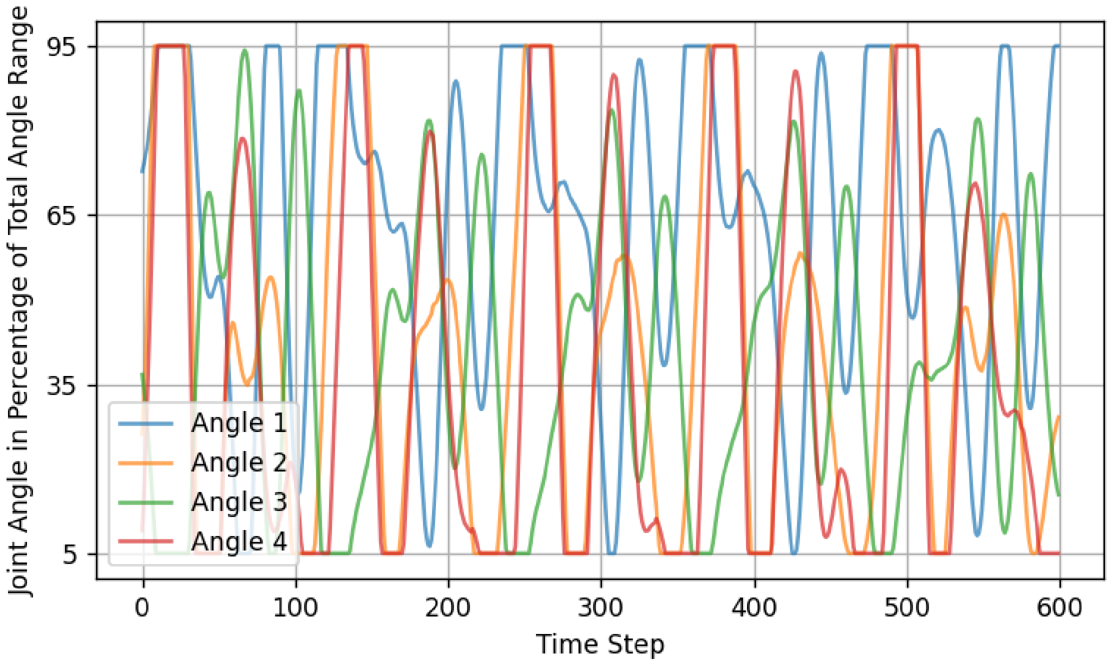


(e) reBASICS "Both" condition, as is

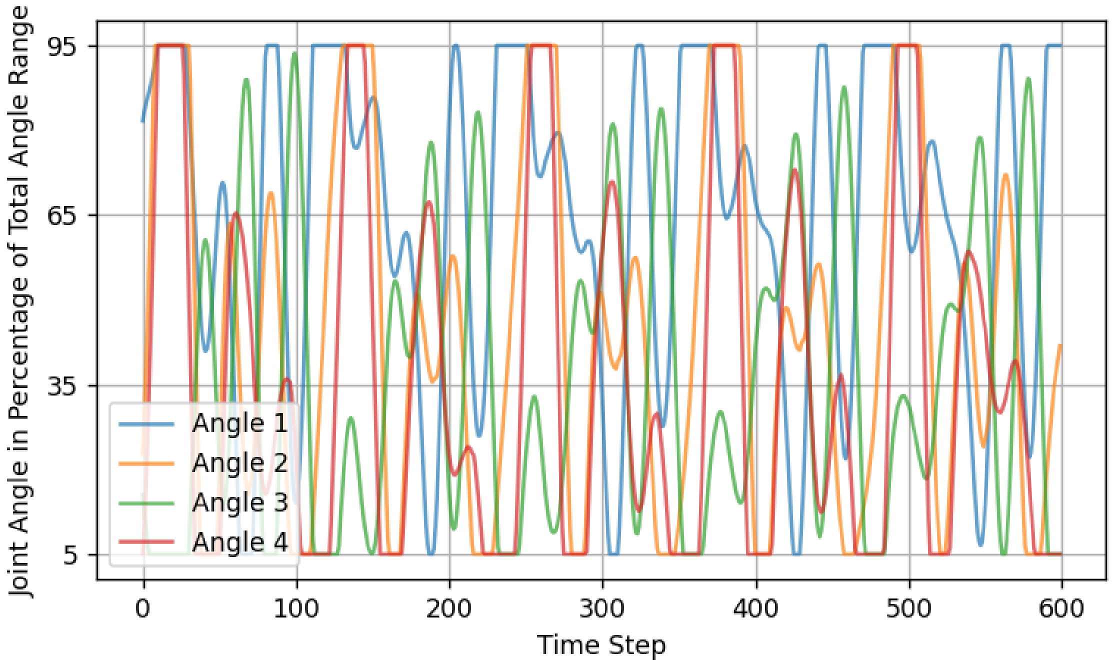


(f) reBASICS "MLP feedback"-only condition

Figure 5.19: Specific Angle Change commanded in Percentage of total Angle Range, for the first of the 5 loops



(a) reBASICS "MLP feedback"-only condition



(b) reBASICS "Both" feedback paths condition

Figure 5.20: Joint Angles commanded per step in percentage of total Angle Range, for the complete movement

6 Conclusion

The work in this thesis expanded on the previous work of Kalidindi et al. (2019) [8], Tan et al. (2022) [13], and Kawai et al. (2024) [17], that consisted of experiments in simulations and a robot with electrical motors, to test the cerebellar-inspired structure on a soft robot with redundant Degrees of Freedom, the type of robot that stands to benefit most from the mathematical-modeling-free inverse kinematics solution that it offers. Applying the cerebellar-inspired structure, which consists of an inverse kinematics approximator MLP forming the controller and an ESN component correcting that MLP controller, did not lead to perfect performance in retracing a target movement but performed better than an MLP controller alone. This shows that the cerebellar-inspired structure works in principle.
Adding a feedback path to the MLP controller's joint angles output (”Angle feedback”) as done in the previous work of Kawai et al. (2024) [17] led to improvements. Adding this correction as a stand-alone significantly improved the accuracy of the movements. This shows that control of the robot is imperfect, which the ”Angle feedback” can learn to compensate. Further, this feedback path may prevent interference from hysteresis by exaggerating joint angle commands.
However, the ”Angle feedback” path also leads to model degradation after just a few training steps. This is likely because adjustments keep being applied even when further corrections to joint angle commands can't be achieved due to physical limitations. This suggests that a regularization term that adjusts corrections proportional to how successful the previous correction was at addressing the error may improve results.
In the work of Kawai et al. (2024) [17], the ”Angle feedback” path and the MLP controller input feedback path (”MLP feedback”) were trained sequentially, with first the ”Angle feedback” training being completed followed by a switch to training the ”MLP feedback” path. In this thesis, both feedback paths were trained simultaneously in the ”Both” experimental condition.
The different feedback paths evidently had an interference effect, which is the likely cause of failure in the standard leaky integrator ESN ensemble ”Both” feedback path experimental condition. This failure was likely caused by the ”Angle feedback” path making adjustments too aggressively, leading to degradation rather quickly. However, with a stronger ESN component in form of reBASICS, the interaction was also positive and lead to a reduction in the training cycles necessary for successful training. This shows that continually training the ”Angle feedback” path may help address newly emerging errors that arise from the new joint angle commands as part of the ”MLP feedback” path training. Future work might explore how to balance the two feedback paths, either by repeatedly switching between them, or by adjusting the amount of adaptation from each feedback path.
With reBASICS as the ESN component, the ”Both” feedback paths experimental condition has shorter training time but does not improve performance compared to the ”MLP

feedback”-only experimental condition. In fact, both experimental conditions end up with very similar joint angle commands, characterized by extreme joint angles commands, or exaggerations. Those exaggerations appear necessary to achieve the joint angles that lead to the desired end-effector positions. This implies that the necessary exaggerations are achieved in the ”MLP feedback” experimental condition solely by leveraging the inverse kinematics the MLP controller can produce. Both experimental conditions seem limited by the strength of the MLP controller. Therefore, future research could explore how the different feedback path options would perform with varying MLP controller strengths.
In this thesis, the MLP controller was trained with a comparatively simple mapping containing only the relationship between achieved end-effector position and angles $y(t) \rightarrow \theta(t)$, leading to a relatively weak MLP controller. This is in contrast to previous work (Kalidindi et al. 2019 [8] and Tan et al. 2022 [13]), who included more information in the mapping such as the previous end-effector location and angle configuration $(y(t-1), y(t), \theta(t-1)) \rightarrow \theta(t)$, which results in stronger MLP controllers.
Further improvements to the MLP controller may be made by gathering the training data similarly to Tan et al. (2022) [13], who added constraints that may result in improved data quality. They limited the random exploration to explore angle combinations to nearby points, by limiting the change in joint angles for the next gathered data point to be between 3% to 10% of the actuators’ maximum range. This additional limitation of exploring only nearby configurations for the next data point to be gathered might be especially useful as it may encode how the actuators behave in moving between nearby points. Because continuous movements are commanded as a series of neighboring desired end-effector locations, an approximate inverse kinematics generator might greatly benefit from its training data being gathered as many nearby points. Future work might take this a step further, by gathering data from continual movements, instead of letting the end-effector settle in place. This recording of associated end-effector positions and joint angles in continual movements would enable time efficient data gathering with hysteresis impacting movement implicitly encoded in the data.
With their stronger MLP controller and less difficult tasks, Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] found the correction by their ESN component versions satisfactory. The work in this thesis attempted to validate their results for a more difficult task and used an ESN type similar to Tan et al. (2022) [13], except for expanding the ESN component to consist of an ensemble of ESNs. To imitate the input style that drove the ESN component in previous work, one ESN Ensemble version was driven by input $u(t) = [y_d(t); \theta(t)]$, which includes the actual joint angles θ. Those angles θ change after each training step due to updated corrections. This is tested against an ESN ensemble with an unchanging input $u(t) = y_d(t)$ containing only the desired end-effector positions y_d. The cerebellar-inspired architecture that used an ESN component with an ESN ensemble driven with input $u(t) = [y_d(t); \theta(t)]$ experienced immediate model collapse. This is in line with my earlier speculations that if the input that drives the classic ESN ensemble changes between training cycles, as $\theta(t)$ gets adapted as a result of training the ESN ensemble output layer, then the reservoir activity would change which invalidates the output weights learned to correct $\theta(t)$ in the first place. In the previous work of Kalidindi et al. (2019) [8] and Tan et al. (2022) [13] this worked for them possibly because they relied on an already near perfect MLP controller which guaranteed that the joint angles $\theta(t)$ did not change enough

between training steps to significantly change the reservoir dynamics on which the ESN component output layer is trained. In any case, the ESN ensemble driven by only the unchanging desired end-effector positions y_d performed better.
Nonetheless, an ESN component consisting of reBASICS vastly outperformed the ESN ensemble with input $u(t) = y_d(t)$. This may be simply because the interference effect that occurs in the ”Both” feedback path experimental condition led to failure for the ESN ensemble. While this implies that the ESN ensemble is weaker than reBASICS when employed on the ”MLP feedback” path, it may be possible that the ESN ensemble also could have fully leveraged this feedback path, albeit slower. This would have to be explored in follow-up studies to properly address.
Comparing the outputs of the two ensemble versions revealed that reBASICS has a much lower degree of correlation between modules, a measure related to how well modules' outputs can be combined into a desired output, also termed representational capacity ([18]. This is opposed to Tan et al.'s (2022) [13] ESN, which is based on the work of Lukosevicius et al. (2006) [49]. In this work, the ESN is optimized for a classification task, which means that this type of ESN is optimized to be maximally discriminative to different input instead of being optimized to have rich dynamics in the face of this use-case present continually similar input of desired coordinates $y_d(t)$. This is in contrast to reBASICS, which is specifically designed to mirror the speculated reservoir dynamics of the basal ganglia. In this thesis, it is the more appropriate ESN ensemble version for this type of task.
A limitation in the work of this thesis is that arguably suboptimal hyperparameters are used for both the ESN ensemble and reBASICS, such as having a connection probability between recurrent neurons of $p = 1$. In a post-experimental additional hyperparameter search with $p = 0.1$, the ESN ensembles showed much lower average degree of correlation, but still had very present patterns caused by the periodicity of the input due to the circular movement being repeated 5 times per complete movement. Follow-up experiments are necessary to make definite statements on wether the average degree of correlation in an ensemble is the main predictor for learning success, or if the differences in architecture between reBASICS and classic ESN ensembles keep favoring reBASICS due to its movement independent input. Another confounding factor is that the ESN ensemble used in experiments had $1/10$ of output neurons compared to reBASICS, weakening representational capacity, which future research may address by equalizing the amount of output neurons.

Bibliography

[1] J. W. Burdick, "On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds," in *Advanced Robotics: 1989: Proceedings of the 4th International Conference on Advanced Robotics Columbus, Ohio, June 13–15, 1989*, pp. 25–34, Springer, 1989.

[2] F. Renda, M. Cianchetti, M. Giorelli, A. Arienti, and C. Laschi, "A 3d steady-state model of a tendon-driven continuum soft manipulator inspired by the octopus arm," *Bioinspiration & Biomimetics*, vol. 7, no. 2, p. 025006, 2012.

[3] D. Trivedi, A. Lotfi, and C. D. Rahn, "Geometrically exact models for soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 24, no. 4, pp. 773–780, 2008.

[4] T. G. Thuruthel, E. Falotico, M. Cianchetti, and C. Laschi, "Learning global inverse kinematics solutions for a continuum robot," in *ROMANSY 21-Robot Design, Dynamics and Control: Proceedings of the 21st CISM-IFToMM Symposium, June 20-23, Udine, Italy*, pp. 47–54, Springer, 2016.

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

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

[7] V. Hlaváč, "Mlp neural network for a kinematic control of a redundant planar manipulator," in *International Conference on the Theory of Machines and Mechanisms*, pp. 24–32, Springer, 2020.

[8] H. T. Kalidindi, T. G. Thuruthel, C. Laschi, and E. Falotico, "Cerebellum-inspired approach for adaptive kinematic control of soft robots," in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 684–689, IEEE, 2019.

[9] M. Desmurget and S. Grafton, "Forward modeling allows feedback control for fast reaching movements," *Trends in Cognitive Sciences*, vol. 4, no. 11, pp. 423–431, 2000.

[10] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[11] F. Piqué, H. T. Kalidindi, L. Fruzzetti, C. Laschi, A. Menciassi, and E. Falotico, "Controlling soft robotic arms using continual learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5469–5476, 2022.

[12] Z. Chen, Y. Xia, J. Liu, J. Liu, W. Tang, J. Chen, F. Gao, L. Ma, H. Liao, Y. Wang, *et al.*, "Hysteresis-aware neural network modeling and whole-body reinforcement learning control of soft robots," *arXiv preprint arXiv:2504.13582*, 2025.

[13] N. Tan, P. Yu, and F. Ni, "A cerebellum-inspired network model and learning approaches for solving kinematic tracking control of redundant manipulators," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 15, no. 1, pp. 150–162, 2022.

[14] T. Yamazaki and S. Tanaka, "Neural modeling of an internal clock," *Neural Computation*, vol. 17, no. 5, pp. 1032–1058, 2005.

[15] T. Yamazaki and S. Tanaka, "Computational models of timing mechanisms in the cerebellar granular layer," *The Cerebellum*, vol. 8, pp. 423–432, 2009.

[16] T. Yamazaki and S. Tanaka, "The cerebellum as a liquid state machine," *Neural Networks*, vol. 20, no. 3, pp. 290–297, 2007.

[17] Y. Kawai, H. Atsuta, and M. Asada, "Adaptive robot control using modular reservoir computing to minimize multimodal errors," in *2024 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, IEEE, 2024.

[18] Y. Kawai, J. Park, I. Tsuda, and M. Asada, "Learning long-term motor timing/patterns on an orthogonal basis in random neural networks," *Neural Networks*, vol. 163, pp. 298–311, 2023.

[19] H. Ishihara and M. Asada, "Design of 22-dof pneumatically actuated upper body for child android 'affetto'," *Advanced Robotics*, vol. 29, no. 18, pp. 1151–1163, 2015.

[20] D. Marr and W. T. Thach, "A theory of cerebellar cortex," *From the retina to the neocortex: selected papers of David Marr*, pp. 11–50, 1969.

[21] D. M. Wolpert, Z. Ghahramani, and M. I. Jordan, "An internal model for sensorimotor integration," *Science*, vol. 269, no. 5232, pp. 1880–1882, 1995.

[22] H. Tanaka, T. Ishikawa, J. Lee, and S. Kakei, "The cerebro-cerebellum as a locus of forward model: a review," *Frontiers in Systems Neuroscience*, vol. 14, p. 19, 2020.

[23] S. L. Palay and V. Chan-Palay, *Cerebellar Cortex: Cytology and Organization*. Springer-Verlag, 1974.

[24] D. M. Wolpert, R. C. Miall, and M. Kawato, "Internal models in the cerebellum," *Trends in Cognitive Sciences*, vol. 2, no. 9, pp. 338–347, 1998.

[25] H. Imamizu, T. Kuroda, S. Miyauchi, T. Yoshioka, and M. Kawato, "Modular organization of internal models of tools in the human cerebellum," *Proceedings of the National Academy of Sciences*, vol. 100, no. 9, pp. 5461–5466, 2003.

[26] R. B. Ivry, R. M. Spencer, H. N. Zelaznik, and J. Diedrichsen, “The cerebellum and event timing,” *Annals of the New York Academy of Sciences*, vol. 978, no. 1, pp. 302–317, 2002.

[27] J. L. Contreras-Vidal, S. Grossberg, and D. Bullock, “A neural model of cerebellar learning for arm movement control: cortico-spino-cerebellar dynamics.,” *Learning & Memory*, vol. 3, no. 6, pp. 475–502, 1997.

[28] T. Yamazaki and S. Nagao, “A computational mechanism for unified gain and timing control in the cerebellum,” *PloS one*, vol. 7, no. 3, p. e33319, 2012.

[29] Herbert Jaeger, “Echo state network,” *Scholarpedia*, vol. 2, no. 9, p. 2330, 2007. revision #196567.

[30] H. Jaeger and H. Haas, “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication,” *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[31] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[32] D. Sussillo and L. F. Abbott, “Generating coherent patterns of activity from chaotic neural networks,” *Neuron*, vol. 63, no. 4, pp. 544–557, 2009.

[33] E. A. Antonelo, B. Schrauwen, and J. Van Campenhout, “Generative modeling of autonomous robots and their environments using reservoir computing,” *Neural Processing Letters*, vol. 26, no. 3, pp. 233–249, 2007.

[34] M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[35] G. B. Stanley, F. F. Li, and Y. Dan, “Reconstruction of natural scenes from ensemble responses in the lateral geniculate nucleus,” *Journal of Neuroscience*, vol. 19, no. 18, pp. 8036–8042, 1999.

[36] G. B. Stanley, “Recursive stimulus reconstruction algorithms for real-time implementation in neural ensembles,” *Neurocomputing*, vol. 38, pp. 1703–1708, 2001.

[37] W. M. Kistler and C. I. De Zeeuw, “Dynamical working memory and timed responses: the role of reverberating loops in the olivo-cerebellar system,” *Neural computation*, vol. 14, no. 11, pp. 2597–2626, 2002.

[38] S. Mussa-Ivaldi, “Real brains for real robots,” *Nature*, vol. 408, no. 6810, pp. 305–306, 2000.

[39] M. Lukoševičius, “A practical guide to applying echo state networks,” in *Neural Networks: Tricks of the Trade: Second Edition*, pp. 659–686, Springer, 2012.

[40] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *Neurocomputing: Foundations of Research*, pp. 123–134, 1988.

[41] M. Rolf and J. 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*, pp. 3440–3446, IEEE, 2012.

[42] Y. Kawai, J. Park, I. Tsuda, and M. Asada, "Self-organization of a dynamical orthogonal basis acquiring large memory capacity in modular reservoir computing," in *International Conference on Artificial Neural Networks*, pp. 635–646, Springer, 2022.

[43] M. D. Mauk and D. V. Buonomano, "The neural basis of temporal processing," *Annu. Rev. Neurosci.*, vol. 27, no. 1, pp. 307–340, 2004.

[44] S. Chevallier, "simple_esn." `https://github.com/sylvchev/simple_esn`, 2020. [Online; accessed 10-Januray-2023].

[45] F. Chollet *et al.*, "Keras," 2015.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[47] D. Thomas, W. Woodall, and E. Fernandez, "Next-generation ROS: Building on DDS," *ROSCon Chicago*, vol. 2014, 2014.

[48] S. Bock, J. Goppold, and M. Weiß, "An improvement of the convergence proof of the ADAM-optimizer," *arXiv preprint arXiv:1804.10587*, 2018.

[49] M. Lukoševicius, D. Popovici, H. Jaeger, U. Siewert, and R. Park, "Time warping invariant echo state networks," *International University Bremen, Technical Report*, 2006.

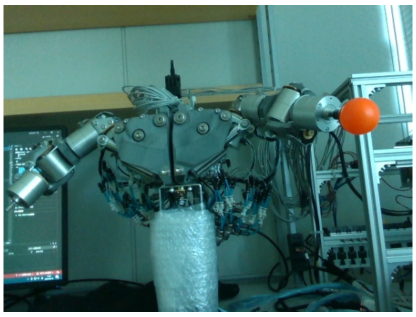
[50] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[51] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

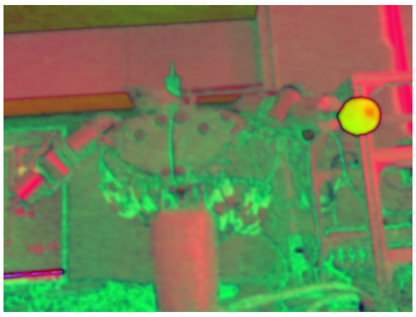
Appendices

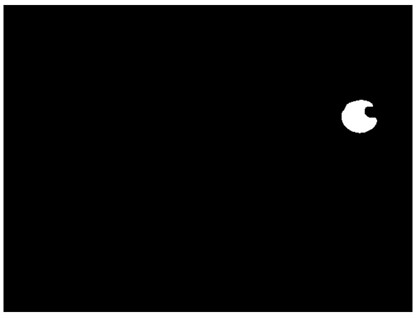
.1 Revising the end-effector tracking to enable continuous tracking

The camera with the internal software worked reliably, including in the end-effector tracking, but had the limitation of providing a frame of data only every 0.48 seconds. For recording continuous movements, which require a high frame rate, this is not fast enough. As a consequence, the choice made was to build code independent of the ”rospy” library, by utilizing the raw camera data which is provided at $30Hz$. Reading in the camera data was done with the python libraries ”OpenCV” [50] and ”Numpy” [51].
The raw data consists of the RGB (Red, Green, and Blue) values as a 2D image, and the respective depth dimension per pixel. The pixels of the 2D image may be projected into 3-dimensional space based on the depth measurement and the camera internal parameters utilized by the camera internal software. With that, the data of each pixel consists of the three RGB values and three location coordinates for each axis.
To track the end-effector, the following object tracking algorithm was implemented. First, a Gaussian blur with a kernel size of 11×11 and a standard deviation of $\sigma = 2.0$ is applied to the image to reduce noise. Next, the RGB color space is transformed to HSV (Hue, Saturation, Value), which is commonly used for object tracking. Next, a color filter capturing HSV values between $(0, 190, 160)$ and $(60, 255, 255)$ is applied. The hue value range of 0 to 60 captures red, orange, and yellowish colors, to detect the orange end-effector. Still, the end-effector is not captured perfectly, as can be seen in the information loss between Subfigure 1c and 1d. This is because the sun shining through the window from the right of the picture whitens a part of the end-effector to such a degree that the whitish color starts to overlap with other colors in the picture. As an additional measure to ensure no noise interferes, erosion is applied on a 3x3 kernel with two iterations. This is followed by equivalent dilation to restore the size of the object. This morphological opening removes small noise artifacts, but also smooths the circular end-effector. Finally, the minimally enclosing circle is applied to the largest found object, which can then be projected onto the original frame. The center of the minimally enclosing circle is used as the end-effector coordinate.


(a) Initial frame

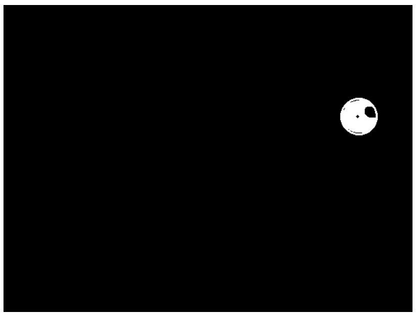
(b) Applied Gaussian blur


(c) Change to HSV color space


(d) Apply the color filter

(e) Apply erosion

(f) Apply dilation


(g) Apply a minimal enclosing circle

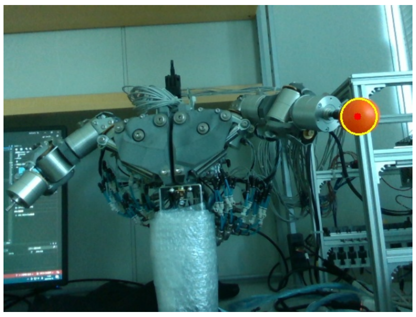

(h) Project contour into original frame

Figure 1: Image processing to detect the end-effector

.2 New data-type gathering for the inverse kinematics approximator

The data gathering algorithm 1 is used again to gather data on the relationship between the raw camera data and the joint angles, for a total of 19,843 data points. In the newly gathered data, the depth measurement contained values that seemed odd, such as a frequent occurrence of ”0”s. To investigate this, the occurrence frequency for different values was analyzed, and visualized in Figures 2, 3, and 4. A ”0” measurement occurs when the camera failed to measure anything, which happens very rarely on the X or Y axis, but somewhat frequently on the Z axis. Furthermore, the X and Y axes are normally distributed, which is in line with previous observations in Figure 4.2. The depth data, however, is not normally distributed, with coordinates between ”400 to 450” and ”950 to 1000” occurring unusually often. This indicates that the depth measurement may be inaccurate for individual frames.

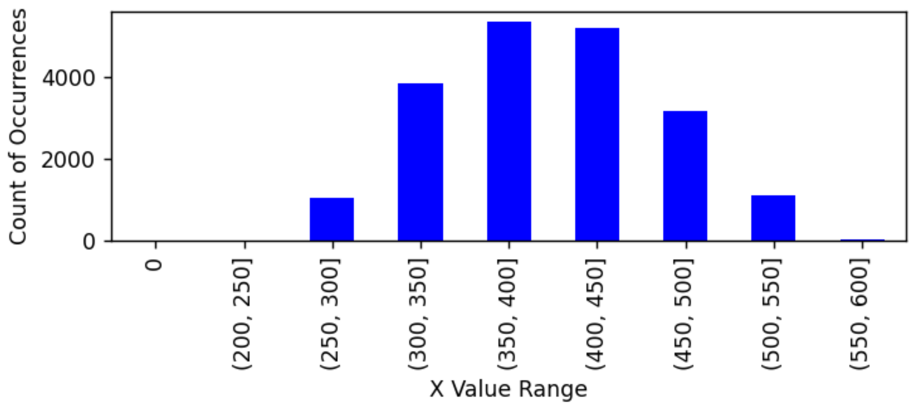


Figure 2: X axis / horizontal axis distribution in training data for the MLP controller

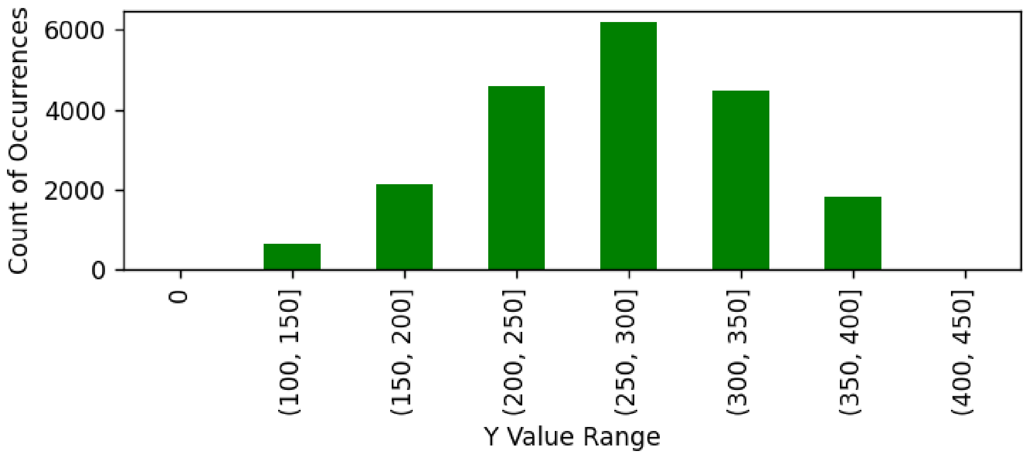


Figure 3: Y axis / vertical axis distribution in training data for the MLP controller

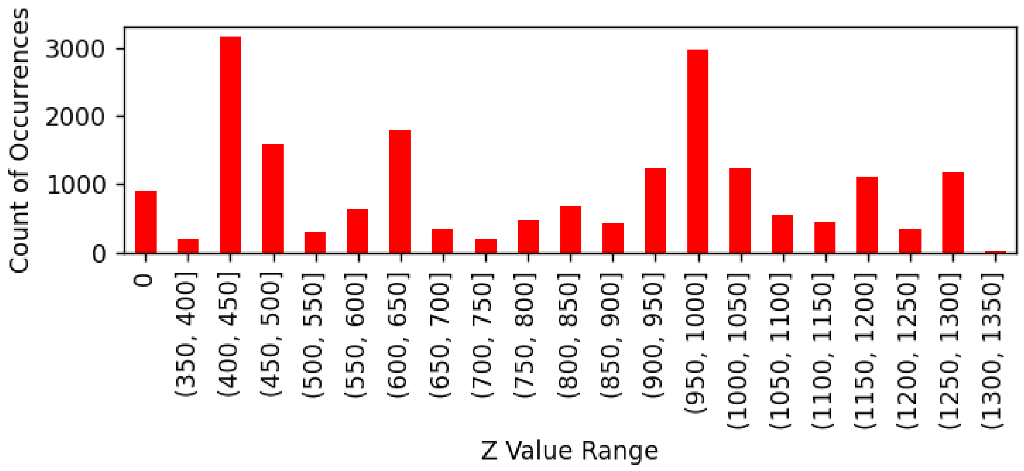


Figure 4: Z axis / depth axis distribution in training data for the MLP controller

.3 Problems with the depth measurement

As the frequent occurrence of ”0” measurements indicating a failed measurement seemed unusual. This is especially the case given that during the data gathering, the algorithm was set up to record the last valid measurement that occurs within the one second ”settling” period of the arm, making it robust to occasional failed measurements. That means that during that one second, consisting of 30 frames, all 30 of them recorded a failed ”0” measurement. This high rate of failure would be particularly problematic for continuous movements, where every frame should contain a valid measurement.
To investigate this, the depth measurement was analyzed more deeply. Looking at the depth measurement frame by frame, there appeared to indeed be a very high occurrence of ”0”s. For better visualization, the surrounding 3x3 and then the 5x5 around the pixel that marks the center of the end-effector location were explored. An example depth measurement looks like this:

$$\begin{bmatrix} 457 & 456 & 456 \\ 0 & 456 & 457 \\ 0 & 458 & 457 \end{bmatrix} \quad \begin{bmatrix} 459 & 0 & 460 & 458 & 460 \\ 459 & 457 & 456 & 456 & 0 \\ 458 & 0 & 456 & 457 & 0 \\ 460 & 0 & 458 & 457 & 456 \\ 460 & 458 & 0 & 459 & 459 \end{bmatrix}$$

Or like this:

$$\begin{bmatrix} 457 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 457 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 460 & 0 & 0 & 0 & 0 \\ 0 & 458 & 0 & 0 & 0 \end{bmatrix}$$

Since the occurrence of ”0”s is frequent, encountering one at the exact center of the end-effector location is common. However, surrounding the center, non-failed measurements may be present. The surface area of the ball-shaped end-effector is large enough to have multiple surrounding measurements record the end-effector’s depth location. Given this, a possible fix seemed to be to average the non-zero values of the centered grid of depth measurements. Implementing this with a 5x5 grid reduced the occurrence of failed measurements per frame from $\sim 50\%$ to $\sim 6\%$.
However, even if this rate of failed depth measurements were acceptable, the remaining values still appeared unreliable, as evidenced by the non-normal distribution in Figure 4. To investigate this, the robot was commanded to perform a continuous looping movement, which was recorded with the adapted depth measurement. An example loop is shown in Figure 5, consisting of 135 measurements taken over 4.5 seconds. The values fluctuate in ways that are physically impossible for the end-effector to achieve, with the visually most prominent example here being the jump from ~ 400 in purple to ~ 800 in turquoise in the bottom left after a failed measurement displayed in black.

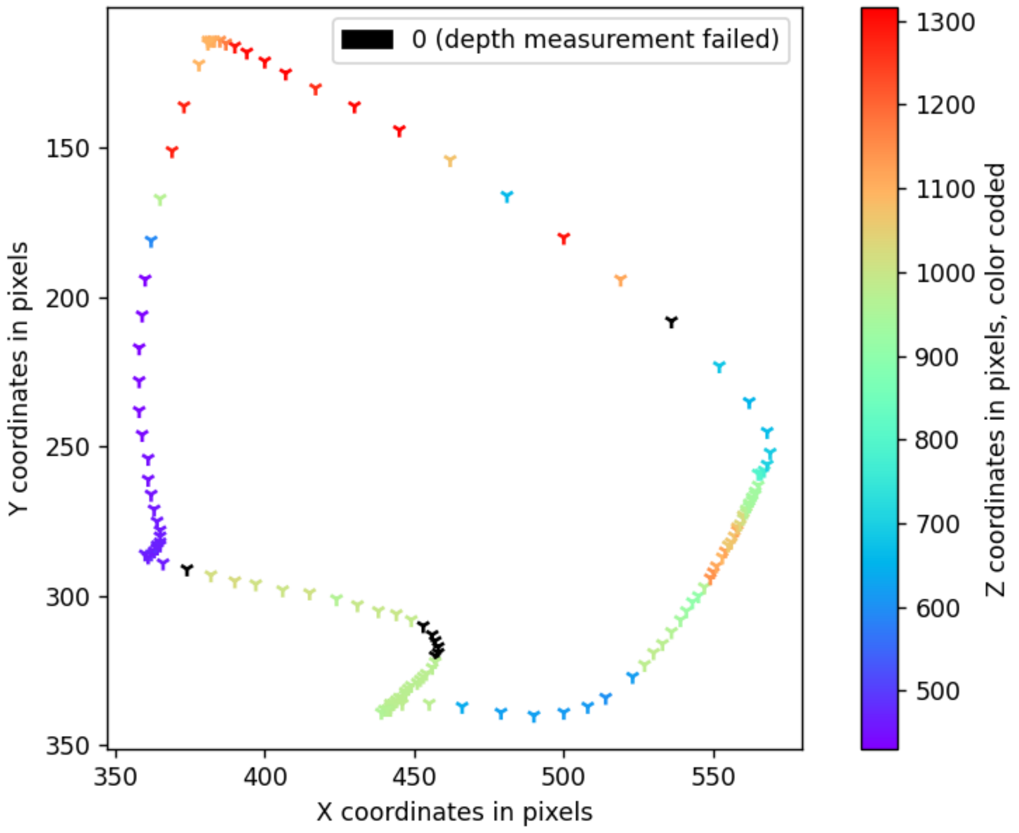


Figure 5: A continuous movement and its failed depth measurements

If the depth measurement were only unreliable in the form of containing failed measurements in between correct ones, the missing values could have been estimated based on their neighboring measurements. The previously used camera internal software managed to achieve this, at the price of a low frame rate. However, with a high frame rate, the depth measurement seems to be too unreliable, and detecting which measurements are valid appears to be a non-trivial matter.