1 2 90

UNIVERSIDADE D
COIMBRA

Margarida de Carvalho Nogueira Amaro

INTELLIGENT COLLABORATIVE ROBOTICS FOR STEEL MOLD POLISHING

Dissertation in Integrated Master's in Electrical and Computer Engineering, supervised by Professor Dr. Rui Pedro Duarte Cortesão and presented to the Department of Electrical and Computer Engineering.

September 2023

1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Margarida de Carvalho Nogueira Amaro

INTELLIGENT COLLABORATIVE ROBOTICS FOR STEEL MOLD POLISHING

Dissertation in Integrated Master's in Electrical and Computer Engineering,

Supervised by Dr. Rui Pedro Duarte Cortesão and presented to

the Department of Electrical and Computer Engineering

September 2023

”I became insane with long intervals of horrible sanity.”

Edgar Allan Poe

Acknowledgments

I would like to express my deepest appreciation to Prof. Rui Cortesão. For accepting me as his mentee and for guiding me through all the exhausting process of constructing this thesis. I’d like to acknowledge Prof. Paulo Peixoto for his guidance in machine learning, in which I had no background.

Special thanks to my friends, the ones walking by my side since elementary school and the ones university gave me, for lifting my spirit when down.

I would like to express my deepest gratitude to my parents, Pedro and Florinda, my sisters, Mafalda and Joana, and my favorite aunt, Fátima, for never giving up on me and helping me throughout my life.

Resumo

A crescente procura pela utilização de objetos de plástico tem promovido a utilização de moldes no seu fabrico e produção. Numa indústria que parece estar limitada apenas pela imaginação humana, os moldes permitem a produção em massa de objetos de plástico. A produção em massa permite reduzir o custo dos objetos de plástico, contribuindo deste modo para um maior crescimento das necessidades desta indústria. Neste contexto, a qualidade do objeto de plástico é fortemente dependente da qualidade do molde. Por conseguinte, a criação de moldes de alta qualidade a baixo custo é uma exigência da indústria. Neste contexto, o polimento é um processo crítico no fabrico de moldes, uma vez que é uma tarefa tipicamente conduzida por operadores humanos especializados, com custos elevados ao longo do tempo.

Nesta tese de mestrado, propõe-se a utilização de Redes Neuronais com Memória de Longo e Curto prazo para abordar o problema do polimento de moldes. Este tipo de redes neuronais profundas são especificamente concebidas para previsão de séries temporais, sendo adequadas para caracterizar a habilidade humana. Estas redes conseguem captar o padrão humano do polimento, gerando referências treinadas para aplicações robóticas. Neste trabalho é utilizado o robô Panda, da Franka Emika, para executar polimentos baseados na demonstração humana. A impedância do robô é controlada tendo em conta a dinâmica de interação do braço humano.

Palavras-chave: Memória de longo e curto prazo, Redes Neuronais Profundas, Aprendizagem Máquina, Arquitetura de Controlo de Impedância, polimento de moldes.

Abstract

Mold relevance in the plastic industry has increased over the years due to the high standards of modern societies. The use of molds allows the mass production of a great variety of objects for our daily lives. In this context, the quality of plastic objects greatly depends on mold characteristics. Therefore, creating high quality molds with low cost is an industry demand. Polishing is a critical process in mold fabrication since it is typically conducted by specialized operators. Mold polishing is a time consuming task, that accounts for an important part of the mold cost.

In this thesis, a Long Short Term Memory (LSTM) Neural Network (NN) is proposed to address the problem of robotic-assisted mold polishing. This type of Deep Neural Network (DNN) is specifically designed for time series forecasting, being appropriate to represent human skills. This work uses a Panda robot, from Franka Emika, to execute polishing tasks based on human demonstration. The robot impedance is designed taking into account human arm interaction dynamics.

Keywords: Long Short Term Memory, Deep Neural Network, Machine learning, Impedance Control Architecture, mold polishing.

Contents

Acronyms

Greatest Common Divisor (GCD)

Least Common Multiple (LCM)

Computer Numerically Controlled (CNC)

Long Short Term Memory (LSTM)

State-of-the-Art (SOTA)

Learning from Demonstration (LfD)

Bi-Directional Long Short Term Memory (BI-LSTM)

Deep Neural Network (DNN)

Autoregressive Integrated Moving Average (ARIMA)

Degrees of Freedom (Dof)

Franka Control Interface (FCI)

Robot Operating System (ROS)

Universal Robot Description Format (URDF)

Artificial Neural Network (ANN)

Neural Network (NN)

Feedforward Neural Networks (FNN)

Recurrent Neural Network (RNN)

Stochastic Gradient Descent (SGD)

Rectified Linear Unit (ReLu)

Mean Squared Error (MSE)

Root Mean Squared Error (RMSE)

Mean Absolute Error (MAE)

Adaptive Moment Estimation (Adam)

Root Mean Square Propagation (RMSprop)

Stochastic Gradient Descent with Momentum (SGDM)

List of Figures

List of Tables

1

Introduction

In this work, the implementation of a Long Short Term Memory (LSTM) Neural Network (NN) is proposed to predict polishing movements. In this chapter the motivation for this dissertation is described, as well as its objectives and contribution. Also, a brief background to the subject of mold and die fabrication processes is proposed. Moreover, this chapter describes the document structure and organization.

1.1 Background

Plastic industry is one of the world's fastest growing industries, ranked as one of the few billion-dollar industries. Almost every product that is used in daily life involves plastic usage and most of these products may be produced by plastic injection molding methods and techniques [8]. Molding is the processes used for the mass production of a large number of consumer products [9]. In the context of the plastic industry, in which the plastic objects are produced using molds, injection molding has been a challenging process for many manufacturers and researchers aiming to develop products to meet requirements at the lowest cost [10].

Besides the plastic industry also glass container fabrication uses molds for mass production of objects. The uses of molds allows the full automation of plastic and glass industries, thus largely increasing the production of this type of objects. In some cases, the reproduction of some art pieces, like sculptures, may be done using molds. The process of plastic injection using molding is well known as the manufacturing technique to produce at low cost, objects with complex geometry and various shapes [8].

Injection molding machines are specialized plastic molding machines, in which a thermoplastic material is used. With this method, the thermoplastic material is melted by heating, and subsequently pushed into the mold cavity by a high pressure injection process. In this way, a variety of shapes of plastic products are formed by holding the injection pressure for a determined time, followed by a certain period of time for cooling [11]. The plastic injection molding process may be characterized by four significant stages. These stages are known by filling, packing, cooling and ejection. This fabrication process of plastic injection molding begins with feeding the resin mixed with the appropriate additives from a container to the heating/injection device [8]. This stage is named the "filling stage" in which the mould cavity is filled with a polymer melted at a pre-determined injection temperature. After the filling of the mold cavity, the "packing stage" follows. In this stage additional melted polymer is packed into the cavity using higher pressures to compensate the expected shrinkage of the cooler solidified polymer. This stage is followed by the "cooling stage" where the mould is cooled until it reaches a point where it is sufficiently rigid to be ejected. In the last stage the mould is opened and the part ejected. This last stage is appropriately named the "ejection stage" [8].

A number of characteristics of the products obtained form molding are dependent on the mold itself. The main characteristic affected by the mold quality, is the outside appearance of the produced objects. Also the product functionality is very dependant on the mold quality and fabrication technique [9]. Building high quality molds is therefore a very relevant topic when addressing mold fabrication industries. In this context, one of the most relevant mold characteristics to improve molded objects is the quality of the mold polishing. The mold polishing consists of removing surface roughness and irregularities from the mold surfaces that contact with the material being molded [9].

The polishing technique is therefore one of the most relevant parts of these object fabrication. Mold polishing is a technique that must be executed with precision, since a mold that may look smooth to naked eye may cause irregularities in the produced plastic object. A defective polishing may produced objects that look worn out and present a defected appearance in the final product, thus causing a decrease in the final product quality and increasing the possibility of consumer rejection of the produced objects. On the other hand, if a mold is too well polished it may lead to vacuum formation between the mold and the plastic object causing their separation to be difficult [12].

The polishing process is usually conducted manually by human operators accounting for 37% to 50% of the total time needed to produce a mold [13] [14]. The fact that is conducted by human labor makes the process of polishing time consuming. This process exposes the specialized workers to high levels of noise as well as high concentration of metallic dust. Promoting the appearance of ”vibration white finger” syndrome, caused by the vibration of the polishing tools after years of exposure. Another characteristic problem of polishing processes is the difficulty in maintaining the polishing tool stable for long periods of time [13] [14]. Furthermore, some companies have difficulties finding skilled workers to perform the polishing task with the desired quality [14].

Currently, the polishing process in the mold industries is still conducted manually since it’s not possible to achieve the same levels of quality with automatic processes. In this context, the skilled worker may decide where to apply more or less pressure in the mold, as opposed to the autonomous system, that only can follow pre-established sequences [12]. Pre-estabelished sequences are naturally limited and therefore may only be used in a reduced number of molds. Complex geometry molds are difficult to polish by robots that execute programmed sequences.

The motivation for this dissertation emerges from the need to automate the polishing process in the mold industry. The automation of this process, even partially, may allow the industry to produce molds faster, thus meeting consumer demand and better fulfilling the aim of Industry 4.0 concept. This dissertation aims at studying and investigate how a robotic arm may perform the work of a human worker, in the process of polishing a metallic part. This way, it may be possible to address the problems identified in this section, namely workers’ health issues.

1.2 Objectives

This work aims at developing an automated mold polishing process. This dissertation objective is to develop strategies to help skilled operators in the mold polishing process.

With this idea in mind, this work aims at using a Panda robot arm in the mold polishing process. Also, the objective of this work is to study the use of Neural Networks, to predict and reproduce human movements for mold polishing processes. A demonstration and demo of the developed work will be carried out.

1.3 Contributions

This study proposes an Long Short Term Memory (LSTM) Neural Network (NN) to code human skills associated with polishing tasks. The designed LSTM drives a robotic arm for autonomous operation.

1.4 Structure of the Document

This document is structured as follows:

- State of the Art (Chapter 2) - Identifies control architectures for robot manipulation, focusing on human-robot transfer skills techniques and the growing use of Neural Network to execute human daily activities.
- Methodology (Chapter 3) - Describes the equipment, tools and techniques that have been used in the development of this work. Also the Cartesian Impedance control architecture for the robot manipulation is described. Moreover, an introduction to Neural Network, focusing on the learning algorithm developed to generate polishing patterns, is proposed. The methodology used to process the data is presented.
- Results (Chapter 4) - Results of several LSTM Neural Network, with different configurations, to generate polishing patterns, are presented. The best identified patterns are tested in simulation environment before being reproduced with the Cartesian Impedance control architecture Panda robot.
- Conclusion and Future Work (Chapter 5) - Improvement suggestion to consider in a future work.

2

State of the Art

This chapter presents a literature review on control architectures for robot manipulators, human robot skill transfer and of Neural Network. In Section 2.1, the identified literature on control architectures for improving robot movements based on human behavior is presented. This literature review was made aiming at better understanding the control architecture for human-robot skill transfer. Section 2.2 analyses literature in the area of human demonstration for robotics demonstration. In Section 2.3 the topic of Neural Network to solving problems of forecasting time series is addressed, including the use of LSTMs.

2.1 Control Architecture

One striking characteristic of the proposals for future factories is related to the idea of bringing humans close to robots [2] at the same workplace. The aim is to promote an efficient collaboration between human and robots in the production process. The proposed future manufacturing processes are integrated in the general designation of Industry 4.0, which includes contributions of Cyber-Physical Systems or The Internet of Things [2]. However, although robots have been successfully used in many tasks and their operations have been considered safe, more complex processes aren't conducted by robots alone due to their reduced cognitive capabilities. Therefore, the combination of human cognitive capabilities with robot collaborative skills allows not only to work together in partial unfamiliar environments but also to collaborate with other agents for accomplishing a common goal [2]. During the last four decades, the problem of control and motion planing in repetitive and burdensome task, such as polishing, have been studied and researched by both industry and academy [2].

The first approach for automatic mold polishing was done with Computer Numerically Controlled (CNC) machines. These machines have a remarkable position accuracy and the ability to simultaneously adjust trajectory, posture and force during the polishing process [13]. However, the limited working space of CNCs requires the process of mold polishing to be divided into several steps and also imposes a limit on the size of the mold part to be polished [13]. Furthermore, molds with complex geometries require specific fixtures and unique techniques and movements. Recently, due to its advantages when compared to CNCs machines, the topic of robotic machining and finishing has been the subject of research of both industry and academy. Robotized machine and finishing methods, present lower costs, greater flexibility, and the ability to integrate actuators and sensors namely by using different types of grippers [13]. In addition, industrial robots work with various types of parts, of varying sizes and with complex geometries without the need for specific accessories. Thus, industrial robots have become an effective and economical solution in subtractive manufacturing industries for objects with complex geometric shapes regardless of the workspace [13].

In Claudio Gaz et al. [15], the authors propose an architecture for a robot which has the capability of sharing the work space with a human worker, without requiring any mutual contact or coordinated actions. The objective is to have a user operating the robot in complex geometry situations of mold, being the manipulation made directly with physical contact and not by an interface. With a three layer architecture, where the intermediate layer allows for coexistence, since it has the ability to prevent collisions in real time, i.e., by monitoring the workspace with cameras and sensors or RGB-Depth devices (which allows to have pixel-to-pixel depth information). The collaboration between the human operator and the robot corresponds to the top layer, where the robot performs complex tasks in direct contact with the human. The assignments are performed with exchange of forces/torques activated by communications such as gestures, voices and direct physical contact with the robot. Collision detection and reflex reaction is implemented in the lower layer without the use of sensors. A model-based scheme that monitors the generalized momentum of the robot. while the robot secures the mold for manual polishing, forces and movements performed by the human operator are reflected in the robot structure, from the gripper to the joints.

Zhou Shenghao e Song Jinchun [16] propose the implementation of a position-based impedance control system to aid human interaction with automobiles. To build a viable application of the interaction between the two it is necessary to control not only the motion, but also the forces of the interaction of the vehicle with the surrounding environment. In a system with human-robot interactions it is necessary to consider the control properties of a human operator as well as the accuracy and performance of the system control, to achieve a natural interaction between the two. This article focuses on admittance and impedance control. The first method is widely used since a position control interface is available in every robotic system. The other method requires a joint torque or joint impedance interface.

Wei Hu et al. [17] propose the use of impedance control to solve the interaction problem for a robotic manipulator. Impedance control not only gets the position or force trajectory, but involves regulating the impedance of the robot end-effector that relates position to force. The biggest problem with impedance control is in situations where it cannot deliver enough power due to non-linear inputs. That can be caused by magnitude constraints and the precise input rate. Thus, it is necessary to design an impedance control that can compensate for the inputs to maintain the stability of the robot.

Hélio Ochoa and Rui Cortesão [2] propose a computed-torque architecture for an assisted polishing robot. The goal is to transfer the polishing techniques and skills from the human operator to the robot, so that techniques taught to a given model can be performed on a model with different characteristics. The control strategy used is impedance control in task space with posture optimization. The goal with impedance control is to assign a predetermined dynamic behavior in the presence of external interactions, coinciding with the dynamics of a mass-spring-damper system.

2.2 Human Transfer Skill

Robot capabilities and the development of ”intelligent machines” has not been able to keep up with the advances in computer technology. This disparity is mainly caused by the difficulty in formalising intelligent human behavior and decision making processes into algorithms [18]. Human perform everyday tasks, such as visual processing, manipulation and mobility with relative ease. However, robots can’t duplicate this performance adequately. Although humans are successfully performing these tasks, they have difficulties describing them [18]. Therefore, studies have been carried out on learning models for the transfer of techniques from humans to robots [18].

Rui Wu et al. [19] propose a framework for transferring the movements of skilled workers to a variable impedance model. A simplified real-time 3D model of the end-point of the human arm is proposed based on the regular operations of humans and the effects of antagonistic muscle contractions and is calculated based on experimental results. A tele-demonstration method for variable impedance skill transfer is then build by calculating the proposed model in real time. The data can be automatically classified by changing the tutor’s arm impedance, and a modular Learning from Demonstration (LfD) method is used to learn the data for different purposes. By focusing on the operating habits of humans, this model helps tutors conveniently adjust the robot’s impedance direction and divide the tasks into several stages.

In [20] a technique to capture movements of specialized operators used in polishing processes studied in [2] is deepened. The methodology proposed in [20] is based on detecting the position/torque of the robot to: 1) Capture a specific technique/movement from operators; 2) Transfer that technique to a robotic system for autonomous execution, i.e. LfD. A Cartesian impedance control is used with posture optimization, where the operator technique is coded by parameters and reference positions. The robot posture is optimized by keeping its joints at values near the center. In order to collect information about the movements made by the workers, the co-manipulation mode in gravity compensation is used. While the free-hand mode allows the force associated with the polishing task to be captured. In this case, the robot impedance control has a rigid design while the operator applies forces that are captured by torque sensors on the joints. Motion and force patterns cannot be captured at the same time since in co-manipulation mode part of the reaction forces are absorbed by the human arm. To test the proposed method a mold with complex geometry is used, where task generalization is designed, analyzed and performed for surfaces in 3 dimensions with different shapes. Additionally, mold polishing was performed with different polishing stones in order to validate the human-robot technique transfer approach.

2.3 Neuronal Networks

Machine Learning, a branch of artificial intelligence, is playing an increasingly important role in the world of science [21]. The development of several areas, such as bioinformatics, physics, mathematical analyses, among others, requires intelligence methods to enrich the content of these disciplines [21]. Artificial Intelligence has been study with greater intensity both academically and commercially, since it allows the resolution of everyday problems. In [22], a method was developed to predict car traffic flow using Machine Learning and image processing algorithms. This methods has been applied to self-driving cars. Machine Learning can also be applied to predict the health state of patients in an intensive care unit [23]. Within the context of this work, Machine Learning may also be applied during the polishing process of molds. Specifically, it may be applied to molds with diverse and complex geometries.

Arif Istiake Sunny et al. [6] proposes the use of two popular models of Recurrent Neural Network for forecasting of stock price gains, thus attracting attention of the financial world by this technology. In this case, stock price prediction may be a relevant factor to increase investor interest in the company's market stocks. A number of proposed uses of Recurrent Neural Network apply LSTM and Bi-Directional Long Short Term Memory (BI-LSTM) models. Motivated by the expanding use of Deep Learning Algorithms to forecast future patterns in different time series applications, hidden structures may be discovered to forecast stock prices. Time series forecasting is a demanding area of research due to its enormous potential in different applications such stock price forecasting, business planing, weather forecasting, resource allocation and numerous others.

Karim Moharm et al. [24] presents the deep learning algorithms, LSTM and BI-LSTM using different configurations and different activation functions to evaluate the experiments and predict the provisional trend of wind speed. The accurate forecast of wind speed is critical in the integration of renewable factors in electric power grid stability, scheduling and planing. Recently, the cleaner production of electrical energy, has gained a lot of attention because of the increased concerns on environmental pollution and the need to reduce the use of non-renewable sources. Wind energy represents a solid clean renewable energy source. However, the stochastic intermittent behavior of wind represents a challenge for this type of energy source management. Hence, the accurate foretelling of the wind speed and relatively wind power can enhance the grid operation. Effective wind energy prediction, results in low-cost efficient and safer operation of the grid. The more accurate wind prediction, the more reliable and efficient power dispatch, energy storage systems, and effective power transmission, may be implemented.

Peng Kaibei et al. [25] proposes a short-term passenger flow in urban rail transit prediction model based on an improved LSTM. This approach, is proposed to solve the traditional Neural Network model problem to predict complex nonlinear data. Using scientific methods to analyse and predict the future trends of passenger flow, is helpful for managers to grasp the evolution trend of passengers in a timely manner and reasonably allocate service resources of stations. With the development of artificial intelligence and the rise of big data, intelligent computing and Machine Learning methods are gradually applied to various application scenarios and achieving good results.

Still within the context of traffic management in intelligent transportation systems, Dilantha Haputhanthri and Adeesha Wijayasiri [7] propose LSTM based deep learning models with different architectures to understand the best forecasting model. In this work, a LSTM model, an encoder-decoder LSTM model, a convolution neural network LSTM (CNN-LSTM) model and a Convolution LSTM (Conv-LSTM) model are designed and developed. Short-term traffic forecasting has become a major asset in the field of real-time traffic management and transportation planning. Traffic volume forecasting can be utilized in optimizing routing strategies and journey planning to reduce traffic congestion and accidents. Long Short Term Memory is one of the commonly used Deep Learning techniques, designed specifically for time series forecasting, by employing memory status and feedback connections, unlike standard feed-forward networks.

3

Methodology

Chapter 1 describes a number of open issues in the mold polishing industry. The introductory chapter illustrates this work's motivation and frames the guidelines for this thesis. In chapter 2 a number of recent literature identified proposals involving control architecture, transfer of skill human-robot and Deep Learning structures for regression problems have been described. This chapter, in Section 3.1, describes the experimental setup by introducing and describing the Panda Robot. Followed by a description of the control architecture used to control the Panda Robot. In Section 3.2, an introduction on Neural Network as an approach to the problem of time series forecasting applied to automatic mold polishing is presented. Moreover, hyperparameters and other NN parameters are analysed to evaluate DNN the performance. Activation functions and optimizers algorithms are discussed and presented as well.

A time series is a discrete or continuous sequence of observations that depends on time. Time series analysis involves working with time based data in order to make predictions about the future [26]. Traditionally, time series forecasting has been dominated by linear methods like Autoregressive Integrated Moving Average (ARIMA), however they present a number of limitations. Machine learning methods may be effective on more complex time series forecasting problems [27].

3.1 Panda Robot

The robotic arm Panda, illustrated in Figure 3.1, was developed by Franka Emika, a German company, focused on bringing human and robots close together.

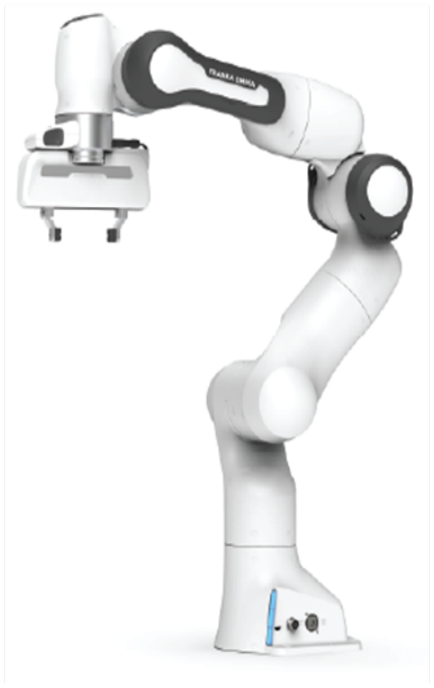


Figure 3.1: Panda robot provided by Franka Emika [1].

The robotic arm is considered a collaborative robot, also known as cobot, that as the name suggests, work well together with human operators. These robots are ”sensible” and aware of their surrounding environment, consequently they don’t need to be in cages and the risk of accidents is reduced. Cobots tend to be small, light and easy to assemble. With human arm-like complexity, that enables sensitive manipulation and force-enabled applications, enabling it to mimic human-like, dexterity-based skill sets [1].

The robot was designed to be user friendly with a easy setup and an intuitive interface that doesn’t requires prior knowledge of programming, the Franka Emika Robot System includes the arm and its control. The robot has 7 Degrees of Freedom (Dof) with torque sensors in each joint, that allows for adjustable stiffness/compliance and advanced torque control. The provided end-effector is a 2-finger gripper with exchangeable fingertips, that is integrated with the software of the Franka Emika Robot System. The gripper has an opening up to 80mm, that allows it to lift 3kg with a continuous force of 70N [1].

The version of the robot used is the Panda Research since it is the version advised for testing control architectures and algorithms [1].

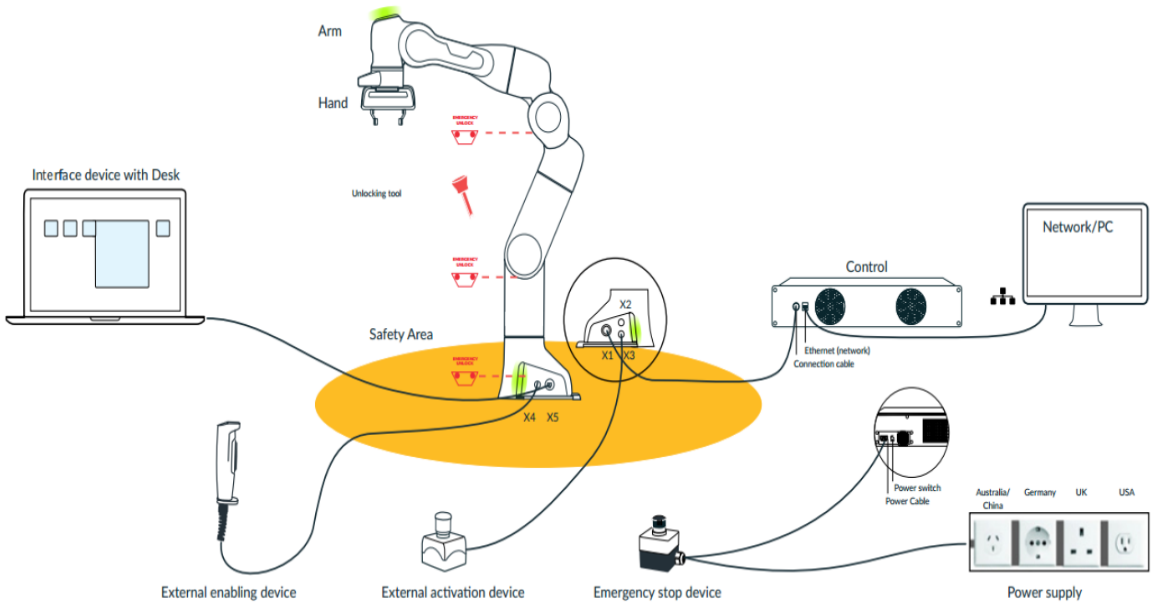


Figure 3.2: The set up of the Panda Robot [1].

3.1.1 Equipment Overview

Figure 3.2 illustrates the set up of the Panda Robot. In case of emergencies the stop device, located between the controller and the internet supply, can safely removes the supply from the robot. At the base of the arm is located the external enabling device (connectort X4). It will activate Panda robot and programs may be iniciated via Desktop, the web-based interface. The external activation device is connected at the base of the Arm (socket X3), in order to consciously authorize movements of the arm from outside the safety area. The Arm is connected via a connection cable to the Control. In order to program the robot via Desktop, the interface device is connected at the base of the arm (socket X5). Otherwise, to program the Panda Robot via Franka Control Interface (FCI), the network Ethernet interface on the front side of the Control should be used [1].

3.1.2 Communication Modes

The Panda Research version allows the user to communicate with the robot with two possible ways, either by using the web application Panda Desk or the Franka Control Interface. The robot's Desktop interface can be accessed from a web browser, Figure 3.3. Simply connect to the robot using an Ethernet cable to establish communication allowing

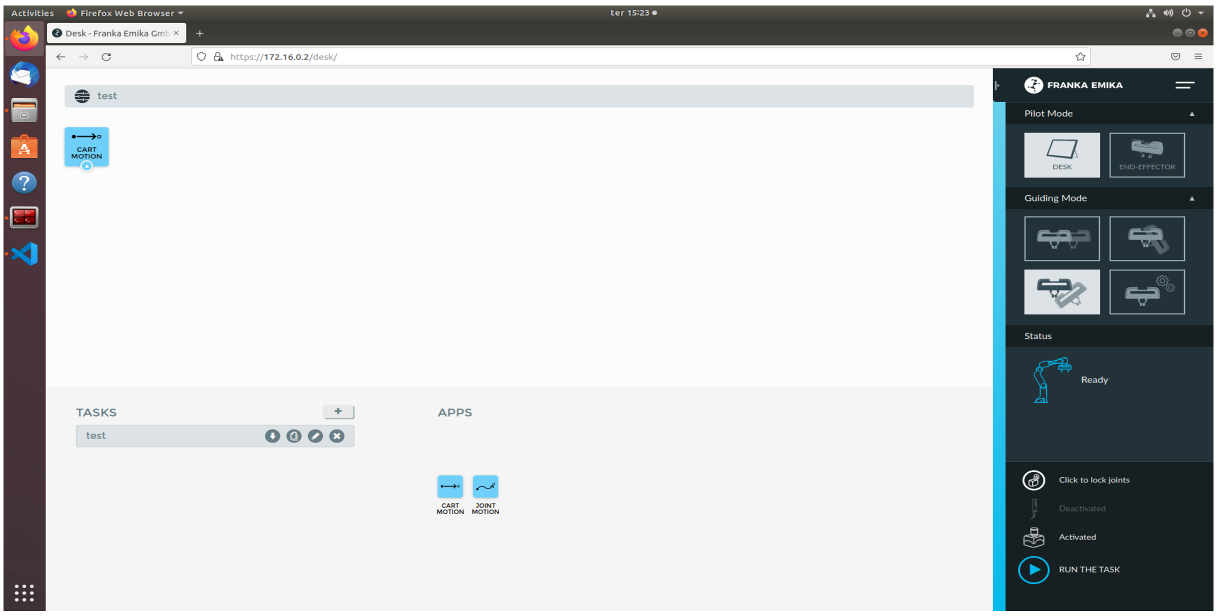


Figure 3.3: Desk top interface of Panda robot.

the manipulation of the Panda arm. Desk allows to create tasks. Tasks consists of chronological sequence of apps. That is, building blocks that describes the basic capabilities of Panda, such as ”grip”, ”put down”, or ”push button” [1].

Franka Emika develop Franka Control Interface as a solution to control and program the Panda robot. This interface allows for real-time, low-level bidirectional connection to the robot’s arm and hand, view Figure 3.4. The FCI enables direct control of the robot and provides its current status. The package consists of a C++ program library named libfranka and franka_ross, in Robot Operating System (ROS) interface with ROS Control, includes Universal Robot Description Format (URDF) models, detailed 3D meshes of the robot and end-effector, allowing visualizations for simulation environments [1].

Libfranka is the implementation of the client side of the FCI, enabling the connection on client applications via standard Ethernet to a Panda with activated FCI. It handles the network communication with Control and provides interfaces to execute non-real time commands to control the hand and configure arm parameters. It also executes real time commands to run user 1 kHz control loops. Allows to read the robot state to get sensor data at 1 kHz. And can access the model library to compute user desired kinematic and dynamic parameters [1].

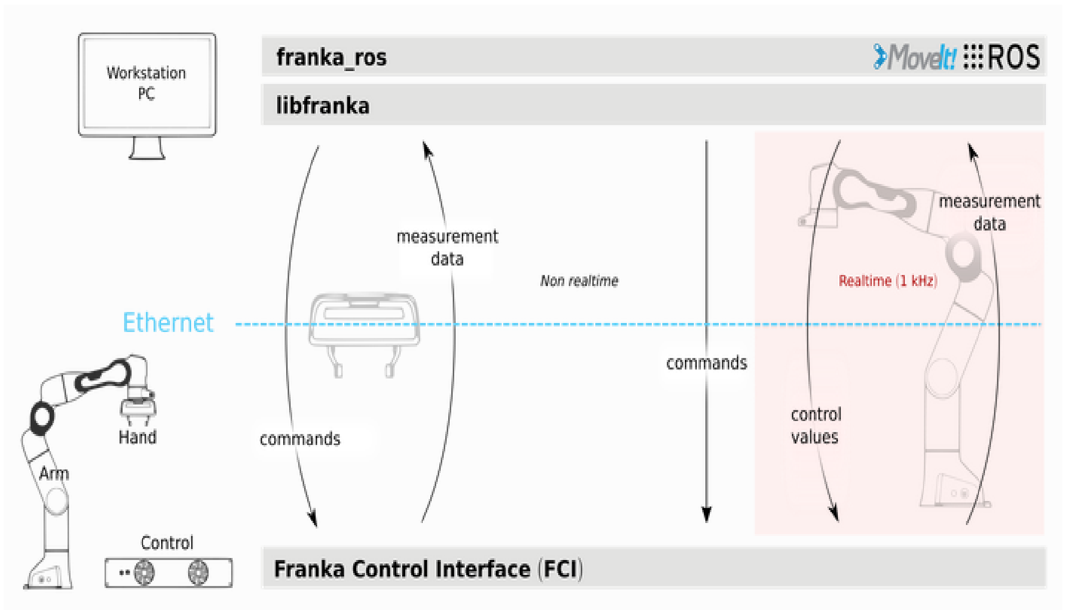


Figure 3.4: The schematic overview of the FCI [1].

The franka_ros is distributed with a Gazebo package, allowing for a robot simulation that is an essential tool, see Figure 3.5. The simulation makes it possible to rapidly test algorithms, design robots, perform regression testing and train system using realistic scenarios.

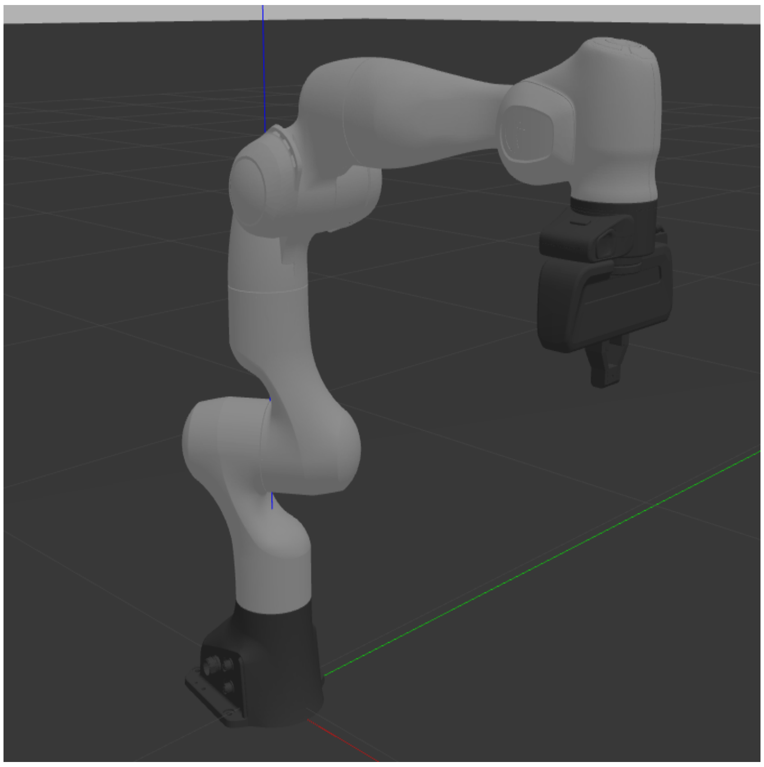


Figure 3.5: Panda robot simulation on Gazebo.

3.1.3 Control Architecture

The scheme of the Cartesian Impedance Control with Posture Optimization is presented in Figure 3.6 [1]. In this figure the impedance control is highlighted. As inputs we have the current and desired position as well as the current and desired orientation. This formulation is used to compute the position and orientation error. In the end-effector frame, the error is filtered with proportional, integral and derivative factors in order to obtain the force of the task space. The torque to perform the task is thereafter computed by multiplying the Jacobian by the force.

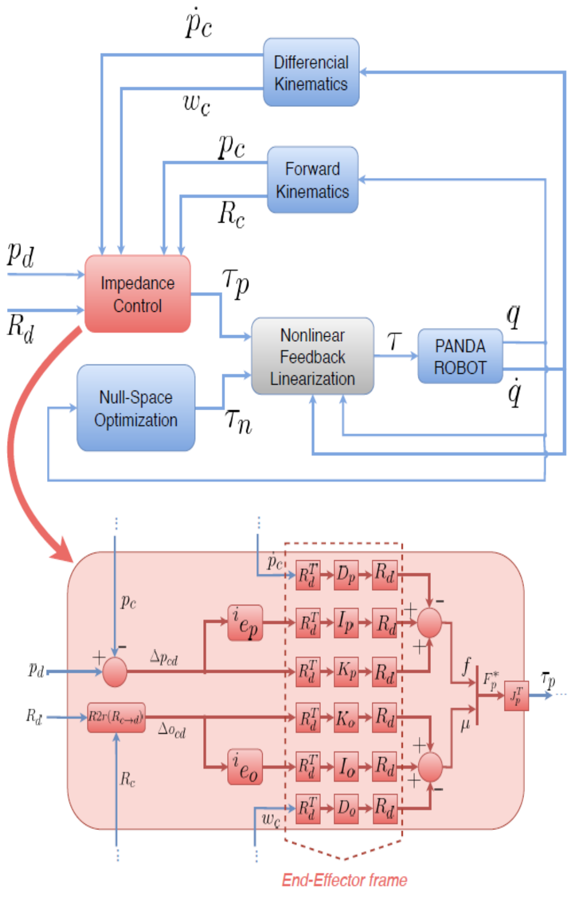


Figure 3.6: Control architecture for mold polishing. A Cartesian impedance controller with posture optimization, where Cartesian positioning is the primary task and posture optimization is performed in the null-space [2].

The Cartesian space allows for a description of the robot pose. In this descriptio, the position and orientation of the end-effector are the main focus. Movements in Cartesian space allow the exact tracking of predefined paths in space, such as straight lines. The changing of position is called translation, while the changing of orientation is named rotation [1].

Impedance describes the robot resistance to movement, being represented by a mass-damper-spring system. This behavior description may be used to safely interact with the surrounding environment, for example by preventing damage of nearby fragile objects. Spring designed allows impedance shaping, an ability similar to the human arm, which codes muscle activation to adapt compliant movements depending on the situation. This ability boosts robustness when executing tasks [1].

3.2 Neural Networks

Artificial Neural Network (ANN) is a commonly used technique to replicate the human brain behavior. Artificial Neural Network (ANN) have the capability to adapt their operation to stochastic uncertainties. These networks inputs are usually expressed as time series data, that is with data that depend of time variables [4].

Neural Network (NN) are appropriate implementations for approximation, classification and also prediction problems [28]. Deep Neural Network has learning capabilities for both straight and non-linear time series. For this reason Deep Neural Network are considered a relevant option for forecasting time series sequences [29].

An ANN can be seen as a structure of neurons or cells with connected with each other [3]. Each neuron can be understood as a mathematical function whose inputs are a set of values given by previous cells. The neuron thus calculates an output value that is conducted to the next neuron in the network [3]. The inputs and outputs of the network are a different types of neurons that have no predecessors or successors, working as input and output interfaces of the ANN [3]. The connections between neurons have associated weights that are changed during the process of training. During this processes the strength of the weights increases or decreases [3]. These concepts are illustrated in Figure 3.7 that shows an example of the simplest neural network: a perceptron neuron [3].

A more complex NN is the feedforwrd, that in the most simple scenery, have 3 layers, on input layer, a hidden layer and an output layer. The number of neurons in the input layer

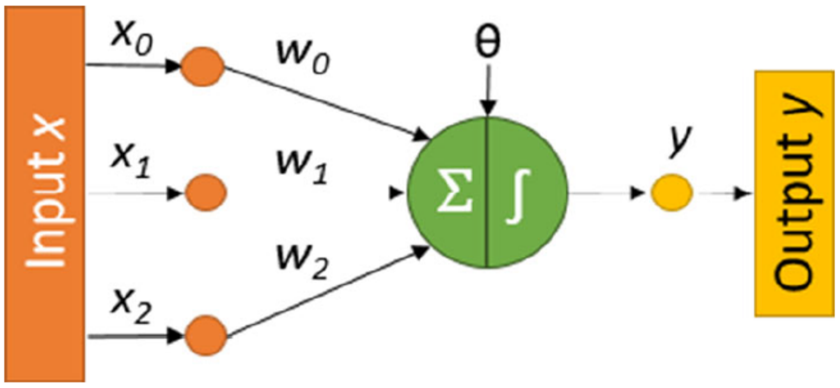


Figure 3.7: The schematic of a perceptron. It receives an input x, which is a vector with three components, though three input cells, and generates an output which is a single value. Note that θ is a hyperparameter that needs to be provided [3].

is dependent of the number of features or attributes to fed to the NN. Once, the number of features effects the network complexity, therefore more or less neurons are necessary. The number of neurons of the output layer is dependent on the prediction or classified items. In the hidden layers, the mathematical expression of the neurons compute non-linear transformations on the input data [26].

The neurons have a threshold value and an activation function. Neurons are activated if the input values are bigger than a threshold. In this case, the output is then computed through the activation function which is then transferred to the next layer in the network. The activation function maintains the output of the neuron between values 0 to 1, or -1 to +1 [26].

The majority of ANNs are supervised learning, where a set of input-output pairs are used to teach the NN so it can predict new outputs from completely new inputs. Supervised learning has two main phases: training and predicting. During training, the input-output pairs from a dataset are used to teach the network until it matches the patterns of input-output [3].

The dataset is divided into two subsets: traing and test datset [3]. The training dataset uses most of the input–output pairs and is used during the training of the ANN, that is to change the weights of the neurons [3]. The test dataset is used after the training process, with the object of evaluate the ANN performance and therefore study its accuracy [3].

The loss is a metric to monitor the quality of the training process and the overfitting, that is the situation in which the ANN is to fitted to the training dataset that is incapable of generalization and can't make good predictions for new data [3]. The loss is the sum of the error produced for each parameter in the dataset, the closest to 0 the best [3].

Contrary to Feedforward Neural Networks (FNN), in Recurrent Neural Network (RNN), the neurons have connections to neurons in the next layers as well as the previous ones, that is with backpropagation links [3]. Therefore, the neurons depend on their input state and internal state [4]. In the backpropagation process the neurons output are given back into the network, causing the ANN to remember information previously received. This kind of neurons are named memory cells [3]. It is therefore used for the study of time series predictions [4]. Figure 3.8 from [4] illustrates the difference between a feedforward ANN and an RNN in a simplified way [4].

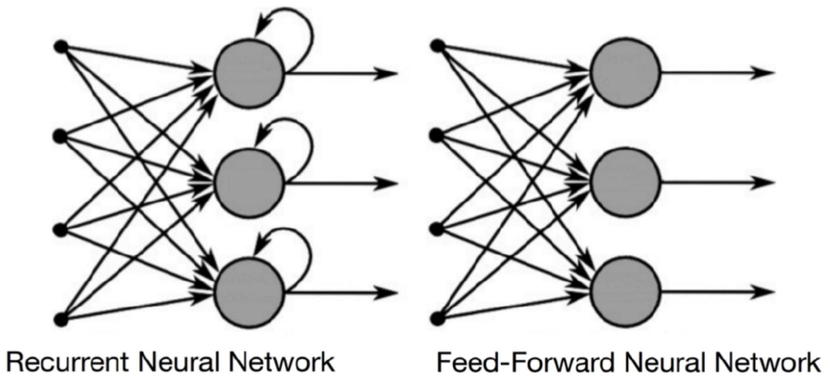


Figure 3.8: A simplified comparative illustration of an ANN and RNN architecture [4].

Basic backpropagation is often implemented in supervised learning tasks. In supervised learning, an artificial neural network is transform so that its actual output (P) becomes close to target outputs (Y) for a training set which contains Tpatterns. They focuses in changing network parameters so that there is no overfitting. Supervised learning is mainly used in pattern recognition problems. In basic backpropagation, the values of weights are initialized with arbitrary values [5]. Next, the outputs (Y) and the errors $E(t)$ are calculated for the set of weights. Then the derivative of E of all the weights is computed, as represented by the dotted lines in Figure 3.9. The weights are adjusted to decrease the error, by decreasing or increasing the values of weights accordingly. This process is repeated until the weights and the errors settled down [5].

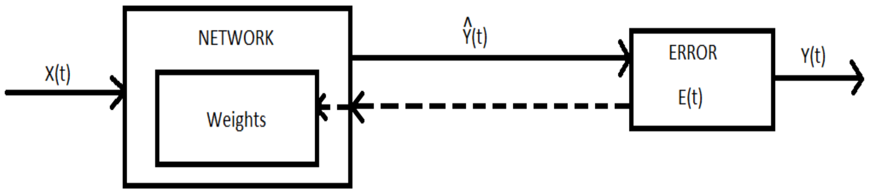


Figure 3.9: Basic backpropagation [5].

The objective is to find the weights and biases that minimize the error function. By using the gradient descent to update the weights and biases interactively to minimize the overall network error. The weights parameters are interactively modified in the direction of the gradient until minimum is reached [26].

In traditional gradient, the all input data is used to calculate the gradient at each iteration. For larger datasets, redundant computations are calculated because gradients for very similar examples are recomputed before parameter updates [26].

Stochastic Gradient Descent (SGD) often converges to a solution much faster then the traditional gradient, because of the lack of redundancy. SGD an example is randomly selected to update the parameters and move the direction of the gradient at each iteration.

The size of the steps to reach the minimum by the gradient descent algorithm is given by the learning rate. With a large learning rate the network may learn very quickly, however the network can miss the global minimum therefore being incapable of learning. On the other hand, a lower learning rate takes longer to find the optimum value [26].

Momentum is another method that can be used to help find the local minimum faster. Taking a value between 0 and 1, it adds this value of the previous weight updates to the current one. With a high value for the momentum parameter the training time may be reduce and may help the network not getting trapped in a local minima. However, a high value for the momentum may increase the risk of overshooting the global minimum. This is escalated with a combination with a high learning rate. On the other hand, setting the momentum parameter with a low value may cause the model to be trapped in a local minimum [26].

3.2.1 LSTM

RNN's architecture are specifically designed for processing problems with sequential data using its internal memory states. These networks, also support feedback connections for backpropagation mechanisms unlike standard feedforward networks [7]. However, RNNs may suffer from problems of vanishing gradient [3].

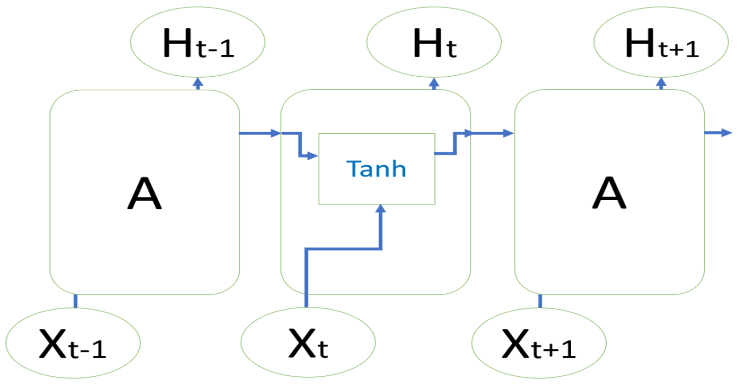


Figure 3.10: The repeating model of an RNN [6].

Vanishing gradient problems come to light when a cell has to remember information for long periods of time. This problem tends to transpired when training a Neural Network using gradient-based learning methods and backpropagation, stopping a weight from changing its value [4]. This becomes a problem because the computations uses finite-precision numbers [4].

Long Short Term Memory NN are a specific type of RNN composed by LSTM cells that were specifically designed to solve the problem of the vanishing gradients [3] by allowing gradients to flow unchanged [4]. Eventhough, LSTM can remember better than their predecessors they are more computational expensive because of the increased number of operations and complexity of its propagation function [3].

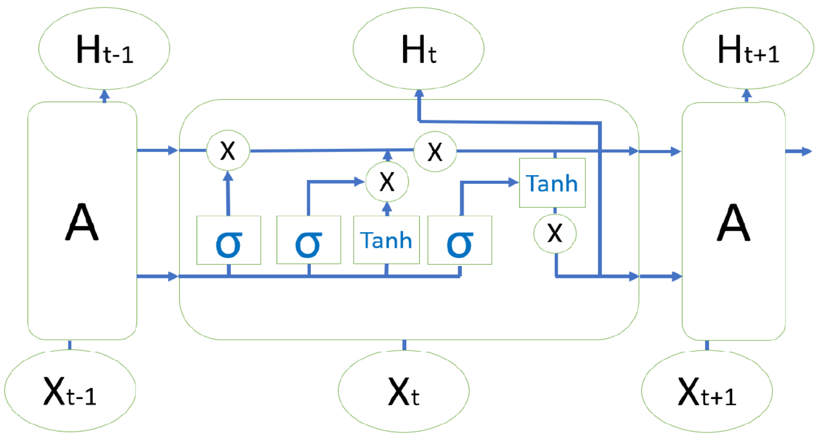


Figure 3.11: The repeating model of an LSTM [6].

LSTMs are widely use in the case of large problems, more specifically in the case of time series prediction. LSTM neuron has gate units and memory cells, that allows to recollect data over a period of time [6].

Cell states store recently expired information inside the memory cells. When the data arrives to a memory cell, the output is controlled through cell state combination, that is refreshed afterwards. If any other information is received by the memory cell it will be used to process the output plus the new cell state. LSTM default conduct is to remember information for long periods of time [6].

The core of the LSTM neural network is the addition of memory neurons and gate units. It allows for a slower rate of information loss and an enchanting of the information storage because the information in the earlier time units can also be transferred to the later time unit [25].

LSTM uses three gates to regulate the flow of information into and out from a cell [4]. Figure 3.12 illustrates a common LSTM neuron or a unit that is composed of a cell, input gate, output gate and forget gate. The cell memorizes the feedback values over time intervals and the gates control the information flow in and out of the cell [7].

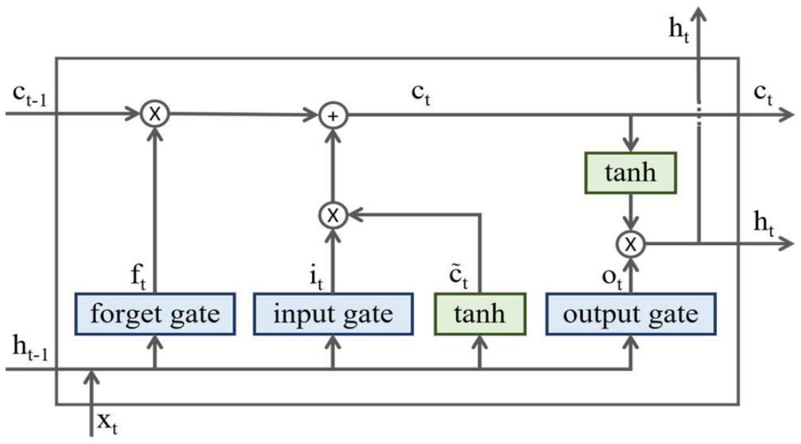


Figure 3.12: Structure of an LSTM unit [7].

Each of the gates in Figure 3.12 can be think about as standard neurons in a feedforward network in which the activation of a weighted sum is calculated. The calculated activations are i_t, o_t, and f_t for the input gate, output gate, and forget gate, respectively. The activation of the gates f_t, i_t and o_t are calculated for a time step t using the activation of the cell at $t-1$ (c_{t-1}) according to the equations 3.1, 3.2 and 3.3 [7].

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.1}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.2}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.3}$$

Here, the W and U represents the weights of the input and the recurrent connections of the input gate (i), output gate (o), forget gate (f) or the memory cell (c). Finally, c_t and h_t are calculated using equations 3.4, 3.5 and 3.6 [7].

$$c_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3.4}$$

$$c_t \ = f_t \otimes c_{t-1} + i_t \otimes \ c_t \tag{3.5}$$

$$h_t \ = o_t \otimes tanh(c_t) \tag{3.6}$$

Here, σ represents the sigmoid function and in both Figure 3.12 and the equations, $\otimes$ denotes the element-wise multiplication. Furthermore, the upward output, h_t usually goes through another activation function, which is then considered as the output of the unit [7].

Gate is an approach to control whether the data can enter into the cell state, or not. Gate is a combination of a sigmoid function, and a point-wise multiplication process. The sigmoid function can generate any number from zero to one. This value controls the passage of data in a way that an estimated zero signifies ”do not pass anything” while an estimated one signifies ”pass everything”. For LSTM model, different gates are used to pass our recently experienced data from one cell to another cell. These gates are known as update gate, forget gate, and output gate [6].

These cells are used to control the memory of LSTM model. Here in LSTM, both the activation values and candidate values were used. Thus, LSTM generates two outputs from the cell, one is the activation, and another is the candidate value. The data is passed through the level line which is the highest point of Figure 3.11. This level line is termed as cell state [6].

3.2.2 Parameters and Hyperparameters

A Neural Network model has two types of coefficients, the parameters and the hyperparameters. Parameters are those coefficients whose values change during the learning process as the model gets optimized with the objective of reducing the errors. An example of such coefficients are weigths and bias. Moreover, hyperparameters are those coefficients that are manual given to the network depending on the optimization strategy and are never updating during the training process [30]. Choosing the right hyperparameters has a critical impact on the performance of the network [3]. Depending on the problem in question and the dataset being used a different configuration of hyperparameters is required, rending each situation unique. Therefore, the task of choosing the this values is a challenging [31].

There are two main approaches for hyperparameter optimization: manual and automatic. In the manual approach, hyperparameters optimization is made by experts, who interprets how the hyperparameters affect the performance of the model, changing accordingly. Automatic hyperparameter optimization methods are difficult to apply due to their high computational cost and time consuming problems [32].

Automatic algorithmic approaches range from simple Grid search and Random search to more sophisticated model-based approaches. Grid search explores all possible combination of hyperparameters values to find the global optima, therefore being very time consuming process. Random search algorithms, which are based on direct search methods, are easy to implement. However, these algorithms are converged slowly and take a long time to find the global optima [32].

DNN models require many hyperparamters to be set for learning [33]. The hyperparameters needed to be considered in this study are as follows:

1. Train-test split ratio;
2. Number of hidden layers;
3. Number of neurons in the hidden layer;
4. Number of epochs;
5. Optimization algorithm;
6. Learning rate in optimization algorithm;
7. Momentum in optimization algorithm;
8. Loss function;
9. Weight initialization;
10. Activation function;

11. Batch size;

12. Time steps.

For a DNN to learn patterns the same input-output pairs from the traing dataset are feed to the model several times during the process of learning. Each time the complete traing dataset passed thougth the network an epoch is completed [3].

In each epoch, the training dataset is randomly shuffled and split into batches of input–output pairs that are passed through the ANN. Each time a batch is passed, an iteration is completed [3].

3.2.3 LSTM Model

3.2.3.1 Environment

The development of Long Short Term Memory Neural Network in this thesis was done with MatLab. The Deep Learning Toolbox provides a framework for designing and implement Deep Neural Network to perform classification and regression on images, time series and text data. It was also use Google Colabboratory, also know as Colab. Colab allows to write and execute arbitrary python code through the browser, and is specially well suited for machine learning and data analysis. Using the open source TensorFlow library for machine learning.

3.2.3.2 Data Collection

In [20] the robot is placed in gravity compensation for the specialized operator to be able to manipulate it. The user handles the robot, which has an end-effector that is similar to the polishing tool used by human operators. As the user performs the polish operation the position (x, y, z) are being recorded into a file text. The capture motion was performed in a horizontal plane, since it was more comfortable for human demonstration.

The free-hand mode of the robot in [20] enables to capture human force patterns associated to the task, by direct manual polishing of a small surface attached to the robot end-effector. In this case, robot impedance control has rigid design and human applied forces are captured by joint torque sensing. Position and force patterns for polishing can therefore be study for human to robot skill transfer. Force and position patterns cannot be capture simultaneously since in co-manipulation mode part of reaction forces are absorbed by the human arm. There is a high frequency noise filtering for both position and force data, using a first-order low pass filter with 100Hz cut off frequency.

Two different patterns were captured by the robot and used to train the Neural Network of this work.

3.2.3.3 Data Processing

The raw data was capture in intervals of 1 millisecond. Creating a sequence of points dependent of variable time. Therefore, establishing the problem of this thesis as a time series problem. The data was capture in relation to the robot Panda, being the origin in the base.

Considering the control architecture of the robot is an impedance control, that is, the robot force is indirectly manipulated by position (z coordinate) it is not necessary to generate new z coordinates from the LSTM. Once the z coordinates values given to the robot are the mold position. Therefore, the values study in this thesis are of x and y coordinates.

Data is captured at a frequency of 1Hz, while the human arm has a bandwidth of 4-6 Hz. Within this context, removing data allows faster Deep Neural Network training without compromising the output. Hence, for pattern one NN 50 successive points of every 51 point blocks are removed from the original dataset. The obtained datasets are illustrated in Figures 3.13 and 3.14 for LSTM with only one feature. Figure 3.15 illustrates the processed dataset for two input feature LSTM. Pattern two removes instead 30 successive points of every 31 point blocks from the original dataset. These sets are illustrated in Figures 3.16 and 3.17 for LSTM with only one feature and Figure 3.18 for two input features. To ensure smoothness of robot movements the LSTM generated data are interpolated before being fed to the Panda robot. Pattern one has an interpolation of 50 points and pattern two of 30 points.

Data normalization is performed to ensure equal weights for all inputs. Without normalization, higher values get favored during training thereby skewing the results and rendering

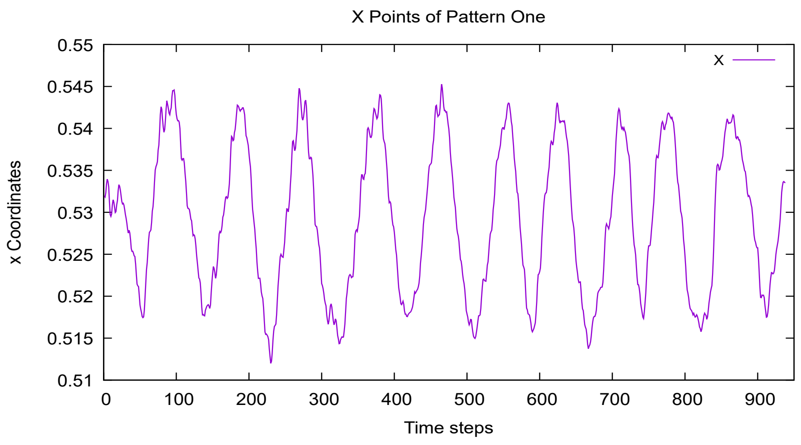


Figure 3.13: Input data of x coordinates of pattern one for LSTM with only one input feature.

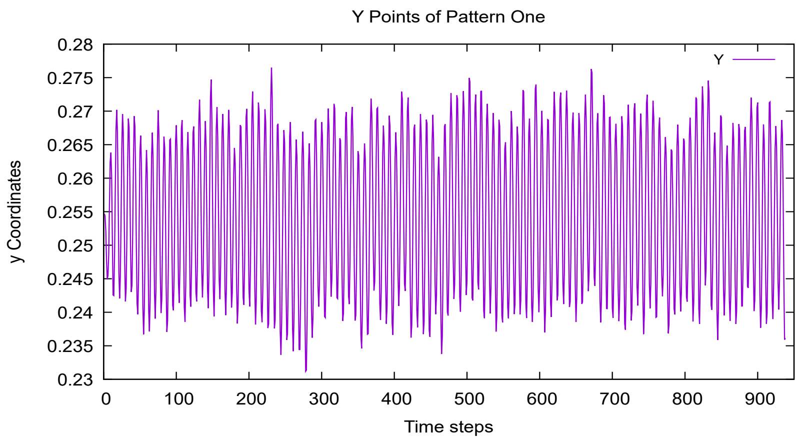


Figure 3.14: Input data of y coordinates of pattern one for LSTM with only one input feature.

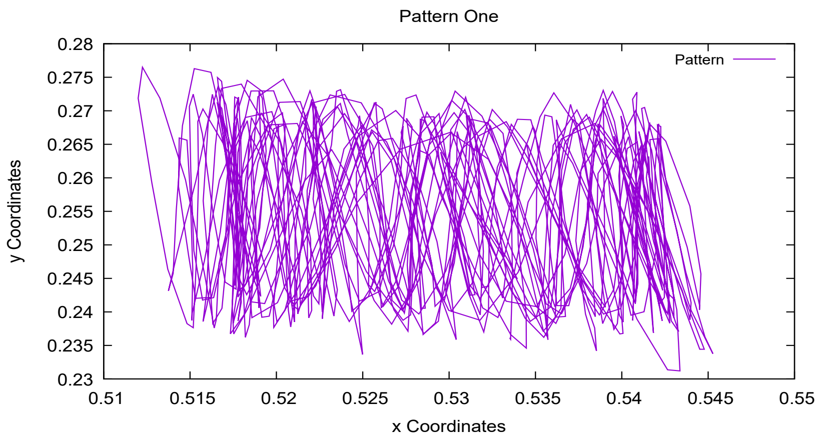


Figure 3.15: Input data of x and y coordinates of pattern one for LSTM with two input features.

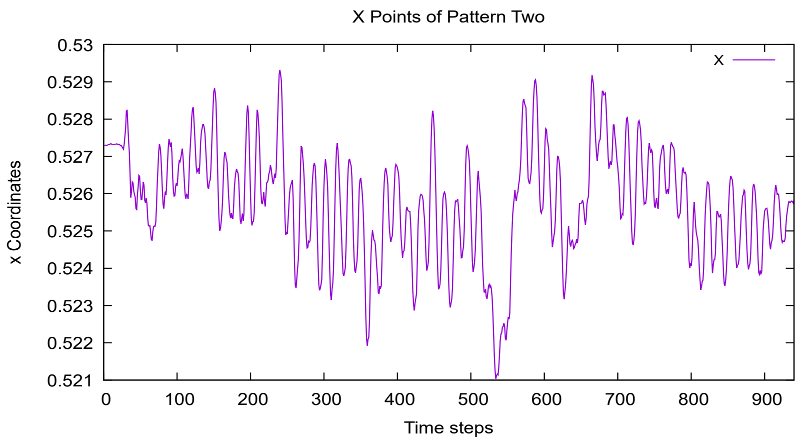


Figure 3.16: Input data of x coordinates of pattern two for LSTM with only one input feature.

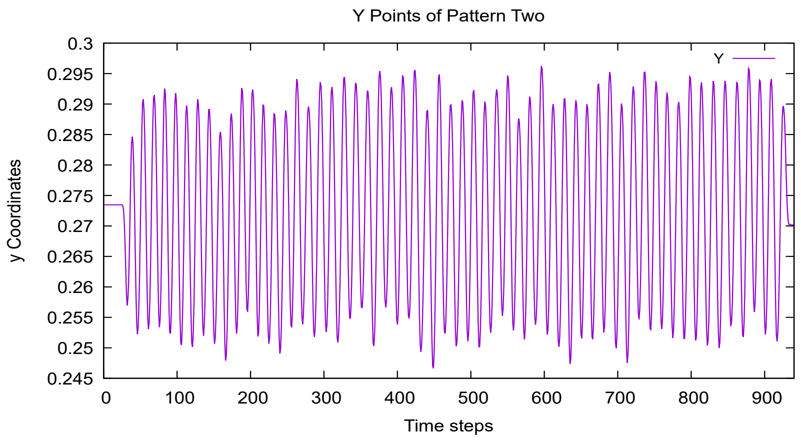


Figure 3.17: Input data of y coordinates of pattern two for LSTM with only one input feature.

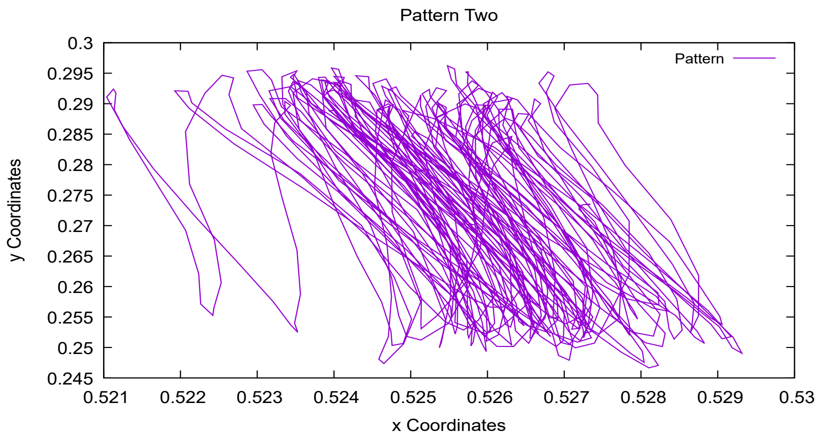


Figure 3.18: Input data of x and y coordinates of pattern two for LSTM with two input features.

the output unreliable [4].

Time series data can be transformed in supervised learning, by using previous and the current time steps to create a input-output set. Given that the order between the observations are preserved [27].

The number of previous time steps used in the input-output set is called the window width or size of the lag. This sliding window is the basic for how any time series dataset can turn into a supervised learning problem [27].

Before the data enters the LSTM one last transformation is required. Since the NN in question receives as input a three dimension structure composed of the following array [Samples, Time Steps, Features] [27].

- Samples: One sequence is one sample. A batch is comprised of one or more samples [27];
- Time Steps: One time step is one point of observation in the sample. One sample is comprised of multiple time steps [27];
- Features: One feature is one observation at a time step. One time step is comprised of one or more features [27].

For the example of a sequence of points such [1 2 3 4 5 6 7 8 9] with a lag window (time step) of 3, a supervised dataset is as follows:

X	y
[1 2 3]	[4]
[2 3 4]	[5]
[3 4 5]	[6]
[4 5 6]	[7]
[5 6 7]	[8]
[6 7 8]	[9]

The previous three time steps are the inputs (X) and the next time step is the output

(y) of the supervised learning problem. There is no previous values that can be used to predict the first three values in the sequence. Also, there is no next values to predict for the last three values in the sequence. In this case, the array [Samples, Time Steps, Features] corresponds to [6, 3, 1].

3.2.3.4 Model

When there is multiple variables measured over time, that is more than one feature as input, it is called multivariate architecture. When only one variable is measured over time, only one feature, it is called univariate. Multivariate data is often more harder to work with and more difficult to model [27]. Considering the existence of two features, coordinates x and y, the problem can be faced with a multivariate LSTM or two univariate LSTM networks, one for each feature, x and y.

When the model only has a single hidden layer followed by an output layer it is said the model has a Vanilla architecture as illustrated in Figure 3.19. When the model has multiple hidden layers stacked one on top of the other it is called a Stacked architecture as shown in Figure 3.20. The less hidden layers the model has the less complex and more fast the calculus is. However, adding hidden LSTM layer can overcome the problem of overfitting. Therefore, a compromise must be made [27]. However, having a more complex model, with several hidden layers or a high number of neurons per layer, can add noise and increase the likelihood of overfitting [34].

In a simple Neural Network with only one hidden layer it may be harder to improve an approximation at one point without making it worse elsewhere given that the interactions between the neurons is global. However, with an architecture of two hidden layers the outcome of the neurons are isolated and the approximations are made in different regions that can be adjusted without effecting each other [35].

A dense layer follows the last LSTM layer to condense its output to the prediction value. Hence, the output dimension of the NN with dense layer is equal to the number of features as input. The dense layer has a linear activation function and is fully connected to the last LSTM layer [36].

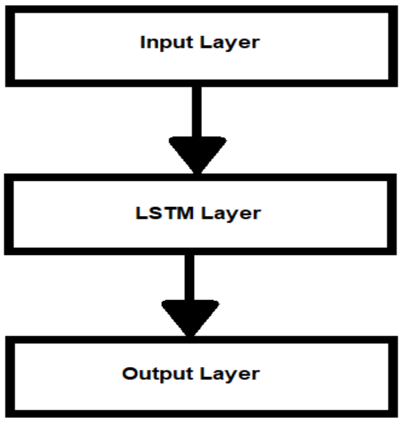


Figure 3.19: Vanilla LSTM diagram.

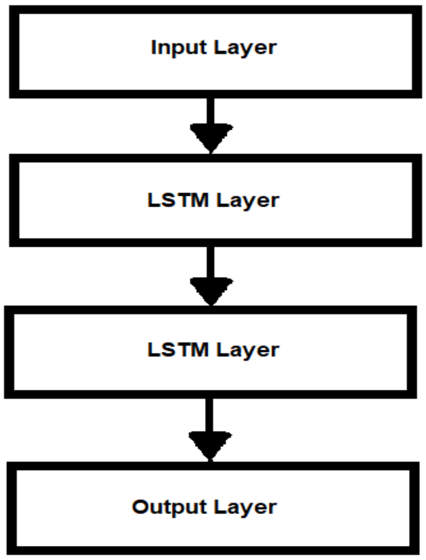


Figure 3.20: Stacked LSTM diagram with two hidden layers.

Activation Function

Activation functions is a hyperparameter that affects the Artificial Neural Network critically, by deciding whether or not a neuron will be activated. Therefore, assisting the network in learning complex patterns existent in the input data and allowing the introduction of non-linearity in the Neural Network [37] [38].

Three common activation function are studied and compered in this work, they are sigmoid function, tanh function and Rectified Linear Unit (ReLu) function.

Sigmoid Function: This function has a range between 0 to 1, given by equation 3.7. This is also called as logistic or squashing function [39].

$$f(x) = \frac{1}{(1+e^{-x})} \tag{3.7}$$

Tanh Function: This function is used in problems of voice recognition and natural language processing with RNN. With scale between -1 to 1 and zero-centered [39].

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.8}$$

ReLu Function: Frequently used in deep learning applications, given by equation 3.9. When compared with Sigmoid and Tanh activation function has generally better performance. The ReLu doesn't compute the divisions and exponentials, thus the computation speed is higher [39].

$$f(x) = max(0,x) \tag{3.9}$$

Loss Function

The Neural Network error is known as loss. The loss function is the procedure used for measuring how distant the NN predictions are of the test dataset [39] [38]. Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are three loss functions chosen as the criteria to evaluate the prediction performance in this study.

$$MSE = \frac{1}{T}\sum_{i=1}^{T}(\hat{x}_t - x_t)^2 \tag{3.10}$$

$$RMSE = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(\hat{x}_t - x_t)^2} \tag{3.11}$$

$$MAE = \frac{1}{T}\sum_{i=1}^{T}|x_t - \hat{x}_t| \tag{3.12}$$

Where T is the number of test set samples, x_t, refers to the real value of the forecasting point, and $\hat{x}_t$, is the corresponding predicted value [40].

Optimizer Algorithm

The goal in learning process is to minimize the loss function and produce better results by adjusting the weights and biases. Optimization algorithms plays an essential part in improving the learning process [37] [41]. The learning process of a Deep Neural Network may be described as an optimization problem whose objective is to find the global optimum by a training trajectory and fast convergence using gradient descent algorithms. Therefore, choosing an inappropriate optimization algorithm can lead the network to reside in a local minima during training, and thus not achieving any advances in the learning process [37].

Two optimizer algorithms studied in this thesis that are recurrently used are Adaptive Moment Estimation (Adam) and Root Mean Square Propagation (RMSprop).

A commonly used optimizer is Stochastic Gradient Descent, already described above in Section 3.2. A more computational faster optimizer when compered to traditional functions is the Root Mean Square Propagation (RMSprop), which uses momentum to find minimums of the loss fucntion. The use of momentum enables previous batches of gradient descent to take the network to a minimum faster [38].

Adam is a commonly used optimizer algorithm that is a combination of RMSprop and Stochastic Gradient Descent with Momentum (SGDM) [42]. A significant advantage of

Adam is the use of exponential moving average of the gradients as momentum term instead of the gradients itself like in SGDM, and uses a parameter update that is similar to RMSprop with an added momentum term [41][42].

Adam is simple to implement, computational efficiency, with small memory requirements and rescales the diagonal of the gradient. Behaving efficiently with large problems in terms of data and/or parameters. The empirical results show that Adam works well in practice and has advantages over other stochastic optimization methods [40]. Thus, making the LSTM model have a accurate prediction higher than other optimizers [40].

4

Results

In this chapter, the results obtained from the implementation of the methodology described in Chapter 3 are presented. With the aim of achieving the best possible generated pattern for polishing a predetermined mold, a number of LSTMs with different hyperparametes and parameters, were developed and investigated. The results are evaluated based on the loss function and the capacity to replicate a similar polishing pattern. Moreover, a description of the mold and the polishing areas chosen for test in both simulation and real life are presented. A demonstration of the Panda robot polishing the mold, when fed the neural network generated pattern, in both simulation and real life environment, is also presented.

4.1 LSTM

4.1.1 Univariate Vanilla Architecture

An LSTM with a Univariate Vanilla architecture is shown in the diagram of Figure 4.1. In this architecture, the NN is implemented with only one LSTM layer and only one input feature. This architecture was trained separately for both x and y points of both recorded patterns, with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer and a train-test split ratio of 90-10%. Tables 4.1, 4.2 and 4.3, illustrate the RMSE, MSE and MAE errors, respectively, for x and y points for different activation functions and optimizers for pattern one. Minimal errors are achieved both x and y coordinates, with tangent as an activation function and RMSprop as the optimizer. And tables 4.4, 4.5 and 4.6, illustrate the RMSE, MSE and MAE errors, respectively, for x and y points for different activation functions and optimizers for pattern two. Minimal errors are achieved both x and y coordinates, with ReLu as an activation function and RMSprop and Adam, respectively, as the optimizer.

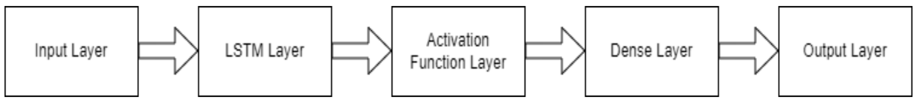


Figure 4.1: Diagram of LSTM with a Vanilla architecture.

activation/optimization	X points	Y points
Sigmoid/Adam	0.1224	0.0828
Sigmoid/RMSProp	0.1365	0.1687
ReLu/Adam	0.0426	0.0699
ReLu/RMSProp	0.0566	0.0746
tanh/Adam	0.0492	0.0675
tanh/RMSProp	0.0410	0.0632

Table 4.1: RMSE errors for Univariate Vanilla LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

activation/optimization	X points	Y points
Sigmoid/Adam	0.0153	0.0071
Sigmoid/RMSProp	0.0186	0.0285
ReLu/Adam	0.0018	0.0051
ReLu/RMSProp	0.0033	0.0057
tanh/Adam	0.0025	0.0049
tanh/RMSProp	0.0017	0.0042

Table 4.2: MSE errors for Univariate Vanilla LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

Figure 4.2 upper image and 4.3 upper image illustrate the open loop forecast, that is the predictions of the next time step using only the test dataset, of x and y points, respectively, for the best achieved results with univariate vanilla architecture, for pattern one as input. In both figures the blue line represents the test dataset and the red line is the forecast output. Moreover, in Figure 4.2 bottom image and 4.3 bottom image the closed loop forecast is shown. Specifically, these figures show the prediction of the subsequent time step by using previous prediction as input. The figures show the best achieved results with this architecture for both x and y points of pattern one, respectively. The blue line represents

activation/optimization	X points	Y points
Sigmoid/Adam	0.0793	0.0483
Sigmoid/RMSProp	0.1289	0.1557
ReLu/Adam	0.0269	0.0423
ReLu/RMSProp	0.0435	0.0492
tanh/Adam	0.0335	0.0346
tanh/RMSProp	0.0301	0.0359

Table 4.3: MAE errors for Univariate Vanilla LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

activation/optimization	X points	Y points
Sigmoid/Adam	0.1049	0.0719
Sigmoid/RMSProp	0.1959	0.1341
ReLu/Adam	0.0648	0.0361
ReLu/RMSProp	0.0641	0.0481
tanh/Adam	0.0679	0.0387
tanh/RMSProp	0.0771	0.0444

Table 4.4: RMSE errors for Univariate Vanilla LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

activation/optimization	X points	Y points
Sigmoid/Adam	0.0111	0.0053
Sigmoid/RMSProp	0.0385	0.0180
ReLu/Adam	0.0043	0.0015
ReLu/RMSProp	0.0041	0.0024
tanh/Adam	0.0047	0.0016
tanh/RMSProp	0.0059	0.0019

Table 4.5: MSE errors for Univariate Vanilla LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations for pattern one may be found in Appendix A, in Figures 6.1 through 6.10.

While figure 4.4 upper image and 4.5 upper image illustrate the open loop forecast of x and y points, respectively, for the best achieved results with univariate vanilla architecture, for pattern two as input. In both figures the blue line represents the test dataset and the red line is the forecast output. Moreover, in Figure 4.4 bottom image and 4.5 bottom image

activation/optimization	X points	Y points
Sigmoid/Adam	0.0639	0.0286
Sigmoid/RMSProp	0.1759	0.1241
ReLu/Adam	0.0412	0.0181
ReLu/RMSProp	0.0474	0.0322
tanh/Adam	0.0439	0.0201
tanh/RMSProp	0.0541	0.0304

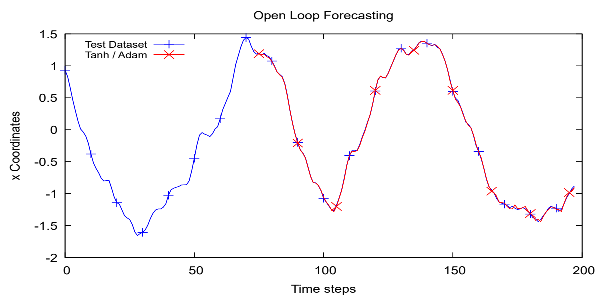
Table 4.6: MAE errors for Univariate Vanilla LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

the close loop forecast is shown. The figures show the best achieved results with this architecture for both x and y points of pattern two, respectively. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations for pattern two may be found in Appendix B, in Figures 7.1 through 7.10.

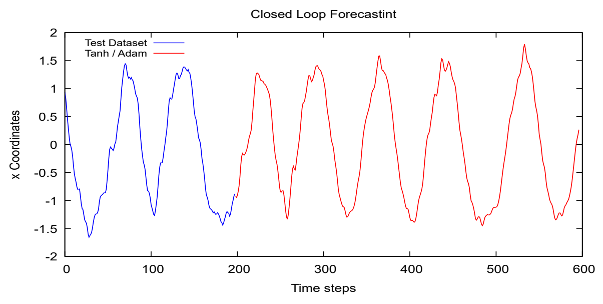
Different values of time steps were study in this work. For values under 200 time steps, the LSTM was unable to retain key characteristics of the train dataset and therefore couldn't reproduce a suitable polishing pattern. Higher values of time steps increased the NN complexity, without a corresponding result improvement. Also, higher values for epochs and the number of neurons in the hidden layer increase the complexity of the NN and thus requiring more training time.

4.1.2 Univariate Stacked Architecture

An LSTM with a Univariate Stacked architecture is show in Figure 4.6 diagram. This NN has one LSTM layer and only one feature for input. This architecture was trained separately for both x and y points of both recorded patterns, with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both hidden layers and a train-test split ratio of 90-10%. In Tables 4.7, 4.8 and 4.9, the RMSE, MSE and MAE errors are presented, respectively, for x and y points of pattern one for different activation functions and optimizers. The best result for x coordinates are achieved with tangent as the activation function and RMSprop as the optimizer. While for y coordinates the best results are obtained with the tangent activation function and the Adam optimizer. Tables 4.10, 4.11 and 4.12 illustrate the RMSE, MSE



(a)



(b)

Figure 4.2: Univariate Vanilla LSTM output for x points of pattern one with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.

and MAE errors, respectively, for x and y points of pattern two for different activation functions and optimizers. Minimal errors are achieved for both x and y coordinates with ReLu as an activation function and RMSprop and Adam, respectively, as the optimizer.

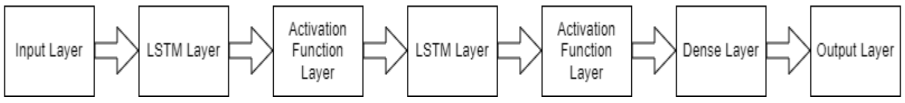
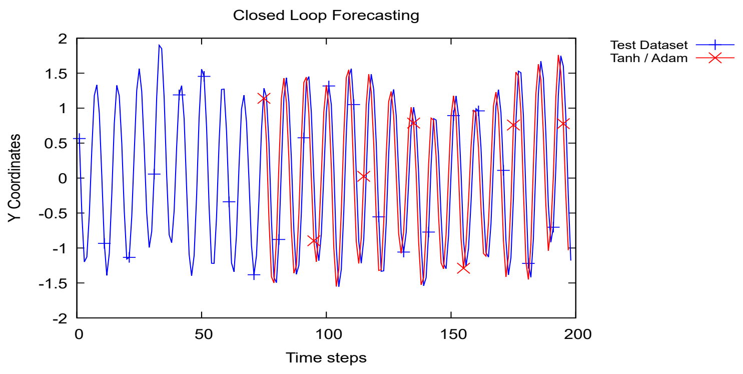


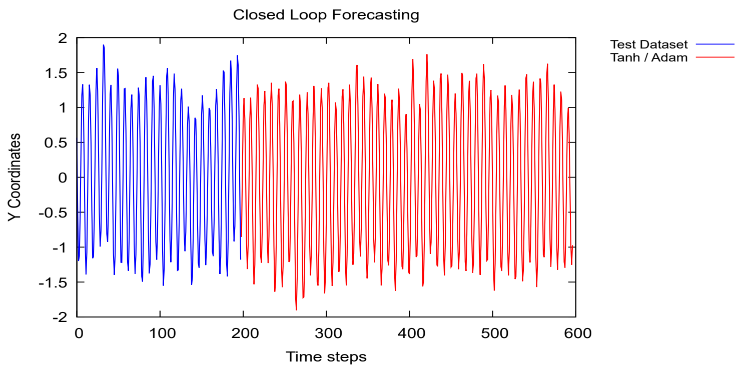
Figure 4.6: Diagram of LSTM with a Stacked architecture, with two hidden LSTM layers.

activation/optimization	X points	Y points
Sigmoid/Adam	0.0746	0.0746
Sigmoid/RMSProp	0.1967	0.1748
ReLu/Adam	0.0575	0.0699
ReLu/RMSProp	0.0659	0.0811
tanh/Adam	0.0565	0.0651
tanh/RMSProp	0.0552	0.0699

Table 4.7: RMSE errors for Univariate Stacked LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both the hidden layers, a train-test split ratio of 90-10%, for different activation functions and optimizer.

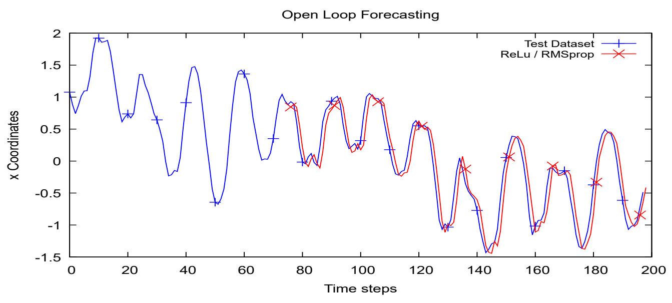


(a)

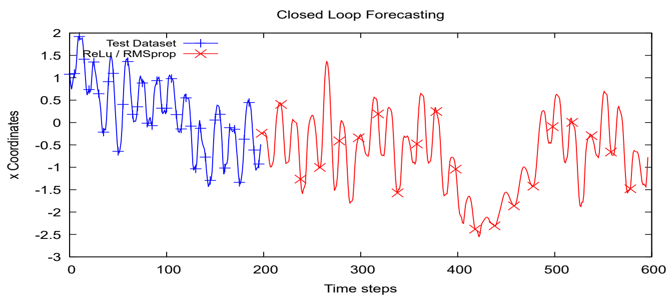


(b)

Figure 4.3: Univariate Vanilla LSTM output for y points of pattern one with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.

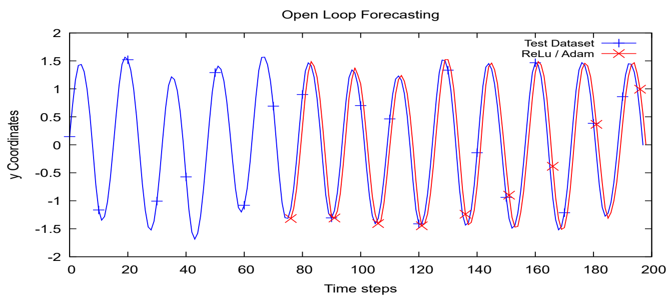


(a)

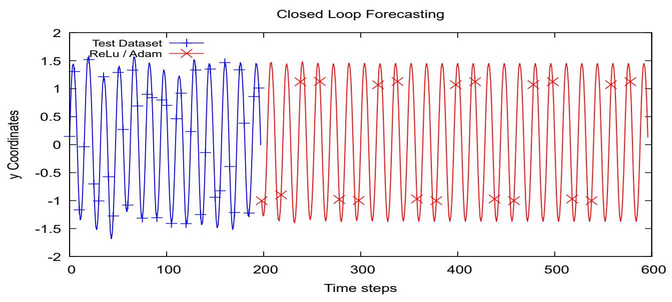


(b)

Figure 4.4: Univariate Vanilla LSTM output for x points of pattern two with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.



(a)



(b)

Figure 4.5: Univariate Vanilla LSTM output for y points of pattern two with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.

activation/optimization	X points	Y points
Sigmoid/Adam	0.0058	0.0090
Sigmoid/RMSProp	0.0387	0.03081
ReLu/Adam	0.0033	0.0051
ReLu/RMSProp	0.0044	0.0067
tanh/Adam	0.0033	0.0045
tanh/RMSProp	0.0031	0.0051

Table 4.8: MSE errors for Univariate Stacked LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both the hidden layers, a train-test split ratio of 90-10%, for different activation functions and optimizer.

activation/optimization	X points	Y points
Sigmoid/Adam	0.0463	0.0630
Sigmoid/RMSProp	0.1725	0.1420
ReLu/Adam	0.0388	0.0419
ReLu/RMSProp	0.0527	0.0566
tanh/Adam	0.0381	0.0351
tanh/RMSProp	0.0416	0.0431

Table 4.9: MAE errors for Univariate Stacked LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both the hidden layers, a train-test split ratio of 90-10%, for different activation functions and optimizer.

activation/optimization	X points	Y points
Sigmoid/Adam	0.0975	0.0719
Sigmoid/RMSProp	0.2261	0.0921
ReLu/Adam	0.0645	0.0346
ReLu/RMSProp	0.0609	0.0612
tanh/Adam	0.0747	0.0429
tanh/RMSProp	0.0748	0.0439

Table 4.10: RMSE errors for Univariate Stacked LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both the hidden layers, a train-test split ratio of 90-10%, for different activation functions and optimizer.

Figures 4.7 upper image and 4.8 upper image illustrate the open loop forecast of x and y points, respectively, of pattern one for the best achieved results for a Stacked LSTM. In both figures, the blue line represents the test dataset and the red line is the forecast output. Moreover, in Figure 4.7 bottom image and 4.8 bottom image, the close loop forecast is shown for the best achieved results with this architecture for both x and y points, respectively, of pattern one. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer

activation/optimization	X points	Y points
Sigmoid/Adam	0.0096	0.0031
Sigmoid/RMSProp	0.0530	0.0086
ReLu/Adam	0.0042	0.0013
ReLu/RMSProp	0.0037	0.0037
tanh/Adam	0.0057	0.0019
tanh/RMSProp	0.0057	0.0020

Table 4.11: MSE errors for Univariate Stacked LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both the hidden layers, a train-test split ratio of 90-10%, for different activation functions and optimizer.

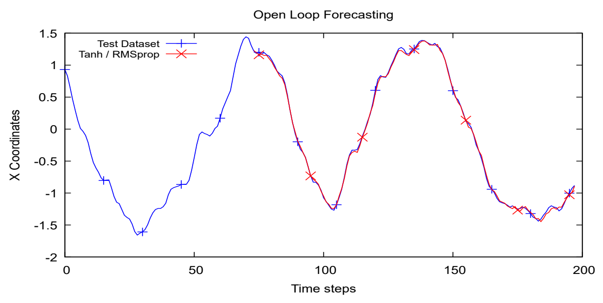
activation/optimization	X points	Y points
Sigmoid/Adam	0.0577	0.0280
Sigmoid/RMSProp	0.2006	0.0700
ReLu/Adam	0.0360	0.0178
ReLu/RMSProp	0.0427	0.0514
tanh/Adam	0.0515	0.0242
tanh/RMSProp	0.0536	0.0278

Table 4.12: MAE errors for Univariate Stacked LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 512 neurons in both the hidden layers, a train-test split ratio of 90-10%, for different activation functions and optimizer.

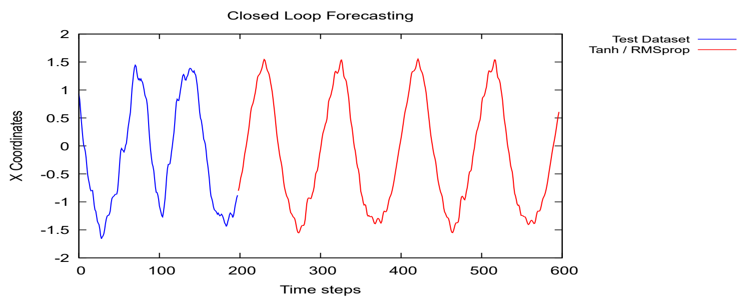
combinations may be found in Appendix A, in Figures 6.11 through 6.20.

Figures 4.9 upper image and 4.10 upper image illustrate the open loop forecast of x and y points, respectively, of pattern two for the best achieved results for a stacked LSTM. In both figures, the blue line represents the test dataset and the red line is the forecast output. Moreover, in Figure 4.4 bottom image and 4.10 bottom image the close loop forecast is shown for the best achieved results with this architecture for both x and y points, respectively, of pattern two. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations may be found in Appendix B, in Figures 7.11 through 7.20.

Increasing the number of hidden layers would create a higher complexity of the NN, thus, being more time consuming during the training process.

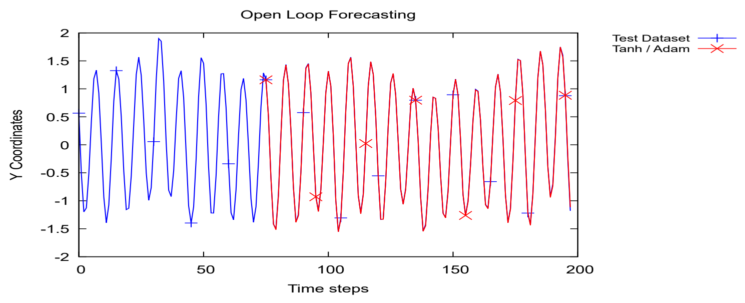


(a)

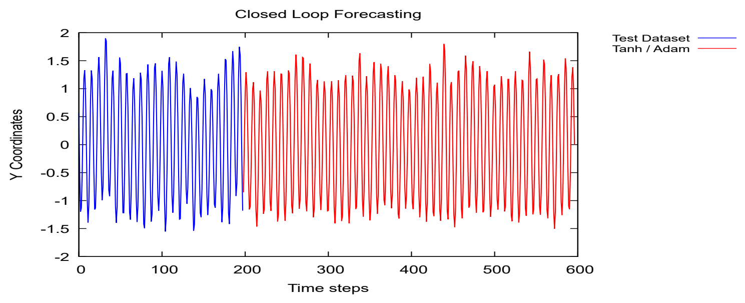


(b)

Figure 4.7: Univariate Stacked LSTM output for x points of pattern two with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.

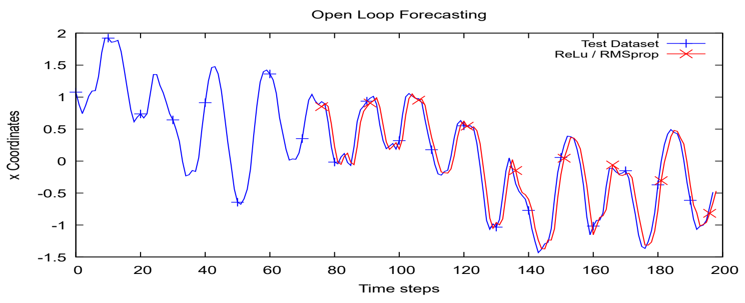


(a)

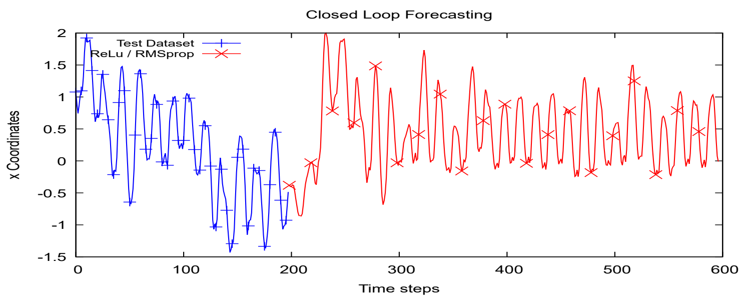


(b)

Figure 4.8: Univariate Stacked LSTM output for y points of pattern two with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.

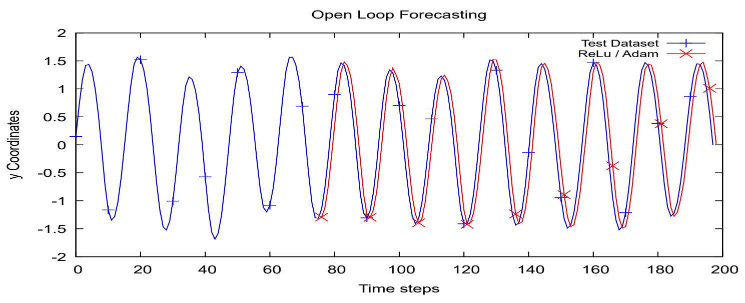


(a)

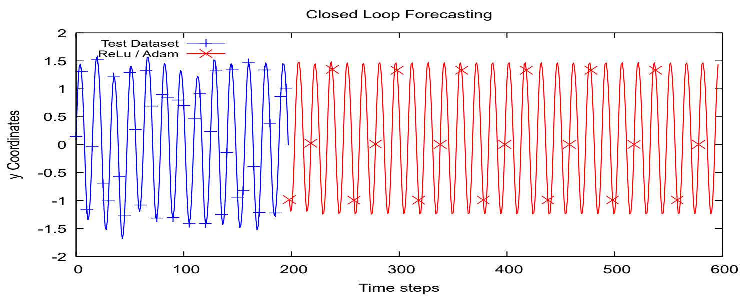


(b)

Figure 4.9: Univariate Stacked LSTM output for x points of pattern two with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.



(a)



(b)

Figure 4.10: Univariate Stacked LSTM output for y points of pattern two with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue line is the input of the trained NN and the red line is the output.

4.1.3 Multivariate Vanilla Architecture

An LSTM with a Multivariate Vanilla architecture is show in the diagram of Figure 4.1. This NN is implemented with only one LSTM layer and two features for input. This architecture was trained with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer and a train-test split ratio of 90-10%. In Table 4.13 are shown the RMSE, MSE and MAE errors for x and y points of pattern one for different activation functions and optimizes. The best results for both x and y coordinates of pattern one, thus showing the minimal error, where achieved with ReLu as an activation function and Adam as the optimizer.

activation/optimization	RMSE	MSE	MAE
Sigmoid/Adam	0.0805	0.0066	0.0454
Sigmoid/RMSProp	0.2536	0.0782	0.2596
ReLu/Adam	0.0496	0.0026	0.0242
ReLu/RMSProp	0.0617	0.0039	0.0419
tanh/Adam	0.0574	0.0034	0.0333
tanh/RMSProp	0.0570	0.0033	0.0376

Table 4.13: RMSE, MSE and MAE errors for Multivariate Vanilla LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

Table 4.14 illustrates the RMSE, MSE and MAE errors for x and y points of pattern two for different activation functions and optimizes. The minimal error obtain for x and y coordinates of pattern two where achieved with ReLu as an activation function and Adam as the optimizer.

Figure 4.11 illustrates the open loop forecast for the best achieved results with Multivariate Vanilla LSTM for points of patter one as input. This figure shows a blue line that represents the test dataset and the dashedt red line that represents the forecast output. Moreover, in Figure 4.12 the close loop forecast is shown. The figure shows the best achieved results with this architecture. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations may be found in Appendix A, in Figures 6.21 through 6.25.

activation/optimization	RMSE	MSE	MAE
Sigmoid/Adam	0.0854	0.0074	0.0489
Sigmoid/RMSProp	0.3019	0.0912	0.2807
ReLu/Adam	0.0488	0.0025	0.0254
ReLu/RMSProp	0.0612	0.0038	0.0455
tanh/Adam	0.0540	0.0030	0.0314
tanh/RMSProp	0.0539	0.0029	0.0351

Table 4.14: RMSE, MSE and MAE errors for Multivariate Vanilla LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

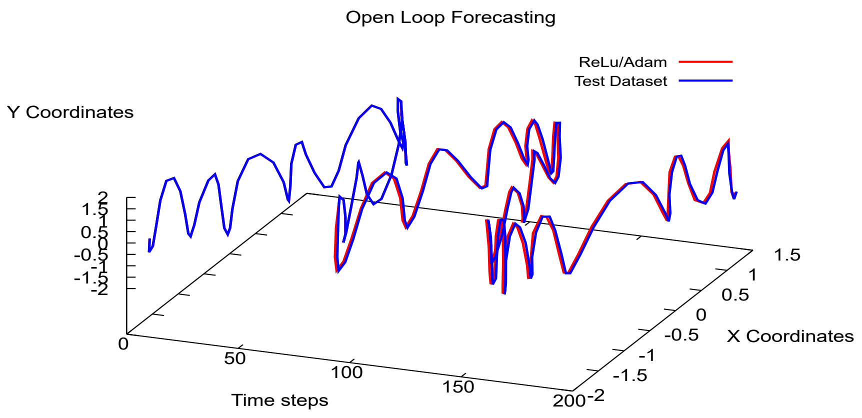


Figure 4.11: Multivariate Vanilla LSTM open loop output for x and y points of pattern one with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue solid line is the input of the trained NN and the red line is the output.

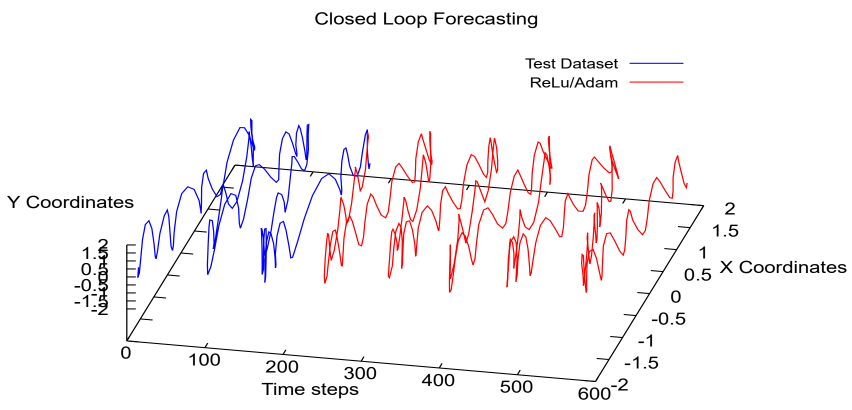


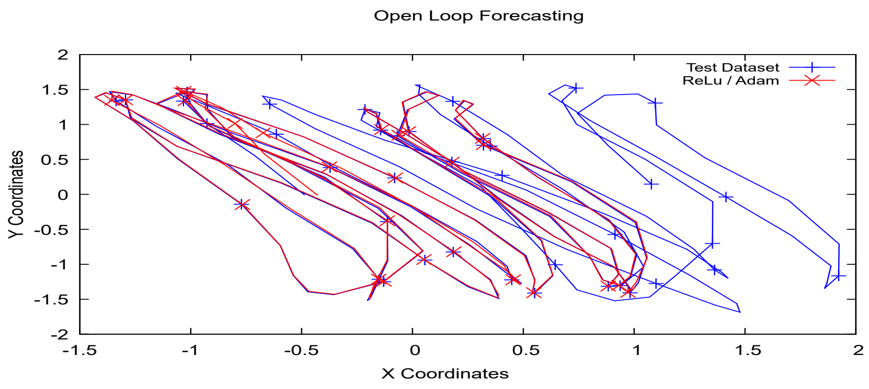
Figure 4.12: Multivariate Vanilla LSTM close loop output for x and y points of pattern one with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue solid line is the input of the trained NN and the red dashed line is the output.

Figure 4.13 upper image illustrates the open loop forecast for the best achieved results with Multivariate Vanilla LSTM for points of patter two as input. This figure shows a blue line that represents the test dataset and the red line represents the forecast output. Moreover, in Figure 4.13 bottom image the close loop forecast is shown. The figure shows the best achieved results with this architecture. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations may be found in Appendix B, in Figures 7.21 through 7.25.

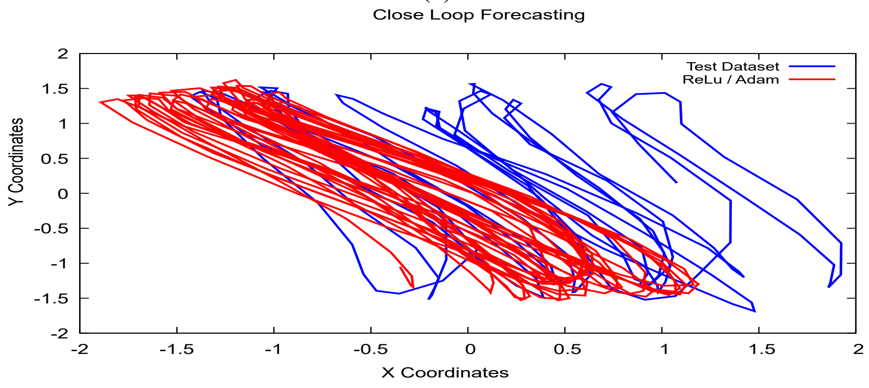
Increasing the number of neurons in the hidden layer would create a higher complexity of the NN, thus, being more time consuming during the training process.

4.1.4 Multivariate Stacked Architecture

An LSTM with a Multivariate Stacked architecture is show in the diagram of Figure 4.6. In this case the NN has two LSTM layers and two features for input. This architecture was trained with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer and a train-test split ratio of 90-10%. In Table 4.15 illustrated the RMSE, MSE and MAE errors for x and y points of pattern one for different activation functions and optimizers. The minimal errors for both x and y coordinates, are achieved with ReLu as an activation function and Adam as the optimizer.



(a)



(b)

Figure 4.13: Multivariate Vanilla LSTM open loop output for x and y points of pattern two with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. The blue solid line is the input of the trained NN and the red line is the output.

activation/optimization	RMSE	MSE	MAE
Sigmoid/Adam	0.1094	0.0123	0.0719
Sigmoid/RMSProp	0.3055	0.0936	0.2555
ReLu/Adam	0.0458	0.0022	0.0221
ReLu/RMSProp	0.0747	0.0056	0.0556
tanh/Adam	0.0533	0.0029	0.0319
tanh/RMSProp	0.0604	0.0037	0.0374

Table 4.15: RMSE, MSE and MAE errors for Multivariate Stacked LSTM for x and y points of pattern one trained with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

While Table 4.16 illustrated the RMSE, MSE and MAE errors for x and y points of pattern two for different activation functions and optimizers. The minimal errors for both x and y coordinates, are achieved with ReLu as an activation function and Adam as the optimizer.

activation/optimization	RMSE	MSE	MAE
Sigmoid/Adam	0.1135	0.0131	0.0766
Sigmoid/RMSProp	0.3292	0.1097	0.2767
ReLu/Adam	0.0482	0.0024	0.0251
ReLu/RMSProp	0.0847	0.0072	0.0669
tanh/Adam	0.0561	0.0033	0.0359
tanh/RMSProp	0.0644	0.0042	0.0461

Table 4.16: RMSE, MSE and MAE errors for Multivariate Stacked LSTM for x and y points of pattern two trained with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, for different activation functions and optimizer.

Figure 4.14 illustrates the open loop forecast for the best achieved results with Multivariate Stacked LSTM for points of pattern one as input. This figure shows a blue line that represents the test dataset and the red line that represents the forecast output. Moreover, in Figure 4.15 the close loop forecast is shown. The figure shows the best achieved results with this architecture. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations may be found in Appendix A, in Figures 6.26 through 6.29.

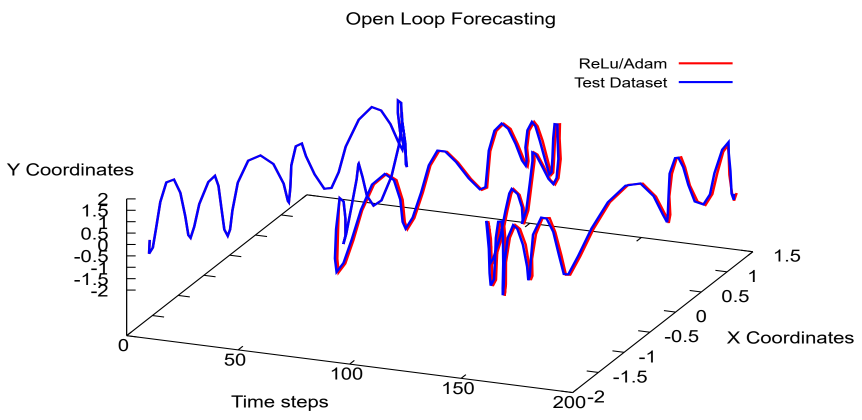


Figure 4.14: Multivariate Stacked LSTM open loop output for x and y points of pattern one with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue solid line is the input of the trained NN and the red line is the output.

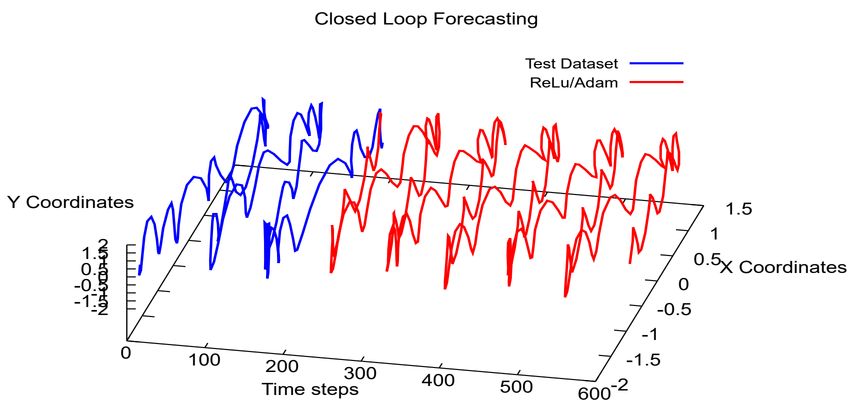
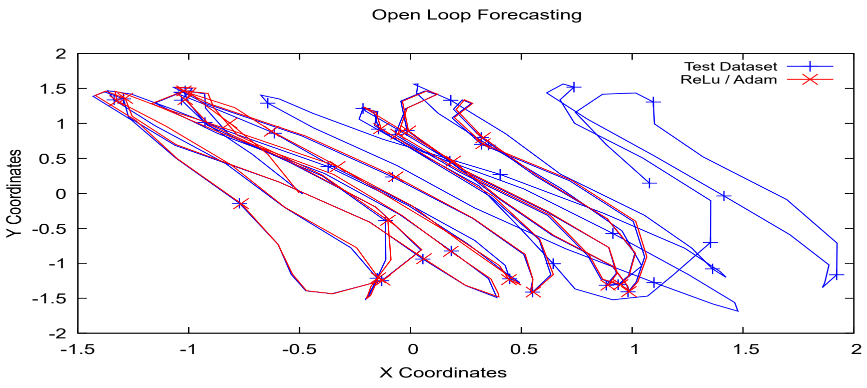
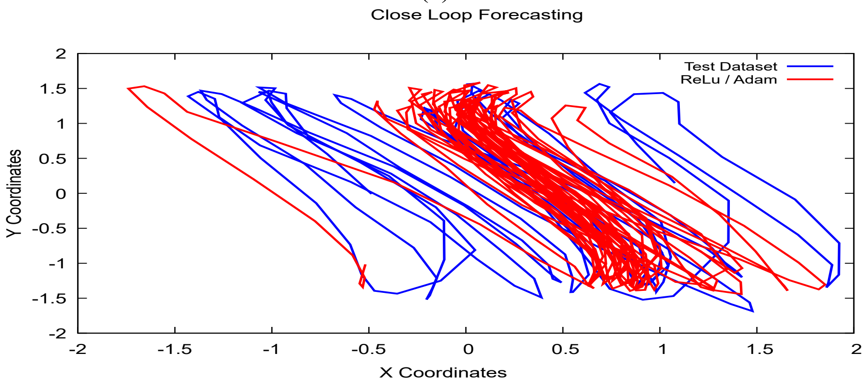


Figure 4.15: Multivariate Stacked LSTM close loop output for x and y points of pattern one with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue solid line is the input of the trained NN and the red line is the output.

Figure 4.16 upper image illustrates the open loop forecast for the best achieved results with Multivariate Stacked LSTM for points of pattern two as input. This figure shows a blue line that represents the test dataset and the red line that represents the forecast output.Moreover, in Figure 4.16 bottom image the close loop forecast is shown. The figure shows the best achieved results with this architecture. The blue line represents the test dataset and the red line represents the forecast pattern. The results for the remaining activation function and optimizer combinations may be found in Appendix B, in Figures 7.26



(a)



(b)

Figure 4.16: Multivariate Stacked LSTM open loop output for x and y points of pattern two with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Image (a) refers to the open loop output while (b) to closed loop output. Where the blue solid line is the input of the trained NN and the red line is the output.

through 7.28.

4.2 Findings and Result Discussion

The obtained results suggest that Sigmoid activation functions present poor performance when compared to ReLu and tangent ones. Poor performance is specially identifiable when Sigmoid functions are combined with RMSprop optimizers.

For univariate architecture scenarios that only use one input feature, the x coordinate presents better behaviour with RMSprop optimizers while the y coordinate performs better with Adam optimizers.

Moreover, ReLu and Adam function combinations show best performance with a multivariate architecture.

Better quantitative results were obtained (i.e., lower errors) with only one hidden layer for multivariate architectures. Nevertheless, stacked architectures (i.e., multiple hidden layers) provided good qualitative results as shown in Figures 4.12 and 4.15 for pattern one and Figures 4.13 (b) and 4.16 (b) for pattern two.

4.3 Demonstration Mold Used for Testing

For simulation and testing purposes a mold was used, as illustrated in Figure 4.17. The mold has a combination of flat and curved surfaces. Also gradient slope surfaces form the geometry of the mold where the developed algorithm was tested. Figure 4.18 shows the mold CAD model that allows for visual representation on RVIZ.

4.4 Simulation of the Polishing Process

MatLab was used to chose the areas to test the performance of the LSTM and retrieve the square to the surface. Figure 4.19 shows a chosen areas for test purposes.

The perpendicular of the chosen areas are retrieved because the robot end-effector orientation during the polishing process should be square to the mold, illustrated by the blue

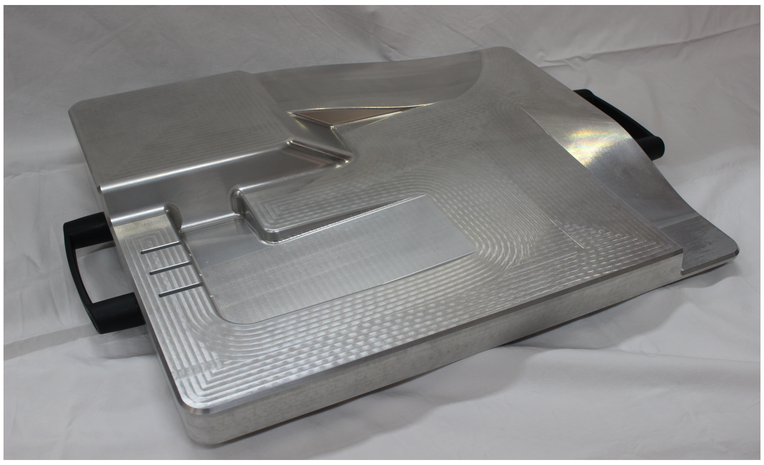


Figure 4.17: Demonstration mold used for testing and simulation purposes.

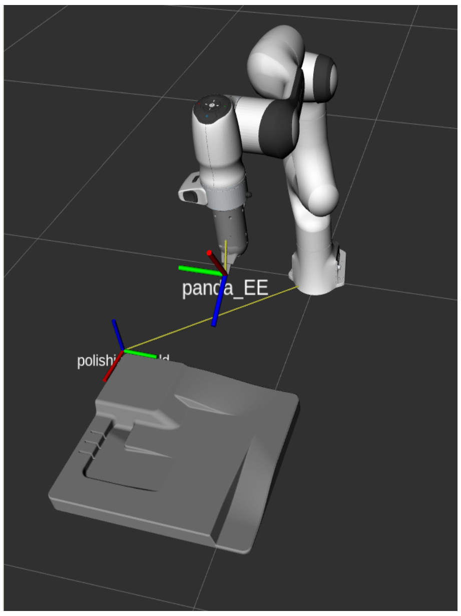


Figure 4.18: RVIZ simulation of the mold and robot Panda.

vectors in Figure 4.19. The desired orientation of the end-effector during the polish process is given by the following equation

$$R_d = R_m R_t \tag{4.1}$$

where (R_m) is the mold rotation and (R_t) is the polishing pattern rotation. (R_t) is chosen depending on mold form.

The plane is defined by four points from the mold. From the given four points, only three

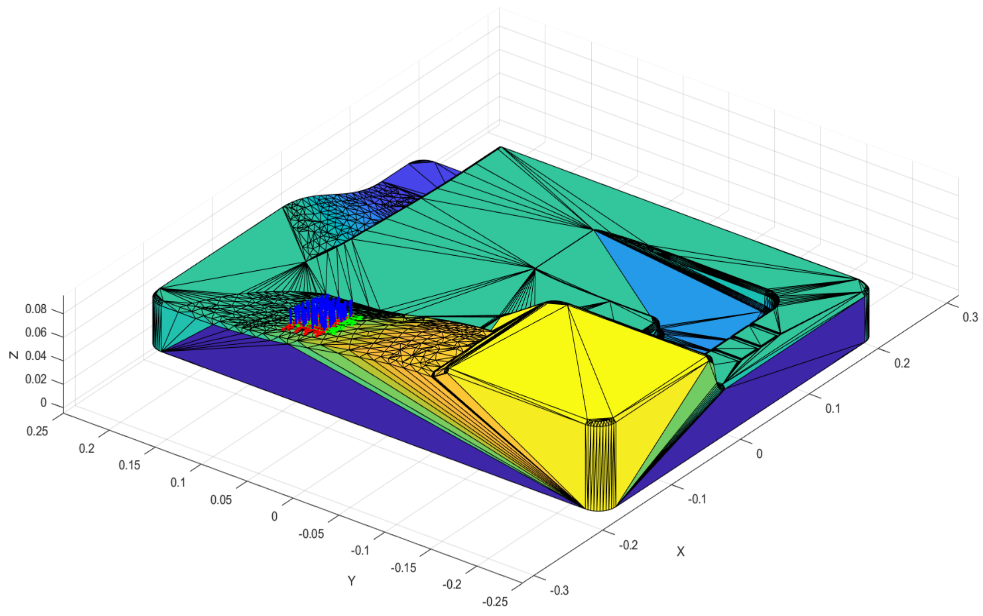


Figure 4.19: MatLab CAD visualization of the mold. The perpendicular inside the chosen area are represented by the blue vector.

are selected to compute the normal of the plane to be polished.

Figure 4.20 shows the RVIZ simulation of the Panda robot polishing pattern generated by the LSTM.

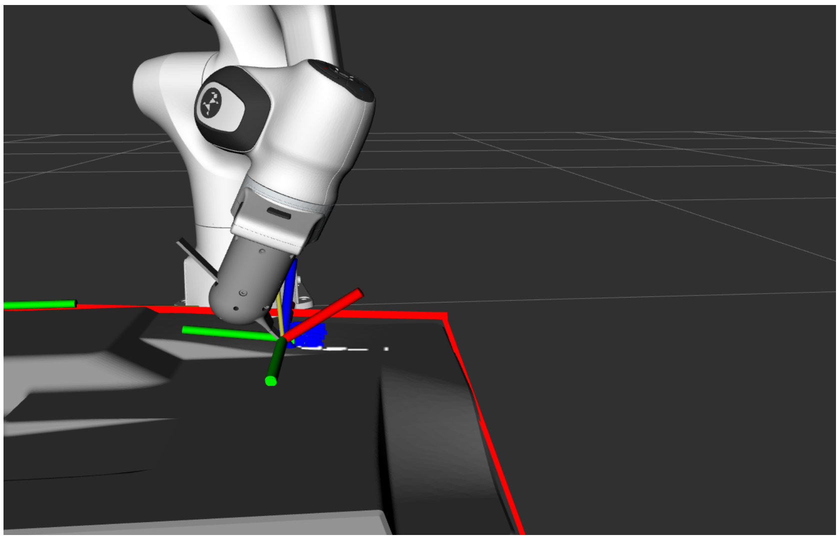


Figure 4.20: RVIZ simulation of Panda robot performing the polishing movements generated by the LSTM.

4.5 Real Life Polishing

Figure 4.21 illustrates a polishing task carried out by the robot Panda on the demonstration mold, using pattern one.

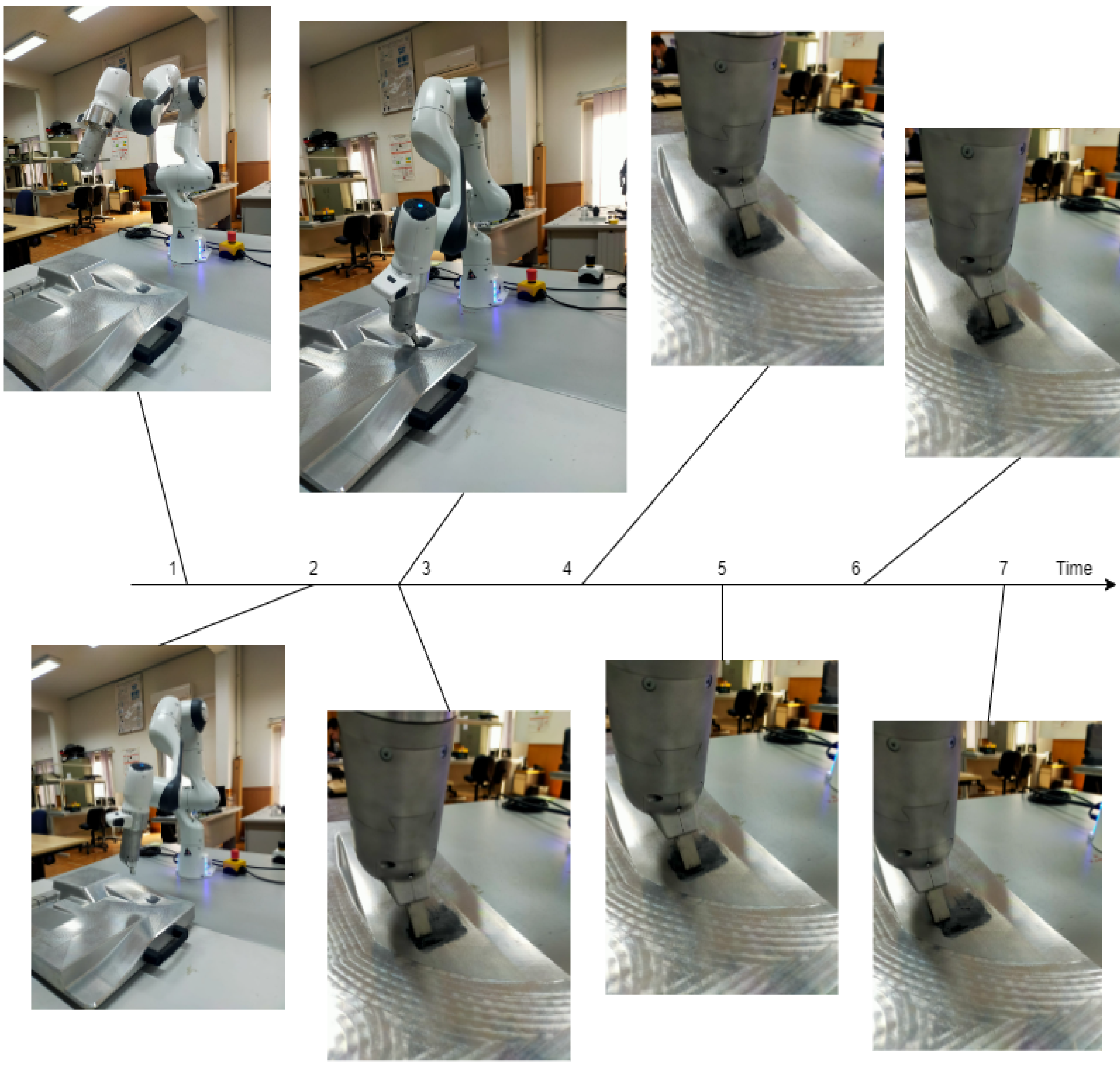


Figure 4.21: Visual representation of Panda robot performing polishing task. From time step 1 to 2 the robot moves to a point in the mold to start the polishing process. In time step 3 to 7 the robot performs the polishing patter generated by the LSTM.

The movement drawn by the robotic arm in the mold is represented in Figure 4.22, where it´s visible the polishing pattern generated by the LSTM in Figure 4.12.

During the polishing process the wear of the stone provokes a decrease in the applied forced by the robot and, therefore, a decrease quality of the polishing. To counter this effect the stiffness of the robot may be manually increased during the polishing process

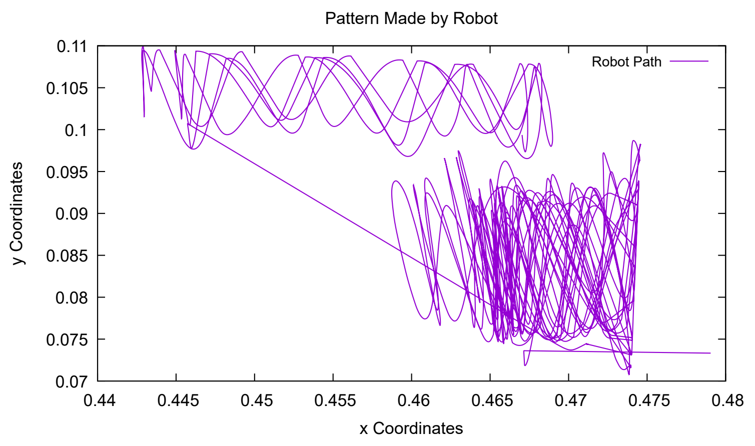


Figure 4.22: Robot movements performed in the mold during the process of polishing a patter generated by LSTM multivariate vanilla architecture with ReLu activation fucntion and Adam optimizer of pattern one.

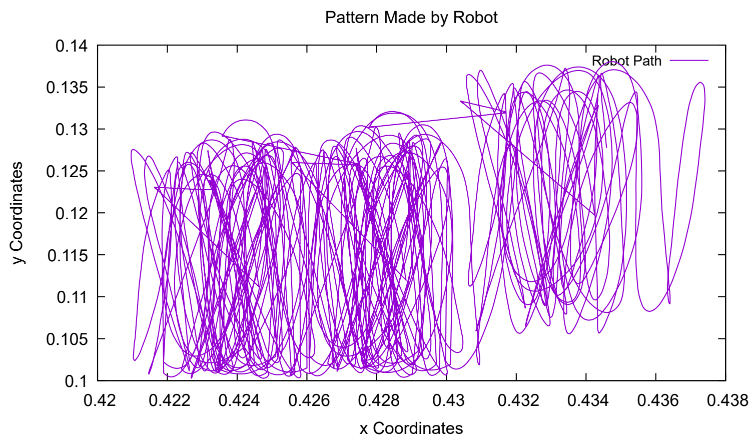


Figure 4.23: Robot movements performed in the mold during the process of polishing a patter generated by LSTM multivariate vanilla architecture with ReLu activation fucntion and Adam optimizer of pattern two.

by increasing the values of Kp in the z axes.

5

Conclusion and Future Work

The main purpose of this thesis is to further develop an autonomous polishing system for plastic molds. This work develops a nn architecture to address the polishing process of steel molds, which are commonly used in the growing plastic industry. Consumer satisfaction demands that plastic objects are rapidly produced in large quantities, without noticeable flaws. Within this context, high quality mold production plays an important role to guaranty objects with high standards. Polishing process are critical in mold fabrication enabling to achieve high quality plastic objects.

In this work, Long Short Term Memory (LSTM) Neural Network were used to generate several polishing patterns. The data for Deep Neural Network training was obtained by recording specialized polishing human operator motions. A number of LSTMs were studied and analysed, including different hyperparameter and architecture designs to obtain desired results. The influence of the number of time steps, neurons, hidden layers and epochs in NN behaviour have been studied. Additionally, the relevance of NN activation functions and optimizers also have been investigated.

From the obtained results, several relevant NN characteristics have been identified. The more neurons the network has - that is the more complex it is - the more time it requires to complete training. Therefore, the number of neurons per layer and the number of hidden layers were chosen making a compromise between satisfactory results and small training times. Multivariate architecture networks took considerable more time to train than univariate ones, as expected since the multivariate architecture has the double of input parameters. The hyperparameter time step value has a key impact on network output. When the time steps are under 200, the Neural Networks are not able to reproduce well desired polishing patterns. Higher time step values increase network complexity without result improvement. The sigmoid activation function had in general bad performance when compared with tangent and ReLu functions, especially when combined with the RMSprop optimizer. For univariate architecture networks the coordinate x has better results with RMSprop optimizer while the y coordinate performed better with Adam. The ReLu activation function combined with Adam optimizer presents the best results for both studied patterns with a multivariate architecture, i.e., with two input features. Moreover, for multivariate architecture with only one hidden layer were observed better quantitative results. Nevertheless, multivariate stacked architectures provided better qualitative results.

The best NN results were fed into a Franka Emika Panda robot arm. A polishing task has been carried out in to demonstration mold polishing. The robot has an impedance control architecture to replicate human arm movements.

5.1 Future Work

The currently implemented process doesn't take into account polishing tool's wear. An autonomous mechanism must therefore be implemented as future work to take into account tool's wear. By increasing the stiffness of the robot and by changing the z coordinates of the end-effector is one solution to tackle this problem.

6

Appendix A Complementary Results

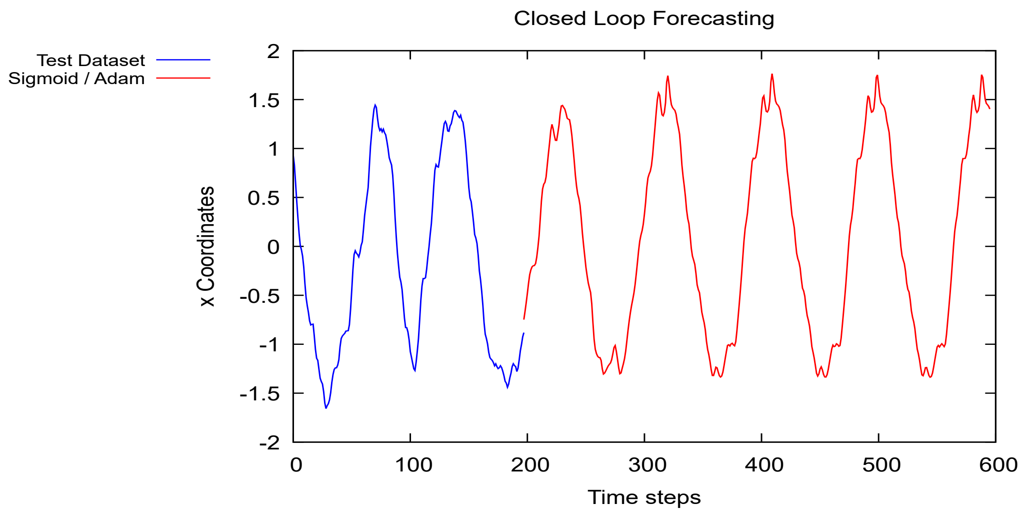


Figure 6.1: Univariate Vanilla LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

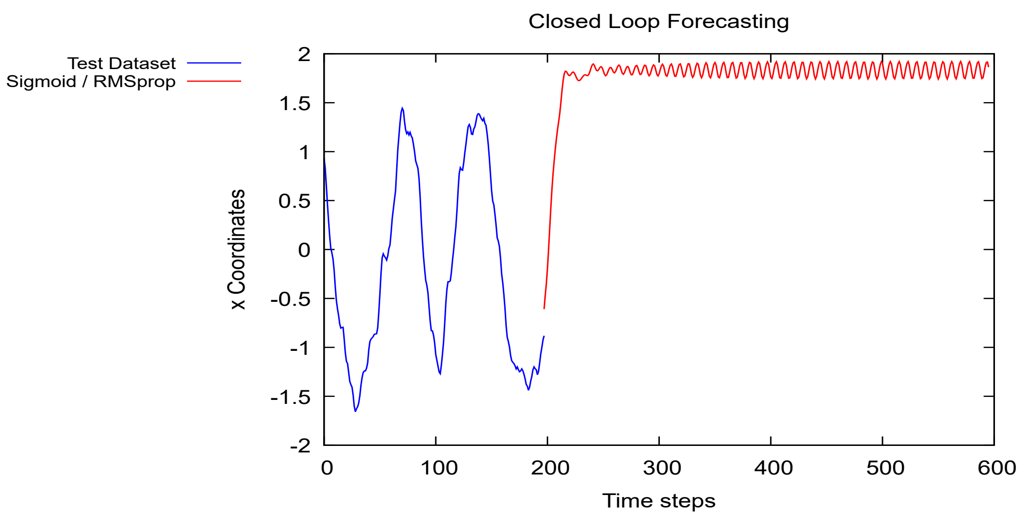


Figure 6.2: Univariate Vanilla LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

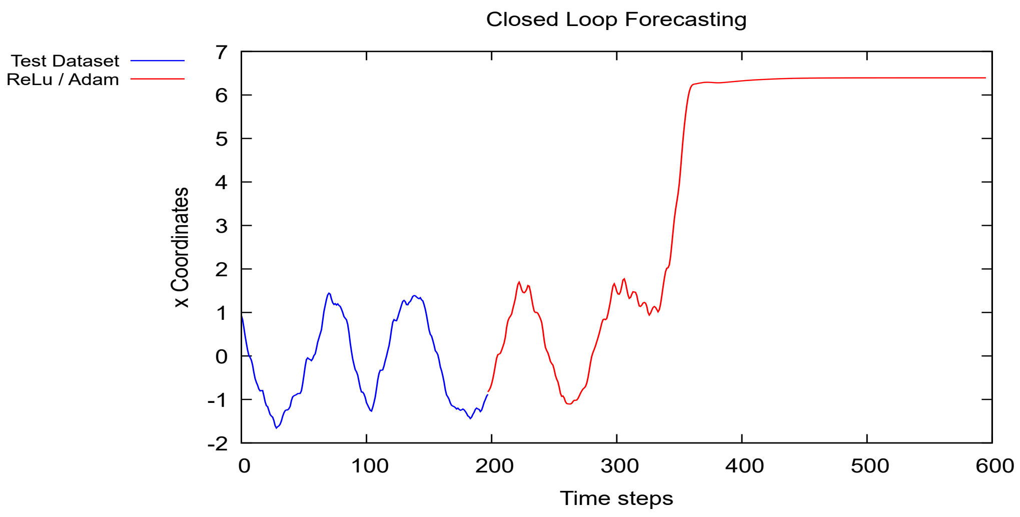


Figure 6.3: Univariate Vanilla LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

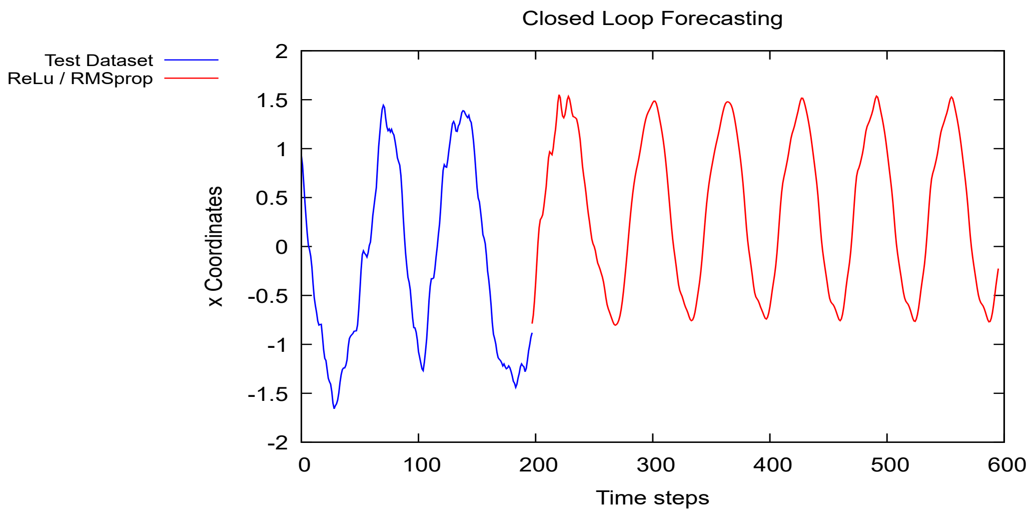


Figure 6.4: Univariate Vanilla LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

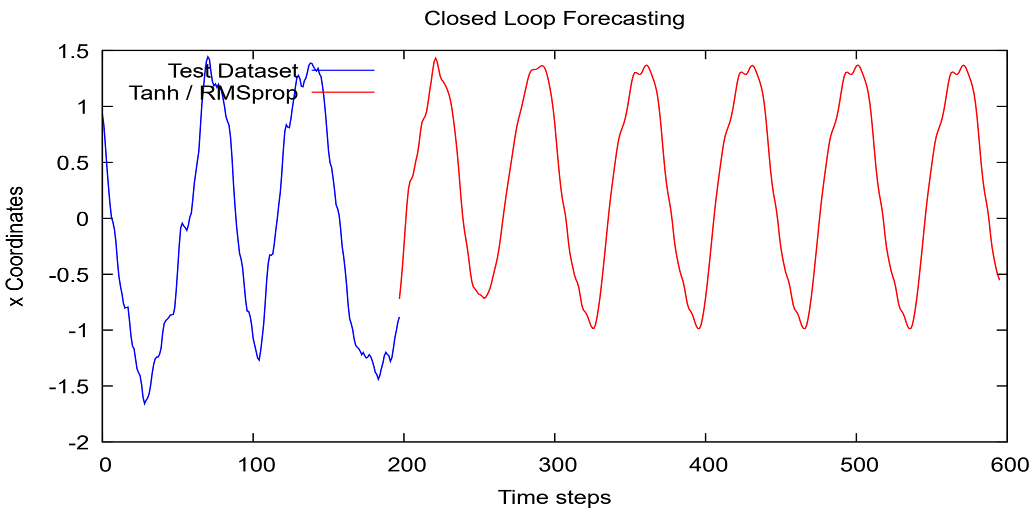


Figure 6.5: Univariate Vanilla LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

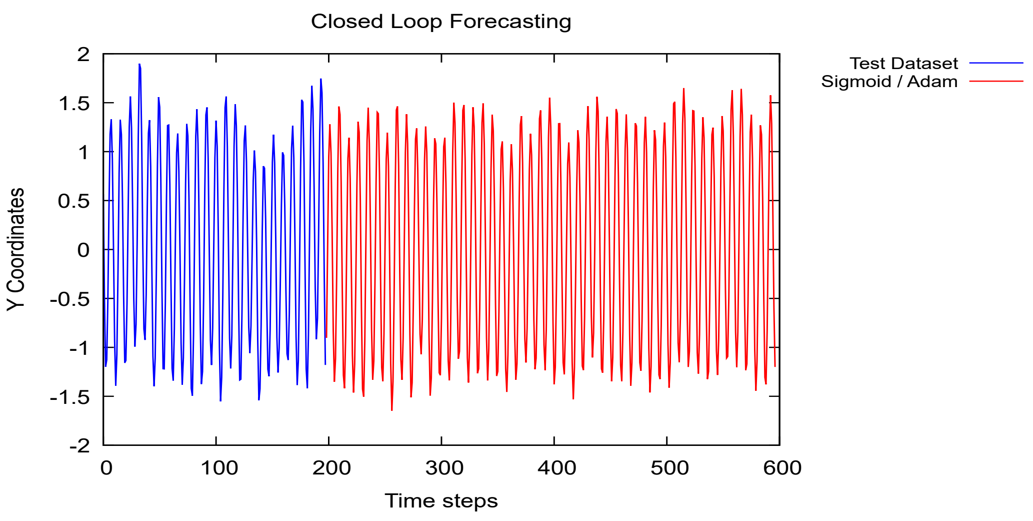


Figure 6.6: Univariate Vanilla LSTM close loop output for y of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

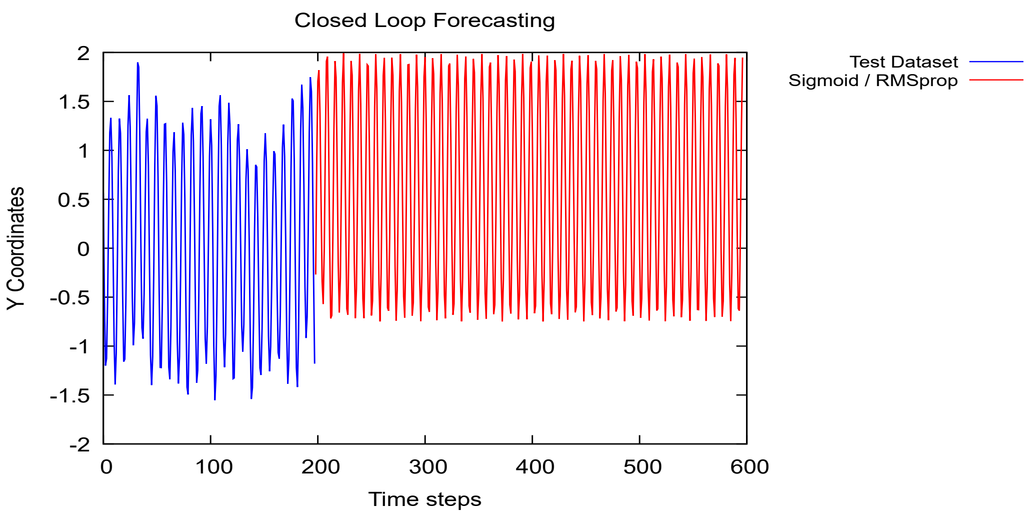


Figure 6.7: Univariate Vanilla LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

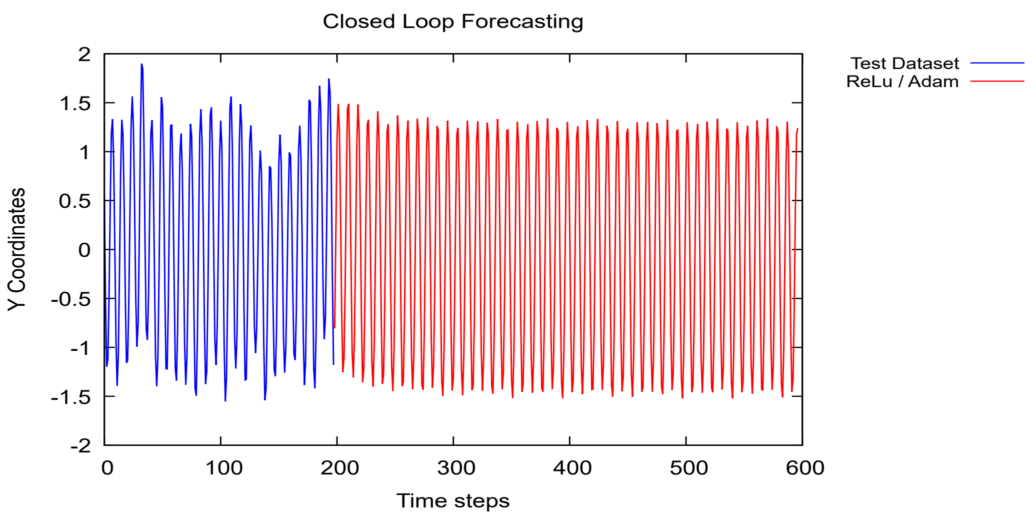


Figure 6.8: Univariate Vanilla LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

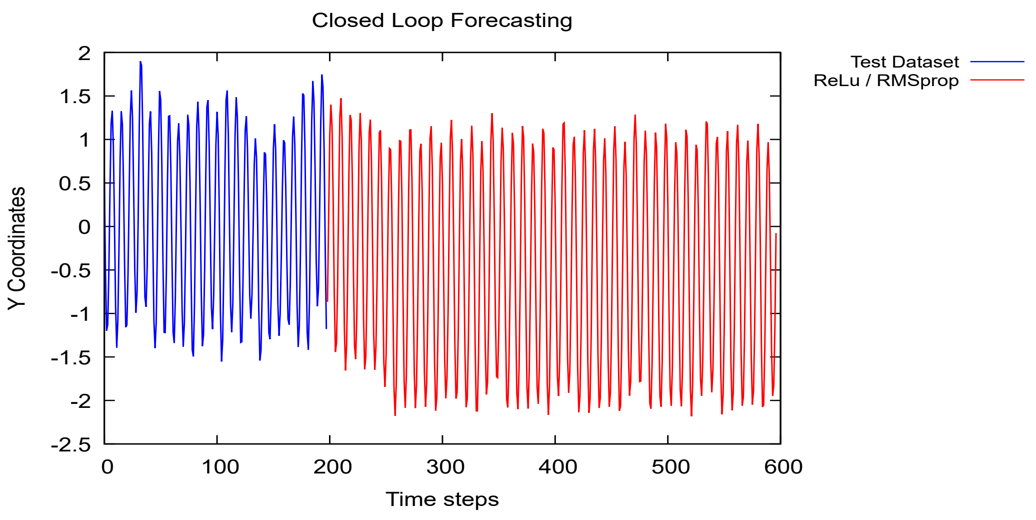


Figure 6.9: Univariate Vanilla LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

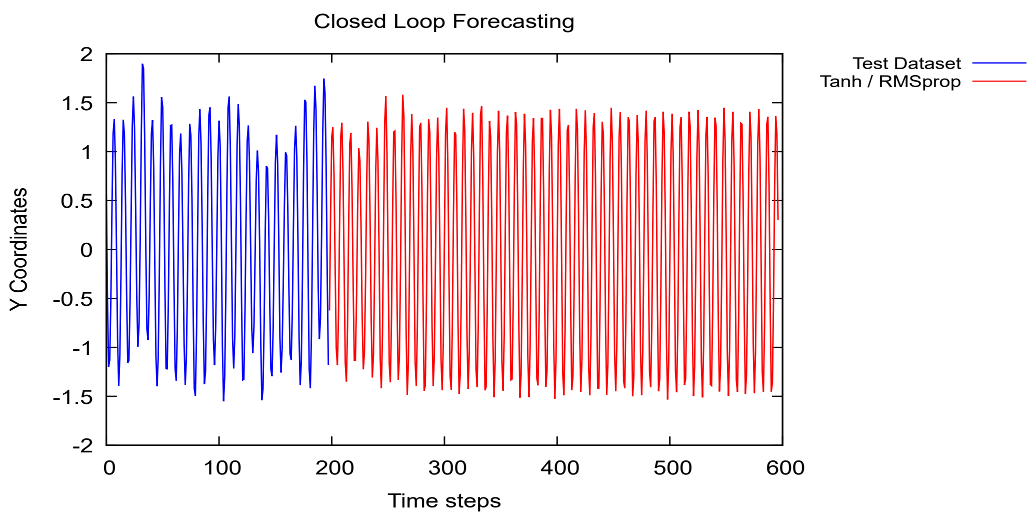


Figure 6.10: Univariate Vanilla LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

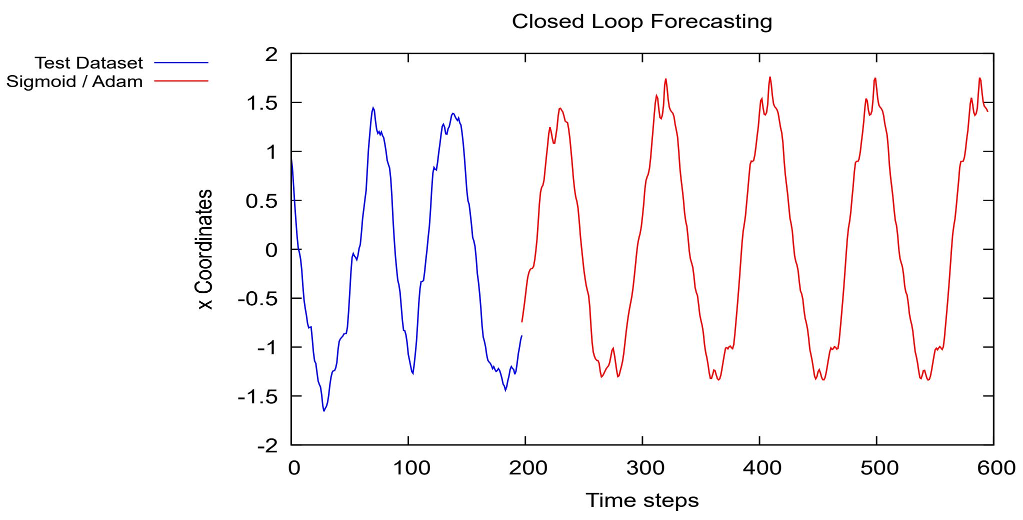


Figure 6.11: Univariate Stacked LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

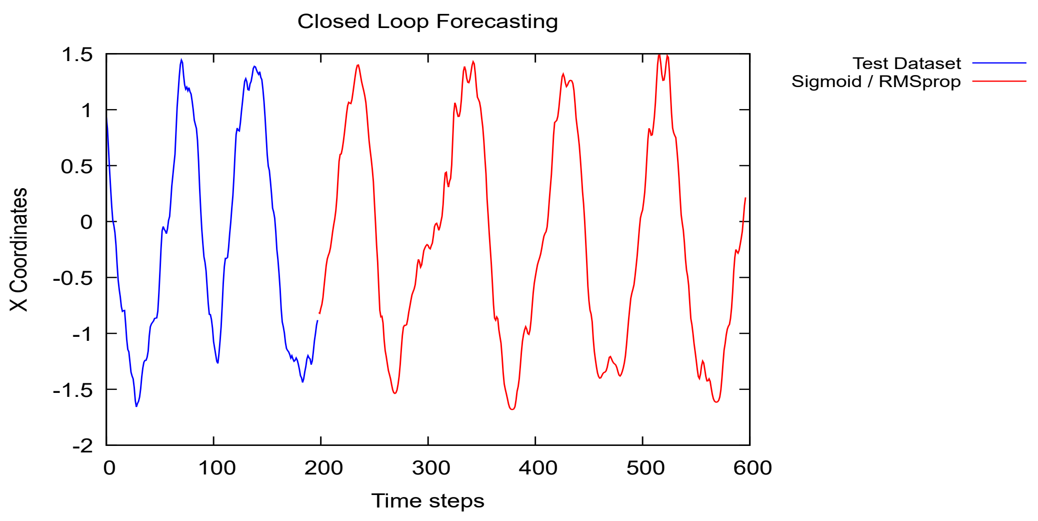


Figure 6.12: Univariate Stacked LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

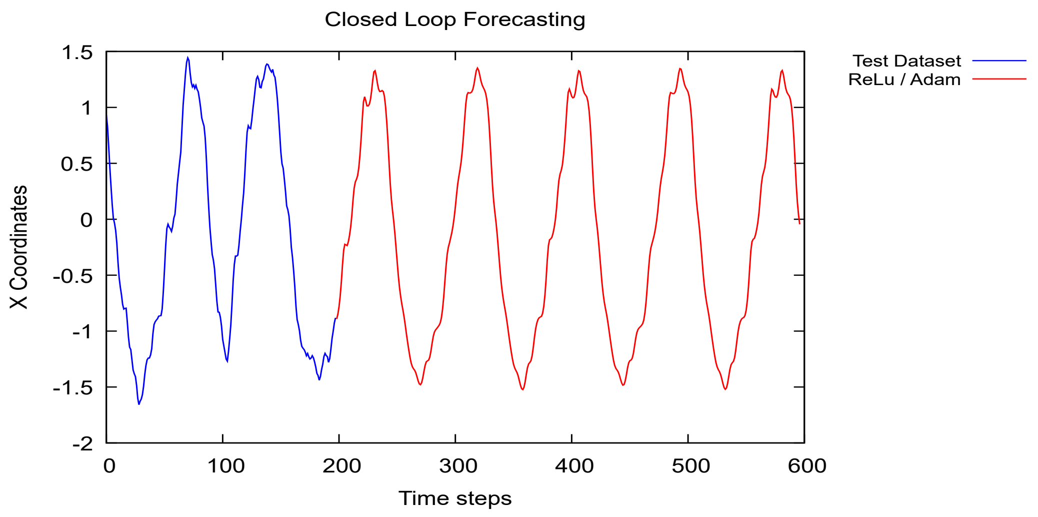


Figure 6.13: Univariate Stacked LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

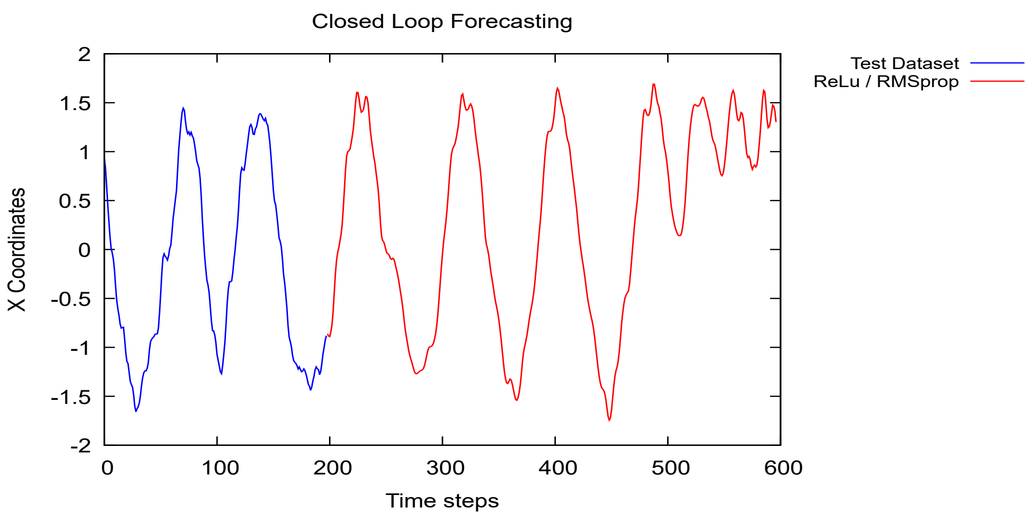


Figure 6.14: Univariate Stacked LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

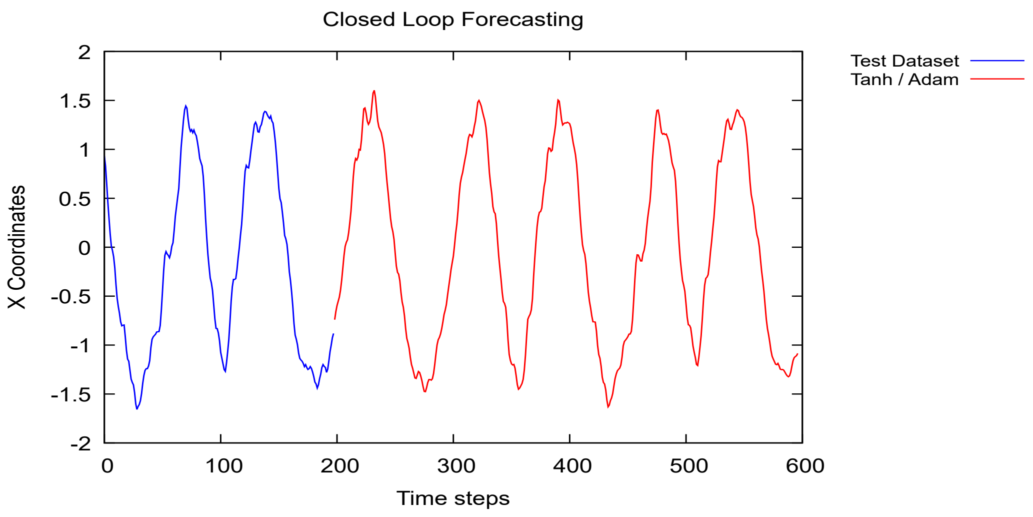


Figure 6.15: Univariate Stacked LSTM close loop output for x points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

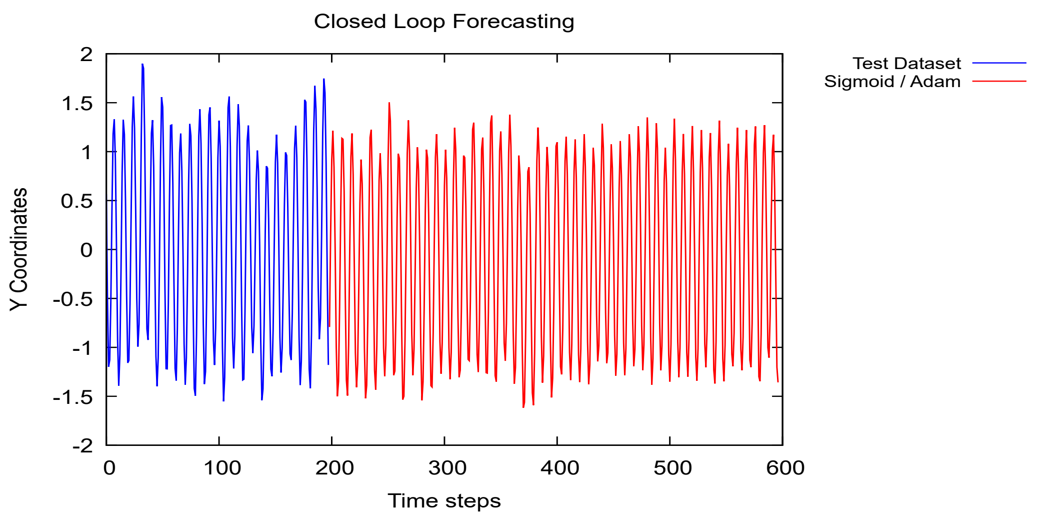


Figure 6.16: Univariate Stacked LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

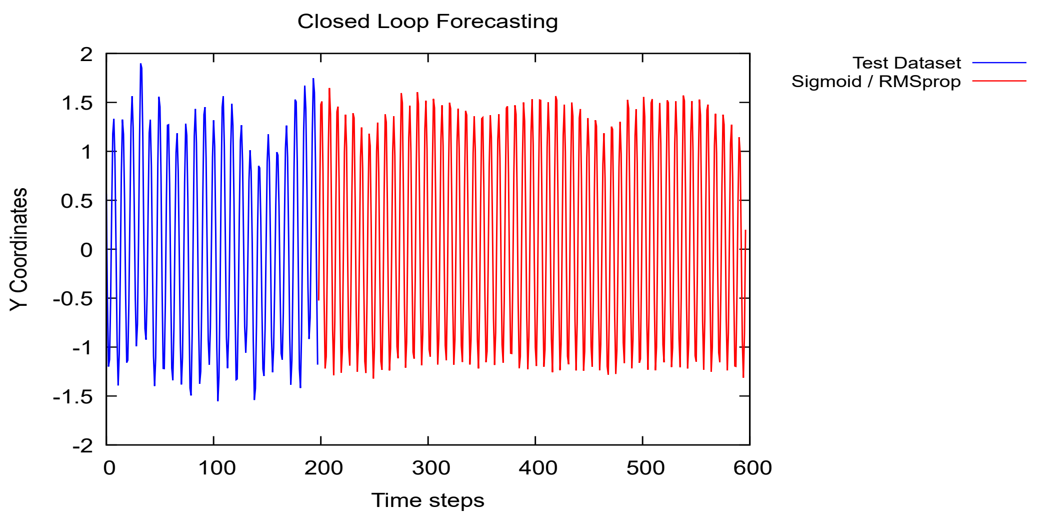


Figure 6.17: Univariate Stacked LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

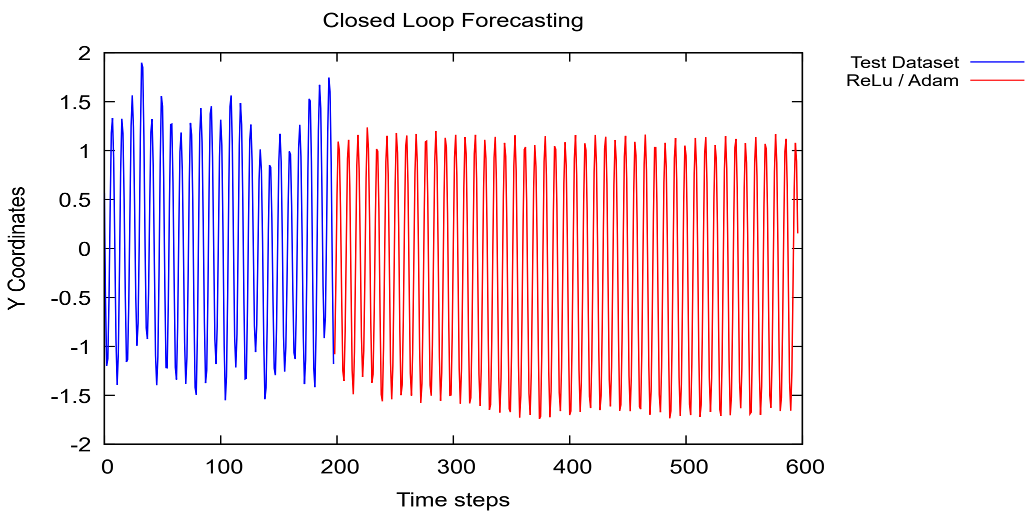


Figure 6.18: Univariate Stacked LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

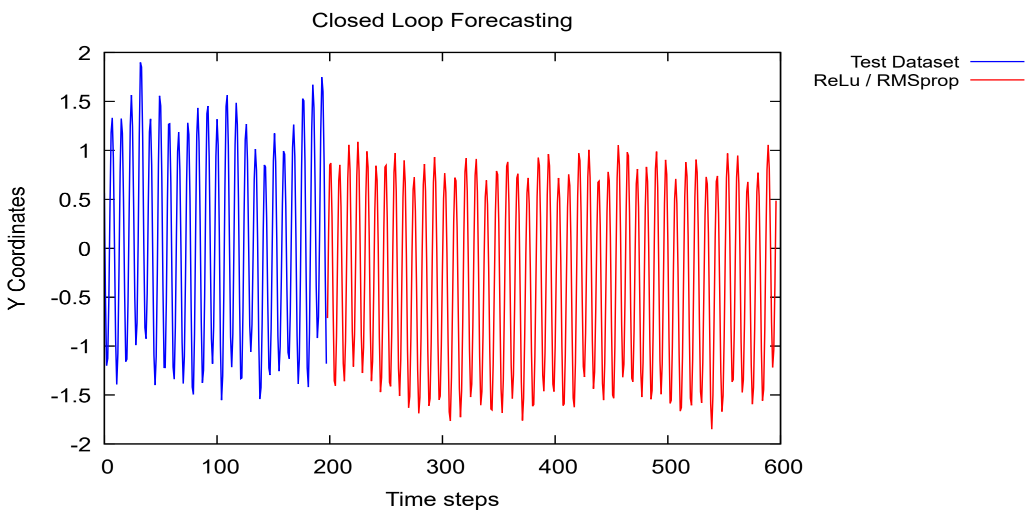


Figure 6.19: Univariate Stacked LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

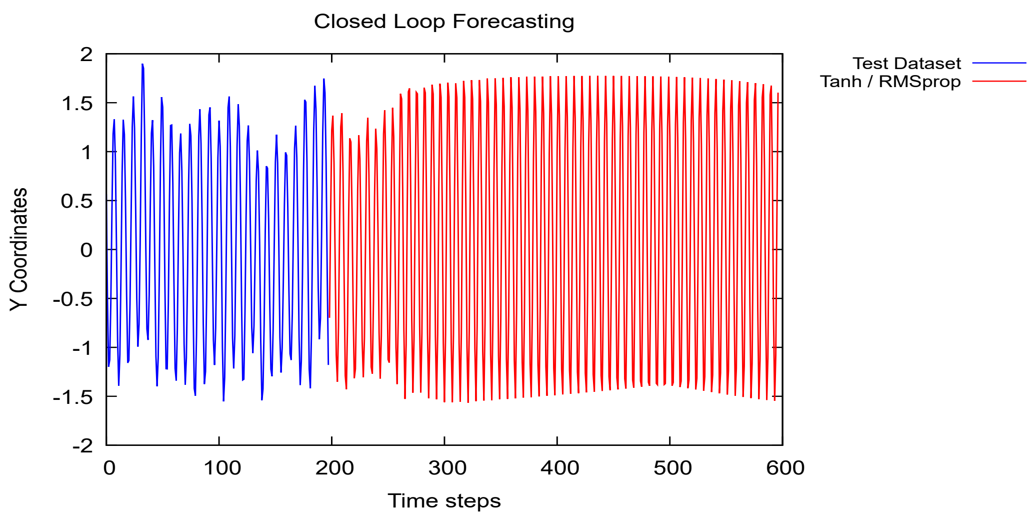


Figure 6.20: Univariate Stacked LSTM close loop output for y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

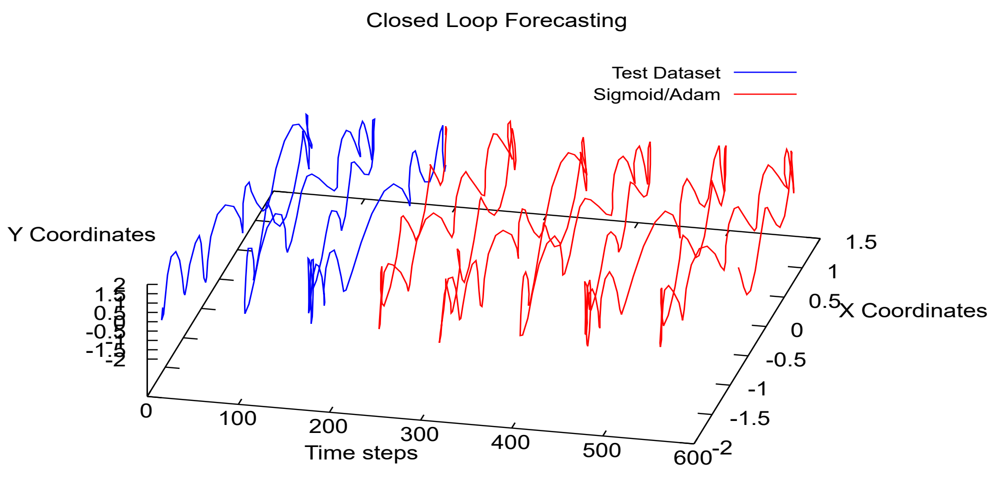


Figure 6.21: Multivariate Vanilla LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

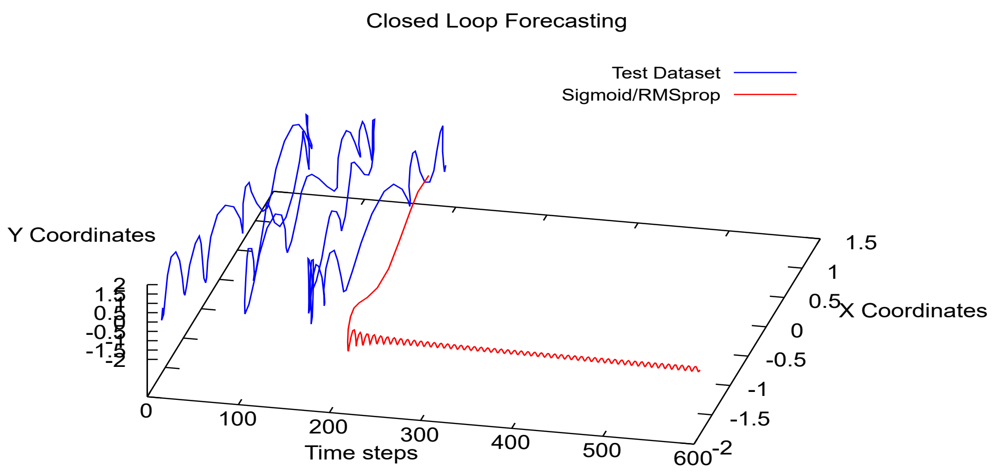


Figure 6.22: Multivariate Vanilla LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

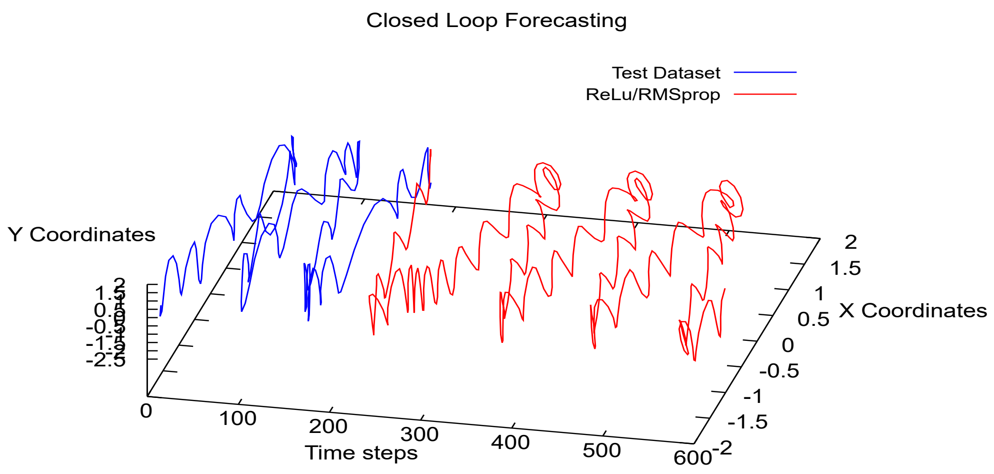


Figure 6.23: Multivariate Vanilla LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

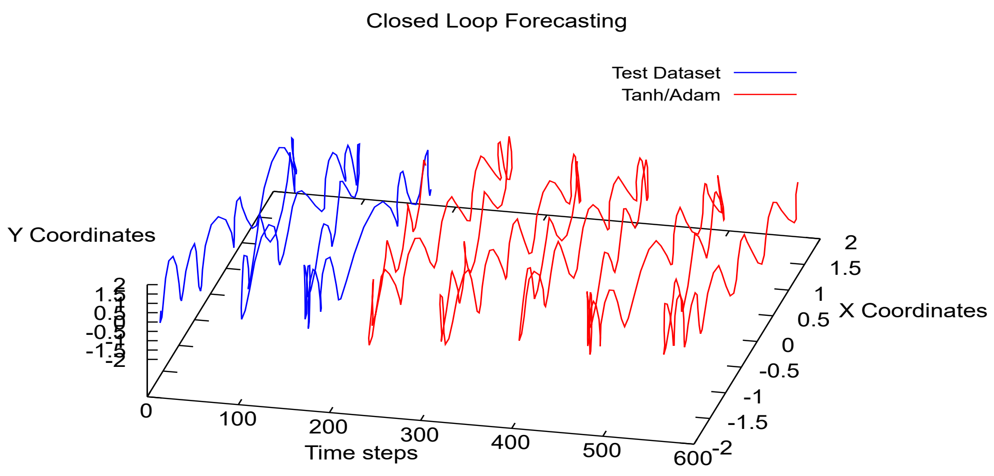


Figure 6.24: Multivariate Vanilla LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tengent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

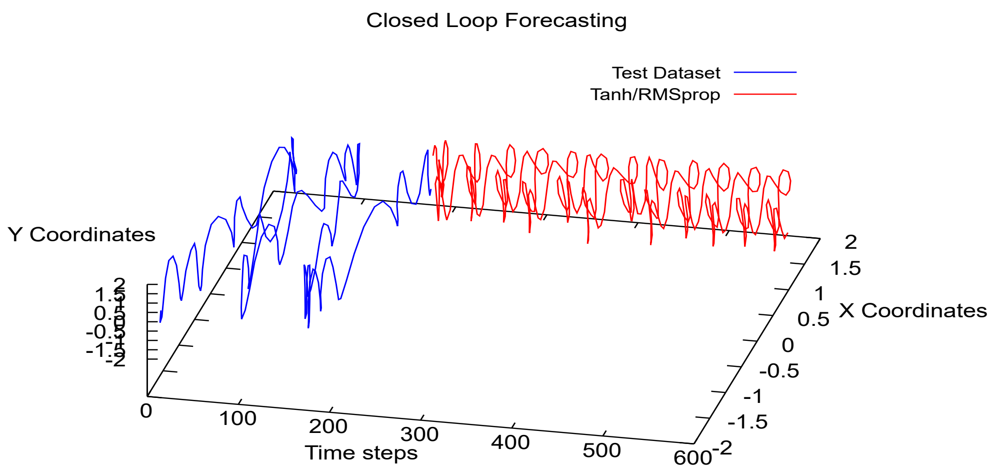


Figure 6.25: Multivariate Vanilla LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

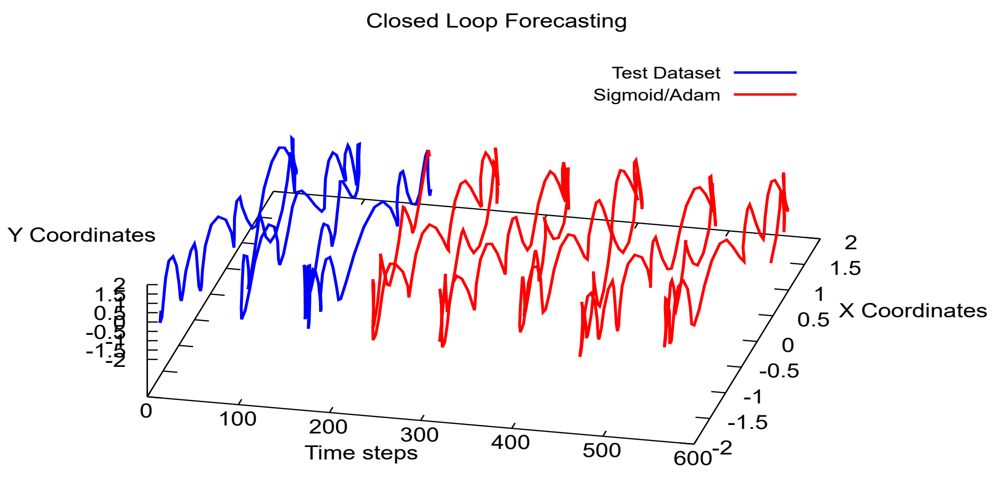


Figure 6.26: Multivariate Stacked LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

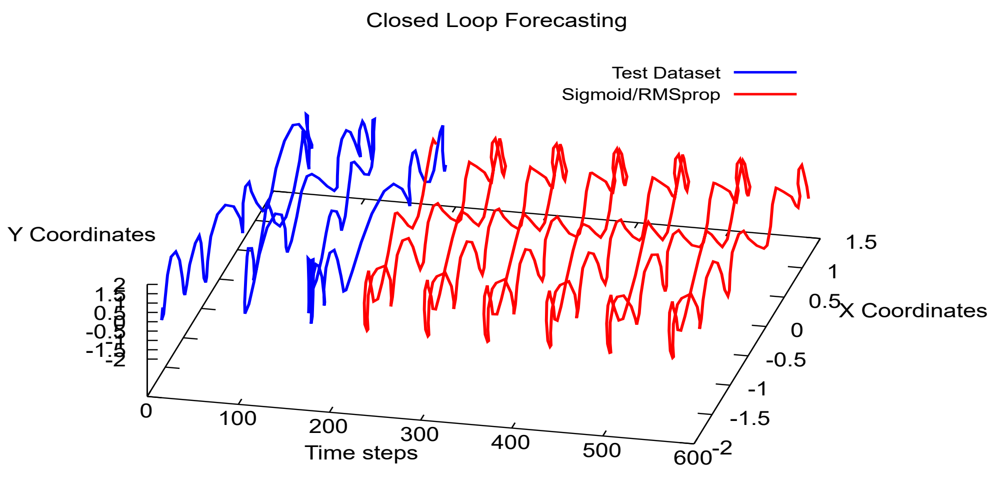


Figure 6.27: Multivariate Stacked LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

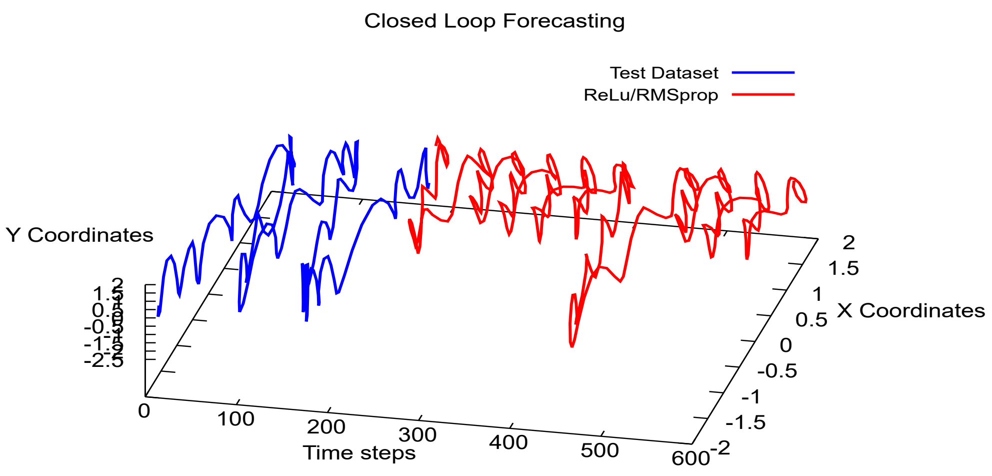


Figure 6.28: Multivariate Stacked LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

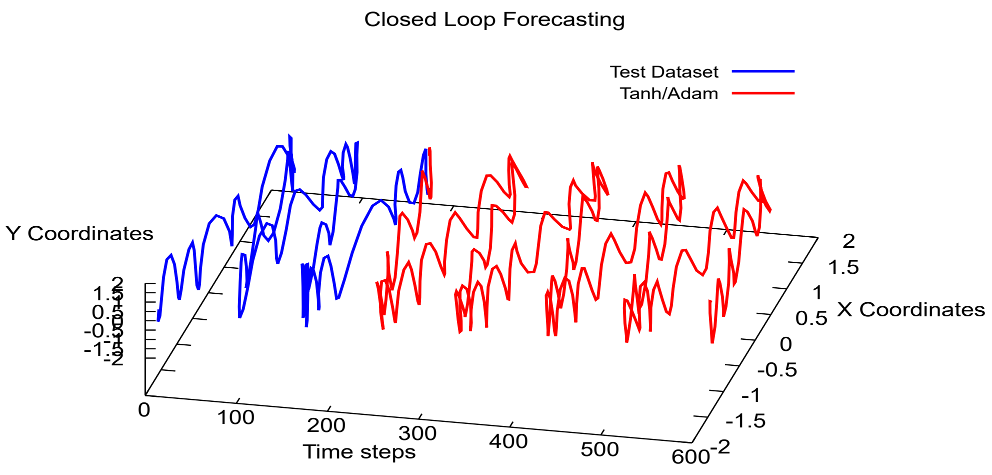


Figure 6.29: Multivariate Stacked LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

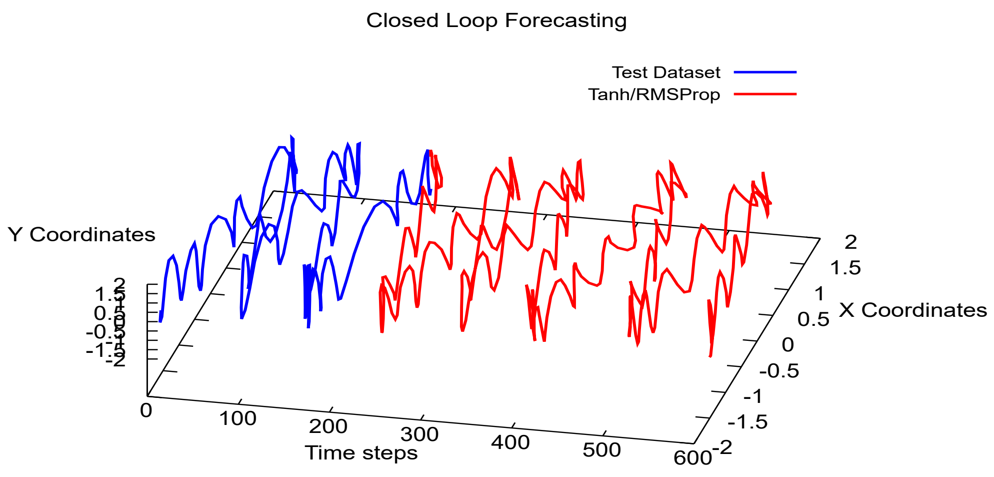


Figure 6.30: Multivariate Stacked LSTM close loop output for x and y points of pattern one with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

7

Appendix B Complementary Results

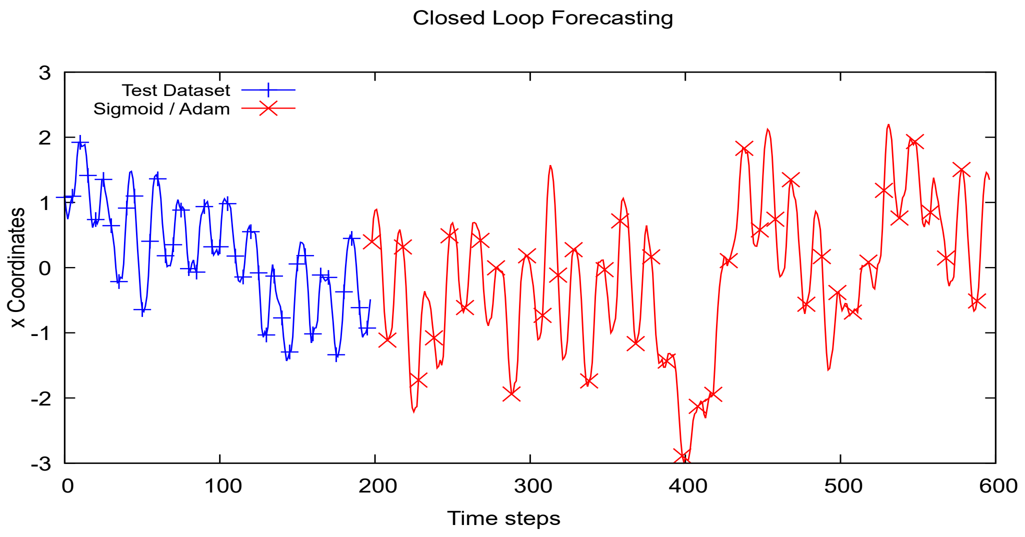


Figure 7.1: Univariate Vanilla LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

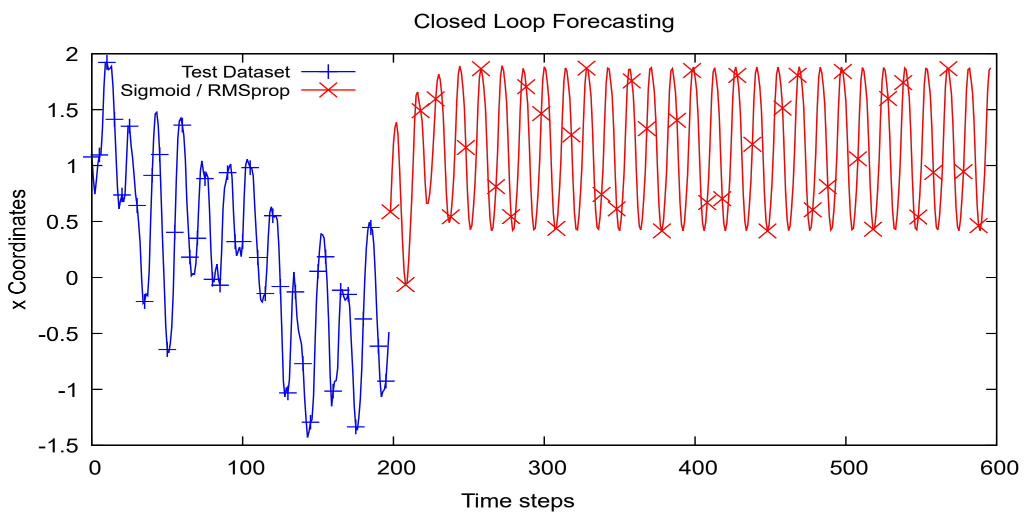


Figure 7.2: Univariate Vanilla LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

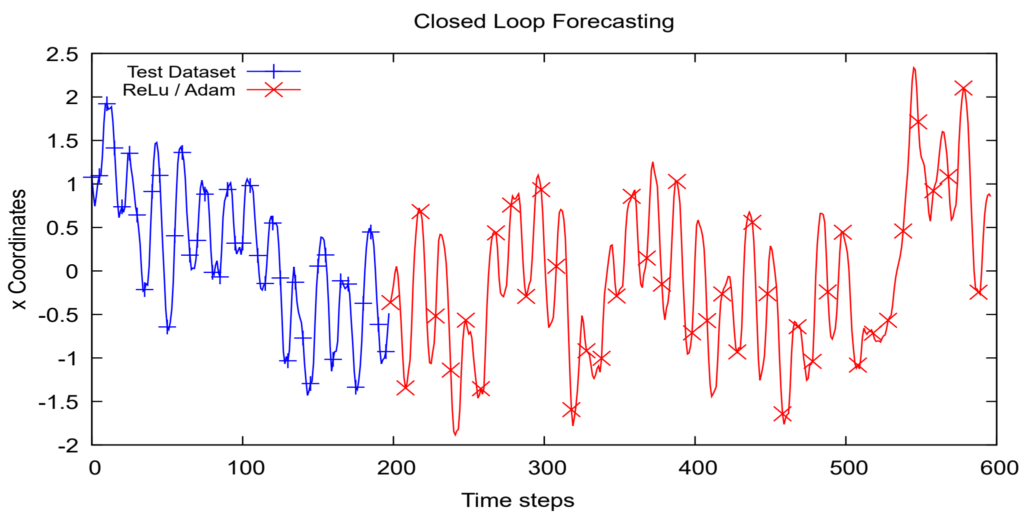


Figure 7.3: Univariate Vanilla LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

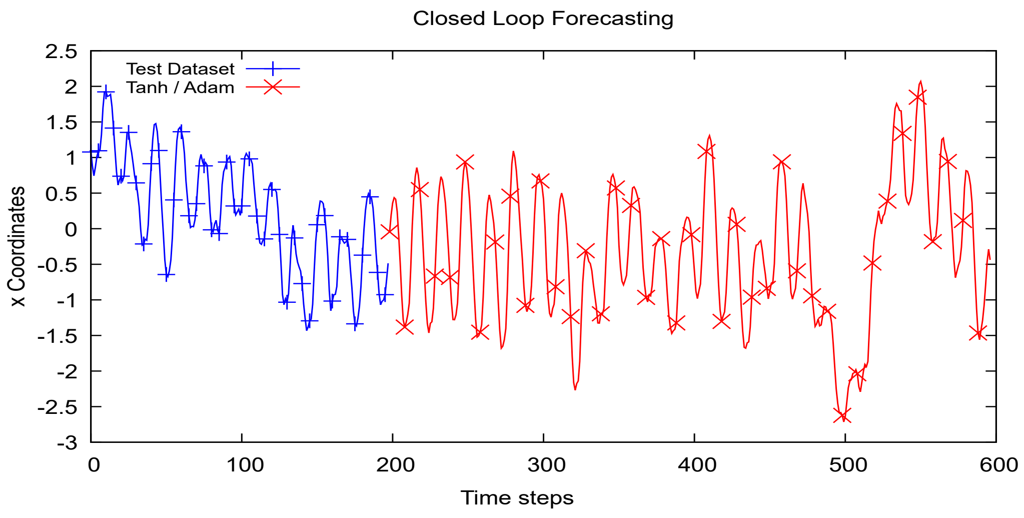


Figure 7.4: Univariate Vanilla LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

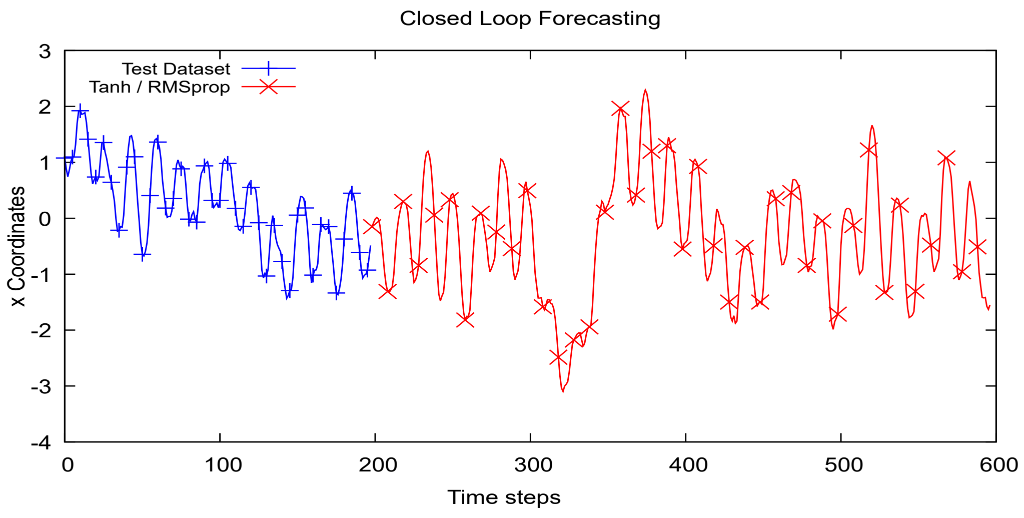


Figure 7.5: Univariate Vanilla LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

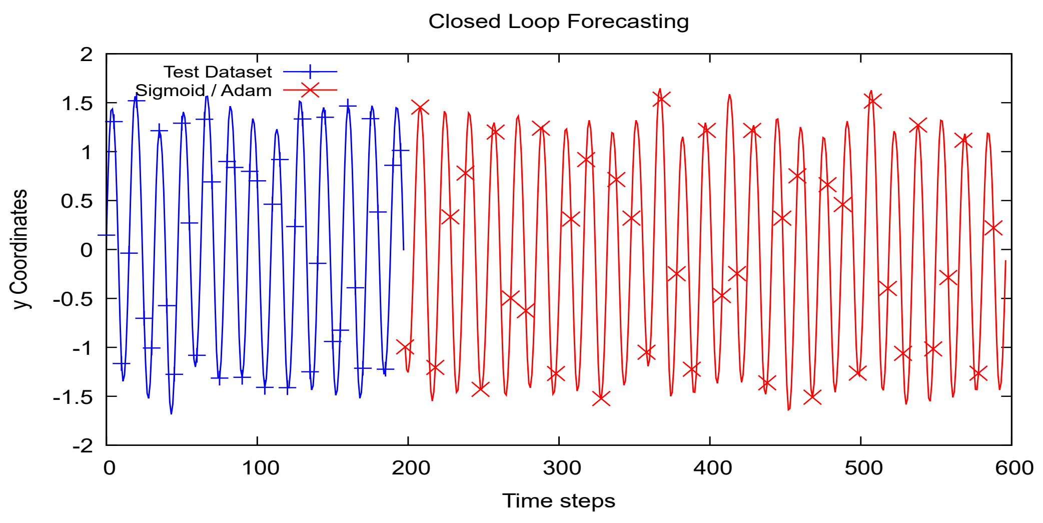


Figure 7.6: Univariate Vanilla LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

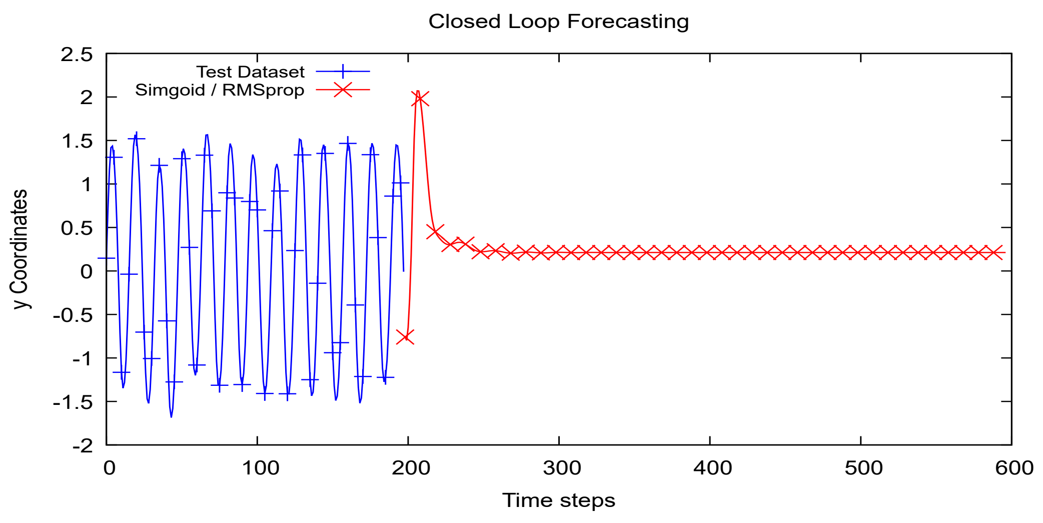


Figure 7.7: Univariate Vanilla LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

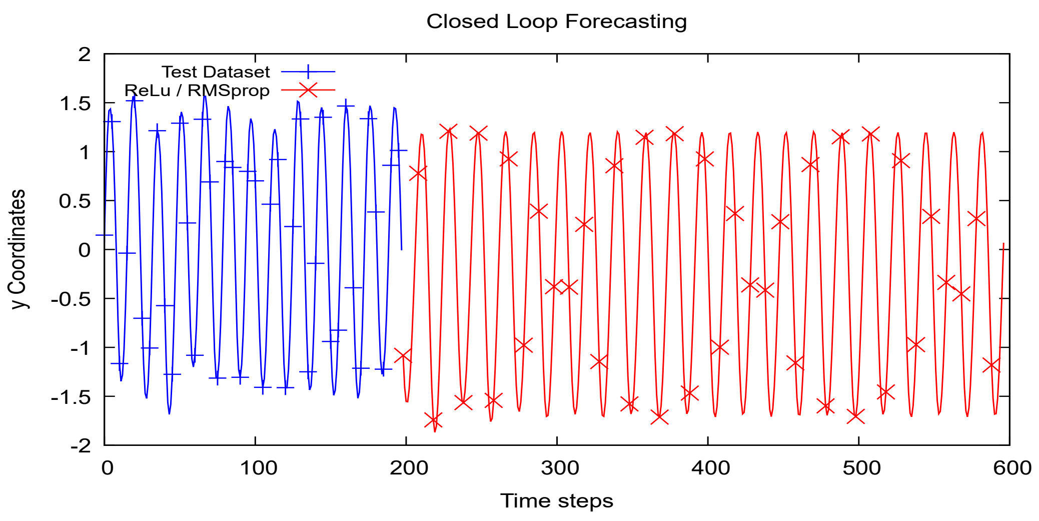


Figure 7.8: Univariate Vanilla LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

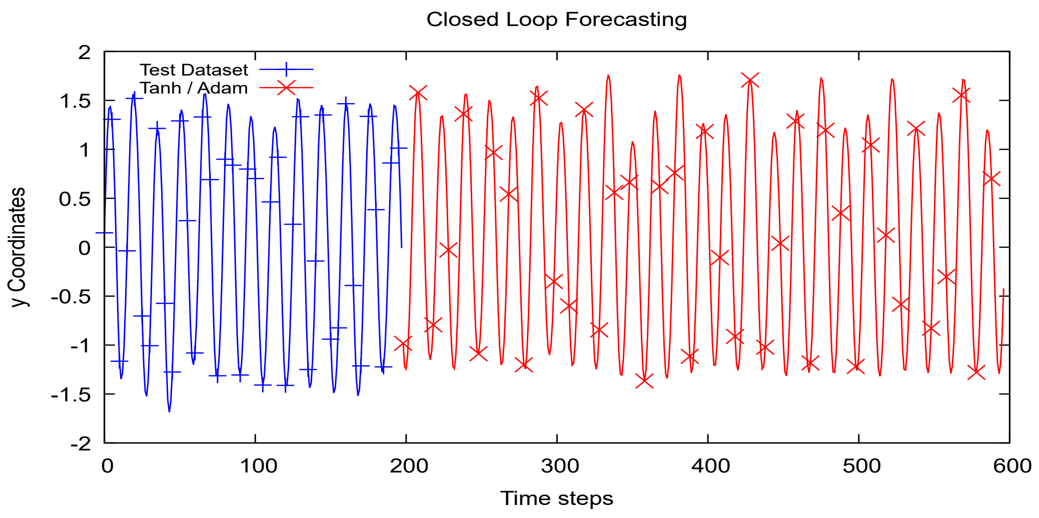


Figure 7.9: Univariate Vanilla LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

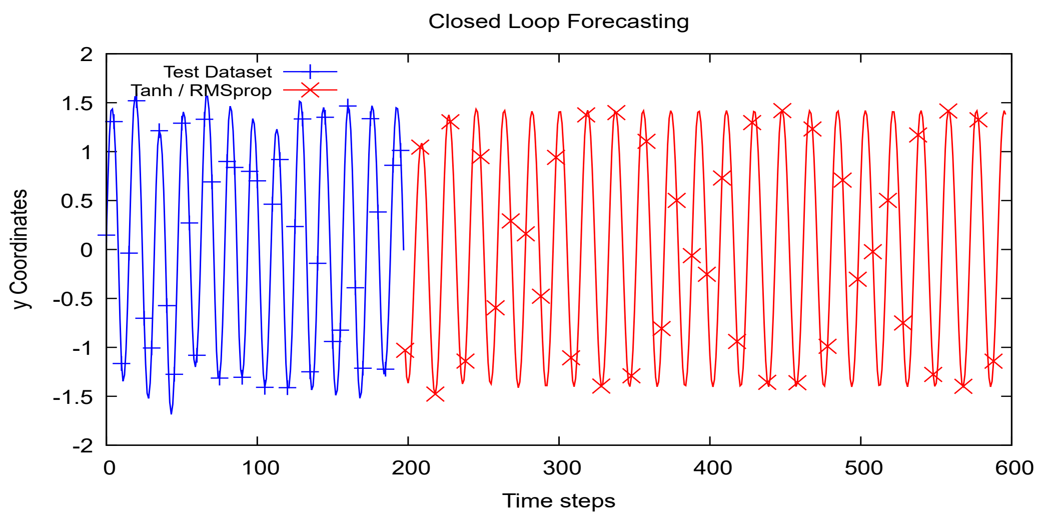


Figure 7.10: Univariate Vanilla LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

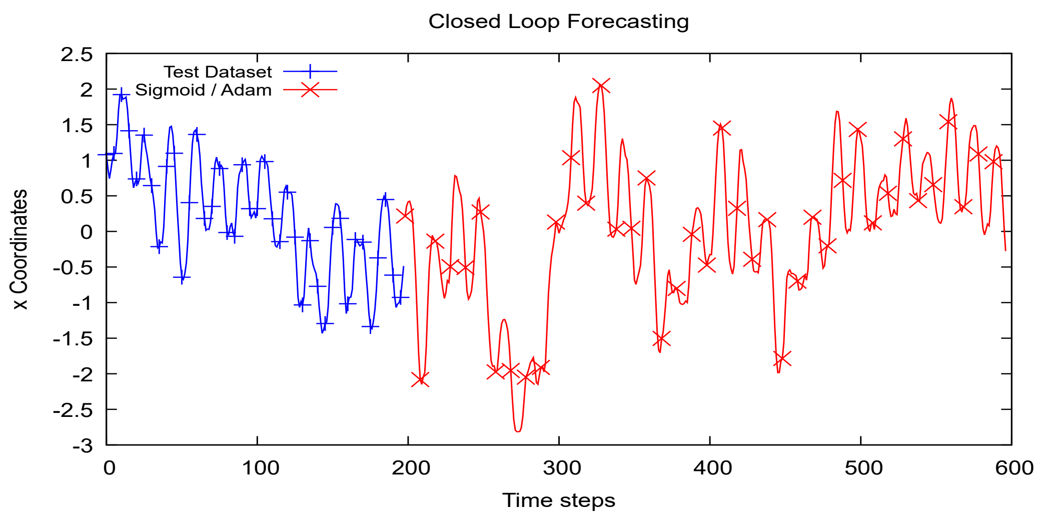


Figure 7.11: Univariate Stacked LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

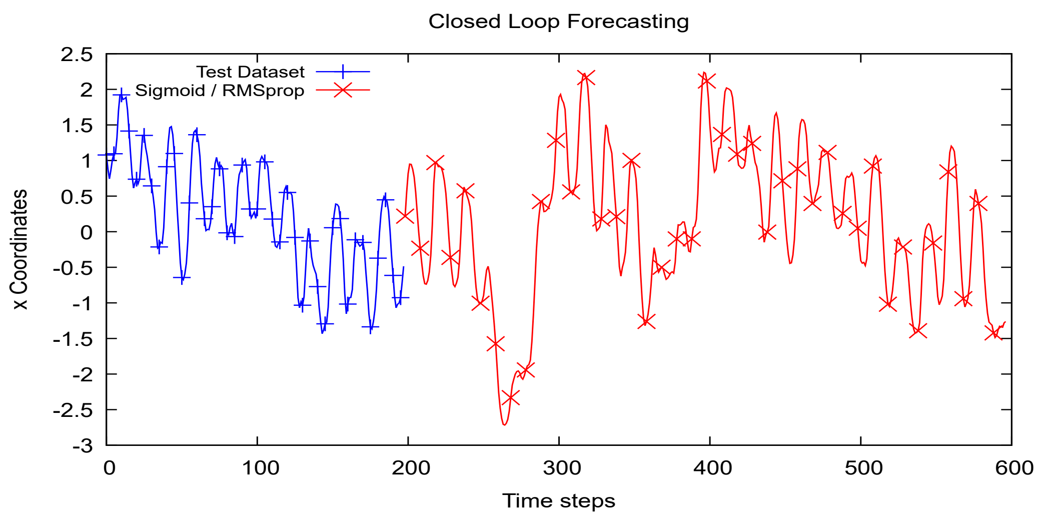


Figure 7.12: Univariate Stacked LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

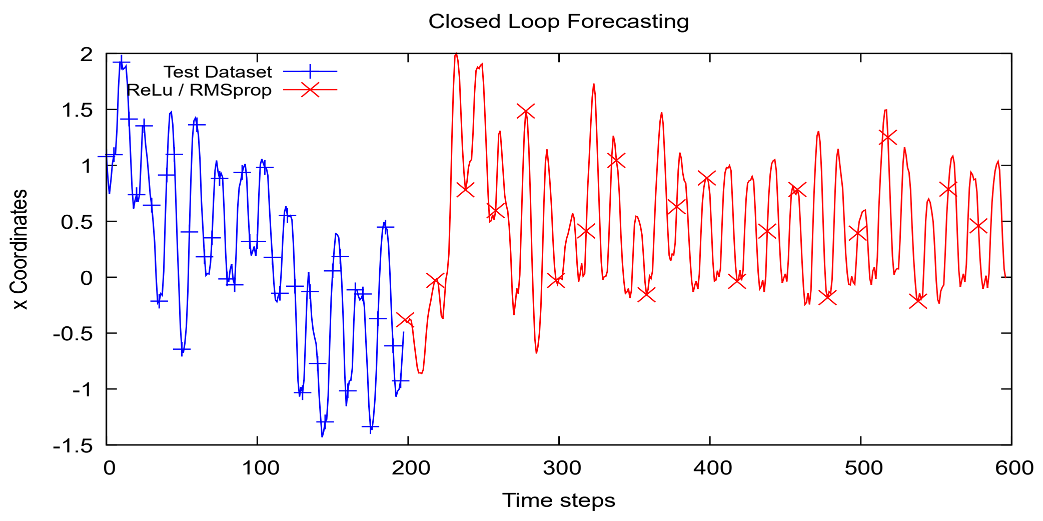


Figure 7.13: Univariate Stacked LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

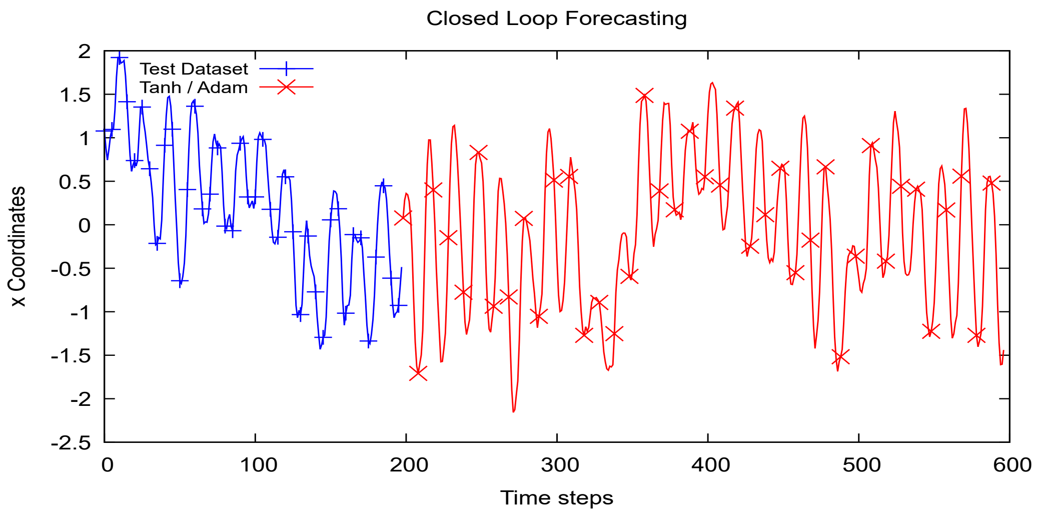


Figure 7.14: Univariate Stacked LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

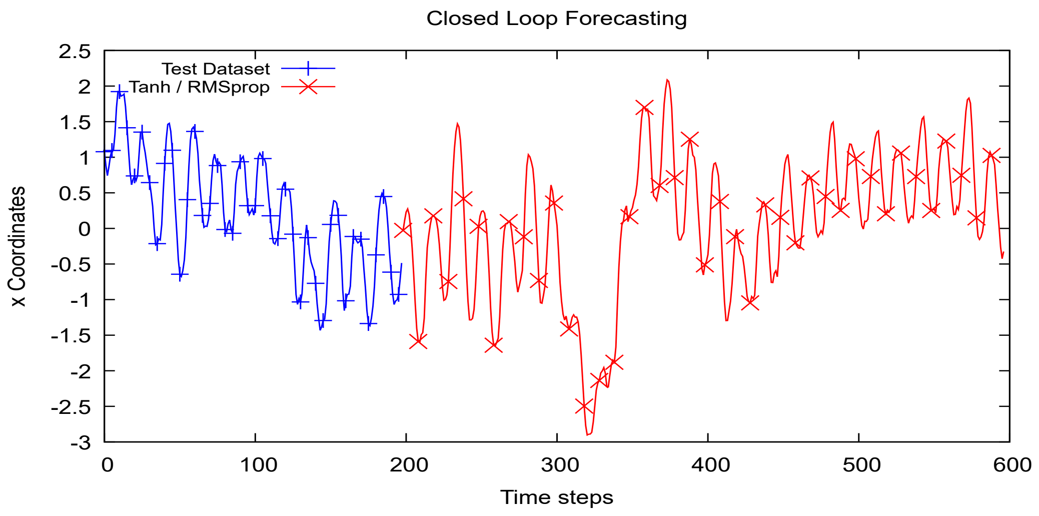


Figure 7.15: Univariate Stacked LSTM close loop output for x points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

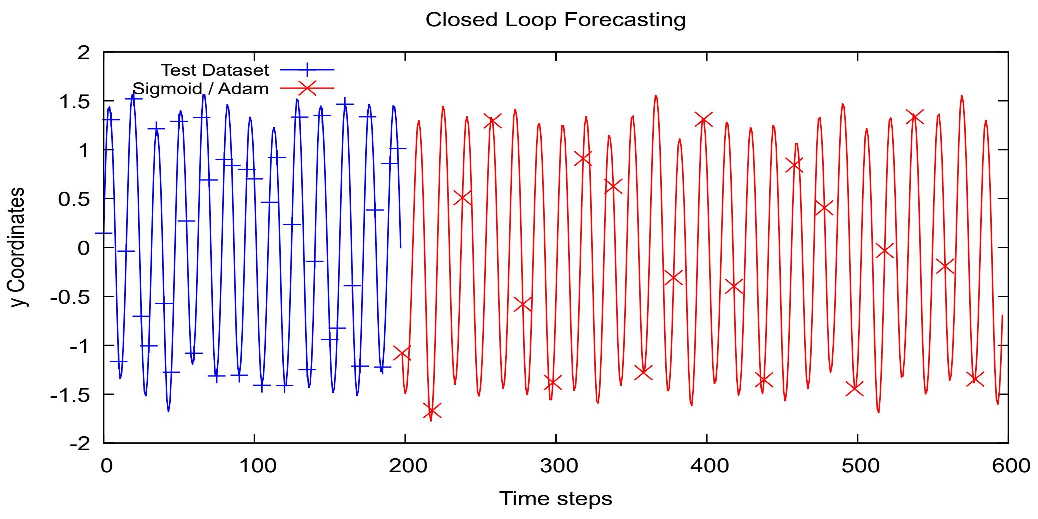


Figure 7.16: Univariate Stacked LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

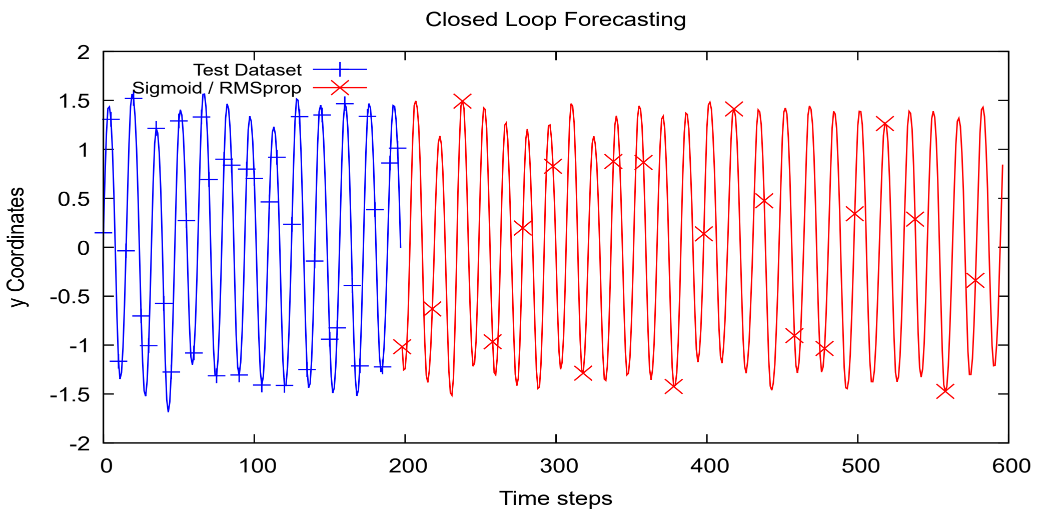


Figure 7.17: Univariate Stacked LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

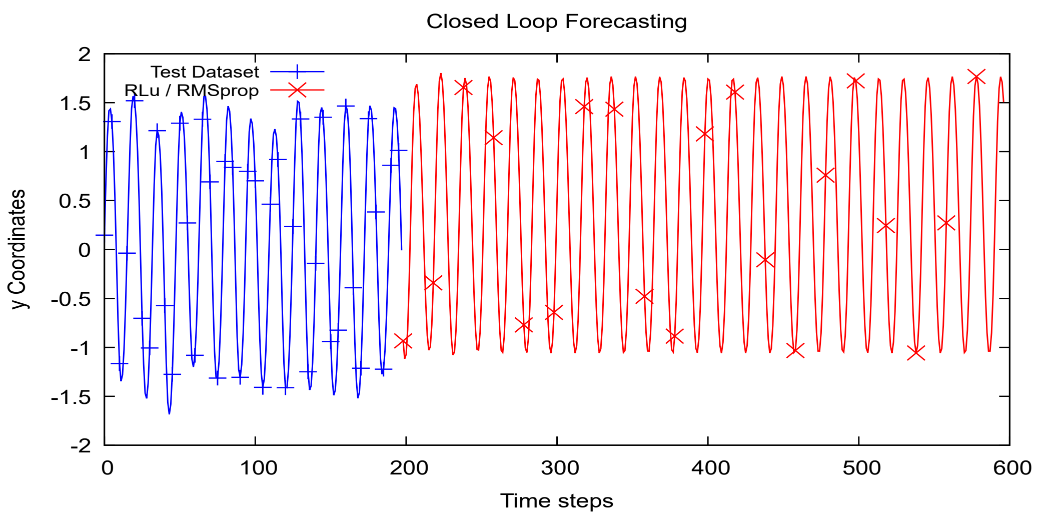


Figure 7.18: Univariate Stacked LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

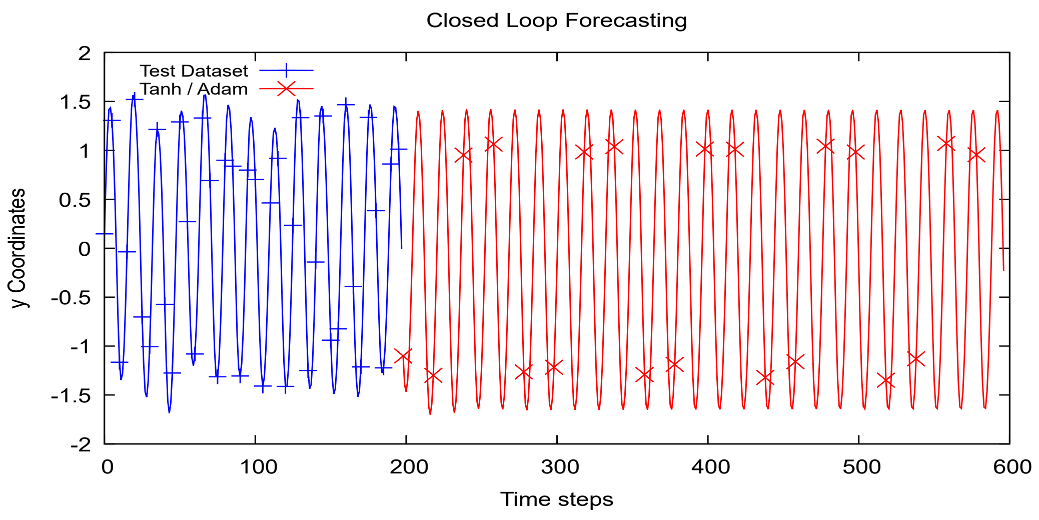


Figure 7.19: Univariate Stacked LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

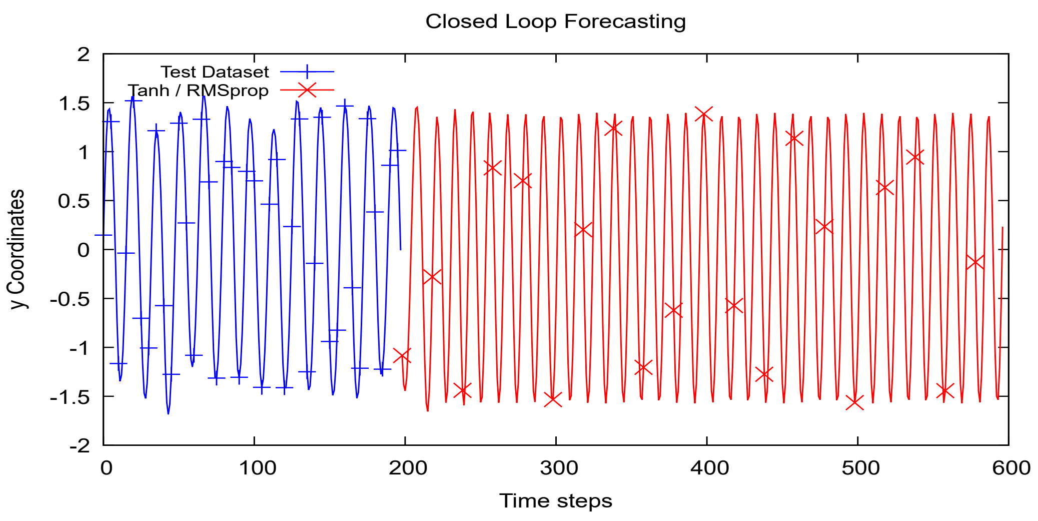


Figure 7.20: Univariate Stacked LSTM close loop output for y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 512 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

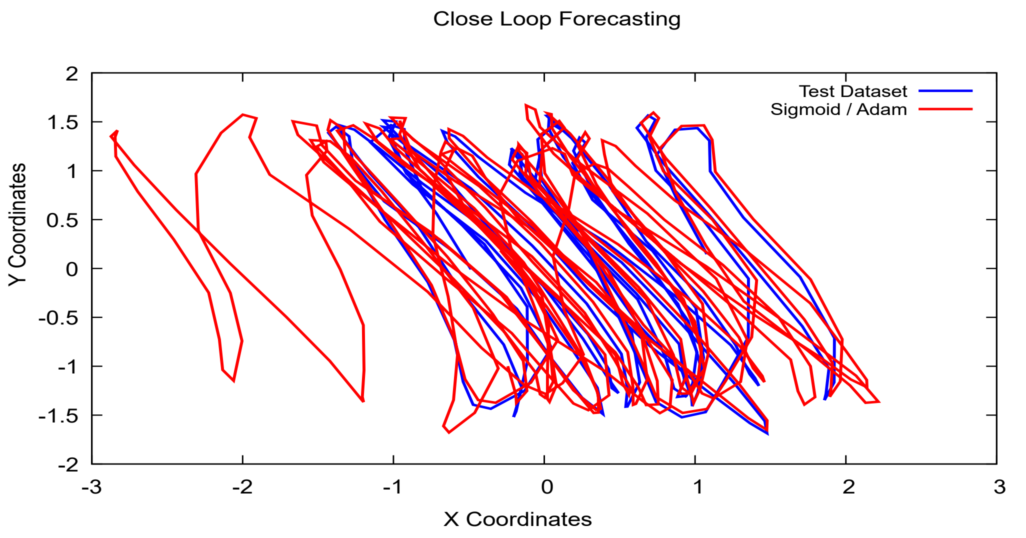


Figure 7.21: Multivariate Vanilla LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

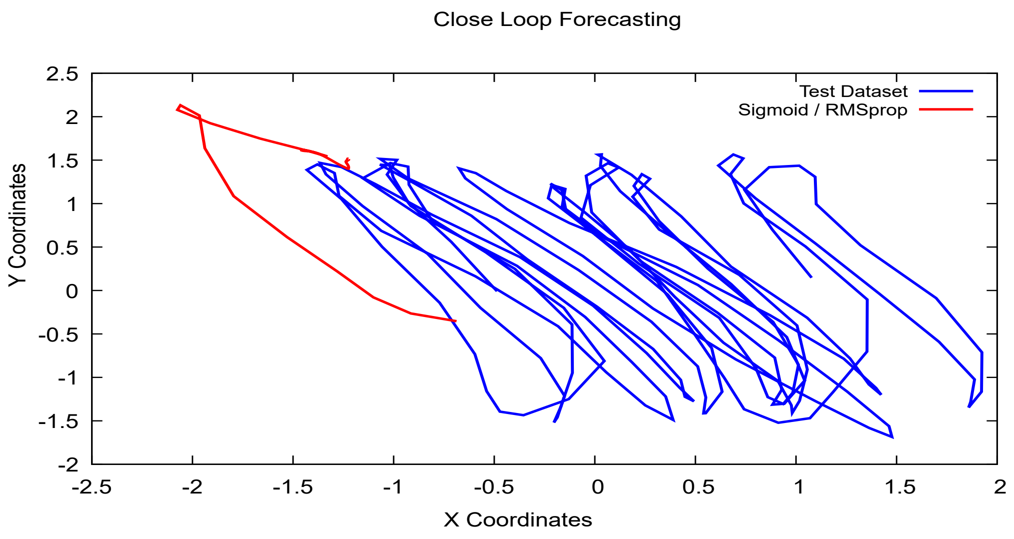


Figure 7.22: Multivariate Vanilla LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

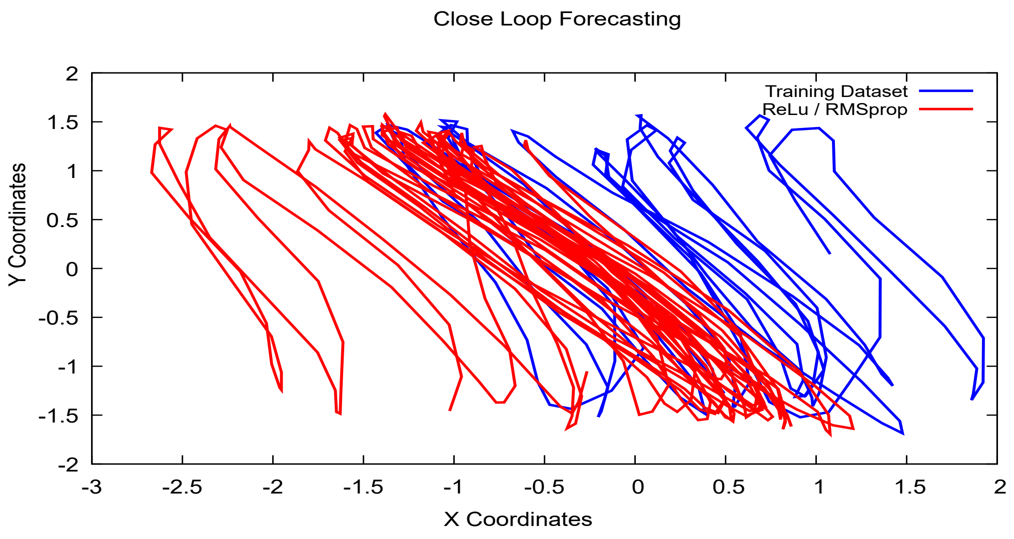


Figure 7.23: Multivariate Vanilla LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

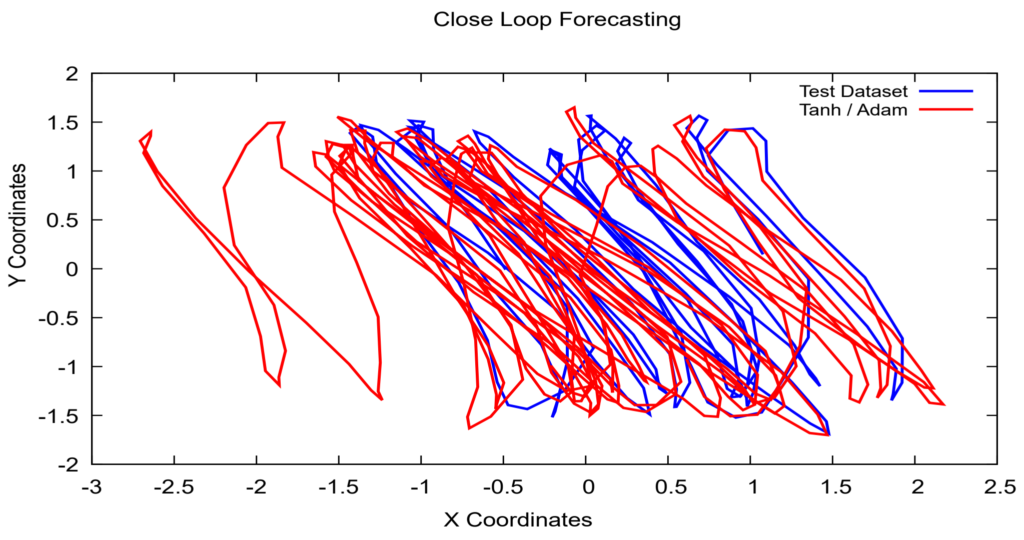


Figure 7.24: Multivariate Vanilla LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

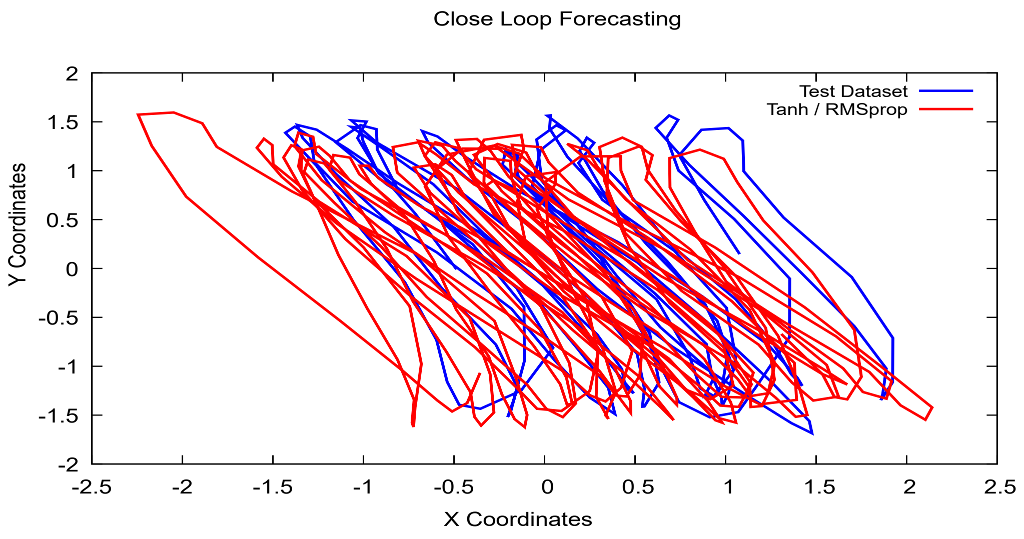


Figure 7.25: Multivariate Vanilla LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

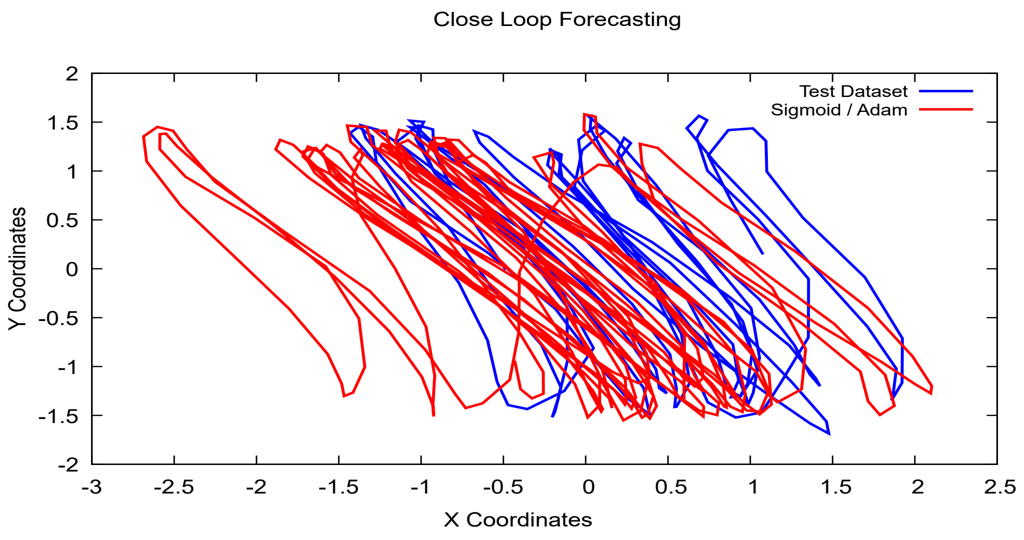


Figure 7.26: Multivariate Stacked LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

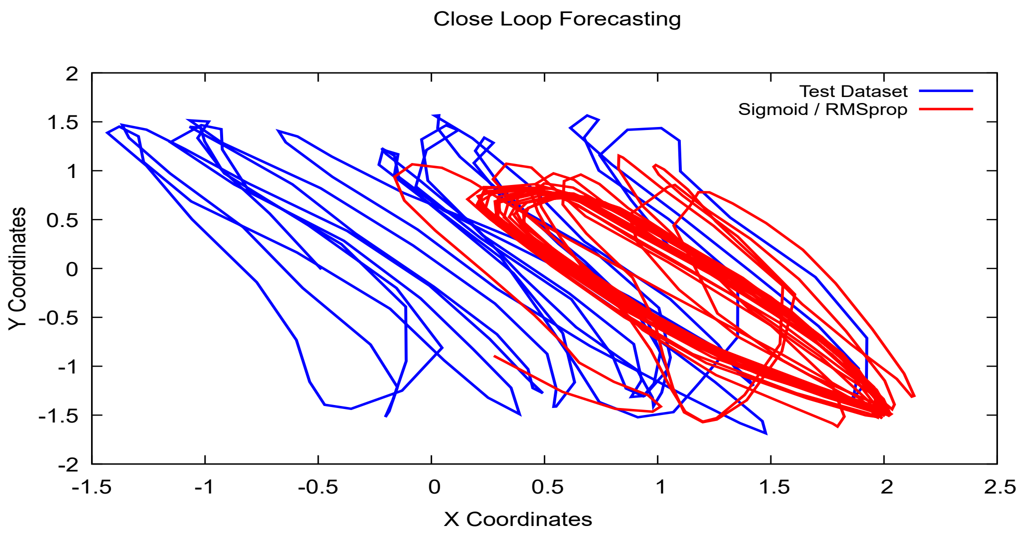


Figure 7.27: Multivariate Stacked LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, sigmoid as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

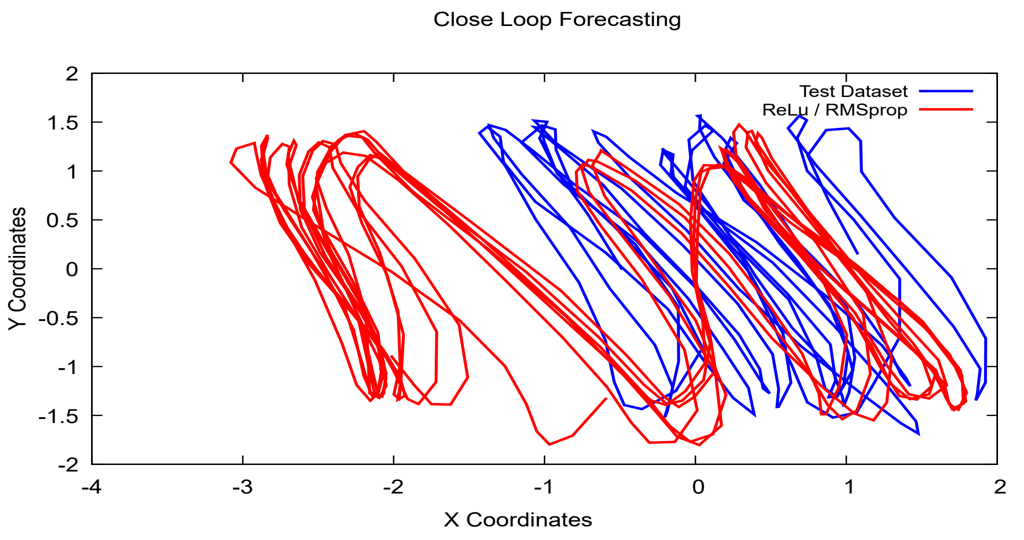


Figure 7.28: Multivariate Stacked LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, ReLu as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

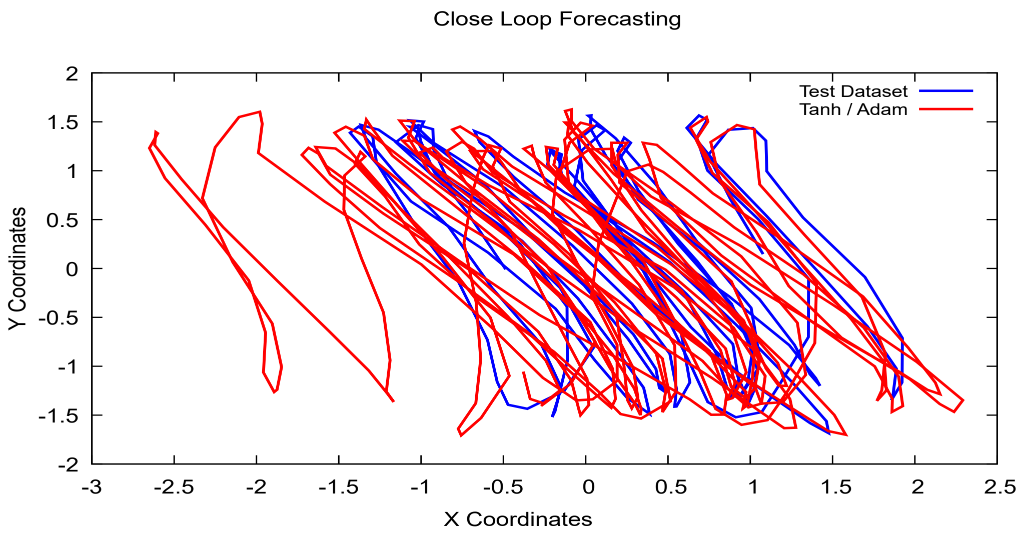


Figure 7.29: Multivariate Stacked LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and Adam as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

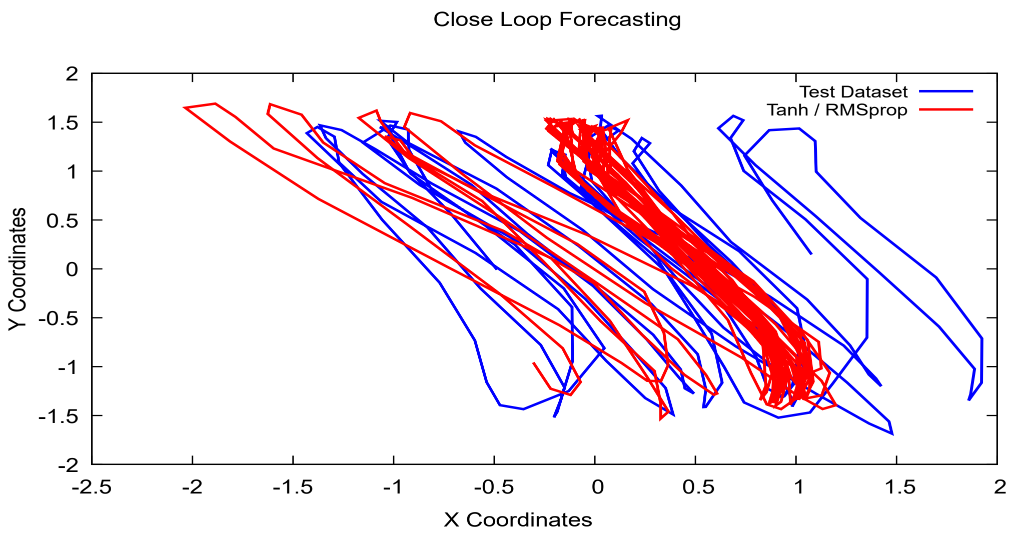


Figure 7.30: Multivariate Stacked LSTM close loop output for x and y points of pattern two with with 200 time step, 70 epochs, 16 for batch size, 1024 neurons in the hidden layer, a train-test split ratio of 90-10%, tangent as activation function and RMSprop as optimizer. Where the blue line is the input of the trained NN and the red line is the output.

Bibliography

[1] *Panda's Instruction Handbook.* April 2020.

[2] H. Ochoa and R. Cortesão, "Control architecture for robotic-assisted polishing tasks based on human skills," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 630–637, 2019.

[3] S. L. Loli Burgueño, Jordi Cabot and S. Gérard, "A generic lstm neural network architecture to infer heterogeneous model transformations," in *Software and Systems Modeling*, vol. 21, pp. 139–156, 2022.

[4] C. Bergström and O. Hjelm, "Impact of time steps on stock market prediction with lstm," 2019.

[5] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[6] M. A. Istiake Sunny, M. M. S. Maswood, and A. G. Alharbi, "Deep learning-based stock price prediction using lstm and bi-directional lstm model," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, pp. 87–92, 2020.

[7] D. Haputhanthri and A. Wijayasiri, "Short-term traffic forecasting using lstm-based deep learning models," in *2021 Moratuwa Engineering Research Conference (MERCon)*, pp. 602–607, 2021.

[8] V. Santhosh and K. R. Babu, "Design and wrapage analysis of plastic injection mould," in *National Conference on Challenges in Research & Technology in the Coming Decades (CRT 2013)*, pp. 1–5, 2013.

[9] Y. Choi, J. Shin, H. Choi, and S. Lee, “Quality management system for web-based collaboration in mold amp; die industry,” in *The 40th International Conference on Computers Indutrial Engineering*, pp. 1–6, 2010.

[10] S. Chen and W. Lai, “Control system software design of injection molding machine based on neural network,” in *2011 Second International Conference on Mechanic Automation and Control Engineering*, pp. 1119–1122, 2011.

[11] A. Tellaeche and R. Arana, “Machine learning algorithms for quality control in plastic molding industry,” in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–4, 2013.

[12] “Moldes injeções plásticos.” `http://moldesinjecaoplasticos.com.br/`, june 2017.

[13] A. E. K. Mohammad, J. Hong, and D. Wang, “Design of a force-controlled end-effector with low-inertia effect for robotic polishing using macro-mini robot approach,” *Robotics and Computer-Integrated Manufacturing*, vol. 49, pp. 54–65, 2018.

[14] M. Tsai, J.-L. Chang, and J.-F. Haung, “Development of an automatic mold polishing system,” *IEEE Transactions on Automation Science and Engineering*, vol. 2, no. 4, pp. 393–397, 2005.

[15] C. Gaz, E. Magrini, and A. De Luca, “A model-based residual approach for human-robot collaboration during manual polishing operations,” *Mechatronics*, vol. 55, pp. 234–247, 2018.

[16] Z. Shenghao and S. Jinchun, “Impedance control for vehicle driving with human operation under unstructured environment,” in *2011 International Conference on Internet Computing and Information Services*, pp. 159–162, 2011.

[17] W. He, Y. Dong, and C. Sun, “Adaptive neural impedance control of a robotic manipulator with input saturation,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 334–344, 2016.

[18] M. Nechyba and Y. Xu, “Human skill transfer: neural networks as learners and teachers,” in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 3, pp. 314–319 vol.3, 1995.

[19] R. Wu, H. Zhang, and J. Zhao, “Robot variable impedance skill transfer and learning framework based on a simplified human arm impedance model,” *IEEE Access*, vol. 8, pp. 225627–225638, 2020.

[20] H. Ochoa and R. Cortesão, “Impedance control architecture for robotic-assisted mold polishing based on human demonstration,” *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 3822–3830, 2022.

[21] M. Lu and F. Li, “Survey on lie group machine learning,” *Big Data Mining and Analytics*, vol. 3, no. 4, pp. 235–258, 2020.

[22] G. Meena, D. Sharma, and M. Mahrishi, “Traffic prediction for intelligent transportation system using machine learning,” in *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*, pp. 145–148, 2020.

[23] M. A. Ahmad, E. A. T. Rivera, P. M. Murray, E. M. Carly, P. M. Anita, and A. Teredesai, “Machine learning approaches for patient state prediction in pediatric icus,” in *2021 IEEE 9th International Conference on Healthcare Informatics (ICHI)*, pp. 422–426, 2021.

[24] K. Moharm, M. Eltahan, and E. Elsaadany, “Wind speed forecast using lstm and bi-lstm algorithms over gabal el-zayt wind farm,” in *2020 International Conference on Smart Grids and Energy Systems (SGES)*, pp. 922–927, 2020.

[25] K. PENG, W. BAI, and L. WU, “Passenger flow forecast of railway station based on improved lstm,” in *2020 2nd International Conference on Advances in Computer Technology, Information Science and Communications (CTISC)*, pp. 166–170, 2020.

[26] N. D. Lewis, *Deep Time Series Forecasting with Python: An Intuitive Introduction to Deep Learning for Applied Time Series Modeling.* Paperback, 2016.

[27] J. Brownlee, *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[28] V. A. Shterev, N. S. Metchkarski, and K. A. Koparanov, "Time series prediction with neural networks: a review," in *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pp. 1–4, 2022.

[29] M. M. Panda, S. N. Panda, and P. K. Pattnaik, "Exchange rate prediction using ann and deep learning methodologies: A systematic review," in *2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN)*, pp. 86–90, 2020.

[30] A. Jaisswal and A. Naik, "Effect of hyperparameters on backpropagation," in *2021 IEEE Pune Section International Conference (PuneCon)*, pp. 1–5, 2021.

[31] N. Gorgolis, I. Hatzilygeroudis, Z. Istenes, and L. □. G. Gyenne, "Hyperparameter optimization of lstm network models through genetic algorithm," in *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–4, 2019.

[32] B. Nakisa, M. N. Rastgoo, A. Rakotonirainy, F. Maire, and V. Chandran, "Long short term memory hyperparameter optimization for a neural network based emotion recognition framework," *IEEE Access*, vol. 6, pp. 49325–49338, 2018.

[33] S. Hwangbo, S. I. Kim, U. Cho, Y.-S. Song, and T. Park, "Identification of hyperparameters with high effects on performance of deep neural networks: application to clinicopathological data of ovarian cancer," in *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 1982–1987, 2019.

[34] B. Cha, Y. Cha, S. An, E. Jeon, S. Park, and J. Kim, "Experimental design for multi-task deep learning toward intelligence augmented visual ai," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1735–1737, 2021.

[35] D. L. Chester, "Why two hidden layers are better than one," in *1990 International Joint Conference on Neural Networks (IJCNN)*, pp. 265–268, 1990.

[36] A. Rose and M. Grotjahn, "Lstm based time-series prediction for optimal scheduling in the foundry industry," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022.

[37] M. H. Essai Ali, A. B. Abdel-Raman, and E. A. Badry, "Developing novel activation functions based deep learning lstm for classification," *IEEE Access*, vol. 10, pp. 97259–97275, 2022.

[38] J. Pomerat, A. Segev, and R. Datta, "On neural network activation functions and optimizers in relation to polynomial regression," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 6183–6185, 2019.

[39] S. Kavitha, N. Sanjana, K. Yogajeeva, and S. Sathyavathi, "Speech emotion recognition using different activation function," in *2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, pp. 1–5, 2021.

[40] Z. Chang, Y. Zhang, and W. Chen, "Effective adam-optimized lstm neural network for electricity price forecasting," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 245–248, 2018.

[41] R. Llugsi, S. E. Yacoubi, A. Fontaine, and P. Lupera, "Comparison between adam, adamax and adam w optimizers to implement a weather forecast based on neural networks for the andean city of quito," in *2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM)*, pp. 1–6, 2021.

[42] S. Y. ŞEN and N. ÖZKURT, "Convolutional neural network hyperparameter tuning with adam optimizer for ecg classification," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pp. 1–6, 2020.