CONTROL AND SYSTEM IDENTIFICATION OF LEGGED LOCOMOTION WITH RECURRENT NEURAL NETWORKS

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Bahadır Çatalbaş

June 2022

CONTROL AND SYSTEM IDENTIFICATION OF LEGGED LOCOMOTION WITH RECURRENT NEURAL NETWORKS
By Bahadır Çatalbaş
June 2022

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Ömer Morgül(Advisor)

Süleyman Serdar Kozat

Selim Aksoy

Mehmet Kemal Leblebicioğlu

İsmail Uyanık

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

CONTROL AND SYSTEM IDENTIFICATION OF LEGGED LOCOMOTION WITH RECURRENT NEURAL NETWORKS

Bahadır Çatalbaş
Ph.D. in Electrical and Electronics Engineering
Advisor: Ömer Morgül
June 2022

In recent years, robotic systems have gained massive popularity in the industry, military, and daily use for various purposes, thanks to advancements in artificial intelligence and control theory. As an exciting sub-branch of robotics with their differences and opportunities, legged robots have the potential to diversify and spread the use of robotic systems to new fields. Especially, legged locomotion is a desirable ability for mechanical systems where agile mobility and a wide range of motions are required to fulfill the designated task. On the other hand, unlike wheeled robots, legged robot platforms have a hybrid dynamical structure consisting of the flight and contact phases of the legs. Since the hybrid dynamical structure and nonlinear dynamics in the robot model make it challenging to apply control and perform system identification for them, various methods are proposed to solve these problems in the literature. This thesis focuses on developing new neural network-based techniques to apply control and system identification to legged locomotion so that robotic platforms can be designed to move efficiently as animal counterparts do in nature.

In the first part of this thesis, we present our works on neural network-based controller development and evaluation studies for bipedal locomotion. In detail, neural controllers, in which long short-term memory (LSTM) type of neuron models are employed at recurrent layers, are utilized in the feedback and feedforward paths. Supervised learning data sets are produced using a biped robot platform controlled by a central pattern generator to train these neural networks. Then, the ability of the neural networks to perform stable gait by controlling the robot platform is assessed under various ground conditions in the simulation environment. After that, the stable walking generation capacity of the neural networks and the central pattern generators are compared with each other. It is shown that the proposed neural networks are more successful gait controllers than the central pattern generator, which is employed to generate data sets used in training.

In the second part, we present our studies on the end-to-end usage of neural networks in system identification for bipedal locomotion. To this end, supervised learning data sets are produced using a biped robot model controlled by a central pattern generator. After that, neural networks are trained under series-parallel and parallel system identification schemes to approximate the input-output relations of the biped robot model. In detail, different neural models and neural network architectures are trained and tested in an end-to-end manner. Among neuron models, LeakyReLU and LSTM are found as the most suitable feedforward and recurrent neuron types for system identification, respectively. Moreover, neural network architecture consisting of recurrent and feedforward layers is found to be efficient in terms of learnable parameter numbers for system identification of the biped robot model.

The last part discusses the results obtained in the control and system identification studies using neural networks. In the light of acquired results, neural networks with recurrent layers can apply control and systems identification in an end-to-end manner. Finally, the thesis is completed by discussing possible future research directions with the obtained results.

Keywords: Robot Locomotion Control, Legged Locomotion, Biped Robot, System Identification, Central Pattern Generator, Machine Learning, Deep Learning, Recurrent Neural Networks, Long Short-Term Memory.

ÖZET

TEKRARLAYAN SİNİR AĞLARI İLE BACAKLI LOKOMOSYONUN KONTROLÜ VE SİSTEM TANIMLANMASI

Bahadır Çatalbaş
Elektrik ve Elektronik Mühendisliği, Doktora
Tez Danışmanı: Ömer Morgül
Haziran 2022

Son yıllarda, yapay zeka ve kontrol teorisindeki gelişmeler sayesinde robotik sistemler endüstride, askeriyede ve çeşitli amaçlarla günlük kullanımda büyük bir popülerlik kazanmıştır. Farklılıkları ve fırsatlarıyla robotiğin heyecan verici bir alt dalı olan bacaklı robotlar, robotik sistemlerin kullanımını çeşitlendirme ve yeni alanlara yayma potansiyeline sahiptir. Özellikle bacaklı hareket, belirlenen görevi yerine getirmek için çevik hareketliliğin ve geniş bir hareket yelpazesinin gerekli olduğu mekanik sistemler için arzu edilen bir yetenektir. Öte yandan, tekerlekli robotlardan farklı olarak bacaklı robot platformları, bacakların uçuş ve temas aşamalarından oluşan hibrit dinamik bir yapıya sahiptir. Robot modelindeki hibrit dinamik yapı ve doğrusal olmayan dinamikler, onlar için kontrol uygulamayı ve sistem tanımlamayı yapmayı zorlaştırdığından, literatürde bu problemlerin çözümü için çeşitli yöntemler önerilmiştir. Bu tez, robotik platformların doğada hayvan benzerlerinin yaptığı gibi verimli hareket edecek şekilde tasarlanabilmesi için bacaklı harekete kontrol ve sistem tanımlaması uygulamak için yeni sinir ağı tabanlı teknikler geliştirmeye odaklanmaktadır.

Bu tezin ilk bölümünde, iki ayaklı hareket için sinir ağı tabanlı denetleyici geliştirme ve değerlendirme çalışmalarımızı sunuyoruz. Ayrıntılı olarak, tekrarlayan katmanlarda uzun kısa süreli bellek (LSTM) tipi sinir modellerinin kullanıldığı sinirsel kontrolcüler, geri besleme ve ileri besleme yollarında kullanılmaktadır. Denetimli öğrenme veri kümeleri, bu sinir ağlarını eğitmek için bir merkezi örüntü üreteci tarafından kontrol edilen iki ayaklı bir robot platformu kullanılarak üretilmektedir. Daha sonra yapay sinir ağlarının robot platformunu kontrol ederek stabil yürüyüş yapabilme kabiliyeti simülasyon ortamında çeşitli zemin koşulları altında değerlendirilmektedir. Ardından sinir ağlarının ve merkezi örüntü üreteçlerinin kararlı yürüme üretme kapasitesi birbirleriyle karşılaştırılmaktadır. Önerilen sinir ağlarının, eğitimde kullanılan veri kümelerini

oluşturmak için kullanılan merkezi örüntü üretecinden daha başarılı yürüyüş kontrolcüleri olduğu gösterilmiştir.

İkinci bölümde, iki ayaklı hareket için sistem tanımlamasında sinir ağlarının uçtan uca kullanımına yönelik çalışmalarımızı sunuyoruz. Bu amaçla, bir merkezi örüntü üreteci tarafından kontrol edilen iki ayaklı bir robot modeli kullanılarak denetimli öğrenme veri setleri üretilir. Bundan sonra, sinir ağları, iki ayaklı robot modelinin girdi-çıktı ilişkilerini yaklaşık olarak tahmin etmek için seri-paralel ve paralel sistem tanımlama şemaları altında eğitilmiştir. Ayrıntılı olarak, farklı sinir modelleri ve sinir ağı mimarileri uçtan uca eğitilip, test edilmektedir. Sinir modelleri arasında, LeakyReLU ve LSTM, sistem tanımlaması için sırasıyla en uygun ileri beslemeli ve tekrarlayan nöron türleri olarak bulunmuştur. Ayrıca, tekrarlayan ve ileri beslemeli katmanlardan oluşan sinir ağı mimarisinin, iki ayaklı robot modelinin sistem tanımlaması için öğrenilebilir parametre sayısı açısından verimli olduğu bulunmuştur.

Son bölümde, sinir ağları kullanılarak yapılan kontrol ve sistem tanımlama çalışmalarında elde edilen sonuçlar tartışılmaktadır. Elde edilen sonuçlar ışığında, tekrarlayan katmanlara sahip sinir ağları, uçtan uca kontrol ve sistem tanımlaması uygulayabilmektedir. Son olarak, elde edilen sonuçlarla gelecekteki olası araştırma yönleri tartışılarak tez tamamlanmaktır.

Anahtar sözcükler: Robot Lokomosyon Kontrolü, Bacaklı Lokomosyon, İki Ayaklı Robot, Sistem Tanımlama, Merkezi Örüntü Üreteci, Makine Öğrenimi, Derin Öğrenme, Tekrarlayan Sinir Ağları, Uzun Kısa Süreli Bellek.

Acknowledgement

First of all, I would like to express my deepest gratitude to my supervisor Prof. Dr. Ömer Morgül, for his precious guidance, patience, encouragement, and continuous support throughout my university life. He was always there to listen, give advice, and encourage me to grow my imagination and scientific curiosity. He also gave me the opportunity of joining one of his research projects which is supported by TÜBİTAK, as a member of his research group. The work in this project contributed to the expansion of the scope of my thesis.

I would also like to thank Prof. Dr. Hitay Özbay for triggering my enthusiasm for control theory education. I am also grateful to Asst. Prof. Dr. Melih Çakmakçı for his support of my thesis. Also, I want to appreciate the understanding of my colleague assistants and professors while working as a teaching assistant for several lectures. In addition, I would like to thank all my instructors throughout my education life, from primary school up to the university lecture halls.

I am indebted to the distinguished members of my thesis jury Prof. Dr. Süleyman Serdar Kozat, Prof. Dr. Selim Aksoy, Prof. Dr. Mehmet Kemal Leblebicioğlu, and Asst. Prof. Dr. İsmail Uyanık for accepting my work and guiding me up to this point. Without their vision and support, this thesis may not have ended as it is today.

In addition, I am very thankful to the current and former members of our research group members Hasan Hamzaçebi, Mustafa Oğuz Yeğin, Ahmet Safa Öztürk, Hasan Eftun Orhon, Dilan Öztürk Şener, Deniz Kerimoğlu, and Elvan Kuzucu Hıdır for our wonderful times at Bilkent. Specifically, I am grateful to Caner Odabaş for his endless patience, valuable cooperation in courses, our Ph.D. Qualifying Exam studies, and support on my thesis. In the same way, I am thankful to Ali Nail İnal for his numerous help, thoughtful advice, and guidance throughout my whole university life.

I want to give special thanks to my friends Murat Aslan, Onur Berkay Gamgam, Emrah Topal, Ahmet Dündar Sezer, Serkan Sarıtaş, and Ali Alp Akyol for listening to me and motivating my academic work. Another appreciation goes to my previous coworkers Murat Özgül, Berker Karagöz, Melik Hüseyin

Hamidioğulları, Yasemin Demircioğlu, Dilşad Bayram, Mevlüt Hürol Mete, and Mehmet Bozdemir for supporting me both in professional and social sense.

I want to thank Mürüvet Parlakay, Aslı Tosuner, Aydan Gençel, and Armağan Gürsoy for their helps on administrative works and thank Ergün Hırlakoğlu, Onur Bostancı, Metin Kayabaşı, and Ufuk Tufan for their technical support.

I am appreciative of the financial support from the Scientific and Technological Research Council of Turkey (TÜBİTAK) in the scope of 2211 National Ph.D. Scholarship Program. The work presented in this thesis was supported by TÜBİTAK through project 120E104. I gratefully acknowledge the support of NVIDIA Corporation with the donations of Titan Xp and Quadro P6000 GPUs used in this thesis work.

Some of the content presented in this thesis is first published in Journal of Intelligent & Robotic Systems, 104-4, 1-30, 2022 by Springer Nature and reproduced with permission from Springer Nature, see [1].

Finally, I am very thankful to my parents Zehra, Sezai, and brother Burak, for their precious lifelong support, unconditional love, patience, and cooperation. Indeed, it would not have been possible to reach where I am today without their support. Besides being a member of the family, my brother has also greatly supported me throughout my doctorate studies as a colleague. I thank him for the fruitful work we have done together and the visionary academic discussions. I feel fortunate to be part of this family.

To my family, with love and gratitude...

Contents

1 Introduction **1**

1.1 Background and Motivation 2

1.2 Key Contributions 10

1.3 Organization of the Dissertation 12

2 Control of Legged Locomotion with Neural Networks **14**

2.1 Problem Definition 14

2.1.1 Biped Robot Model 15

2.1.2 Central Pattern Generator 17

2.1.3 Neural Network Based Controller Design 18

2.1.3.1 Forward Propagation 23

2.1.3.2 Back-Propagation 25

2.1.3.3 Adam Optimizer 27

2.2 Methodology 29

2.2.1 Data Set Preparation 29

2.2.1.1 Torque Control Data Sets 31

2.2.1.2 Position Control Data Sets 32

2.2.1.3 PID Controller Data Sets 33

2.2.2 Torque Controller Implementation 33

2.2.2.1 Neural Network Architecture Design 34

2.2.3 Position Controller Implementation 37

2.2.3.1 PID Controller Design 38

2.2.4 Neural Network Replacement of PID Controller 40

2.3 Results and Discussion 43

2.3.1 Torque Control Simulations 43

2.3.2 Position Control Simulations 50
2.3.3 PID Replacement Simulations 54
2.3.4 Controller Sensitivity Analysis 57
2.3.5 Joint Torque Limitation Analysis 60
2.3.6 Stability Analysis 63
2.3.7 Discussion 67
2.4 Conclusion 72

3 Identification of Legged Locomotion with Neural Networks 74
3.1 Problem Definition 75
3.2 Methodology 76
3.2.1 System Identification Models 76
3.2.1.1 Parallel System Identification Model 77
3.2.1.2 Series-Parallel System Identification Model 78
3.2.2 Data Set Preparation 79
3.2.2.1 Parallel Model Data Sets 82
3.2.2.2 Series-Parallel Model Data Sets 82
3.2.3 System Identification Neural Network Architecture 83
3.3 Results and Discussion 87
3.3.1 Series-Parallel System Identification Simulations 87
3.3.2 Parallel System Identification Simulations 93
3.3.3 Discussion 97
3.4 Conclusion 99

4 Conclusion and Future Works 101

A Equations of Biped Robot Model 117

List of Figures

1.1 Illustrations of the proposed neural network architecture and control diagrams in the Chapter 2 11

2.1 Representation of biped robot model joints, limbs, torques, limb angles, and interaction with the ground 16

2.2 Demonstration of neural rhythm generator. Weights of interconnections are represented with w_{fe}, w_{rl}, w_{hka} and neural rhythm generator torque multipliers are shown with p_f^h, p_e^h, p_f^k, p_e^k, p_f^a, p_e^a. 19

2.3 Feedforward NN where W_1, W_2, W_3 and W_4 show the weight matrices of layers 20

2.4 Recurrent layer with LSTM neuron cells where z^{-1} denotes one step time delay 22

2.5 Proposed HNN-based controller structure. Inputs and outputs of recurrent layer (LSTM layer) are presented with $\boldsymbol{x}[t]$ and $\boldsymbol{y}[t]$. In a similar way, inputs and outputs of the feedforward layer (regression layer) are shown with $\hat{\boldsymbol{y}}[t]$ and $\boldsymbol{o}[t]$. 23

2.6 LSTM neuron model internal structure 24

2.7 Torque control diagram 34

2.8 Position control diagram with closed-loop PID controller 37

2.9 Position control diagram with closed-loop neural network controller that takes the difference between reference and feedback inputs. 40

2.10 Position control diagram with a closed-loop neural network controller takes the reference and feedback inputs separately. 41

2.11 Walking success change of selected neural controller on data sets during training 45
2.12 Calculated torque output difference between CPG and the selected controller for all patterns in each data set 46
2.13 Successful locomotion generation capability of selected controller and CPG for different ramp angle and speed excitation value combinations in the data sets 48
2.14 Successful walking percentages of the selected controller and CPG for different roughness multipliers 50
2.15 Walking success change of selected neural controller on data sets during training 52
2.16 Calculated limb trajectory difference between CPG controlled robot model and the selected controller output for all patterns in each data set 53
2.17 Successful locomotion generation capability of selected controller and CPG for different ramp angle and speed excitation value combinations in the data sets 55
2.18 Calculated torque output difference between PID and the selected controller for all patterns in each data set 56
2.19 Successful locomotion generation capability of selected controller and CPG for different ramp angle and speed excitation value combinations in the data sets 58
2.20 Limit cycle trajectories of CPG and proposed NNBCs driven biped robot model hip states. Blue and orange lines show (x_2, $\dot{x}_2$) and (x_2, $\dot{x}_1$) limit cycle trajectories, respectively. Note that these limit cycles are generated for 0 degree ramp angle and 6.41 excitation value. All controllers acquire successful locomotion for this ramp angle and excitation value pair. 67

3.1 Parallel system identification model 77
3.2 Series-parallel system identification model 78
3.3 Distribution of successful and unsuccessful walking patterns with respect to speed excitation level and ramp angle value 80

3.4 Distribution of successful walking patterns to the training and testing sets 81
3.5 Neural network architectures employed in the "System Identification Neural Network" block in the system identification models . . 86

List of Tables

2.1 Explanation of terms used in robot model 17
2.2 Data set generation with central pattern generator driven biped robot platform 30
2.3 NNBCs architecture for torque control scenario. Neuron number of layers are given in parentheses. 35
2.4 Neural replacement of PID controller trials 42
2.5 NN driven walking success comparison for torque control scenario 44
2.6 Rough terrain torque control experiments 49
2.7 NN driven walking success comparison for position control scenario 51
2.8 Rough terrain position control experiments 59
2.9 Robot weight increase experiments 61
2.10 Joint torque limitation experiments 63
2.11 Center of gravity (COG) and zero moment point (ZMP) analysis for NN driven walking scenarios 65
2.12 Computed eigenvalues via the limit cycle analysis for success walking patterns obtained by each controller. Note that these eigenvalues are computed for walking 0 degree ramp angle and 6.41 excitation value walking conditions. All controllers acquire successful locomotion for this ramp angle and excitation value pair. 66
2.13 Results of dropout regularization technique implementation on new neural network-based controller training trials 70
2.14 Results of CPG tuning experiments and new neural network-based controller training trials 71

3.1 System identification performance comparisons of neural networks with one hidden layer for the series-parallel model 89
3.2 System identification performance comparisons of neural networks with various numbers of hidden layers for the series-parallel model 92
3.3 Results of dropout and L_2 regularization implementation to the selected system identification neural network for series-parallel model 94
3.4 System identification performance comparisons of neural networks with various numbers of hidden layers for the parallel model . . . 96
3.5 Results of L_2 regularization implementation to the selected system identification neural network for parallel model 97

Chapter 1

Introduction

This dissertation first introduces the novel uses of recurrent neural networks as a controller for the biped robot locomotion in various control diagrams. To this end, the performances of proposed neural controllers are assessed under the different ground conditions in the simulation environment. The rest of this dissertation presents the use of recurrent neural networks in the system identification for legged locomotion. Finally, we conclude the dissertation by discussing obtained results and possible future research directions.

This section starts with a summary of related literature on robot locomotion, control theory, and neural network learning concepts. Moreover, we explain our motivation for employing recurrent neural networks on the control and system identification tasks by criticizing the advantages and drawbacks of summarized methods in the literature. The subsequent section describes the key contributions of the dissertation using recurrent neural networks on the legged locomotion control and system identification problem. Finally, we conclude this chapter by presenting the organization of the dissertation.

1.1 Background and Motivation

Nowadays, robotic solutions have gained massive popularity due to the recent advancements in artificial intelligence and control theory. For this reason, different robotic systems are being developed for various purposes in the industry and military. Legged robots are an exciting sub-branch of robotics with their differences and opportunities. Humanoid robot Atlas showed that legged robots have the capability of agile mobility and wide range of motion with their design close to the humans [2,3]. Unfortunately, their hybrid dynamical structure, which is inevitable for legged locomotion, includes nonlinear components that make it challenging to model and control these platforms, [4, 5]. In the control of single-legged robots, deadbeat type controllers can be applied by dividing the motion into phases, see e.g., [6]. Moreover, different control theory techniques such as proportional-derivative (PD) and model predictive controllers can be used in the control of biped robots, see e.g., [7]. In addition, different dynamic models have been developed for the control and identification of legged robots with nonlinear structures, such as the Spring-Loaded Inverted Pendulum model, see e.g., [8, 9].

The stability of walking behavior is a very complex problem for legged robots due to their highly nonlinear models. Various methods are available in the literature to analyze the stability of walking behavior. Among these, Center of Gravity (COG) method requires that the center of mass of the legged robot be above the support polygon for the stability of walking behavior, [10]. In the Zero Moment Point (ZMP) approach, stability is achieved when the projection of the point where the sum of active forces is equal to zero is over the support polygon, [11]. In addition, Foot Rotation Indicator (FRI) determines stability and level of instability depending on the location of the rotation forces acting on the support foot at the ground, [12]. A more advanced technique, Centroidal Moment Pivot (CMP) method, eliminates the single foot limitation of FRI by analyzing stability depending on the location of the pivot point, which is defined as the point where a parallel line to reaction force passing through the center of mass of the robot intersects with the ground, [13]. Unfortunately, these approaches become more complex when dynamically stable walking with an increasing number of robot

limbs is desired compared to statically stable motion.

There is also diversity in actuation techniques which affect the whole design of the legged robot. As an illustration of these, Ankaralı and Saranlı [14] introduce an energy regulation method via hip torque actuation on a one-legged spring-mass hopper robot. Kerimoğlu et al. [15] utilize series-elastic ankle actuation to the compass gait model and analyze the stability of walking behavior for the resulting system. Spröwitz et al. [16] propose an open-loop motion control for a quadruped Cheetah robot by utilizing appropriate knee and hip joint actuation. These varieties of actuation also require different controller design methods. In addition to the classical controllers, there are also different legged locomotion control approaches, such as central pattern generators and neural networks.

The inherent nonlinearity and the resulting complexity make it difficult to utilize exact analytic methods for the analysis and control of legged robots. As a remedy to this problem, Sproewitz et al. [17] proposed a central pattern generator (CPG) based controller implementation in multi-legged robotic systems. Likewise, Crespi and Ijspeert [18] presented a CPG-based control method for a snake robot with a high degree of freedom. CPGs, which mimic the reflex spring in the spinal cord, offer a control approach that does not rely on exact solutions and approximation-based methods for gait control of legged robots [19–24]. CPG parameter tuning is generally performed with manual methods, which can be thought of as a supervised learning procedure. During this process, continuity of oscillation and phase difference between joint angles are carefully set by the designer. For this reason, the tuning procedure may become complicated and prolonged with the increasing number of limbs. In addition to supervised methods, Ijspeert and Kodjabachian [25], Nakamura et al. [26], showed that non-supervised learning techniques such as evolutionary algorithms and reinforcement learning can be used to find the parameters of CPGs. There are various oscillatory models which could be utilized in the construction of CPG models, see e.g., [18]. Such a well-known model is given by the Matsuoka oscillator, which is employed to generate rhythmic patterns, and the output of the oscillator can be easily modified with tonic inputs, [27]. Due to these and some other properties, it is preferred in many CPG-controlled biped robot systems, see e.g., [28–31].

CPGs need limited parameter space to create stable walking in the high degree of freedom robots. Unfortunately, this requirement limits the possible range of motions of these systems to periodic trajectories. On the other hand, CPGs may require manual fine-tuning of their parameters due to their structures. In addition to this, it may not be possible to determine a stable movement range due to the inclusion of the robot plant dynamic equations into the equations of CPG. Various neural network-based controllers (NNBCs) have also been developed to avoid some of these disadvantages, see e.g., [32].

Neural networks (NNs), which have found vast application areas today as an essential field of the machine learning theory, are inspired by biological neuron cells, see e.g., [33]. Basically, NNs consist of interconnected simplified neural cell models, which are building blocks of the network. There is a wide range of neuron cell models from as simple as perceptron to complex ones such as leaky integrator and long short-term memory (LSTM), [34–36]. Some of the NN architectures can perform better in different problems. Feedforward NNs are successfully applied to classification and object detection types of problems, see e.g., [37–39]. In detail, Krizhevsky et al. [37], Redmon and Farhadi [39] showed successful utilization of convolutional neural networks (CNNs) in object classification and detection problems, respectively. There are successful implementations of recurrent neural networks (RNNs) in natural language processing and next character or word prediction in the text problems, see e.g., [40, 41]. At the same time, RNN-based motion controllers can provide biped robot gait control solutions, which do not have the disadvantages of CPG, such as limited motion range and manual tuning requirement, in cases where other analytical-based exact and approximated solutions are not desired or cannot be applied, see e.g., [42, 43]. Unlike the CPG parameter tuning case, there are well-defined methods to adjust the parameters of NNs, which are generally called learning algorithms. Various learning algorithms, such as supervised, semi-supervised or unsupervised, could be utilized in the training of NNs, see [34,44]. Among these, supervised learning algorithms are frequently utilized for training NNs when input and output data sets are available, see e.g., [33]. In order to achieve this, different optimizers are developed

and utilized, such as Adam, AdamW, and SinAdaMax, see e.g., [45–48]. In addition, weight initialization techniques are proposed to enhance the performance of neural networks, such as Glorot and He initializers with various probability distributions see e.g., [49–51].

Recently, spiking neuron models which are employed in neuromorphic engineering studies are successfully applied to robot control problems to generate CPGs and NNs. Rostro-Gonzalez et al. [52] proposed a CPG system based on spiking neurons which are trained with Simplex method for hexapod robot locomotion such as walking, jogging, and running gait types. This work focused on generating digital hardware-compatible locomotion controllers with high computational efficiency. In the proposed controller, a different connection topology is required to change gait type, and it was performed by changing synaptic weight matrices. Jaramillo-Avila et al. [53] benefited from a spiking neural network (SNN) to process sensorial information from address event representation-based camera images to decide the movement direction of biped robot. In [53], weights of the proposed SNN were found experimentally. Guerra-Hernandez et al. [54] performed real-life experiments on the CPG system based on spiking neurons in biped, quadruped, and hexapod robots. They benefited from grammatical evolution and Victor-Purpura distance-based fitness function to tune connection weights. In this controller, the gait transition can be performed by resetting all neuron membrane potentials and then setting the initial states of spike trains for desired gait type. Gutierrez-Galan et al. [55] proposed a combination of three spiking central pattern generators (SCPGs) to realize online gait type changes with reasonable transition time by using the limited number of spiking neurons. In the proposed structure, while selected SCPG begins to generate spike train, previous SCPG is inhibited quickly. With the developed controller, the locomotion of an arthropod-like robot was successfully controlled in real-time. Even though each SCPG has a low number of neurons, the combined system requires high redundancy to change between gait types, and also, neuron weights were manually adjusted in this study. In addition, the sensorial information was not taken into account, likewise aforementioned SCPG-based locomotion controllers.

In these successful studies, the spiking neuron model is utilized to control locomotion at different abstraction levels. Generally, sensorial information, which can be beneficial, especially in inclined or rough terrain, was not processed in SCPG-based locomotion controllers. This design selection may be related to the limited processing or learning capability of the spiking neuron model compared to LSTM type recurrent neuron models and the necessity of parameter tuning to extract information from sensorial inputs. Recurrent neural networks have well-defined learning techniques, and they may accomplish switching between different gait types by using the same network weights via suitable training set pattern selections. For more information on CPGs and their applications, the reader is referred to e.g., [20, 56], and the references therein.

Researchers seek optimum combinations of different control techniques and learning algorithms to benefit from the powerful sides of each of them in the legged locomotion control problem. Auddy et al. [57] suggest a novel hierarchical control mechanism consisting of a CPG and a feedforward neural network (FNN) as low and high-level controllers for bipedal locomotion, respectively. In [57], CPG was modulated by FNN to correct walking direction by using only two connections. In detail, a combination of manual tuning and genetic algorithm was utilized to find the parameters of CPG first. After basic walking behavior is acquired via a tuned low-level controller, parameters of FNN are found by using deep reinforcement learning via further walking simulation experiments. In this control scheme, lateral deviations from a straight direction were corrected with two gains, which were the outputs of FNN, multiplied by sagittal hip oscillator CPG outputs. Mandava and Vundavilli [58] benefit from a newly proposed modified chaotic invasive weed optimization (MCIWO) and well-known particle swarm optimization (PSO) algorithms to find weights of a feedforward neural network that is responsible for adaptively tuning gains of torque-based proportional-integral-derivative (PID) controller during the bipedal locomotion. First, ZMP and inverse kinematics concepts were utilized to find dynamically balanced walking gaits. Later, neural networks are trained with MCIWO, PSO, and the traditional steepest descent algorithm to estimate the best PID gains to track walking gaits. After that, the performances of neural networks are tested

in the simulation environment. Finally, this study employed selected neural network weights to estimate PID gains of a real biped robot walking on ascending and descending slope surfaces between [-5, 5] degree intervals. To sum up, these recent studies needed to follow multiple parameter tuning stages because the proposed controllers consisted of a combination of different types of systems such as PID, CPG, and FNN. Unfortunately, these design selections may bring their own disadvantages, such as controller design complexity and training difficulty. One reason for the requirement of controller combination may be related to the absence of the memory property of employed feedforward neural networks. On the other hand, the LSTM neuron model has the ability to solve long-term relationships with its memory property, and it is resistant to vanishing gradient problems. For these reasons, LSTM recurrent layer has some advantages compared to feedforward layers which are utilized in these two studies, and its usage may enhance the abilities of proposed controllers in [57] and [58].

In a similar way, well-known Wiener and Hammerstein system identification models employ a combination of a feedforward neural network and polynomial to represent plant dynamics, see e.g., [59]. Since there is no memory term in feedforward neural networks, a polynomial block accompanies the feedforward neural network block to add memory property to the system identification scheme in a similar manner to CPG in the hierarchical control mechanisms and integrator term in PID controllers. From this perspective, it is seen that feedforward neural network blocks found usage in both control and system identification literature. In our previous study, we compared the performance and parameter efficiency of feedforward and recurrent neural networks for a CPG-controlled biped robot system identification problem in an end-to-end manner, [60]. Here, the series-parallel system identification scheme was used when comparing neural networks of various depths to facilitate the memory requirement and provide a fair comparison. As a result, the recurrent neural network with one regression layer and one LSTM layer reached the lowest system identification error compared to all feedforward network architectures. In this way, the potential of LSTM layers has been investigated in the legged locomotion identification problem, and it has been found that their end-to-end usage is possible and efficient for the bipedal locomotion

identification problem. In this context, the study in [60] requires further research to find an optimum combination of recurrent and feedforward layers. In addition to this, the effect of the type of nonlinearities in the neuron model should be investigated for bipedal locomotion system identification. The results given in [60] may also make sense for the legged locomotion control problem because legged locomotion includes hybrid dynamics, so tracking the walking phase changes is required for high-performance locomotion control. For this reason, the usage of recurrent neural networks with LSTM layers needs to be investigated extensively in the biped locomotion control problem.

In this dissertation, we propose utilizing neural network-based controllers consisting of feedforward and recurrent neuron layers as torque and position controllers for biped robot locomotion. The proposed NNBC can behave like a meta-learner compared to the CPG type controller. In detail, it may generalize the input and output relations of the CPG controllers under suitable training conditions. We note that we utilized CPGs to generate joint angle and torque data in this work to train the NNBCs in achieving successful walking or biped robots. Naturally, if similar data could be generated or obtained by other means, our approach could also be utilized using such data. Simulation or computer animation tools are natural candidates to generate such data, and recently their usage in robotics and related areas has become an important research area [61]. Such an approach could be utilized to generate appropriate joint angle or torque values [62], as well as in controlling robots [63]. Since we focus on CPG-based data, we do not pursue such an approach in this work. It should be noted that the utilization of such data, along with data obtained by CPG's, is an exciting research problem that deserves further investigation. The literature is rich on this subject, and for further information, see e.g., the following works and the references therein [64–68].

Analytical approaches are frequently employed to sustain stable legged locomotion with classical control techniques in the literature [56]. First, gait type is determined by analytical methods or observation in the path planning stage. Then joint trajectories are derived by benefiting from inverse kinematics equations under selected stability constraints such as COG, ZMP, FRI, CMP, etc.

Later on, required joint torques may be calculated by inverse dynamic equations, if possible, or classical control techniques such as PID see e.g., [69].

Unfortunately, legged robot dynamics include highly nonlinear terms, and they show hybrid dynamic behavior due to the nature of walking, so the resulting equations contain complex expressions. For these reasons, various simplification methods are proposed to ease the analysis of the robot model and the application of classical control techniques for controlling legged locomotion in the literature. One of them is applied by ignoring extremities with low weight, such as legs in the robot model see e.g., [69–72]. In this way, the analysis of the robot model gets more straightforward while the model accuracy decreases. However, the performance of some control methods may depend on the accuracy of the utilized robot model. In other words, inaccuracies due to simplification may decrease control performance or cause unstable behavior depending on the level of simplification [56]. For instance, ignoring limb weight to ease COM calculations may violate stability criteria such as reported in the [70].

On the other hand, the limited parameter space of classical controllers may not perform well for varying robot dynamic properties or terrain conditions, so various model predictive control (MPC) architectures are proposed in the literature for biped locomotion, see e.g., [72, 73]. In classical control techniques such as PID, there is a limited number of controller parameters that can be tuned to track trajectories for low-level control scenarios. Even though MPC architectures relax this limitation, they may require complex controller designs, and an external gait generator block is added to the control diagram to drive a low-level controller. On the other hand, recurrent neural networks can be utilized as controllers for both low-level and high-level control problems simultaneously with their large tuneable parameter space like in their biological counterparts. Moreover, proposed NNBCs in this study are trained by using the original robot model dynamics with all nonlinear terms. This situation may increase harmony between the controller and the robot model. Beyond these advantages of the proposed NNBCs compared to classical control approaches, utilizing LSTM layers in the neural controllers such as our proposed controllers may make the design of high-performance neural model reference control architectures possible.

1.2 Key Contributions

As the main contribution of this dissertation, we propose NNBCs, which involve feedforward and recurrent layers rather than classical feedforward neural network-based controllers for biped robot locomotion control as exemplified in Figure 1.1(A). Since the biped robot model shows hybrid dynamic behavior due to changes between flight and stance phases for each leg, we expect that the recurrent layer, which consists of LSTM-type neurons, may contribute to tracking these changes and control locomotion successfully.

As a second contribution to support this idea, the proposed hybrid neural controllers are utilized in various combinations placed in the feedback loop and feedforward paths as shown in Figure 1.1(B) and Figure 1.1(C), respectively. Moreover, the latter is employed with a PID controller for gait control.

As a third contribution, we proposed the utilization of two NNBCs, which generate position and torque outputs, in the same controller diagram together, as shown in Figure 1.1(D). We have performed various simulations under varying ground conditions using these neural controllers. The results of these simulations are analyzed within the scope of this dissertation to demonstrate the superiority of proposed NNBCs against their CPG and PID-type controller alternatives.

As another contribution, the generalization abilities of proposed NNBCs are demonstrated. To do this, the effects of hyperparameter selection such as network size, mini-batch size, and L_2 regularization are reported in the generalization performance, depending on the placement of the controller. Our results indicate that the proposed NNBCs perform better in many cases for a wide range of ramp angles, walking speed intervals, and rough terrain environments than their CPG and PID-based counterparts.

As a final contribution, we analyze the performance of recurrent neural networks in system identification for a biped robot model to a broader extent. In detail, we compare the effectiveness of different neural network architectures and

neuron layer types for this purpose under parallel and series-parallel models separately. Our results show that it is possible to reach lower system identification error rates by combining feedforward and recurrent layers.

Figure 1.1: Illustrations of the proposed neural network architecture and control diagrams in the Chapter 2

1.3 Organization of the Dissertation

This dissertation consists of four chapters. The first chapter begins with a brief introduction to the dissertation content. After that, we summarize related literature on the legged locomotion and our motivation for this study. Then, we present the key contributions of the dissertation. Finally, the chapter is finished by explaining the organization of the dissertation.

The second chapter introduces the biped robot model on which control and system identification will be applied using neural networks and the CPG utilized to generate supervised learning data sets. Then, we revisit feedforward and recurrent neural networks with explaining forward and back propagation equations in detail. After that, we describe our methods for preparing data sets and forming neural network architectures. However, designing such NNBCs to control bipedal locomotion requires extensive research on the determination of optimum neural network architecture, neuron layer type, and training procedures such as regularization techniques. For this reason, we propose the use of various types of neural networks in feedback and feedforward paths. Afterward, we train these neural networks and perform extensive experimental studies to measure the performance of proposed NNBCs with each other and CPG. In the end, we discuss the results in detail.

In the third chapter, our studies are presented on the use of recurrent neural networks for the system identification of bipedal locomotion due to the successful results encountered in the previous chapter. First, we define the system identification problem for bipedal locomotion, then explain the data set preparation for this study. Later on, parallel and series-parallel system identification models are introduced, and candidate neural network architectures are reported. After that, proposed neural networks are trained to converge to the input-output relation of the biped robot model. Afterward, trained neural networks are tested to measure system identification performance, and neural network architectures are compared. Lastly, the obtained results are discussed here.

In the fourth chapter, we review what has been done in the dissertation and debate the results obtained in the control and system identification studies in Chapters 2 and 3. Finally, the dissertation is completed by determining open problems and possible future research directions according to the obtained results.

Chapter 2

Control of Legged Locomotion with Neural Networks

This chapter focuses on the NNBC design problem for two-legged robot motion control in various terrain conditions. To yield this, we utilize LSTM neurons at the recurrent layers and linear neurons at the feedforward layers in the proposed neural controller architecture. Furthermore, NNBCs, which generate either joint torques or limb angles to achieve stable walking depending on the control scenario, are trained with varying training options. Then, their stable walking performances are evaluated and compared in the simulation environment.

Some of the content in this chapter is first published in Journal of Intelligent & Robotic Systems, 104-4, 1-30, 2022 by Springer Nature and reproduced with permission from Springer Nature, see [1].

2.1 Problem Definition

This section presents the biped robot model, CPG, and the basics of neural networks employed in this study. The biped robot model is utilized in both data set

generation together with CPG and evaluation of proposed NNBCs in the various walking environments. To this end, forward propagation, back-propagation, and weight update concepts are visited for NNBCs that consist of feedforward and recurrent layers.

2.1.1 Biped Robot Model

The two-legged robot used in this study consists of a point mass hip, four rigid limbs, and six joints, as shown in Figure 2.1. Its equations of motion, which are given in the appendix for the sake of completion, can be obtained by using standard methods of mechanics, see e.g., [30] for further information. Moreover, horizontal and vertical friction and ground reaction forces are also modeled with parallel spring and damper between the ground-contacting foot and the ground as seen in Figure 2.1.

The generalized coordinate vector of the biped robot system can be found in a similar way to [74, 75] and it is represented with the 6 degrees of freedom as shown below:

$$\boldsymbol{q} = [x_1, x_2, \theta_1, \theta_2, \theta_3, \theta_4]^T. \tag{2.1}$$

The robot model is controlled with the torques created in the six joints, which are at the hip, knee, and ankle joints on both legs, as follows:

$$\boldsymbol{T} = [T_{r1}, T_{r2}, T_{r3}, T_{r4}, T_{r5}, T_{r6}]^T. \tag{2.2}$$

In the single support phase, the ankle torque of the foot which is in the flight phase does not affect robot model dynamics, so the ankle torque of that foot is excluded from the torque vector in (2.2).

The elastic ground contact model, which is defined by parallel spring and damper, permits horizontal and vertical movement of the support feet, such as slipping over the walking surface and sinking through the ground. Thus, the degree of freedom of the robot model does not decrease at single or double support phases, which is different from [74, 75] where the hard ground contact model is

employed. Under these conditions, the biped model becomes a fully-actuated system at the double support phase. However, the removal of ankle torque in the flight phase makes the overall system underactuated in the single support phase. Moreover actuated ankle joint makes it possible to perform static stable walking [76]. In addition to joint torques, ground reaction and frictional forces also affect the robot movement during the stance phase of each leg.

The parameters utilized in the Figure 2.1 are described in Table 2.1. The ramp angle between the walking plane and the ground plane is not shown here in order not to complicate the figure.

Figure 2.1: Representation of biped robot model joints, limbs, torques, limb angles, and interaction with the ground

Table 2.1: Explanation of terms used in robot model

Paramater	Description
x_1, x_2	Horizontal and vertical hip coordinates
$\theta_1, \theta_2, \theta_3, \theta_4$	Limb angles
l_1, l_2	Lower and upper leg limb lengths
m_1, m_2	Mass of upper and lower leg limbs
g	Gravitational acceleration
M	Body mass
$T_{r1}, T_{r2}, T_{r3}, T_{r4}, T_{r5}, T_{r6}$	Torque values and directions
$F_{g1}, F_{g2}, F_{g3}, F_{g4}$	Ground reaction and friction forces
k	Spring coefficient
d	Damping coefficient

2.1.2 Central Pattern Generator

Since we aim to utilize NN structures to control the motion of biped robots, we need meaningful data representing successful walking for the training of our proposed networks. CPGs, which consist of combination of rhythm generators, produce periodic waveforms for joint angles and/or torques to provide stable and periodic locomotion, see e.g., [77]. Taga et al. [30] showed that coupled structure of CPG and biped robot is capable of performing stable locomotion by choosing the parameters of CPG appropriately. Unfortunately, an analytical method to select appropriate parameters is unknown, and the whole tuning should be done manually. We utilize this CPG-driven two-legged robot model to obtain the data sets related to successful walking, see [30]. The utilized CPG equations take feedback from the biped model, which interacts with the walking surface, and calculate the torques applied to robot model joints. In detail, the CPG model utilized in this work consists of one Matsuoka oscillator per joint; hence the entire structure contains six such oscillators, as demonstrated in Figure 2.2. Moreover, each oscillator includes one primary and one supplementary leaky integrator neuron. Accordingly, the CPG structure contains 12 coupled leaky integrator models

as given below ($i = 1, ..., 12$):

$$\tau_i \dot{u}_i = - u_i + \sum_{i,j=1}^{12} w_{ij} y_j - \beta v_i + u_0 + Feed_i(\boldsymbol{x}, \dot{\boldsymbol{x}}), \tag{2.3}$$

$$\tau_i' \dot{v}_i = - v_i + y_i, \tag{2.4}$$

$$y_i = max(0, u_i), \tag{2.5}$$

where variables u_i, w_{ij}, y_j, v_i, u_0, $Feed_i$, τ_i and τ_i' correspond to internal state of ith neuron, coupling coefficient from jth to ith neuron, output of jth neuron, self-inhibition of the ith neuron, speed excitation level input, feedback from the biped platform to ith neuron where $(\boldsymbol{x}, \dot{\boldsymbol{x}})$ are the biped internal states, and time constants, respectively. Robot model feedback connections are taken into CPG model with $Feed_i(\boldsymbol{x}, \dot{\boldsymbol{x}})$ function which is detailed in the Appendix. The outputs y_i of CPG are utilized to generate the joint torque inputs for the biped robot as

$$T_{r1} = p_e^h y_2 - p_f^h y_1, \; T_{r2} = p_e^h y_4 - p_f^h y_3, \tag{2.6}$$

$$T_{r3} = p_e^k y_6 - p_f^k y_5, \; T_{r4} = p_e^k y_8 - p_f^k y_7, \tag{2.7}$$

$$T_{r5} = (p_e^a y_{10} - p_f^a y_9) max(0, F_{g2}), \tag{2.8}$$

$$T_{r6} = (p_e^a y_{12} - p_f^a y_{11}) max(0, F_{g4}), \tag{2.9}$$

where p_f^h, p_e^h, p_f^k, p_e^k, p_f^a, and p_e^a coefficients correspond to torque multipliers which are again manually hand-tuned to generate stable locomotion. The speed excitation value u_0 is a tonic input that has an amplifying effect on the oscillation amplitude of CPG as given in (2.3)-(2.5). Increase in oscillation amplitude results in higher joints torques which cause higher acceleration of limb movements as shown in (2.6)-(2.9). Hence, the CPG driven robot model walking speed increases with increasing speed excitation value. For further information, see [30].

2.1.3 Neural Network Based Controller Design

The use of NN as a controller, which is currently a widely investigated research topic, allows the addition of nonlinear plant dynamics that are ignored or linearized because of the limited parameter space of classical controllers during the

Figure 2.2: Demonstration of neural rhythm generator. Weights of interconnections are represented with w_{fe}, w_{rl}, w_{hka} and neural rhythm generator torque multipliers are shown with p_f^h, p_e^h, p_f^k, p_e^k, p_f^a, p_e^a.

controller design process, see e.g., [78]. With this motivation, we propose hybrid neural network (HNN) structures consisting of feedforward and recurrent layers as a controller for biped locomotion control.

The generic form of a multi-layer feedforward NN is demonstrated in Figure 2.3. Typically inputs of a layer are multiplied with weights, and their summation is passed through a nonlinear activation function to obtain the outputs of this layer. This process is repeated till the output is calculated.

Figure 2.3: Feedforward NN where W_1, W_2, W_3 and W_4 show the weight matrices of layers

Let us denote the input vector as $\boldsymbol{x}$ and the output vector as $\boldsymbol{o}$. Their relation may be given symbolically as:

$$\boldsymbol{o} = F(\boldsymbol{W}, \boldsymbol{x}), \tag{2.10}$$

where F is a nonlinear function, and $\boldsymbol{W}$ symbolically represents the weights of NN. Let $(\boldsymbol{x}, \boldsymbol{d})$ be an input/output pair in our training set and $\boldsymbol{E}$ be any meaningful cost function that measures the closeness of $\boldsymbol{d}$ and $\boldsymbol{o}$. The basic problem can be recast as an optimization problem as given below:

$$\boldsymbol{W}_{opt} = \min_{\boldsymbol{W}} \boldsymbol{E}. \tag{2.11}$$

This optimization can be achieved by using the well-known back-propagation algorithm, see [34].

Figure 2.4 shows the generic structure of a LSTM recurrent layer where neurons have recurrent connections from other neurons in the same layer and also external input connections. In this case, each LSTM cell has an output $\boldsymbol{y}[t]$ as well as an internal state $\boldsymbol{c}[t]$. Symbolically, if $\boldsymbol{W}$ represents the weights of the RNN shown in Figure 2.4, the input-output relation can be formally represented as:

$$\boldsymbol{y}[t] = F(\boldsymbol{W}, \boldsymbol{y}[t-1], \boldsymbol{x}[t]), \tag{2.12}$$

where F is a nonlinear function that depends on the activation functions utilized in the LSTM cell. Now assume that $(\boldsymbol{x}[t], \boldsymbol{d}[t])$ is an input/output sequence in the training set. The problem is to choose appropriate weights $\boldsymbol{W}$ such that when the input sequence is $\boldsymbol{x}[t]$, the output sequence $\boldsymbol{y}[t]$ is sufficiently close to the desired output sequence $\boldsymbol{d}[t]$. Similar to (2.11), this could also be recast as an optimization problem, and this optimization can be achieved by using the well-known back-propagation through time algorithm, see [79].

A classical way of solving the optimization problem given by (2.11) is to utilize the well-known gradient descent approach. In this case, the weights are symbolically updated as follows:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \alpha \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_k}, \tag{2.13}$$

where the term $\partial \boldsymbol{E}/\partial \boldsymbol{W}_k$ is called the error gradient, and $\alpha > 0$ is the learning coefficient. It can be shown that under certain conditions, the iterations defined by (2.13) solve the optimization problem given by (2.11) provided that $\alpha > 0$ is sufficiently small. This basic approach is utilized in the NN training algorithm with slightly more complex modifications, see e.g., Adam optimization given in the sequel.

In short, the weight update outlined above is achieved in two steps that sequentially follow each other. First, an input pattern from the training set is applied to the NN input, and the outputs of all layers are calculated. This phase is called forward propagation. After this step, the cost (or error) in (2.11) can

Figure 2.4: Recurrent layer with LSTM neuron cells where z^{-1} denotes one step time delay

be evaluated. Following this step, the error gradients starting the output layer are calculated. In this case, it turns out that the error gradients of one layer are quite helpful in evaluating the error gradient of one preceding layer. This phase is called (error) back-propagation. After all error gradients are calculated, the weights are updated as given by (2.13) or slightly more complex methods, and this process is repeated for other input/output pairs in the training set.

2.1.3.1 Forward Propagation

The general structure of NNBCs to be used in this study is presented in Figure 2.5 where the combination of feedback connections from the robot platform and external reference inputs constitute input sequence $\boldsymbol{x}[t]$ which is given to the LSTM layer. Then, the output of the recurrent layer is given to the feedforward layer as input. After that, the feedforward layer performs the linear regression process by taking the weighted average of the inputs, and it constitutes regression output sequence $\boldsymbol{o}[t]$ which is the joint torques and/or limb angles as the output of the NNBC depending on the control scenario. Throughout this work, NNs have been trained with $N = 1000$ steps long patterns starting at $t_0 = 0$, ending at $t_1 = 10$ seconds and sampled with $\Delta t = 0.01$ second time intervals.

Figure 2.5: Proposed HNN-based controller structure. Inputs and outputs of recurrent layer (LSTM layer) are presented with $\boldsymbol{x}[t]$ and $\boldsymbol{y}[t]$. In a similar way, inputs and outputs of the feedforward layer (regression layer) are shown with $\hat{\boldsymbol{y}}[t]$ and $\boldsymbol{o}[t]$.

The details of the LSTM neuron model, whose internal structure is shown in Figure 2.6, consists of four inner gates and two states given between equations (2.14)-(2.19) see e.g. [36, 80].

Input gate:

$$\boldsymbol{i}[t] = \sigma(\boldsymbol{W_i x}[t] + \boldsymbol{U_i y}[t-1] + \boldsymbol{b_i}). \tag{2.14}$$

Forget gate:

$$\boldsymbol{f}[t] = \sigma(\boldsymbol{W_f x}[t] + \boldsymbol{U_f y}[t-1] + \boldsymbol{b_f}). \tag{2.15}$$

Figure 2.6: LSTM neuron model internal structure

Cell candidate gate;

$$\boldsymbol{a}[t] = tanh(\boldsymbol{W_a}\boldsymbol{x}[t] + \boldsymbol{U_a}\boldsymbol{y}[t-1] + \boldsymbol{b_a}). \tag{2.16}$$

Output gate:

$$\boldsymbol{o}[t] = \sigma(\boldsymbol{W_o}\boldsymbol{x}[t] + \boldsymbol{U_o}\boldsymbol{y}[t-1] + \boldsymbol{b_o}). \tag{2.17}$$

In forward propagation, weight matrices $\boldsymbol{W_i}$, $\boldsymbol{W_f}$, $\boldsymbol{W_a}$, $\boldsymbol{W_o}$ are multiplied with LSTM layer input $\boldsymbol{x}[t]$ at time t and added to related gate activations. Likewise, the parameter matrices $\boldsymbol{U_i}$, $\boldsymbol{U_f}$, $\boldsymbol{U_a}$, $\boldsymbol{U_o}$ are multiplied by the hidden state vector of LSTM layer expressed by $\boldsymbol{y}[t-1]$ at time step $t-1$ and added to the gate activations. Then bias terms, expressed as $\boldsymbol{b_i}$, $\boldsymbol{b_f}$, $\boldsymbol{b_a}$, $\boldsymbol{b_o}$, are added to the gate activations. After this, nonlinear activation functions which are hyperbolic tangent ($tanh$) and sigmoid (σ) in LSTM gates are applied to the gate activations [80].

After calculating the values of the internal gates, the update of the cell state and the hidden state is done as shown in Equation (2.18) and (2.19), see e.g. [36, 80].

Cell state:

$$\boldsymbol{c}[t] = \boldsymbol{f}[t] \cdot \boldsymbol{c}[t-1] + \boldsymbol{i}[t] \cdot \boldsymbol{a}[t]. \tag{2.18}$$

Hidden state:

$$\boldsymbol{y}[t] = \boldsymbol{o}[t] \cdot tanh(\boldsymbol{c}[t]). \tag{2.19}$$

Cell state is demonstrated with the horizontal line in Figure 2.6 and information that is carried by cell state can be easily modified with linear interactions or preserved between time steps. Moreover, the nonexistence of nonlinear operation in the cell state line helps to avoid gradient vanishing type problems which may be seen in RNN training, see [81]. The output of the LSTM cell is denoted with the hidden state, and it is expressed by $\boldsymbol{y}[t]$ at time step t. In our proposed network architecture, the hidden state $\boldsymbol{y}[t]$ is given as input to the subsequent feedforward regression layer.

In this study, the feedforward layer at the output of the network is used as a regression layer. The parameter matrix of the feedforward layer neurons is expressed by the variable $\boldsymbol{W_{fnn}}$. Feedforward layer inputs are chosen as an expanded version of outputs of the previous recurrent layer. Torque/angle output of the neural controller $\boldsymbol{o}[t]$ at the time t is calculated as:

$$\boldsymbol{o}[t] = \boldsymbol{W_{fnn}}\hat{\boldsymbol{y}}[t], \tag{2.20}$$

where $\hat{\boldsymbol{y}}[t] = [\boldsymbol{y}[t]\ 1]^T$.

2.1.3.2 Back-Propagation

As explained above, the weights of NN are chosen to minimize an appropriate cost function. Assume that $(\boldsymbol{x}[t], \boldsymbol{d}[t])$ is a given input/output pattern in the training set. As mentioned above, we sample the signals with $\Delta t = 0.01$ second-long time steps for 10 seconds. Let $\boldsymbol{o}[t]$ be the time output of NN to the input, again sampled similarly. Then, the error metric to be used in (2.11) is chosen as given below:

$$\boldsymbol{E} = \frac{\Delta t}{2}\sum_{t=1}^{N}(\boldsymbol{o}[t] - \boldsymbol{d}[t])^2. \tag{2.21}$$

where N=1000. Hence, to update any weight $\boldsymbol{W}$ in the NN, we need to compute the error gradients $\partial\boldsymbol{E}/\partial\boldsymbol{W}_k$, see (2.13). To this end, error gradient with respect to regression layer parameters is calculated as given in (2.22) under the half mean square error metric and it is used to update the parameters of $\boldsymbol{W_{ffn}}$ matrix via

Adam optimizer, see [45].

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W_{ffn}}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{W_{ffn}}} = \sum_{t=1}^{N} \Delta t(\boldsymbol{o}[t] - \boldsymbol{d}[t])\hat{\boldsymbol{y}}[t]^T. \tag{2.22}$$

After the error gradients of the last layer weights are computed, we need to relate the cost function with the previous layer weights in the network. To achieve this, the partial derivative of the cost function with respect to the output of the previous layer is computed by using the chain rule. Thus, the error gradient to be used in the update of recurrent layer weight arising from the recurrent layer output ($\boldsymbol{y}[t]$) is calculated as:

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{y}[t]} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{o}[t]} \frac{\partial \boldsymbol{o}[t]}{\partial \boldsymbol{y}[t]} = \boldsymbol{W_{ffn}}^T \Delta t(\boldsymbol{o}[t] - \boldsymbol{d}[t]). \tag{2.23}$$

Back-propagation through time (BPTT) algorithm is an extension of standard back-propagation used in RNNs. The BPTT equations which are used to update the LSTM weights are given between (2.24)-(2.35), see e.g. [82]. Since the neuron model contains four gates, some parameters are similarly processed in equations. Therefore, to simplify the update equations, we first define the following:

$$\boldsymbol{g}[t] = \begin{bmatrix} \boldsymbol{i}[t] \\ \boldsymbol{f}[t] \\ \boldsymbol{a}[t] \\ \boldsymbol{o}[t] \end{bmatrix}, \ \boldsymbol{W} = \begin{bmatrix} \boldsymbol{W_i} \\ \boldsymbol{W_f} \\ \boldsymbol{W_a} \\ \boldsymbol{W_o} \end{bmatrix}, \ \boldsymbol{U} = \begin{bmatrix} \boldsymbol{U_i} \\ \boldsymbol{U_f} \\ \boldsymbol{U_a} \\ \boldsymbol{U_o} \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b_i} \\ \boldsymbol{b_f} \\ \boldsymbol{b_a} \\ \boldsymbol{b_o} \end{bmatrix}. \tag{2.24}$$

The sum of the error gradient at time step t in the recurrent layer is shown as:

$$\delta \boldsymbol{out}[t] = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{y}[t]} + \boldsymbol{\Delta out}[t]. \tag{2.25}$$

Error gradient is composed of two main components. The first component comes from the next layer. In our case, this term is given by (2.23). The second error gradient is related to the BPTT algorithm, which is denoted as $\boldsymbol{\Delta out}[t]$ and is the backpropagated error gradient from the next time step of LSTM.

Gradients of LSTM internal gates are given in (2.26)-(2.30).

$$\boldsymbol{\delta s}[t] = \boldsymbol{\delta out}[t] \cdot \boldsymbol{o}[t] \cdot (1 - tanh^2(\boldsymbol{s}[t])) + \boldsymbol{\delta s}[t+1] \cdot \boldsymbol{f}[t+1], \tag{2.26}$$

$$\boldsymbol{\delta i}[t] = \boldsymbol{\delta s}[t] \cdot \boldsymbol{a}[t] \cdot \boldsymbol{i}[t] \cdot (1 - \boldsymbol{i}[t]), \tag{2.27}$$

$$\boldsymbol{\delta f}[t] = \boldsymbol{\delta s}[t] \cdot \boldsymbol{s}[t-1] \cdot \boldsymbol{f}[t] \cdot (1 - \boldsymbol{f}[t]), \tag{2.28}$$

$$\boldsymbol{\delta a}[t] = \boldsymbol{\delta s}[t] \cdot \boldsymbol{i}[t] \cdot (1 - \boldsymbol{a}[t]^2), \tag{2.29}$$

$$\boldsymbol{\delta o}[t] = \boldsymbol{\delta out}[t] \cdot tanh(\boldsymbol{s}[t]) \cdot \boldsymbol{o}[t] \cdot (1 - \boldsymbol{o}[t]). \tag{2.30}$$

For simplicity, the error gradients given in (2.27)-(2.30) can be collected in a single error gradient vector as follows:

$$\boldsymbol{\delta g}[t] = \begin{bmatrix} \boldsymbol{\delta i}[t] \\ \boldsymbol{\delta f}[t] \\ \boldsymbol{\delta a}[t] \\ \boldsymbol{\delta o}[t] \end{bmatrix}. \tag{2.31}$$

The error gradient, which is backpropagated to the previous LSTM time step, that is used in (2.25), is given below:

$$\boldsymbol{\Delta out}[t-1] = \boldsymbol{U}^T \cdot \boldsymbol{\delta g}[t]. \tag{2.32}$$

Finally, error gradients to update parameters are calculated as follows:

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}} = \sum_{t=1}^{N} \boldsymbol{\delta g}[t] \boldsymbol{x}[t]^T, \tag{2.33}$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{U}} = \sum_{t=1}^{N-1} \boldsymbol{\delta g}[t] \boldsymbol{out}[t-1]^T, \tag{2.34}$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}} = \sum_{t=1}^{N} \boldsymbol{\delta g}[t]. \tag{2.35}$$

For details, see e.g. [82].

2.1.3.3 Adam Optimizer

Adaptive Moment Estimation (Adam) is a first-order algorithm that can be applied to the optimization problems as given by (2.11), see [45]. As the name

suggests, Adam uses first and second moment estimates to determine the individual adaptive learning rates for the parameters to be optimized, see (2.13).

In this study, modified Adam optimizer without a bias correction term is employed to update NN parameters, as given in equations (2.36)-(2.39), see [45]. The variables $\boldsymbol{X}$ used in equations (2.36), (2.37) and (2.39) represent any of parameter matrices/vectors in feedforward ($\boldsymbol{W_{ffn}}$) and recurrent ($\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$) NN layers. $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X_k}}$ is the error gradient to be used to update the matrix/vector $\boldsymbol{X}$ and k subscript means the error gradient of kth training pattern.

At the end of an epoch, calculated error gradients for each training set pattern are utilized to optimize parameters. Initially, $\boldsymbol{M}_0$ and $\boldsymbol{R}_0$ first and second moment estimation matrices which have the same dimension with the parameter to be updated are set to zero. After that, moment estimates and parameters are updated for each training set pattern. Between (2.36)-(2.39), all operations on vectors are performed in element-wise.

$$\boldsymbol{M}_k = \beta_1 \boldsymbol{M}_{k-1} + (1-\beta_1)\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X_k}}, \tag{2.36}$$

$$\boldsymbol{R}_k = \beta_2 \boldsymbol{R}_{k-1} + (1-\beta_2)(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X_k}} \odot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X_k}}), \tag{2.37}$$

$$\alpha_k = \alpha \frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k}, \tag{2.38}$$

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k - \alpha_k \frac{\boldsymbol{M}_k}{\sqrt{\boldsymbol{R}_k} + \epsilon} - \alpha_k \frac{L_2}{n}\boldsymbol{X}_k. \tag{2.39}$$

where β_1, β_2, α and ϵ are various constants which controls the first, second order moments, learning constant and offset in denominator, respectively, and are chosen in our work as $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\alpha = 0.001$ and $\epsilon = 10^{-8}$.

The variables $\boldsymbol{X}_k$, $\boldsymbol{X}_{k+1}$, L_2, n correspond to the weight matrix to be updated, the post-update weight matrix, the L2-regularization parameter, and the number of training set patterns, respectively. The effective learning coefficient α_k is reduced in the later stages of training, as shown in Equation (2.38).

Note that the last term in (2.39) is due to the addition of L2-regularization term $(L_2 X_k^T X_k)/(2n)$ to (2.21); indeed the last term in (2.39) is the gradient of this term. Also note that (2.39) is a sophisticated version of (2.13). Indeed, if we set $\beta_1 = 0, \beta_2 = 1$, then (2.39) became the same as (2.13).

2.2 Methodology

In this section, we explain three different scenarios generated to evaluate the stable locomotion generation performance of NNBCs. To this end, training, validation, and testing data sets are generated for each scenario. Then, NNs are trained to produce desired output data for the input data of patterns in the training set. Lastly, the most successful NN architecture is determined for the locomotion control problem.

2.2.1 Data Set Preparation

The two-legged robot driven by the CPG with the manual tuning of its parameters as in [30] is used for different speed excitation values and ramp angles to generate discrete data sets to be used in the training, validation, and testing of NNs. In the scope of this work, ramp angle and speed excitation values are chosen as constants during walking simulations. Here, the ramp angle value determines the slope of the walking environment, and the speed excitation value affects walking velocity by amplifying oscillations of CPG. Due to the limitations of the CPG model, the robot model is not expected to achieve stable locomotion for each ramp angle and speed excitation value combination. Therefore, CPG-driven movements need to be classified as successful and unsuccessful gait patterns. To achieve this, CPG-driven locomotion patterns are examined by an expert, and motion patterns that have continuity are selected as successful locomotion patterns. Then, important metrics are determined to automate the detection of walking success, such as 0.6 m/sec minimum average walking velocity, 0.6 m minimum hip height, 0.2 m maximum jumping height, and ankle positions. After that, these metrics are utilized

in both data set generation, and NN controlled walking success evaluation.

Training, validation, and testing data sets are separately produced using different ranges of ramp angles and speed excitation values to prevent over-fitting problems. In this context, 40 uniformly distributed speed excitation values in the range of [3 10] and 31 uniformly distributed ramp angles in the range of [-7 7] degrees are selected to form the training data set. A CPG controller drives the robot model for 1240 different combinations of these ramp angles and speed excitation levels. As a result, 398 of these 1240 combinations are determined as stable walking patterns. Similarly, for the validation data set, 30 different speed excitation values in the range of [2 11] and 30 different ramp angles in the range of [-8 8] degrees are run together. Before the robot model is driven for 900 different combinations of these ramp angle and speed excitation levels by the CPG controller, conflicting ramp angle and speed excitation value combinations with the previous 1240 combinations are eliminated from the trial set. Hence, independence between data sets is preserved. Under these conditions, 195 stable walking patterns are obtained for the validation data set. Finally, in order to construct a test data set, 31 different speed excitation values in the range of [1 12] and 30 different ramp slopes in the range of [-9 9] degrees are selected. Conflicting combinations with previous sets are eliminated again. Thus, 148 stable walking patterns which are different from the previous data sets were obtained on the 930 configurations. According to successful locomotion criteria, the CPG-driven biped robot model becomes successful at 32.1% of the training set combinations, 21.67% of the validation set combinations, and 15.91% of the test set combinations. Details are summarized in Table 2.2.

Table 2.2: Data set generation with central pattern generator driven biped robot platform

	Excitation Value Interval	**Ramp Angle Interval**	**Experimented Configuration Number**	**Successful Walking Number**	**Successful Walking Percentage**
Training	[3 10]	$[-7^\circ\ 7^\circ]$	1240	398	32.1%
Validation	[2 11]	$[-8^\circ\ 8^\circ]$	900	195	21.67%
Testing	[1 12]	$[-9^\circ\ 9^\circ]$	930	148	15.91%

To speed up the training of the NNBCs, which are described in the following subsections, produced data set input and output patterns are passed through decimation operation, which is a signal processing technique for decreasing sample points in sequences without causing aliasing distortion. Inputs and outputs of the CPG-driven biped robot model which is simulated at a rate of 10 KHz for 10 seconds long are filtered with an equiripple low pass filter which has 100 Hz passband cutoff frequency. Then, downsampling operation at a rate of 100 to 1 is applied to filtered data. In this way, the decimation operation is completed without losing important information. After that, these locomotion data sets are re-scaled to fit the [-0.5 0.5] range to facilitate the training process and allow the usage of different neuron activation functions. Thus, robot model input and output data which are re-scaled and re-sampled at 100 Hz are produced to generate various supervised learning data sets.

Within the scope of this work, the performance of NNBCs is evaluated in three scenarios. In the first scenario, the joint torque generation capability of the neural controller is evaluated in a closed-loop system where the neural controller takes feedback connections from the robot model and speed excitation input directly. In the second scenario, the reference limb angle producing ability of neural controller structure is examined with respect to ramp angle and speed excitation value inputs. In this scenario, proposed limb angles are tracked with a PID controller. In the third scenario, the neural controller replacement of the PID controller is performed, and the joint torque generation capability of the neural controller is evaluated by taking reference and actual limb angles. To this end, three different input-output data sets have been produced from the re-scaled and re-sampled locomotion data sets, as detailed in the following subsections.

2.2.1.1 Torque Control Data Sets

In the first scenario, NNs are assigned to generate torque patterns according to speed excitation value input and feedback inputs taken from the biped robot platform, as listed in detail below:

- 6 Output data sequence [Dimensions: 6x1000]
  - 6 Joint torques: T_{ri} where $i = 1, ..., 6$
- 32 Input data sequence [Dimensions: 32x1000]
  - 12 Feedback connections: $Feed_i(\boldsymbol{x}, \dot{\boldsymbol{x}})$ where $i = 1, ..., 12$
  - 4 Limb angle states: θ_i where $i = 1, ..., 4$
  - 14 Velocity states: $\dot{x}_i$ where $i = 1, ..., 14$
  - 1 Speed excitation level
  - 1 Bias term

2.2.1.2 Position Control Data Sets

In the second scenario, NNs are assigned to generate limb angle patterns according to speed excitation value and ramp angle inputs, as given in detail below:

- 4 Output data sequence [Dimensions: 4x1000]
  - 6 Limb angles: θ_i where $i = 1, ..., 4$
- 3 Input data sequence [Dimensions: 3x1000]
  - 1 Speed excitation value
  - 1 Ramp angle in degree
  - 1 Bias term

Note that, unlike the torque control scenario, NN takes ramp angle information as an input because the neural controller needs to run without feedback connections from the robot platform, so it does not have the opportunity to observe the walking environment.

2.2.1.3 PID Controller Data Sets

In the third scenario, NNs are assigned to generate torque patterns according to reference limb angles and limb angle feedback taken from the biped robot, as shown in detail below:

- 6 Output data sequence [Dimensions: 6x1000]
  - 6 Joint torques: T_{ri} where $i = 1, ..., 6$
- 8 Input data sequence [Dimensions: 8x1000]
  - 4 Reference limb angle states: θ_i^{nn} where $i = 1, ..., 4$
  - 4 Limb angle states: θ_i where $i = 1, ..., 4$

Different from two previous data sets, joint torques are outputs of the PID controller tuned to track CPG-driven robot limb angles in a closed-loop system. Details of the PID controller are given in the related subsection.

2.2.2 Torque Controller Implementation

In the first control scenario, closed-loop NNBCs are assigned to generate joint torque values. Note that in this scenario, the neural controller receives (external) speed excitation level and (internal feedback) biped robot states signals as its inputs, as shown in Figure 2.7. In this scenario, NNs are trained via torque control data set extracted from the CPG-driven biped robot model walking data as explained previously. We intuitively expect that taking feedback directly from the biped robot and indirectly from the environment should increase the stability of locomotion and robustness against external disturbances. Thus, larger action space may be obtained while stability is preserved due to the generalization ability of NNBCs.

Figure 2.7: Torque control diagram

2.2.2.1 Neural Network Architecture Design

NN architecture has key importance in control performance. For this reason, different network architectures are trained and tested to find the most appropriate architecture for the torque control problem. Since the torque control requires continuous output in an interval, the regression layer is added to the end of the NN architecture, as shown in Figure 2.5. Moreover, the robot model has a hybrid dynamic structure due to changes between flight and stance phases for each leg. Ankle joint torque does not affect system dynamics while a foot is not in contact with the ground in the flight phase. However, ankle joint torque, ground reaction, and friction forces affect system dynamics while the foot is in contact with the ground in the stance phase. Therefore, these walking phase changes must be tracked for high-performance control. For this purpose, at least one recurrent layer is decided to be added to NN architecture. By considering these points, five different NN architectures in which different numbers of feedforward and recurrent layer with various neuron numbers are used to find the most appropriate NNBC architecture as reported in Table 2.3. In addition to these, the feedforward layer between LSTM layers is also used to diversify the tested architectures. Finally, all networks are ended with six neurons, including the feedforward regression layer.

Table 2.3: NNBCs architecture for torque control scenario. Neuron number of layers are given in parentheses.

	Config. #1	**Config. #2**	**Config. #3**	**Config. #4**	**Config. #5**
Learning constant	0.001	0.001	0.001	0.001	0.001
L_2 constant	0	0	0	0	0
Mini-batch size	1	20	20	20	20
LSTM layer number	1	2	2	2	1
Feedforward layer number	1	1	2	2	1
Feedforward layer activation	Linear	Linear	Linear	Linear	Linear
Regression neuron number	6	6	6	6	6
Neural network structure					
Epoch number	20000	20000	18000	18000	20000
Training Set Walking Success	26.85%	30.16%	24.27%	21.05%	26.21%
Validation Set Walking Success	27.56%	29.78%	19.11%	13.44%	17.44%
Testing Set Walking Success	14.73%	11.08%	6.88%	7.74%	12.9%
Training Set Approximation Error	$5.89 \cdot 10^{-4}\ Nm$	$5.06 \cdot 10^{-4}\ Nm$	$5.43 \cdot 10^{-4}\ Nm$	$5.1 \cdot 10^{-4}\ Nm$	$5.15 \cdot 10^{-4}\ Nm$
Validation Set Approximation Error	$1.02 \cdot 10^{-3}\ Nm$	$1.05 \cdot 10^{-3}\ Nm$	$1.13 \cdot 10^{-3}\ Nm$	$1.2 \cdot 10^{-3}\ Nm$	$8.89 \cdot 10^{-4}\ Nm$
Testing Set Approximation Error	$1.02 \cdot 10^{-3}\ Nm$	$8.61 \cdot 10^{-4}\ Nm$	$8.75 \cdot 10^{-4}\ Nm$	$9.17 \cdot 10^{-4}\ Nm$	$8.84 \cdot 10^{-4}\ Nm$

After training, two different tests are applied to measure the competence of these hybrid networks.

In the first test, trained HNN structures are tested to determine their stable locomotion controlling capability on training, validation, and testing set configurations in the simulation environment. In detail, neural controllers are tested for all configurations used in forming data sets independent of the CPG success. In these simulations, the bias term, excitation input values, and feedback connections taken from the robot model are given to NN as input. Then, the output of the neural controller, which consists of six torque values, is given to the robot model. To avoid over-fitting, neural network weights that perform the highest validation set walking success percentages throughout the training process are chosen, and they are utilized to find walking success percentages in other data sets. After the simulations, robot model movements are evaluated with successful walking criteria, which are determined in data set generation. Then, results are reported in Table 2.3. Hence, we aim to observe whether the neural controller generates stable locomotion for the cases where CPG is unsuccessful. In the second test, input sequences in torque control training, validation, and test data sets are given to the network, and outputs are compared with the desired output sequence. The mean squared error metric is applied to the difference and reported as approximation error in Table 2.3. Unlike the first test, the second test is applied only for patterns where CPG successfully generates stable walking. In contrast, the first test is applied for each configuration of data sets regardless of CPG walking success.

The smallest NN architecture consists of one recurrent layer with 50 neurons and one regression layer with six neurons in Configuration #1. Configuration #2 includes two recurrent layers different from Configuration #1. While a high number of recurrent layer utilization is beneficial for decreasing training set approximation error, it also reduces the test set walking success percentage. Feedforward layer addition between recurrent layers is tested with Configuration #3 and #4. A small number of intermediate feedforward neuron utilization generally becomes more successful than a high number of neuron utilization. Finally, a high number of neuron utilization in the single recurrent layer is examined in

Configuration #5. It is interesting to see that approximation error and walking success percentage metrics do not show high correlation in data sets. Although Configuration #1 has the highest testing set approximation error, it has the highest testing set walking success rate. Moreover, Configuration #1 can be trained faster than other controller networks because of its small network size. In this context, Configuration #1 is selected as the most appropriate neural controller architecture due to its high testing set walking success rate and small network size. Therefore, this controller architecture is utilized in the continuation of this study.

2.2.3 Position Controller Implementation

In the second control scenario, NNBCs are assigned to produce limb angles while taking speed excitation and ramp angle external inputs. Produced limb angles are tracked by a second closed-loop controller, as shown in Figure 2.8. In this scenario, NNs are trained via position control data sets extracted from CPG-driven biped robot walking data, as explained in the previous related subsection. Since the NN architecture Configuration #1 in Table 2.3 is successfully employed in torque control scenario which requires relating 32 inputs with 6 outputs, it is reasonable to assume that the same configuration should also perform sufficiently well for position control scenario which includes only 3 inputs and 4 outputs.

Figure 2.8: Position control diagram with closed-loop PID controller

2.2.3.1 PID Controller Design

In the position control simulations, reference limb angles ($\theta_i,\ i = 1,2,3,4$) for the limbs of biped robot in Figure 2.1 are produced by the neural controller in feedforward path and joint torques ($T_{ri},\ i = 1,2,...,6$) are calculated with PID controller as shown in Figure 2.8. To achieve this, four limb angle input is associated with six torque output by six PID controllers, which use the same controller parameters, as given in (2.40)-(2.48):

$$e_i[t] = \theta_i^{nn}[t] - \theta_i[t],\ for\ i = 1,2,3,4, \tag{2.40}$$

$$\dot{e}_i[t] = \dot{\theta}_i^{nn}[t] - \dot{\theta}_i[t],\ for\ i = 1,2,3,4, \tag{2.41}$$

$$se_i[t] = \sum_{i=0}^{t} \Delta t(\theta_1^{nn}[i] - \theta_1[i]),\ for\ i = 1,2,3,4, \tag{2.42}$$

$$T_{r1}[t+1] = I_1(K_p e_1[t] + K_i se_1[t] + K_d \dot{e}_1[t]), \tag{2.43}$$

$$T_{r2}[t+1] = I_1(K_p e_2[t] + K_i se_2[t] + K_d \dot{e}_2[t]), \tag{2.44}$$

$$\begin{aligned} T_{r3}[t+1] =& I_1(K_p e_1[t] + K_i se_1[t] + K_d \dot{e}_1[t]) \\ & - I_2(K_p e_3[t] + K_i se_3[t] + K_d \dot{e}_3[t]), \end{aligned} \tag{2.45}$$

$$\begin{aligned} T_{r4}[t+1] =& I_1(K_p e_2[t] + K_i se_2[t] + K_d \dot{e}_2[t]) \\ & - I_2(K_p e_4[t] + K_i se_4[t] + K_d \dot{e}_4[t]), \end{aligned} \tag{2.46}$$

$$T_{r5}[t+1] = I_2(K_p e_3[t] + K_i se_3[t] + K_d \dot{e}_3[t]), \tag{2.47}$$

$$T_{r6}[t+1] = I_2(K_p e_4[t] + K_i se_4[t] + K_d \dot{e}_4[t]), \tag{2.48}$$

where $\theta_i^{nn},\ i = 1,2,3,4$ is the output of neural position controller and $\theta_i,\ i = 1,2,3,4$ is the actual limb angles of right upper leg, left upper leg, right lower leg, left lower leg, respectively. I_1 and I_2 are inertia of lower and upper legs, respectively.

First, it is required to determine suitable performance metrics for tuning controller parameters used in (2.43)-(2.48). In our problem, one intuitive performance metric may be considered as approaching the torque output of CPG ($\boldsymbol{T_r^{CPG}}$) with the PID controllers for each of $n = 398$ training data set patterns

as:

$$
\begin{aligned}
\min_{K_p,K_i,K_d} \quad & \sum_{j=1}^{n}\sum_{t=1}^{N}\frac{\Delta t}{2}(T_1[t]-T_2[t])^2 \\
\text{s.t.} \quad & K_p, K_i, K_d \geq 0 \\
\text{where} \quad & T_1[t] = \boldsymbol{T_r}^{\boldsymbol{PID}(j,K_p,K_i,K_d)}[t] \\
& T_2[t] = \boldsymbol{T_r}^{\boldsymbol{CPG}(j)}[t] \\
& \boldsymbol{T_r^{PID}} : \text{ Joint torques of PID controller} \\
& \boldsymbol{T_r^{CPG}} : \text{ Joint torques of CPG controller.}
\end{aligned} \tag{2.49}
$$

Unfortunately, due to the highly nonlinear structure of biped robot dynamics and CPG, the optimization problem given by (2.49) is not tractable and not efficient as well for such a tuning process. For this reason, analysis of all walking duration of PID controlled driven biped robot is chosen to determine appropriate performance metrics. To achieve this, desired robot limb angles ($\theta_i,\ i = 1,2,3,4$), which exist in position control training data set as output data sequence, are given as reference input to the PID including closed-loop system instead of neural controller outputs ($\theta_i^{nn},\ i = 1,2,3,4$) in Figure 2.8. After the simulation, walking data is classified as successful or not by using criteria such as minimum displacement, maximum jumping height, falling, etc. By using these ideas, we developed a heuristic metric $f_{successs}$ which tries to maximize the successful walking over the training set patterns, and the associated optimization process could be given as:

$$
\begin{aligned}
\max_{K_p,K_i,K_d} \quad & \sum_{j=1}^{n} f_{success}(r, \boldsymbol{\theta}^{(j)}, K_p, K_i, K_d) \\
\text{s.t.} \quad & K_p, K_i, K_d \geq 0 \\
\text{where} \quad & r : \text{ ramp angle} \\
& \boldsymbol{\theta}^{(j)} : [\theta_1^{(j)}, \theta_2^{(j)}, \theta_3^{(j)}, \theta_4^{(j)}].
\end{aligned} \tag{2.50}
$$

In this way, the highest success rate is acquired with $K_p = 2000$, $K_i = 400$, and $K_d = 150$. Therefore these parameters are employed in the PID controllers in the remaining part of the study.

2.2.4 Neural Network Replacement of PID Controller

In the third control scenario, PID controller is replaced with neural controllers as shown in Figures 2.9 and 2.10 as an extension to neural position controller experiments in feedforward path. Thus, it is aimed to track produced limb angles by a neural controller in the feedforward path by a second neural controller, which is in the closed-loop. For this purpose, NN architecture Configuration #1 in Table 2.3 is utilized because of its success in closed-loop torque control problem.

Figures 2.9 and 2.10 show that the proposed closed-loop neural controller configurations take four or eight inputs, respectively. The proposed controller in Figure 2.9 takes the difference of four reference limb angle values ($\boldsymbol{\theta}^{nn}[t]$) from the neural controller in the feedforward path and four biped robot platform limb angles ($\boldsymbol{\theta}[t-1]$) at time t as input. This classical input scheme is same with the input scheme of PID controller. Unlike this, the proposed controller in Figure 2.10 takes four reference limb angle values ($\boldsymbol{\theta}^{nn}[t]$) from the neural controller in the feedforward path and four biped robot platform limb angles ($\boldsymbol{\theta}[t-1]$) at time t as input, separately. Then, both of the proposed neural controllers calculates six torque outputs ($\boldsymbol{T_r}[t]$) at time t.

Figure 2.9: Position control diagram with closed-loop neural network controller that takes the difference between reference and feedback inputs.

The PID controller is utilized in the training set generation of the closed-loop NN over position control training data set patterns. To this end, reference limb angles, instant biped robot limb angles, and PID torque outputs are recorded for training patterns. After that, five different neural network configurations

Figure 2.10: Position control diagram with a closed-loop neural network controller takes the reference and feedback inputs separately.

are trained as shown in Table 2.4. Among these, LSTM layer or feedforward layers with hyperbolic tangent activation functions are utilized at hidden layers. When feedforward neurons are utilized in hidden layers, neuron numbers are chosen to give a similar number of tunable neural network weights to LSTM layer utilization. Adam optimizer is utilized in training with a technique similar to the early stopping method.

Under these conditions, the lowest error rate was obtained with a combination of $L_2 = 0$, 50 LSTM neurons, and eight separate inputs configuration. This NN is utilized in position control experiments, and results are reported in the later subsection. It is interesting to see that, different from the torque control problem nonzero L_2 regularization constant resulted in higher training set error and lower walking control success than zero L_2 training configurations. This situation may be related to the inability to learn training set patterns with L_2 regularization. Moreover, neural network configuration with eight inputs (Config. #2) in Figure 2.10 reaches higher performance than neural network configuration with four inputs (Config. #1) in Figure 2.9 as reported in Table 2.4. For this reason, the neural network configuration with eight inputs is employed in the remaining studies on the neural network replacement of the PID controller.

Table 2.4: Neural replacement of PID controller trials

	Config. #1	**Config. #2**	**Config. #3**	**Config. #4**	**Config. #5**
Learning constant	0.001	0.001	0.001	0.001	0.001
L_2 constant	0	0	0.05	0	0
Mini-batch size	199	199	199	199	199
Hidden layer number	1	1	1	2	4
Neuron number at each hidden layer	50	50	50	103	61
Hidden layer neuron type	LSTM	LSTM	LSTM	FNN (tanh)	FNN (tanh)
Neural network input number	4	8	8	8	8
Regression layer neuron number	6	6	6	6	6
Epoch number	27000	51000	60000	15000	10000
Training set walking success percentage	64.6%	75.4%	0%	39.27%	31.85%
Validation set walking success percentage	64.78%	68%	0%	35.44%	33.22%
Testing set walking success percentage	46.34%	55.48%	0%	30.22%	22.37%
Training set approximation error (Nm)	$5.33 \cdot 10^{-5}$	$5.17 \cdot 10^{-5}$	$3.7 \cdot 10^{-4}$	$5.2 \cdot 10^{-4}$	$5.2 \cdot 10^{-4}$
Validation set approximation error (Nm)	$5.57 \cdot 10^{-5}$	$5.47 \cdot 10^{-5}$	$3.76 \cdot 10^{-4}$	$5.26 \cdot 10^{-4}$	$5.26 \cdot 10^{-4}$
Testing set approximation error (Nm)	$5.3 \cdot 10^{-5}$	$5.21 \cdot 10^{-5}$	$3.59 \cdot 10^{-4}$	$5.1 \cdot 10^{-4}$	$5.11 \cdot 10^{-4}$

2.3 Results and Discussion

In this section, proposed NNBCs' stable locomotion controlling capacity with a biped robot platform is evaluated under varying ground conditions and various metrics. Then, NNBCs are compared with each other and the CPG controller, which is utilized in training set generation. Finally, the simulation results are discussed.

2.3.1 Torque Control Simulations

The generalization capability limits of the proposed NNBC architecture are investigated using different L_2 regularization constants and mini-batch sizes at the training stage. Mini-batch size (M) means the number of patterns utilized to update network weights together. L_2 regularization is a weight decay method that is employed to improve the generalization capability of the network, see e.g., [83]. Employed mini-batch sizes and L_2 regularization constants are given in Table 2.5. NN parameters are recorded at specific epoch intervals during training. After completing the training stage, the recorded network parameters are used to measure the capability of stable gait generation on the training, validation, and testing data set configurations.

Reported success rates in Table 2.5 are obtained by performing walking simulation for each ramp angle and speed excitation value combinations given in Table 2.2. For this reason, reported success rates for each data set need to be compared by CPG values in Table 2.2 while concluding about the superiority of proposed NNBC architectures. To this end, the NN training configurations that achieved higher success than the CPG are marked in bold in Table 2.5. Note that CPG is utilized in obtaining the data set with which the NNs are trained. Clearly, the higher performance of neural controllers compared to CPG is a result of their generalization ability.

The highest validation set walking success percentage is obtained for $L_2 = 0.5$

Table 2.5: NN driven walking success comparison for torque control scenario

	Data Set	**Mini-Batch Size**		
		M=1	M=40	M=199
$L_2 = 0$	Training	29.03%	27.9%	30.4%
	Validation	**24.44%**	14.11%	18.44%
	Testing	11.51%	10.32%	9.68%
$L_2 = 0.0005$	Training	28.06%	-	-
	Validation	19.44%	-	-
	Testing	12.04%	-	-
$L_2 = 0.005$	Training	28.55%	30.73%	24.44%
	Validation	**28.11%**	18.56%	17.67%
	Testing	**17.96%**	15.16%	11.29%
$L_2 = 0.05$	Training	14.76%	**35.65%**	**48.47%**
	Validation	**35%**	13.44%	**22.33%**
	Testing	**24.41%**	15.81%	**23.66%**
$L_2 = 0.5$	Training	4.19%	**44.35%**	**53.06%**
	Validation	10.89%	**30.44%**	**31.22%**
	Testing	2.69%	**25.91%**	**33.55%**

and $M = 199$ configuration. The NN controlled walking success rate changes throughout the training process, as shown in Figure 2.11. One reason for it could be the emergence of the over-fitting problem in the following stages of training. The NN weights that provide the highest validation set walking success rate are chosen from recorded network weights for each training configuration to avoid this problem. Then, selected network weights are employed to reach reported walking success percentages on training, validation, and testing data sets in Table 2.5. Thus, the problem of over-fitting is avoided by applying a technique similar to early stopping.

The training configuration of $L_2 = 0.5$ and $M = 199$ reached the highest success in all data sets at the 4000th epoch. Therefore, the NN which uses these weights is found as the best torque controller, and it is referred to as "selected controller" in the remaining part of the subsection. The mean squared error between torque outputs of CPG and the selected controller is given for each training, validation, and testing data set pattern in Figure 2.12 to examine the performance

Figure 2.11: Walking success change of selected neural controller on data sets during training

of the selected controller. When Figure 2.12 is examined, it is observed that the difference between the two controller outputs is higher for patterns at the outer boundaries of the data sets. It is also observed that the selected controller is more sensitive to the change of ramp angle than the speed excitation value changes. The difference between the two controllers tends to increase for high speed excitation and ramp angle value combinations.

To understand the effect of observed differences between controllers on the locomotion controlling performance, walking tests are performed in the simulation environment for ramp angle and speed excitation configurations shown in Figure 2.13. As can be seen in Figure 2.13, the selected controller has higher success than CPG in all data sets. In detail, the selected controller reaches higher walking success compared to the CPG for downhill ramp angles in the training, validation,

Figure 2.12: Calculated torque output difference between CPG and the selected controller for all patterns in each data set

and test data sets. Similarly, the selected controller shows higher success than CPG for uphill ramp angles in the training and testing data sets. CPG offers higher success for uphill ramp angles than the selected controller in the validation data set. In this context, the increasing difference between the selected controller and the CPG for patterns at the outer boundaries of data sets in Figure 2.12 apparently contributes positively to the walking control performance.

To evaluate the noise rejection performance of the selected controller, the walking success of the selected controller and CPG have been compared under rough terrain conditions. For this purpose, zero mean and unit variance Gaussian distribution is employed in the generation of the random numbers to make the rough terrains used in the simulation environment. Nine different walking surfaces with varying roughness levels have been modeled using the same random numbers to ensure equal conditions between experiments. To achieve this, random numbers are scaled with the multipliers in [0 0.001 0.002 0.005 0.01 0.015 0.02 0.03 0.04] array. Then, scaled random numbers are added to the height of the walking surface at 0.1 meter-wide intervals to obtain rough terrain. After that, manufactured rough terrain models are added to the inclined ground model on which the selected controller and CPG-driven robot move. The variation of the walking success of the robot platform, which is driven by the selected neural controller and CPG, operated under these conditions depending on the level of ground roughness is shown in Figure 2.14 and Table 2.6. For each roughness level, the selected controller reached equal or higher walking success rates than CPG at all data sets, and higher success rates are marked in bold in Table 2.6. As shown in Figure 2.14, both controllers form a monotonically decreasing walking success for roughness multipliers of 0.002 and above. It is seen that the selected neural controller and the CPG cannot perform successful locomotion on any data set for roughness multipliers of 0.04 and above.

Figure 2.13: Successful locomotion generation capability of selected controller and CPG for different ramp angle and speed excitation value combinations in the data sets

Table 2.6: Rough terrain torque control experiments

Roughness Multiplier	**Data Set**	**Configuration**	
		CPG	**NN**
0	Training	32.1%	**53.06%**
	Validation	21.67%	**31.22%**
	Testing	15.91%	**33.55%**
0.001	Training	34.92%	**54.68%**
	Validation	22.56%	**30.78%**
	Testing	17.1%	**31.61%**
0.002	Training	34.84%	**53.31%**
	Validation	23.22%	**29.78%**
	Testing	17.42%	**31.29%**
0.005	Training	30.4%	**48.47%**
	Validation	21.33%	**25.44%**
	Testing	15.38%	**25.81%**
0.01	Training	17.42%	**32.58%**
	Validation	11.67%	**16.67%**
	Testing	8.17%	**17.31%**
0.015	Training	6.37%	**14.6%**
	Validation	4.44%	**8.44%**
	Testing	3.23%	**8.71%**
0.02	Training	2.58%	**6.85%**
	Validation	1.22%	**2.56%**
	Testing	1.4%	**4.95%**
0.03	Training	0.08%	**0.16%**
	Validation	0%	0%
	Testing	0%	**0.43%**
0.04	Training	0%	0%
	Validation	0%	0%
	Testing	0%	0%

Figure 2.14: Successful walking percentages of the selected controller and CPG for different roughness multipliers

2.3.2 Position Control Simulations

The generalization capability limits of the NN controller in the feedforward path architecture are investigated with the utilization of different L_2 regularization constants and mini-batch sizes in the training process. Employed mini-batch sizes and L_2 regularization constants are given in Table 2.7. NN parameters are recorded at specific epoch intervals during training. After completing the training stage, the recorded network parameters are used to measure the successes of stable gait controlling in training, validation, and testing data sets. During locomotion tests, the PID controller is utilized in the closed-loop.

The highest validation set walking success rate is obtained for $L_2 = 0.05$ and $M = 199$ configuration. The NN controlled walking success rate changes through the training process, as shown in Figure 2.15. As compared to Figure 2.11, now we have more oscillatory behavior in the success rate change during training. The

Table 2.7: NN driven walking success comparison for position control scenario

	Data Set	**Mini-Batch Size**	
		M=40	M=199
$L_2 = 0$	Training	**59.76%**	**49.03%**
	Validation	**46%**	**45.56%**
	Testing	**41.61%**	**34.84%**
$L_2 = 0.005$	Training	**62.1%**	**53.15%**
	Validation	**53%**	**44.56%**
	Testing	**37.96%**	**40.75%**
$L_2 = 0.05$	Training	**68.87%**	**71.69%**
	Validation	**56.22%**	**64.67%**
	Testing	**38.39%**	**55.16%**
$L_2 = 0.5$	Training	**39.35%**	**69.84%**
	Validation	**22.67%**	**39.56%**
	Testing	14.62%	**31.08%**

network weights that provide the highest validation set walking success rate are chosen from recorded network weights for each training configuration to avoid the over-fitting possibility. After that, selected network weights are used to obtain walking success percentages on training, validation, and testing data sets given in Table 2.7. In Table 2.7, the NN training configurations that achieved higher success than the CPG are marked in bold. Note that CPG is utilized in obtaining the training data set. Clearly, the better performance of neural controllers as compared to CPG is a result of their generalization ability over all data sets. Therefore, NN that uses weights for configuration of $L_2 = 0.05$ and $M = 199$ at the 37500th epoch is the best position controller, and it is referred to as the "selected controller" in the remaining part of the subsection.

To examine the performance of the selected controller, the mean squared error between limb angle trajectories of the CPG controlled robot model and the selected controller outputs is given for each training, validation, and testing data set pattern in Figure 2.16. When Figure 2.16 is examined, it is observed that the difference between the two controller outputs is higher for patterns at the outer boundaries of the data sets and patterns at the speed excitation values in

Figure 2.15: Walking success change of selected neural controller on data sets during training

the range of [6.5 9]. It is also observed that the selected controller is sensitive to the change in ramp angle and the speed excitation value changes. The difference between the two controllers tends to increase for high speed excitation and ramp angle value combinations.

The selected controller has higher success than CPG in all data sets, as shown in Figure 2.17. In detail, the selected controller reaches higher walking success than the CPG for uphill and downhill ramp angles in all data sets. Similarly, the selected controller also shows higher success than CPG for high and low-speed excitation values in all data sets. It is observed that for some speed excitation values in the range of [6 7.5] selected controller failed, as can be seen in Figure 2.17. When the reasons for this behavior are examined, it is seen that for these

Figure 2.16: Calculated limb trajectory difference between CPG controlled robot model and the selected controller output for all patterns in each data set

speed excitation values, the robot platform shows similar behaviors to jumping instead of walking. This behavior may possibly be due to the PID controller structure used to control the robot platform. This problem may possibly be solved by optimizing the PID controller given in equations between (2.40)-(2.48). Another possible reason for jumping behaviors may be related to observed high mean squared error rates for some speed excitation values in the range of [6.5 8] in Figure 2.16. On the other hand, the increasing difference between selected controller output and CPG-controlled robot limb trajectories for the patterns at the outer boundaries of data sets in Figure 2.16 apparently contributes positively to the walking control performance.

2.3.3 PID Replacement Simulations

In the third control scenario, the PID controller is replaced with a NNBC as an extension to the second scenario. Thus, the selected controller in the feedforward path is utilized together with closed-loop NNBC instead of the PID controller. The training configuration of $L_2 = 0$ and $M = 199$ reached the highest success in the validation set at the 51000th epoch. Therefore, the NN which uses these weights is found as the best controller, and it is referred to as the "selected controller" in the remaining part of the subsection. The mean squared error between torque outputs of PID and the selected controller is given for each training, validation, and testing data set pattern in Figure 2.18 to examine the performance of the selected controller. When Figure 2.18 is examined, it is observed that the difference between the two controllers tends to increase for combinations of high speed excitation and low ramp angle values.

In order to understand the effect of observed differences between controllers on the locomotion controlling performance, walking tests are performed in the simulation environment for ramp angle and speed excitation configurations shown in Figure 2.19. As can be seen in Figure 2.19, the selected controller has higher success than CPG in all data sets. In detail, the selected controller shows higher success than CPG for uphill ramp angles in the training, validation, and testing

Figure 2.17: Successful locomotion generation capability of selected controller and CPG for different ramp angle and speed excitation value combinations in the data sets

Figure 2.18: Calculated torque output difference between PID and the selected controller for all patterns in each data set

data sets. Similarly, the selected controller reaches higher walking success than the CPG for high and low speed excitation values in all data sets.

In order to analyze the robustness of the proposed NNBC, CPG controller, selected controller with PID (NN+PID; Figure 2.8) and selected controller with closed-loop neural controller (NN+NN; Figure 2.10) have been tested on rough terrain conditions which are the same with the torque control rough terrain experiments and results are given in Table 2.8. For each roughness level, the selected controller with configuration in Figure 2.8 and 2.10 reached equal or higher walking success rates than CPG at all data sets. Among two configurations of the selected controller, closed-loop neural controller utilization in Figure 2.10 mostly reaches higher success rates for roughness multiplayer of 0.01 and below. After this level, PID utilization as closed-loop controller in Figure 2.8 generally give better results on data sets. Here, we note that NN is trained with flat terrain data, but nevertheless, it still shows acceptable performance for rough terrain walking. This behavior may also be related to the generalization ability of NNBCs.

When controller performance on position control scenario in Figure 2.8 and 2.10 and closed loop controller performance on torque control scenario in Figure 2.7 are compared with each other, position control structure given in Figure 2.10 performs better than the other configurations. In addition to this, for the high roughness multiplier values position control structure given in Figure 2.10 is less successful compared to PID controller.

2.3.4 Controller Sensitivity Analysis

In most control applications, there are differences between the mathematical model used to design the controller and the actual plant that needs to be controlled, so good controllers should work in a harmonious and resistant way against slight variations in plant dynamics and external disturbance, respectively. This subsection reports sensitivity analysis of proposed NNBCs against changing robot weight in the simulation environment. To this end, it is decided to increase the

Figure 2.19: Successful locomotion generation capability of selected controller and CPG for different ramp angle and speed excitation value combinations in the data sets

Table 2.8: Rough terrain position control experiments

Roughness Multiplier	**Data Set**	**Configuration**		
		CPG	**NN+PID**	**NN+NN**
0	Training	32.1%	71.69%	**75.4%**
	Validation	21.67%	64.67%	**68%**
	Testing	15.91%	55.16%	**55.48%**
0.001	Training	34.92%	71.94%	**75.4%**
	Validation	22.56%	64.33%	**67.89%**
	Testing	17.1%	54.84%	**55.38%**
0.002	Training	34.84%	71.53%	**74.76%**
	Validation	23.22%	64%	**67.33%**
	Testing	17.42%	**54.52%**	54.09%
0.005	Training	30.4%	69.44%	**73.06%**
	Validation	21.33%	61.78%	**65.78%**
	Testing	15.38%	**54.19%**	52.26%
0.01	Training	17.42%	62.98%	**67.26%**
	Validation	11.67%	57.78%	**58.89%**
	Testing	8.17%	**49.78%**	47.1%
0.015	Training	6.37%	56.05%	**58.06%**
	Validation	4.44%	**52%**	48.11%
	Testing	3.23%	**41.94%**	40%
0.02	Training	2.58%	**49.6%**	44.35%
	Validation	1.22%	**41.11%**	37.89%
	Testing	1.4%	**35.91%**	31.94%
0.03	Training	0.08%	**25.24%**	23.31%
	Validation	0%	**22%**	19.67%
	Testing	0%	11.61%	**17.2%**
0.04	Training	0%	0%	0%
	Validation	0%	0%	0%
	Testing	0%	0%	0%

weight of each mass on the robot model and compare the resulting walking success rates for CPG and proposed NNBCs. This analysis can be thought of as another way of checking the existence of over-fitting to robot model dynamics in the simulation environment.

NNBCs are not retrained to satisfy fair comparison with weighted robot models during this analysis. In order to analyze the sensitivity of the proposed NNBCs, CPG controller, selected controller for torque control scenario (NN; Figure 2.7), selected controller with PID for position control scenario (NN+PID; Figure 2.8) and selected controller with closed-loop neural controller for position control scenario (NN+NN; Figure 2.10) have been tested with increasing robot model weight and obtained results are given in Table 2.9. For each data set and robot weight increase level, the highest walking success rates are marked in bold in Table 2.9.

The selected NNBCs reached higher walking success rates than CPG at all data sets for each weight level. Among three configurations of the selected controllers, the selected controller with a closed-loop neural controller for the position control scenario (NN+NN; Figure 2.10) reaches the highest training and validation data set performances. Later, selected controller with PID for position control scenario (NN+PID; Figure 2.8) reaches the highest testing data set performance. Finally, the selected controller for the torque control scenario (NN; Figure 2.7) outperforms CPG for each weight increase percentage, but it reaches lower walking success rates than position controllers, similar to rough terrain experiment results. These results may be related to the generalization ability of NNBCs.

2.3.5 Joint Torque Limitation Analysis

Successful real-life implementations of legged locomotion control algorithms have to work harmoniously with various physical constraints of the robot plant, such as actuator torque and speed limitations. For this reason, a locomotion control algorithm needs to be able to work under some constraints with adequate performance. To this end, we applied torque optimization to proposed NNBCs, and we tested CPG and NNBCs under torque limitation, which is the 25% of the

Table 2.9: Robot weight increase experiments

		Configuration			
Robot Weight Increase	**Data Set**	**CPG**	**Torque Controller (NN)**	**Position Controller (NN+PID)**	**Position Controller (NN+NN)**
0%	Training	32.1%	53.06%	71.69%	**75.4%**
	Validation	21.67%	31.22%	64.67%	**68%**
	Testing	15.91%	33.55%	55.16%	**55.48%**
2%	Training	31.29%	54.11%	71.05%	**75%**
	Validation	21.78%	29.89%	63.56%	**67.67%**
	Testing	15.81%	32.9%	**54.84%**	54.73%
4%	Training	31.13%	54.11%	70.16%	**74.44%**
	Validation	20.67%	28.22%	62.89%	**67%**
	Testing	14.62%	31.29%	53.01%	**53.44%**
5%	Training	30%	53.15%	69.52%	**74.84%**
	Validation	20.89%	27.89%	62.33%	**66.89%**
	Testing	14.84%	31.4%	52.37%	**52.9%**
6%	Training	29.35%	54.44%	68.79%	**74.27%**
	Validation	19.67%	25.22%	61.33%	**67.11%**
	Testing	14.19%	30.86%	**52.58%**	51.94%
8%	Training	27.74%	52.34%	67.74%	**73.63%**
	Validation	18.89%	24%	61.11%	**65.44%**
	Testing	13.76%	28.71%	**50.75%**	49.78%
10%	Training	26.37%	50.73%	67.02%	**72.42%**
	Validation	18%	22.22%	59.44%	**64.22%**
	Testing	13.23%	26.67%	**49.68%**	48.71%
15%	Training	20.4%	43.47%	63.47%	**69.27%**
	Validation	13.56%	16.89%	55.22%	**61%**
	Testing	9.68%	23.23%	**46.02%**	43.66%
20%	Training	14.52%	33.63%	57.66%	**64.19%**
	Validation	10.56%	13.78%	50.56%	**55.11%**
	Testing	7.31%	17.63%	**41.83%**	37.63%
25%	Training	9.03%	27.5%	47.74%	**53.71%**
	Validation	6.11%	9.11%	42.89%	**47.22%**
	Testing	4.41%	14.09%	**35.38%**	30.43%

highest joint torques generated by the CPG controller during the generation of the training data set.

To make a fair comparison, CPG is run for training, validation, and testing set ramp angle and excitation value combinations under the aforementioned torque limitation, and resulting success rates are given in Table 2.10. For the same purpose, the selected controller for the torque control scenario (NN; Figure 2.7) is trained for mini-batch size 199 and various L_2 regularization constants by using the original training set patterns that are truncated with respect to torque limit again. Later on, the selected controller with PID for the position control scenario (NN+PID; Figure 2.8) is optimized by reselecting PID coefficients. To this end, optimization in (2.50) is repeated under torque limitation. In this way, the highest success rate is acquired with $K_p = 4500$, $K_i = 0$, and $K_d = 600$. After that, the selected controller with a closed-loop neural controller for the position control scenario (NN+NN; Figure 2.10) is trained for mini-batch size 199 by using the original training set patterns that are truncated with respect to torque limit again. Finally, optimized NNBCs were tested under torque limitation, and resulting success rates are reported in Table 2.10. The NNBCs that achieved higher success than the CPG are marked in bold in Table 2.10.

Under the 25% joint torque limitation, CPG and selected controller for torque control scenario encounter a decline in gait control success, but the neural controller outperformed the CPG for L_2 regularization constant 0.05 over all data sets. Interestingly, the selected controller with PID for the position control scenario reaches even higher success rates than its unlimited joint torque performances. This success increase may be related to the under-optimized situation of the PID controller. Similar to the torque control scenario, the selected controller with a closed-loop neural controller for the position control scenario encounters a decline in walking control success. Still, the neural controller outperformed the CPG again. Based on this data, it may be concluded that proposed NNBCs outperform CPG for 25% joint torque limitation constraint, and they may work under constraints with various success rates.

Table 2.10: Joint torque limitation experiments

Data Set	Configuration						
	CPG	**Torque Controller (NN)**				**Position Controller (NN+PID)**	**Position Controller (NN+NN)**
		($L_2 = 0$)	($L_2 = 0.005$)	($L_2 = 0.05$)	($L_2 = 0.5$)	($L_2 = 0.05$)	($L_2 = 0$)
Training	26.77%	**27.5%**	**27.18%**	**37.58%**	6.13%	**92.26%**	**51.94%**
Validation	18.78%	13%	**21.67%**	**22.11%**	3.78%	**83.11%**	**49.22%**
Testing	15.59%	7.74%	**21.29%**	**26.67%**	2.04%	**68.39%**	**40.54%**

2.3.6 Stability Analysis

Bipedal locomotion stability analysis is a complex problem due to utilized highly nonlinear robot models and the diversity of gait types. For this reason, various ways of measuring walking stability are proposed in the literature [10–13, 84]. These methods generally impose artificial constraints on walking at various levels, such as static walking. Unfortunately, these constraints may negatively affect the performance of locomotion in terms of speed, efficiency, disturbance rejection, etc. metrics [84].

To avoid these side effects, we classify stable locomotion patterns by measuring average walking velocity, hip height and detecting falling events as explained in Subsection 2.2.1. In this way, the least amount of artificial constraints have been imposed on the walking in this study. Even though center of gravity (COG), zero moment point (ZMP), and limit cycle analyses impose some artificial constraints to gait and some of them cannot be fully applicable because of point feet of the employed robot model in this study, they still may help us to understand the behavioral characteristic of proposed NNBCs. For this reason, we utilize COG, ZMP, and limit cycle analyses in this subsection.

The employed robot model has point feet, so a support polygon does not exist in the single support phase. To make an analysis similar to COG and ZMP, we redefine support polygon as the distance between horizontal projections of feet onto the flat surface independently from ground contact. Even under this assumption, support polygon shrinks to a single point when feet are aligned vertically. With this assumption, we track COG and ZMP throughout walking and calculate the

percentage of walking duration that satisfies COG and ZMP stability criteria. Table 2.11 lists COG and ZMP stability criteria satisfied walking duration percentages for CPG and proposed NNBCs on each data set. It is important to note that these analyses are performed only for successful walking patterns.

In terms of Table 2.11, proposed NNBCs have a lower COG stability criteria satisfying percentage than CPG. Among NNBCs, position controllers satisfy COG stability criteria for a slightly lower percentage than the torque controller. Based on this finding, it can be concluded that NNBCs may perform less statically stable locomotion than CPG, and this situation can be related to the increasing locomotion success of NNBCs. Unexpectedly, ZMP stability criteria are fulfilled for slightly shorter durations than COG, as shown in Table 2.11. When we seek the possible reasons, we determine that ground reaction forces generated at the stance phase of the associated leg include high amplitude fluctuations. This situation causes noise in the second derivatives of state variables utilized in the calculation of ZMP. When trajectories of ZMP are analyzed, it is seen that ZMP oscillates to the back and front of the support polygon and does not diverge from the support polygon permanently. The highest ZMP percentages are observed for torque controller NNBCs, and the lowest percentages are acquired for position controller NNBCs. Unfortunately, noise arising from ground reaction forces prevents further analysis of our control scenarios.

Another method of determining the stability of legged locomotion is the well-known limit cycle analysis. It imposes fewer constraints on the gait compared to COG and ZMP methods, [84]. Thus, more efficient and natural gaits can be classified as stable walking. In detail, trajectories of the state variables in the successive steps generate closed trajectories which are called limit cycles in state space, see e.g., [15, 84].

To analyze the characterization of limit cycles, the Poincaré section which is a subset of system states at step n is taken as follows for our case:

$$\boldsymbol{h}[n] = [x_2[n], \dot{x}_1[n], \dot{x}_2[n]]^T. \tag{2.51}$$

Here, we take Poincaré section when the right thigh is in front of the hip, and

Table 2.11: Center of gravity (COG) and zero moment point (ZMP) analysis for NN driven walking scenarios

	Data Set	**COG**	**ZMP**
CPG	Training	88.23%	47.14%
	Validation	88.02%	47.40%
	Testing	88.45%	47.47%
Torque Controller (NN)	Training	86.77%	50.69%
	Validation	84.38%	52.38%
	Testing	85.41%	50.29%
Position Controller (NN+PID)	Training	80.25%	43.77%
	Validation	83.08%	42.63%
	Testing	76.50%	39.04%
Position Controller (NN+NN)	Training	82.27%	42.00%
	Validation	81.35%	39.99%
	Testing	76.15%	36.62%

the hip is in the highest position during a step. We describe this combination as an apex point. Poincaré sections of successive apex points can be mapped by a stride function '$\boldsymbol{S}(.)$' as shown below:

$$\boldsymbol{h}[n+1] = \boldsymbol{S}(\boldsymbol{h}[n]). \tag{2.52}$$

For a stable periodic motion that converges to a limit cycle, there are fixed points of $\boldsymbol{S}$ function such as $\boldsymbol{h}^*$ and it repeatedly passes from these fixed points between consecutive steps which satisfy the following:

$$\boldsymbol{h}^* = \boldsymbol{S}(\boldsymbol{h}^*). \tag{2.53}$$

Then stability of the limit cycle can be determined by linearizing function $\boldsymbol{S}$ around the fixed point $\boldsymbol{h}^*$ as given below:

$$\boldsymbol{S}(\boldsymbol{h}^* + \boldsymbol{\Delta_h}) = \boldsymbol{h}^* + \boldsymbol{D}\boldsymbol{\Delta_h}, \tag{2.54}$$

where $\boldsymbol{\Delta_h} = [\Delta_{x_2}, \Delta_{\dot{x}_1}, \Delta_{\dot{x}_2}]^T$ is a small deviation vector and $\boldsymbol{D}$ is the Jacobian matrix which consists of partial derivatives of $\boldsymbol{S}$ function with respect to states variables in the Poincaré section. If the eigenvalues of the Jacobian matrix are found within the unit circle, it means the limit cycle is locally stable.

Unfortunately, the highly nonlinear model of the biped robot model and controllers do not let us calculate the Jacobian analytically. Under these conditions, a popular way of evaluating the Jacobian is using numerical methods, see e.g., [15, 84, 85]. The partial derivatives in Jacobian matrix $\boldsymbol{D}$ are calculated with numerical methods as explained in [85]. As an example of this approach, we considered a simple successful walking configuration of 0 degree ramp angle and 6.41 excitation value. In this way, limit cycles of the CPG and NNBCs driven biped robot models for the selected walking patterns, which are shown in the Figure 2.20, are analyzed, and eigenvalues of corresponding Jacobian matrices are reported in Table 2.12. All eigenvalues are found in the unit circle as expected, which is in line with the results of the rough terrain experiments. Moreover, eigenvalues with the lowest amplitude are found in CPG, which may be interpreted as CPG showing faster recovery characteristics against applied perturbation. This observation is also in line with the stability expectations stated in [30], but such a limit cycle calculation was not performed there. Among NNBCs, the torque controller has the highest eigenvalue amplitude, and this is in line with the observation of slow recovery and lower walking success rates than position controller NNBCs.

Table 2.12: Computed eigenvalues via the limit cycle analysis for success walking patterns obtained by each controller. Note that these eigenvalues are computed for walking 0 degree ramp angle and 6.41 excitation value walking conditions. All controllers acquire successful locomotion for this ramp angle and excitation value pair.

	Configuration			
	CPG	**Torque Controller (NN)**	**Position Controller (NN+PID)**	**Position Controller (NN+NN)**
Eigen Values	-0.0954 0.13 0.0502	0.585 -0.0913 -0.0062	-0.3113 0.0857 0.0062	$-0.3790 - 0.2750i$ $-0.3790 + 0.2750i$ 0.0008

Figure 2.20: Limit cycle trajectories of CPG and proposed NNBCs driven biped robot model hip states. Blue and orange lines show (x_2, $\dot{x}_2$) and (x_2, $\dot{x}_1$) limit cycle trajectories, respectively. Note that these limit cycles are generated for 0 degree ramp angle and 6.41 excitation value. All controllers acquire successful locomotion for this ramp angle and excitation value pair.

2.3.7 Discussion

The over-fitting issue is undesired and needs to be avoided to increase the real performance of the machine learning algorithms. To avoid this, three separate data sets named training, validation, and test sets are produced by using CPG driven robot model. While producing these sets, speed excitation and ramp angle values are selected from relatively close range. However, data set patterns are placed far enough from each other to show diversity which can be observed by examining Figure 2.11 and 2.15. In this way, the success rate of neural controllers changes in different directions on different data sets throughout the training process. Moreover, it is very common that neural controllers show changing success rates on each data set for different training configurations as shown in Table 2.5, 2.7 and 2.8. In addition to these, trained neural controllers can perform stable locomotion for speed excitation and ramp angle values that do not exist in any of

these three data sets. Thus, it is validated that generated three data sets carry the necessary properties to avoid the over-fitting problem. Another important issue is the success differences between training, validation, and testing data sets. In the generation of data sets, different intervals of speed excitation and ramp angle values were employed to analyze behaviors of NNBC in larger input ranges. For this reason, successful locomotion percentages show diversity, as explained in the data set preparation subsection. While the highest walking success is obtained in the training data set, the testing data set reaches the lowest success percentage due to enlarging input intervals.

During NNBC-driven biped locomotion experiments, the feedback taken from the robot and environment are given to NNBCs, so inputs diverge from training set inputs starting from the first control output, which is different from CPG output. These dynamics may cause misunderstanding that NN training does not converge throughout the training process, as seen in Figure 2.11. In fact, NNBC successfully learns the relation between input and output data in the training set. Since walking simulations are performed with different inputs from inputs of data sets, first learning and then over-fitting occurs as shown in Figure 2.11.

Even though NNs are highly nonlinear structures, they reach a very high generalization performance in walking experiments. Significantly, L_2 regularization contributed positively to walking success generalization. For instance, 30 of 32 inputs given to the neural controller are calculated during locomotion simulations in the torque control scenario, but NNBC can interpret these previously unseen inputs to produce required torque values for stable walking. Similarly, the neural controller reaches more than two times higher walking success for speed excitation and ramp angle combinations employed in data set generation at the position control experiments. In addition to these, the generalization success of NNs is also validated with rough terrain, robot weight increase, and joint torque limitation experiments. Although neither CPG nor neural controller is trained in rough terrain environments, there are significant walking success differences between them in rough terrain environments. Differences generally tend to increase with increasing roughness. In a similar way, neural controllers outperform CPG in robot weight experiments, and success differences increase with

increasing robot weight. It is interesting to see that proposed NNBCs become more successful in position control scenarios than in torque control scenario. The main reason for this difference may be related to simplifying the legged locomotion problem by separating it into joint torque calculation and limb trajectory planning subparts similar to hierarchical control. Thanks to this separation, one NNBC interpolates limb angles while the other focuses on torque generation for biped locomotion control. After that, the successful walking generation ability of the NNBCs is tested under the torque limitation constraint, and NNBCs outperform the CPG again. Hence, we see that the proposed NNBCs can be optimized to work in small torque intervals, and this ability may ease their usage in real-life biped locomotion control applications.

Remark 1 *Unlike the acquired generalization success with L_2 regularization, Dropout is another well-known regularization technique that does not enhance neural controller performance for every control scenario, see [86]. To measure the performance increment that can be obtained with the dropout method, we retrain torque and position controller neural networks with their best training configurations by adding 0.1 and 0.2 dropout rates to the output of the LSTM layer. In our simulations, dropout does not increase the success rates for torque controller, but some improvement is observed in certain cases for position controller, as shown in 2.13. Obviously, this point deserves further investigation. This situation may be related to limited layer numbers in our proposed neural network architecture. We think that the dropout method would be more beneficial for the performance of neural networks if we utilized a larger number of feedforward layers.* □

Due to the limited parameter space of CPG, it has limited adaptability to biped robot dynamics than NNBC. Even though Matsuoka oscillator is capable of sustaining oscillations, NNBC is more successful in evaluating the feedback taken from the robot model and environment. Moreover, it seems that the LSTM layer in NNBC is capable of tracking phase changes of hybrid dynamics of the biped robot platform thanks to the internal structure of the LSTM neuron model. The information carried by the cell state can be easily modified or preserved with linear interactions between time steps. Moreover, the nonexistence of nonlinear

Table 2.13: Results of dropout regularization technique implementation on new neural network-based controller training trials

	Mini-Batch Size	***L2 Constant***	***Dropout Rate***	***Training Set***	***Validation Set***	***Testing Set***
CPG	-	-	-	*32.1%*	*21.67%*	*15.91%*
Torque	*199*	*0.5*	*0*	*53.06%*	*31.22%*	*33.55%*
Controller	*199*	*0.5*	*0.1*	*31.94%*	*25.33%*	*16.67%*
(NN)	*199*	*0.5*	*0.2*	*37.58%*	*26.11%*	*19.14%*
Position	*199*	*0.05*	*0*	*71.69%*	*64.67%*	*55.16%*
Controller	*199*	*0.05*	*0.1*	*73.79%*	*65.78%*	*52.15%*
(NN+PID)	*199*	*0.05*	*0.2*	*77.1%*	*54.44%*	*57.2%*

operation in the cell state line helps to avoid gradient vanishing type problems which may be seen in RNN training, see [81].

Remark 2 *We perform further tuning on CPG parameters to test our assumption about the dependence of the generalization ability of neural networks on the tuning level of CPGs. Thus, we enlarge the walkable ramp angle range from [-6.53, 6.53] to the [-8.18, 9.55] degree interval for the training set walking patterns. That means an approximately 35.8% increase in the ramp angle range of CPG. A somewhat summary of our simulations by using the patterns generated by the improved CPG parameters are listed in Table 2.14. Here, we train torque controller, position controller, and PID replacement neural networks by using found best training configurations in previous experiments. All of them reach higher success rates than CPG, as expected. As can be seen, although the CPG success rates have increased, so are the NN-based controller performances, which was one of the main results of our work.* □

Feedforward layers are employed in the classical neural network-based controllers due to their quick training ability. However, feedforward layers do not have a dynamic structure, so they may require several previous time step information from the controlled plant in the closed-loop control schemes. In addition, the number of required time steps may increase with nonlinearities in the plant. In our proposed structures, the recurrent layer eliminates the need for previous time step information. We employed fully feedforward neural networks with 3

Table 2.14: Results of CPG tuning experiments and new neural network-based controller training trials

	Excitation Value Interval	***Ramp Angle Interval***	***Improved CPG Walking Percentage***	***Torque Controller Neural Network Walking Percentage (L_2=0.5)***	***Position Controller Neural Network Walking Percentage (L_2=0.05)***	***PID Replacement Neural Network Walking Percentage (L_2=0)***
Training	*[3 10]*	*[−10° 10°]*	*31.22%*	*45.17%*	*55.94%*	*48.5%*
Validation	*[2 11]*	*[−11° 11°]*	*22.58%*	*30.42%*	*46%*	*40.75%*
Testing	*[1 12]*	*[−12° 12°]*	*16.67%*	*27.42%*	*43.01%*	*36.34%*

and 5 layers that have similar parameter counts with our proposed NNBC for position control scenarios as an ablation analysis work to see the importance of recurrent connections in the proposed NNBC, see e.g., [87]. As shown in Table 2.4, fully feedforward neural networks could not succeed in providing successful legged locomotion as much as proposed NNBCs. As a result of this ablation analysis, we see that fully feedforward neural networks are not suitable for the same schemes that we used for the proposed NNBC that includes a LSTM recurrent neural layer. We think that when previous time inputs are added to these feedforward controllers, their success rates will increase, but the number of the required time steps is unknown, and it may increase for hybrid dynamical systems such as biped robots. On the other hand, the inclusion of previous time step information requires placing a higher number of network parameters in the input layer. So, the total neuron count will decrease in the feedforward neural network, and this situation will also limit the performance of the controller. For these reasons, using the LSTM neuron model in the recurrent layer also seems efficient in terms of the number of parameters.

Note that the LSTM neuron model is developed as a memory unit, so its mathematical operation modeling capability is limited. When the internal structure of the LSTM neuron model is examined, it is seen that there is no derivative block to determine the rate of change at the error signal and integrator block to accumulate the error signal. Unlike the PID controller, these mathematical operations have to be performed in-between time steps by collaborating with other LSTM cells at the recurrent layer. To organize this collaboration, significant

numbers of network weight need to be adjusted, and this parameter adjustment may take long training durations. As a remedy to this weakness, the closed-loop control performance of the LSTM cell may be increased with internal structural modifications such as adding integrator and derivative gates.

2.4 Conclusion

In this Chapter, we have focused on the NNBC design problem for two-legged robot motion control. We utilized LSTM neurons at the recurrent layer and linear feedforward neurons at the regression layer in the proposed neural controller architecture. We proposed different neural controller structures which generate either joint torques or limb angles to achieve stable walking. These neural controllers were trained with varying options of training. Then, their stable walking performances were evaluated and compared in a simulation environment.

As the main contribution of this work, we proposed NNBCs with an LSTM recurrent layer instead of classical fully feedforward NNBCs for biped robot locomotion control. We demonstrated that they might achieve stable walking control in various walking environments. Note that biped robots have hybrid dynamics, and as a result, they exhibit different behavior during flight and stance phases for each leg. Since LSTM networks have certain memory, we expect that they might be able to track these changes, and this property may contribute to increasing the stable walking performance. Secondly, proposed hybrid neural controllers were utilized in the feedback loop and feedforward paths to support this idea. In this way, their robot dynamic change tracking ability of the recurrent layer in the NNBC was analyzed depending on controller placement. Thirdly, the performances of proposed controllers were validated in the simulation environment to show the robustness of the proposed structures under varying ground roughness conditions, robot weight changes, and joint torque limitations for position and torque control scenarios. Throughout these simulations, we showed that the proposed NNBCs perform better than CPG and PID type controller alternatives in the legged locomotion control problem. Fourthly, we benefit from well-known

stability analysis methods COG, ZMP, and limit cycle analysis to understand the behavioral characteristic of proposed NNBCs as much as the robot model allows. As a final contribution, the generalization abilities of proposed NNBCs were demonstrated, and training properties that may affect generalization performance were investigated with walking simulations. As a result, L_2 regularization was the most critical factor in the training of networks to reach higher walking success. Mini-batch sizes were determined as less effective but an important factor in generalization. Depending on the control scenario, the contribution of L_2 regularization showed diversity. In the replacement of the PID controller with a NNBC, L_2 regularization affected the performance of the controller negatively as compared to other trained neural controllers. To sum up, proposed NNBCs performed better for a wide range of ramp angles, walking speeds, rough terrain environments, robot weight changes, and joint torque limitations than their counterparts in terms of simulation results.

Finally, the propriety of data set generation, over-fitting issues, and limitations of the LSTM neuron model are discussed in detail. Then, possible improvements are proposed related to the neuron model and training options. The use of the LSTM recurrent layer allows the neural controller detect phase changes between stance and flight without explicitly giving foot contact information to the NN. Also, the stable walking conditions are widened by the generalization ability of RNNs. Thus, the advantages of RNN usage in controlling the hybrid dynamical systems are exemplified by a biped robot platform walking control problem.

Chapter 3

Identification of Legged Locomotion with Neural Networks

This chapter focuses on the neural network design problem for system identification of a two-legged robot in various terrain conditions. We benefit from neural networks consisting of different types of recurrent and feedforward layers to yield this. In detail, we search for efficient neural network architectures in the sense of parameter number with respect to identification estimation error. To this end, supervised training and testing data sets are generated using biped model presented in Subsection 2.1.1 and central pattern generator Subsection 2.1.2 in a similar way presented in Subsection 2.2.1. Furthermore, neural networks are trained under parallel and series-parallel system identification models, and then their estimation performances are compared with each other at the end of this chapter.

3.1 Problem Definition

Legged robot models include nonlinear dynamics due to the nature of the actuation method. Moreover, legged robots also show hybrid dynamical behavior depending on the flight and contact phases of the legs. In other words, the dynamics of the robot model exhibits difference depending on the occurrence of foot contact besides the nonlinear components. Hence, the nature of legged locomotion makes the system identification problems even more difficult for legged robotic systems.

Neuron models, which are building blocks of neural networks, have nonlinearities in the neuron activation functions. From this perspective, neural networks have similarities to nonlinear dynamic systems. For this reason, nonlinearities in neuron activation functions may promise advantages in identifying system dynamics of nonlinear systems, see [59, 60, 88, 89].

To this end, feedforward neural networks are employed in various types of system identification schemes depending on the complexity of the identified model, see [59,88]. Different from feedforward counterparts, recurrent neuron models can store information by either the memory unit inside the neuron model or mutual connection with other recurrent neurons in the same layer through consecutive time steps. This characteristic promises further advantages in identifying hybrid dynamical systems compared to feedforward networks.

Since both hybrid dynamical structure and nonlinear dynamics in the legged robot model make it challenging to apply control and perform system identification, recurrent neural networks also have significant potential in the biped robot model identification. As expected, we show that the recurrent neural layer has advantages over feedforward layer utilization in system identification of legged locomotion [60]. Because as is reported in the NNBC studies in Chapter 2, the LSTM layer is capable of tracking phase changes of hybrid dynamics of the biped robot platform thanks to the internal structure of the LSTM neuron model. Since recurrent neural networks have memory, we expect that they might be able to

track these changes, and this property may contribute to increasing the system identification performance.

With this motivation, this section searches for advantageous neural network architectures, neuron models, and regularization techniques in system identification of the biped robot model presented in Subsection 2.1.1.

3.2 Methodology

In this section, we explain two different system identification models utilized to identify the dynamics of bipedal locomotion. Later on, supervised training and testing data sets are generated for each identification model using this robot model. Then, NNs architectures that will be evaluated are illustrated, and after that, these NNs are trained to produce desired output data for the input data in the training set. Lastly, the most successful NN architectures in each identification model are found for the biped robot locomotion system identification problem.

3.2.1 System Identification Models

In this study, parallel and series-parallel models, which are defined in the following subsections in detail, are employed to perform system identification using neural networks. These two models differ by the series connection taken out from the plant and given to the neural network. This functional connection may increase the performance of the series-parallel model and prevent divergence of neural network predictions compared to the parallel model in time.

In both models, the prediction error is defined by the difference between outputs of identified plant and identification neural network by using a suitable metric such as the mean squared error (MSE) definition. Then, the neural network is trained to minimize this error metric. Hence, it is aimed that the neural

network presents the input and output relation of the plant.

3.2.1.1 Parallel System Identification Model

Parallel system identification model utilized in this study is demonstrated in Figure 3.1. Upper branch includes biped robot model and CPG defined in Subsections 2.1.1, 2.1.2, respectively. Basically robot model is driven by CPG depending on excitation level input $u[t]$ and ramp angle $\phi[t]$ of terrain. In this scheme, $\boldsymbol{y}[t+1]$ denotes extended robot state vector which is combination of robot states $\boldsymbol{x}[t+1]$ and derivatives of robot states $\dot{\boldsymbol{x}}[t+1]$ for the ease of explanation.

Figure 3.1: Parallel system identification model

Throughout the walk, ramp angle $\phi[t]$ and robot model joint torques $\boldsymbol{T_r}[t]$ are given to the system identification neural network (SINN) as inputs, and then succeeding extended robot states $\boldsymbol{y}[t+1]$ are estimated by SINN. After that, the difference between succeeding extended robot states $\boldsymbol{y}[t+1]$ and the output of the SINN $\hat{\boldsymbol{y}}[t+1]$ is denoted as the estimation error $\boldsymbol{e}[t+1]$.

After obtaining the estimation error $\boldsymbol{e}[t+1]$, mean squared error $\boldsymbol{E_j}$ is calculated

for the jth walking pattern as shown in (3.1) to be used in (2.11):

$$\boldsymbol{E}_j = \sum_{t=1}^{N-1} \frac{(\boldsymbol{y}[t+1] - \hat{\boldsymbol{y}}[t+1])^2}{N-1} = \sum_{t=1}^{N-1} \frac{(\boldsymbol{e}[t+1])^2}{N-1} \tag{3.1}$$

Later on, SINN weights are updated with respect to calculated error gradient as thoroughly explained in Subsection 2.1.3. Thus, it is aimed to minimize the estimation error given by (3.1) throughout the training procedure.

3.2.1.2 Series-Parallel System Identification Model

Series-parallel system identification model utilized in this study is demonstrated in Figure 3.2. Basically, the series-parallel model differs from the parallel model with a series connection that conveys delayed extended robot state vector $\boldsymbol{y_{delayed}}[t+1, \boldsymbol{D}]$ for previous time steps to the SINN.

Figure 3.2: Series-parallel system identification model

The delayed extended robot state vector $\boldsymbol{y_{delayed}}[t+1, \boldsymbol{D}]$ for previous time

steps is defined as:

$$\boldsymbol{y_{delayed}}[t+1, \boldsymbol{D}] = \begin{bmatrix} \boldsymbol{y}[t+1-\boldsymbol{D}[1]] \\ \boldsymbol{y}[t+1-\boldsymbol{D}[2]] \\ ... \\ \boldsymbol{y}[t+1-\boldsymbol{D}[M]] \end{bmatrix}, \ \boldsymbol{D} = [1, 2, ..., M], \tag{3.2}$$

where input delay vector $\boldsymbol{D}$, which is the length of M, includes monotonically increasing indexes to be used to get state variables of previous time steps.

Thus, SINN takes extended robot states for previous time steps so that divergence of predictions from actual robot model states is partially prevented with this connection through time. In addition to this, ramp angle $\phi[t]$ and robot model joint torques $\boldsymbol{T_r}[t]$ are also given to the SINN as other inputs, and then succeeding extended robot states $\boldsymbol{y}[t+1]$ are estimated by SINN. Later on, the estimation error is calculated depending on the maximum number of previous time steps as follows:

$$\boldsymbol{E}_j = \sum_{t=D[M]}^{N-1} \frac{(\boldsymbol{y}[t+1] - \hat{\boldsymbol{y}}[t+1])^2}{N - D[M]} = \sum_{t=D[M]}^{N-1} \frac{(\boldsymbol{e}[t+1])^2}{N - D[M]} \tag{3.3}$$

3.2.2 Data Set Preparation

Similar to Subsection 2.2.1, the CPG controls the locomotion of a two-legged robot model with reported parameters at [30] for different speed excitation levels and ramp angles. Thus, the walking data set is produced, and then it is divided into the training and testing sets. In this study, speed excitation levels and ramp angles are selected as constants during walking simulations. After that, CPG-driven locomotion patterns are classified as either successful or unsuccessful gait patterns as explained in Subsection 2.2.1.

In detail, 40 uniformly distributed speed excitation values in the range of [3 10] and 39 uniformly distributed ramp angles in the range of [-9 9] degrees are selected to form the data set. The CPG controller drives the biped robot model for 1560 combinations of these ramp angles and speed excitation levels. As a

result, 393 of these combinations, which corresponds to 25.19% of them as shown in Figure 3.3, are determined as stable walking patterns according to successful locomotion criteria in the Subsection 2.2.1.

Figure 3.3: Distribution of successful and unsuccessful walking patterns with respect to speed excitation level and ramp angle value

To speed up the training of the neural networks, which are described in the Subsection 3.2.3, data set inputs and outputs are passed through decimation operation, which is a signal processing technique for decreasing sample points in sequences without causing aliasing distortion. Inputs and outputs of the CPG-driven biped robot model simulated at a rate of 10 KHz for 10 seconds long are filtered with an equiripple low pass filter with a 100 Hz passband cutoff frequency. Then, a downsampling operation at 100 to 1 is applied to filtered data. Thus, the decimation operation is completed without losing important information. After

that, these locomotion data are re-scaled to fit the [-1/3 1/3] range to facilitate the training process and use different neuron activation functions. Thus, robot model input and output data, which are re-scaled and re-sampled at 100 Hz, are produced to generate various supervised learning data sets. After that, successful walking patterns are divided into two to constitute the training set from 320 patterns and the testing set from 73 patterns with the motivation of preventing the over-fitting problem, as shown in Figure 3.4.

Figure 3.4: Distribution of successful walking patterns to the training and testing sets

Within the scope of this work, the performance of NNs is assessed in two system identification models as explained in Subsection 3.2.1. In summary, one of these models is the series-parallel system identification model. In this model, the system identification neural network takes the ramp angle of the ground, joint torques, state variables, and derivatives of state variables of the biped robot model as input. Then, the capability of the neural network to estimate the subsequent

time step state variables and their derivatives is measured by comparing them with the actual values of the biped robot model. The other model is the parallel system identification model that differs from the previous one by not providing the state variables and derivatives of the bipedal robot model as input to the system identification neural network. In this model, estimation errors are inevitably accumulated through time, so the output of the parallel model may deviate from the biped robot model easier than in the series-parallel model. In this context, two different input-output data sets have been produced from the re-scaled and re-sampled locomotion data sets, as detailed in the previous subsections.

3.2.2.1 Parallel Model Data Sets

In the parallel model, NNs are assigned to estimate next time step state variables and their derivatives of the biped robot model for number of $N-1$ time steps according to ramp angle and joint torques inputs as listed in detail below:

- 28 Output data sequence [Dimensions: 28x(N-1)]
  - 14 States: $x_i[n+1]$ where $i = 1, ..., 14$, $n = 1, ..., N-1$
  - 14 Velocity states: $\dot{x}_i[n+1]$ where $i = 1, ..., 14$, $n = 1, ..., N-1$
- 7 Input data sequence [Dimensions: 7x(N-1)]
  - 6 Joint torques: $T_{ri}[n]$ where $i = 1, ..., 6$, $n = 1, ..., N-1$
  - 1 Ramp angle in degree

3.2.2.2 Series-Parallel Model Data Sets

In the series-parallel model, NNs are assigned to estimate next time step state variables and their derivatives of the biped robot model according to ramp angle, joint torques, state variables, and derivatives of state variables. If the selected time delay step indexes are denoted in the array $\boldsymbol{D}$, which is at the length of M,

state variables and their derivatives of previous time steps will be given to the neural network as listed below. Here, $\boldsymbol{D}[M]$ denotes the maximum number of time delays, and $N - \boldsymbol{D}[M]$ shows the number of time steps.

- 28 Output data sequence [Dimensions: 28x(N-D[M])]
  - 14 States: $x_i[n+1]$ where $i = 1, ..., 14$, $n = \boldsymbol{D}[M], ..., N-1$
  - 14 Velocity states: $\dot{x}_i[n+1]$ where $i = 1, ..., 14$, $n = \boldsymbol{D}[M], ..., N-1$
- (7+28*M) Input data sequence [Dimensions: (7+28*M)x(N-$\boldsymbol{D}$[M])]
  - 14*M States: $x_i[n+1-D[j]]$ where $i = 1, ..., 14$, $n = \boldsymbol{D}[M], ..., N-1$, $j = 1, ..., M$
  - 14*M State derivatives: $\dot{x}_i[n+1-D[j]]$ where $i = 1, ..., 14$, $n = \boldsymbol{D}[M], ..., N-1$, $j = 1, ..., M$
  - 6 Joint torques: $T_{ri}[n+1-D[1]]$ where $i = 1, ..., 6$, $n = \boldsymbol{D}[M], ..., N-1$
  - 1 Ramp angle in degree

3.2.3 System Identification Neural Network Architecture

The system identification neural network block, which exists in both parallel and series-parallel model diagrams, is aimed to imitate the input-output relations of the biped robot model in a way described in Figures 3.1 and 3.2.

The biped robot model is modeled with seven inputs which are six joint torques and one ramp angle, and 28 outputs which are 14 state variables and 14 derivatives of these state variables in this study. Due to the multi-input multi-output (MIMO) model of the biped robot, the SINN block should also have a capable neural network architecture to perform MIMO system identification. For this reason, we decided to use multi-layered neural network architectures with up to 5 hidden layers, as shown in Figure 3.5. Different feedforward and recurrent layer types are utilized in these hidden layers in the scope of this study.

Furthermore, due to the dynamics of the biped robot, states are continuous variables, so the neural network needs a regression layer at the output of the neural network to minimize estimation errors. For this reason, all candidate neural network architectures in Figure 3.5 are ended with a regression layer including 28 neurons with linear activation functions to represent state variables and their first derivatives.

Let N denote the number of time steps of the generated walking pattern. $\boldsymbol{D}$ is the input delay vector, M is the length of the input delay vector ($\boldsymbol{D}$). $\boldsymbol{D}[M]$ is the maximum number of time delays for the series-parallel model and equals to one for the parallel model. Accordingly, $N - \boldsymbol{D}[M]$ successive estimations should be performed in parallel and series-parallel models. From this perspective, the identification problem can be considered as a MIMO time series prediction problem for input and output series are defined in Figure 3.5. Basically, the input vector at the length of $(7 + 28 * M)$ is given to NN as input, then forward propagation is applied, and 28 output is calculated at the output layer of NN for each time step. This procedure is repeated for each of the $(N - \boldsymbol{D}[M])$ time steps. Then, identification errors are found by comparing outputs of NN with desired output series for the training set patterns. After that, the error gradient with respect to network weights is calculated, and it is propagated from the output layer to the previous layers of the NN. Finally, NN weights are updated with optimizers such as Adam and AdamW in a similar way to Subsection 2.1.3.

One of the most significant drawbacks of neural networks is the possibility of trapping into local minimums. That is why it is recommended to repeat the training procedure with different weight initializations. In addition to this, when the limited number of the training patterns is considered, we decided to use 5-fold cross-validation in the training and testing of proposed SINNs. For this purpose, we divide 320 training set patterns into five batches randomly. One of these five batches is chosen as the validation set, and the remaining four of them consist training set. With this training, validation, testing set separation, and random weight initialization, SINN is trained five times by changing the validation set at each fold. After that, the average of the system identification error is found for the proposed SINN architecture by averaging the results of each fold for each

data set. In this chapter, all reported error rates in the tables are the result of the 5-fold cross-validation process. In addition, mini-batch size is chosen as 64 in training, and patterns in the mini-batches are shuffled at the end of each epoch throughout training.

Figure 3.5: Neural network architectures employed in the "System Identification Neural Network" block in the system identification models

3.3 Results and Discussion

This section assesses the system identification capacity of different types of neuron models and network architectures under varying ground conditions and the mean squared error metric. To this end, NNs are trained under series-parallel and parallel models, and then their identification performances are compared with each other. Finally, the obtained results are discussed in detail.

3.3.1 Series-Parallel System Identification Simulations

At first, the series-parallel system identification model is studied, because of the series connection in the model which makes it possible to employ the fully feed-forward neural network in the biped robot model system identification problem. We start by investigating the suitability of different neural network layer types, in other words, neuron models, to the system identification problem. To do this, we start with a neural network with one hidden layer, as shown in Column (a) in Figure 3.5. In the scope of this initial study, 21 feedforward and 4 recurrent neuron models are utilized in this hidden layer as shown in the first column of Table 3.1. In detail, LSTM [36], GRU [90], RNN [91] with Tanh activation function, and RNN [91] with ReLU activation function are used as the recurrent neuron models in Table 3.1. In addition to these, ReLU [92], Tanh, ELU [93], Sigmoid, Hardshrink [94], Hardsigmoid [95], Hardtanh [96], Hardswish [97], LeakyReLU [98], LogSigmoid [99], SELU [100], PReLU [101], ReLU6 [102], RReLU [103], CELU [104], GELU [105], SiLU [106], Mish [107], Softplus [108], SoftShrink [109] and Softsign [110] feedforward neuron models are employed in the hidden layers in Table 3.1. Regardless of the type of neural model used in the hidden layer, the hidden layer is followed by a regression layer with linear activation functions.

In these neural networks, the hidden layer neuron number is determined in a way that the neural network has an equal or higher number of parameters than the parameter number of a neural network with 50 LSTM neurons employed in

its hidden layer. It means that compared neural networks have at least 18828 learnable parameters. In this context, types and numbers of hidden layer neurons are given in the first column of Table 3.1. Under these conditions, 25 different NNs are trained using the Adam optimizer with a learning constant of 0.01 throughout 30000 epochs. Network weights are stored at every 100 epochs, and then they are used to evaluate the training, validation, and testing set MSE rates of the trained neural networks. After that, training and testing set error rates which are computed with neural network weights at the learning step that gives the lowest validation set are noted. This whole process is repeated five times to apply 5-fold cross-validation. Finally, average of each data set error rate is reported in Table 3.1.

Table 3.1 shows that LSTM recurrent layer with 50 neurons reaches the lowest MSE error rates in all data sets among tested hidden layer types. After LSTM, LeakyReLU feedforward layer with 294 neurons obtains the second-lowest MSE error rates in all data sets. Thirdly, Hardsigmoid feedforward layer with 294 neurons achieves the third-lowest MSE error rates at each data set. Even though single hidden layer utilization is not enough to conclude, these results are enough to expect that LSTM and LeakyReLU have the most promising performances in the recurrent and feedforward layers, respectively. For this reason, we employ LSTM and LeakyReLU neuron models in the hidden layers in the rest of the chapter.

For the sake of completeness, we express the definition of the LeakyReLU neuron model employed in this study as follows:

$$\boldsymbol{o} = \begin{cases} \boldsymbol{W}\boldsymbol{x}, & \text{if } \boldsymbol{W}\boldsymbol{x} \geq 0 \\ 0.01 * \boldsymbol{W}\boldsymbol{x}, & \text{otherwise} \end{cases} \tag{3.4}$$

where $\boldsymbol{x}$, $\boldsymbol{W}$, and $\boldsymbol{o}$ represent the input vector, neuron model weights, and output of the neuron model, respectively.

Using the results from Table 3.1, we extend our research to find optimum neural network architectures by using feedforward and recurrent layers under the series-parallel model. To do that, the listed SINNs in Table 3.2, which are designed

Table 3.1: System identification performance comparisons of neural networks with one hidden layer for the series-parallel model

Hidden Layer Architecture	**Input Delay**	**Parameter Number**	**Data Set**		
			Training	**Validation**	**Testing**
50 LSTM	D=[1]	18828	$2.23 \cdot 10^{-6}$	$2.62 \cdot 10^{-6}$	$2.90 \cdot 10^{-6}$
60 GRU	D=[1]	19168	$5.23 \cdot 10^{-6}$	$5.44 \cdot 10^{-6}$	$5.41 \cdot 10^{-6}$
109 RNN (Tanh)	D=[1]	18994	$5.38 \cdot 10^{-5}$	$5.21 \cdot 10^{-5}$	$5.48 \cdot 10^{-5}$
109 RNN (ReLU)	D=[1]	18994	$6.81 \cdot 10^{-6}$	$7.18 \cdot 10^{-6}$	$7.02 \cdot 10^{-6}$
294 ReLU	D=[1]	18844	$4.81 \cdot 10^{-6}$	$5.58 \cdot 10^{-6}$	$5.13 \cdot 10^{-6}$
294 Tanh	D=[1]	18844	$2.42 \cdot 10^{-5}$	$2.47 \cdot 10^{-5}$	$2.47 \cdot 10^{-5}$
294 ELU	D=[1]	18844	$2.18 \cdot 10^{-5}$	$2.22 \cdot 10^{-5}$	$2.23 \cdot 10^{-5}$
294 Sigmoid	D=[1]	18844	$8.77 \cdot 10^{-6}$	$1.01 \cdot 10^{-5}$	$9.53 \cdot 10^{-6}$
294 Hardshrink	D=[1]	18844	$3.22 \cdot 10^{-3}$	$3.21 \cdot 10^{-3}$	$3.31 \cdot 10^{-3}$
294 Hardsigmoid	D=[1]	18844	$4.13 \cdot 10^{-6}$	$5.18 \cdot 10^{-6}$	$4.73 \cdot 10^{-6}$
294 Hardtanh	D=[1]	18844	$5.78 \cdot 10^{-5}$	$5.82 \cdot 10^{-5}$	$5.98 \cdot 10^{-5}$
294 Hardswish	D=[1]	18844	$2.72 \cdot 10^{-5}$	$2.76 \cdot 10^{-5}$	$2.79 \cdot 10^{-5}$
294 LeakyReLU	D=[1]	18844	$3.88 \cdot 10^{-6}$	$5.02 \cdot 10^{-6}$	$4.49 \cdot 10^{-6}$
294 LogSigmoid	D=[1]	18844	$9.39 \cdot 10^{-6}$	$1.08 \cdot 10^{-5}$	$1.03 \cdot 10^{-5}$
294 SELU	D=[1]	18844	$1.89 \cdot 10^{-5}$	$1.93 \cdot 10^{-5}$	$1.94 \cdot 10^{-5}$
294 PReLU	D=[1]	18845	$6.3 \cdot 10^{-6}$	$7.53 \cdot 10^{-6}$	$6.86 \cdot 10^{-6}$
294 ReLU6	D=[1]	18844	$4.81 \cdot 10^{-6}$	$5.58 \cdot 10^{-6}$	$5.13 \cdot 10^{-6}$
294 RReLU	D=[1]	18844	$5.11 \cdot 10^{-5}$	$5.13 \cdot 10^{-5}$	$5.28 \cdot 10^{-5}$
294 CELU	D=[1]	18844	$2.18 \cdot 10^{-5}$	$2.22 \cdot 10^{-5}$	$2.23 \cdot 10^{-5}$
294 GELU	D=[1]	18844	$1.31 \cdot 10^{-5}$	$1.38 \cdot 10^{-5}$	$1.34 \cdot 10^{-5}$
294 SiLU	D=[1]	18844	$2.17 \cdot 10^{-5}$	$2.23 \cdot 10^{-5}$	$2.2 \cdot 10^{-5}$
294 Mish	D=[1]	18844	$2.18 \cdot 10^{-5}$	$2.22 \cdot 10^{-5}$	$2.21 \cdot 10^{-5}$
294 Softplus	D=[1]	18844	$9.28 \cdot 10^{-6}$	$1.08 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$
294 SoftShrink	D=[1]	18844	$2.75 \cdot 10^{-3}$	$2.74 \cdot 10^{-3}$	$2.83 \cdot 10^{-3}$
294 Softsign	D=[1]	18844	$1.65 \cdot 10^{-5}$	$1.69 \cdot 10^{-5}$	$1.69 \cdot 10^{-5}$

to have up to at most 710428 learnable parameters, are trained and tested to evaluate the identification capability of deep neural networks (DNNs). The first column lists the hidden layer architecture from the input to the last layer before the regression layer. The second column demonstrates the input delay vector, which determines the number of previous time step biped robot model state information. The third column shows the learnable parameter number inside the NN. Finally, the last three columns show the MSE error rates obtained using network weights that give the least MSE rate over the validation set at all data sets. Table 3.2 compares the results of utilization of fully feedforward layers, fully recurrent layers, and combinations of feedforward and recurrent layers as the hidden layers in the system identification of the biped robot model.

Fully feedforward layer utilization is tested starting from one layer up to four-layer with an increasing number of the previous time step robot state information. First, we use one LeakyReLU layer with one previous time step state information and obtained lower MSE rates than in Table 3.1 thanks to increasing hidden layer neuron number. Then, we increase the previous time step number to two to analyze the effect of the number of previous time step state information. We see that the total number of neurons in the hidden layer decreases due to the learnable parameter number limitation, and SINN is accomplished to reach much lower error rates. After that, three previous time step numbers are tested, but a much-limited error rate decrease is acquired in the testing set by adding one more time step state information than the previous increment. For this reason, we limit the previous time step number to three and start to increase the number of hidden layers. While doing that, we design to have the same number of neurons at each hidden layer to comply with the upper parameter limit. Two hidden layer utilization decreases error rates, but three and four hidden layers cause increases in testing data set. At the end of the conducted experiments with fully feedforward hidden layers, it is concluded that the robot model state information for three previous time steps and two hidden layers with 784 LeakyReLU neurons at each reached the lowest error rate in the testing set.

Fully recurrent layer utilization is tested starting from one layer up to three-layer with one previous time step robot state information. First, we use one

LSTM layer with one previous time step state information and obtain lower MSE rates than in Table 3.1 thanks to increasing hidden layer neuron number, again. Then, we increase the number of hidden layers to two by using the same number of LSTM neurons at each layer. With this NN architecture, SINN is succeeded in reaching much lower error rates than the one LSTM layer. Thus, it acquires a lower testing set MSE rate than all tested fully feedforward hidden layer architectures. After that, three LSTM layer utilization is tested, but a much-limited error rate decrease is acquired in the testing set by adding this layer, and MSE on the validation set increases. These results conclude that utilization of two LSTM layers obtains a critical amount of error decrease, and third layer utilization is also helpful, but gain is marginal.

Later on, we search for system identification performance of various combinations of feedforward and recurrent layers as the hidden layers. For this purpose, the performances of NNs with up to five hidden layers are evaluated. In this study, feedforward layer neuron number is limited to 70, and neuron numbers of LSTM layers are selected as equal and not exceeding the number of learnable parameters limitation. As a result of the comparisons between many neural network architectures in Table 3.2, the hidden layer architecture in the shape of 233 LSTM, 233 LSTM, and 70 LeakyReLU acquires the lowest amount of MSE in validation and testing data sets. For this reason, the neural network with this hidden layer architecture is named as selected system identification neural network (SSINN) in the rest of this subsection. SSINN consists of two LSTM layers and two feedforward layers, including the regression layer at the end of the network. This shows the potential of NNs which includes combinations of recurrent and feedforward layers for system identification problems.

Table 3.2: System identification performance comparisons of neural networks with various numbers of hidden layers for the series-parallel model

Hidden Layer Architecture	**Input Delay**	**Parameter Number**	**Data Set**		
			Training	**Validation**	**Testing**
400 LSTM	D=[1]	710428	$1.02 \cdot 10^{-6}$	$2.29 \cdot 10^{-6}$	$2.21 \cdot 10^{-6}$
235 LSTM, 235 LSTM	D=[1]	705968	$5.22 \cdot 10^{-7}$	$1.51 \cdot 10^{-6}$	$1.39 \cdot 10^{-6}$
183 LSTM, 183 LSTM, 183 LSTM	D=[1]	704944	$2.44 \cdot 10^{-7}$	$1.54 \cdot 10^{-6}$	$1.39 \cdot 10^{-6}$
70 LeakyReLU, 383 LSTM	D=[1]	710332	$7.64 \cdot 10^{-7}$	$1.9 \cdot 10^{-6}$	$1.75 \cdot 10^{-6}$
70 LeakyReLU, 70 LeakyReLU, 381 LSTM	D=[1]	708558	$6.57 \cdot 10^{-7}$	$1.7 \cdot 10^{-6}$	$1.56 \cdot 10^{-6}$
70 LeakyReLU, 229 LSTM, 229 LSTM	D=[1]	706036	$4.68 \cdot 10^{-7}$	$1.57 \cdot 10^{-6}$	$1.45 \cdot 10^{-6}$
70 LeakyReLU, 70 LeakyReLU, 228 LSTM, 228 LSTM	D=[1]	705198	$4.39 \cdot 10^{-7}$	$1.94 \cdot 10^{-6}$	$1.78 \cdot 10^{-6}$
70 LeakyReLU 180 LSTM, 180 LSTM, 180 LSTM	D=[1]	710308	$2.21 \cdot 10^{-7}$	$1.76 \cdot 10^{-6}$	$1.61 \cdot 10^{-6}$
70 LeakyReLU, 70 LeakyReLU, 179 LSTM, 179 LSTM, 179 LSTM	D=[1]	707766	$2.87 \cdot 10^{-7}$	$2.29. \cdot 10^{-6}$	$2.1 \cdot 10^{-6}$
394 LSTM, 70 LeakyReLU	D=[1]	708894	$6.87 \cdot 10^{-7}$	$1.65 \cdot 10^{-6}$	$1.49 \cdot 10^{-6}$
392 LSTM, 70 LeakyReLU, 70 LeakyReLU	D=[1]	707140	$4.53 \cdot 10^{-7}$	$1.53 \cdot 10^{-6}$	$1.42 \cdot 10^{-6}$
233 LSTM, 233 LSTM, 70 LeakyReLU	D=[1]	706184	$4.32 \cdot 10^{-7}$	$1.39 \cdot 10^{-6}$	$1.27 \cdot 10^{-6}$
232 LSTM, 232 LSTM, 70 LeakyReLU, 70 LeakyReLU	D=[1]	705348	$3.84 \cdot 10^{-7}$	$1.54 \cdot 10^{-6}$	$1.44 \cdot 10^{-6}$
182 LSTM, 182 LSTM, 182 LSTM, 70 LeakyReLU	D=[1]	707126	$3.73 \cdot 10^{-7}$	$1.49 \cdot 10^{-6}$	$1.34 \cdot 10^{-6}$
181 LSTM, 181 LSTM, 181 LSTM, 70 LeakyReLU, 70 LeakyReLU	D=[1]	704602	$3.15 \cdot 10^{-7}$	$1.8 \cdot 10^{-6}$	$1.62 \cdot 10^{-6}$
70 LeakyReLU, 377 LSTM, 70 LeakyReLU	D=[1]	708060	$6.88 \cdot 10^{-7}$	$1.62 \cdot 10^{-6}$	$1.55 \cdot 10^{-6}$
70 LeakyReLU, 227 LSTM, 227 LSTM, 70 LeakyReLU	D=[1]	706008	$5.09 \cdot 10^{-7}$	$1.73 \cdot 10^{-6}$	$1.53 \cdot 10^{-6}$
70 LeakyReLU, 178 LSTM, 178 LSTM, 178 LSTM, 70 LeakyReLU	D=[1]	704830	$3.05 \cdot 10^{-7}$	$1.96 \cdot 10^{-6}$	$1.73 \cdot 10^{-6}$
266 LSTM, 70 LeakyReLU, 266 LSTM	D=[1]	708190	$6.8 \cdot 10^{-7}$	$2.1 \cdot 10^{-6}$	$2.14 \cdot 10^{-6}$
70 LeakyReLU, 258 LSTM, 70 LeakyReLU, 258 LSTM	D=[1]	709022	$4.89 \cdot 10^{-7}$	$2.39 \cdot 10^{-6}$	$2.34 \cdot 10^{-6}$
263 LSTM, 70 LeakyReLU, 263 LSTM, 70 LeakyReLU	D=[1]	706968	$3.73 \cdot 10^{-7}$	$1.57 \cdot 10^{-6}$	$1.61 \cdot 10^{-6}$
70 LeakyReLU, 255 LSTM, 70 LeakyReLU, 255 LSTM, 70 LeakyReLU	D=[1]	707428	$5.77 \cdot 10^{-7}$	$2.36 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$
11100 LeakyReLU	D=[1]	710428	$1.43 \cdot 10^{-6}$	$3.34 \cdot 10^{-6}$	$2.99 \cdot 10^{-6}$
7722 LeakyReLU	D=[1, 2]	710452	$1.07 \cdot 10^{-6}$	$2.74 \cdot 10^{-6}$	$2.29 \cdot 10^{-6}$
5920 LeakyReLU	D=[1, 2, 3]	710428	$1.03 \cdot 10^{-6}$	$2.6 \cdot 10^{-6}$	$2.28 \cdot 10^{-6}$
784 LeakyReLU, 784 LeakyReLU	D=[1, 2, 3]	709548	$4.39 \cdot 10^{-7}$	$2.25 \cdot 10^{-6}$	$2.13 \cdot 10^{-6}$
566 LeakyReLU, 566 LeakyReLU, 566 LeakyReLU	D=[1, 2, 3]	709792	$2.17 \cdot 10^{-7}$	$2.24 \cdot 10^{-6}$	$2.16 \cdot 10^{-6}$
466 LeakyReLU, 466 LeakyReLU, 466 LeakyReLU, 466 LeakyReLU	D=[1, 2, 3]	708814	$1.71 \cdot 10^{-7}$	$2.33 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$

Various regularization techniques are proposed to increase the generalization performance of neural networks in the literature. Table 3.3 presents the results of the effects of L_2 and dropout regularization techniques in the training of SSINN. L_2 regularization constant of 0.001 reduces the MSE rates in validation and testing data sets. However, higher and lower regularization constants increase the MSE rates. Moreover, the dropout technique is applied by adding a dropout layer between LSTM and LeakyReLU layers. The dropout rate of 0.001 reached the lowest error rate for validation and testing among all regularization trials in Table 3.3. In addition to these, dropout and L_2 regularizations are also applied together to the SSINN training. Combinations of the dropout rates of 0.001 and 0.0001, and L_2 constants 0.001 and 0.0001 decrease MSE rates in the testing set. Nevertheless, the lowest validation and testing set MSE rates are obtained using only dropout regularization.

3.3.2 Parallel System Identification Simulations

Second, the parallel system identification model is studied where LSTM and LeakyReLU neuron types are utilized to measure the system identification performance of various hidden layer architectures. Table 3.4 summarizes the performance of usage of fully recurrent layers and combinations of feedforward and recurrent layers as the hidden layers in the system identification of the biped robot model. To do that, the listed SINNs in Table 3.4, which are designed to have up to at least 712108 learnable parameters, are trained and tested to evaluate the identification capability of deep neural networks (DNNs).

Different from series-parallel system identification simulations reported in Subsection 3.3.1, fully feedforward layers are not tested in this subsection due to the inexistence of series connection in the parallel model because the identification problem becomes even more difficult without this series connection. Only if a high number of the previous time step state information is introduced to the fully feedforward neural network, the hybrid dynamical model of the biped robot system can be identified with reasonable error rates. Since the usage of a high

Table 3.3: Results of dropout and L_2 regularization implementation to the selected system identification neural network for series-parallel model

L2	**Dropout**	**Data Set**		
Constant	**Rate**	**Training**	**Validation**	**Testing**
0	0	$4.32 \cdot 10^{-7}$	$1.39 \cdot 10^{-6}$	$1.27 \cdot 10^{-6}$
0.0001	0	$4.81 \cdot 10^{-7}$	$1.4 \cdot 10^{-6}$	$1.34 \cdot 10^{-6}$
0.0005	0	$4.08 \cdot 10^{-7}$	$1.41 \cdot 10^{-6}$	$1.34 \cdot 10^{-6}$
0.001	0	$4.57 \cdot 10^{-7}$	$1.36 \cdot 10^{-6}$	$1.25 \cdot 10^{-6}$
0.005	0	$4.77 \cdot 10^{-7}$	$1.48 \cdot 10^{-6}$	$1.31 \cdot 10^{-6}$
0.01	0	$6.1 \cdot 10^{-7}$	$1.46 \cdot 10^{-6}$	$1.34 \cdot 10^{-6}$
0.1	0	$2.74 \cdot 10^{-6}$	$4.05 \cdot 10^{-6}$	$3.68 \cdot 10^{-6}$
0	0.0001	$4.11 \cdot 10^{-7}$	$1.39 \cdot 10^{-6}$	$1.27 \cdot 10^{-6}$
0	0.001	$4.44 \cdot 10^{-7}$	$1.33 \cdot 10^{-6}$	$1.21 \cdot 10^{-6}$
0	0.002	$5.5 \cdot 10^{-7}$	$1.37 \cdot 10^{-6}$	$1.27 \cdot 10^{-6}$
0	0.01	$8.03 \cdot 10^{-7}$	$1.61 \cdot 10^{-6}$	$1.53 \cdot 10^{-6}$
0	0.1	$9.63 \cdot 10^{-6}$	$9.86 \cdot 10^{-6}$	$9.9 \cdot 10^{-6}$
0.001	0.001	$5.09 \cdot 10^{-7}$	$1.45 \cdot 10^{-6}$	$1.25 \cdot 10^{-6}$
0.001	0.0001	$4.17 \cdot 10^{-7}$	$1.37 \cdot 10^{-6}$	$1.25 \cdot 10^{-6}$
0.0001	0.001	$5.35 \cdot 10^{-7}$	$1.44 \cdot 10^{-6}$	$1.25 \cdot 10^{-6}$
0.0001	0.0001	$3.88 \cdot 10^{-7}$	$1.38 \cdot 10^{-6}$	$1.26 \cdot 10^{-6}$

number of previous time steps requires the utilization of most of the network parameters at the input layer, which will limit the performance of SINN, we do not perform this in the scope of our study.

Fully recurrent layer utilization is tested starting from one layer up to three-layer. At first, we use one LSTM layer and obtain reported MSE rates in Table 3.4. Due to the lack of series connection in the parallel model, MSE rates for single LSTM hidden layer usage are much higher than in Table 3.2 and even Table 3.1. Then, we increase the number of hidden layers to two by using the same number of LSTM neurons at each layer. With this NN architecture, SINN is succeeded in reaching a lower error rate than the one LSTM layer. After that, three LSTM layer utilization is tested, but MSE rates on the validation and testing sets increased instead of decreasing.

Later on, we search for system identification performance of various combinations of feedforward and recurrent layers as the hidden layers. For this purpose, the performances of NNs with up to five hidden layers are evaluated. In this study, the feedforward layer neuron number is limited to 70, and neuron numbers of LSTM layers are selected as equal so that the number of learnable parameters does not fall below the limitation. As a result of the comparisons between many neural network architectures in Table 3.2, the hidden layer architecture in the shape of 70 LeakyReLU, 231 LSTM, and 231 LSTM acquires the lowest amount of MSE in validation and testing data sets. For this reason, the neural network with this hidden layer architecture is named selected system identification neural network (SSINN) in the rest of this subsection. Thus, SSINN consists of one feedforward input layer, two LSTM layers, and one regression layer at the end of the network. This shows the potential of NNs, which includes combinations of recurrent and feedforward layers for system identification problems again.

Table 3.4: System identification performance comparisons of neural networks with various numbers of hidden layers for the parallel model

Hidden Layer	**Parameter**	**Data Set**		
Architecture	**Number**	**Training**	**Validation**	**Testing**
414 LSTM	712108	$4.99 \cdot 10^{-6}$	$9.25 \cdot 10^{-6}$	$8.96 \cdot 10^{-6}$
241 LSTM, 241 LSTM	714352	$4.58 \cdot 10^{-6}$	$9.01 \cdot 10^{-6}$	$8.29 \cdot 10^{-6}$
187 LSTM, 187 LSTM, 187 LSTM	714368	$4.33 \cdot 10^{-6}$	$9.48 \cdot 10^{-6}$	$8.95 \cdot 10^{-6}$
409 LSTM, 70 LeakyReLU	714536	$4.39 \cdot 10^{-6}$	$9.91 \cdot 10^{-6}$	$9.03 \cdot 10^{-6}$
239 LSTM, 239 LSTM, 70 LeakyReLU	714756	$4.28 \cdot 10^{-6}$	$9.01 \cdot 10^{-6}$	$8.60 \cdot 10^{-6}$
238 LSTM, 238 LSTM, 70 LeakyReLU, 70 LeakyReLU	713888	$4.3 \cdot 10^{-6}$	$9.51 \cdot 10^{-6}$	$8.94 \cdot 10^{-6}$
271 LSTM, 70 LeakyReLU, 271 LSTM, 70 LeakyReLU	715400	$4.37 \cdot 10^{-6}$	$9.18 \cdot 10^{-6}$	$8.79 \cdot 10^{-6}$
70 LeakyReLU, 385 LSTM	715148	$4.45 \cdot 10^{-6}$	$8.6 \cdot 10^{-6}$	$8.3 \cdot 10^{-6}$
70 LeakyReLU, 231 LSTM, 231 LSTM	715764	$4.03 \cdot 10^{-6}$	$8.73 \cdot 10^{-6}$	$8.11 \cdot 10^{-6}$
70 LeakyReLU, 70 LeakyReLU, 230 LSTM, 230 LSTM	714878	$4.34 \cdot 10^{-6}$	$8.83 \cdot 10^{-6}$	$8.11 \cdot 10^{-6}$
70 LeakyReLU, 260 LSTM, 70 LeakyReLU, 260 LSTM	716698	$4.16 \cdot 10^{-6}$	$9.43 \cdot 10^{-6}$	$8.95 \cdot 10^{-6}$
70 LeakyReLU, 229 LSTM, 229 LSTM, 70 LeakyReLU	715724	$3.89 \cdot 10^{-6}$	$8.74 \cdot 10^{-6}$	$8.42 \cdot 10^{-6}$
70 LeakyReLU, 257 LSTM, 70 LeakyReLU, 257 LSTM, 70 LeakyReLU	715092	$4.26 \cdot 10^{-6}$	$9.63 \cdot 10^{-6}$	$8.67 \cdot 10^{-6}$

Table 3.5 presents the results of the effects of L_2 regularization in the training of SSINN. Different from the series-parallel model, L_2 regularization cannot succeed in reducing the MSE rate on any data set. This result may be related to the properties of the parallel model that makes the problem harder.

Table 3.5: Results of L_2 regularization implementation to the selected system identification neural network for parallel model

L_2	**Data Set**		
Constant	**Training**	**Validation**	**Testing**
0	$4.03 \cdot 10^{-6}$	$8.73 \cdot 10^{-6}$	$8.11 \cdot 10^{-6}$
0.00001	$4.03 \cdot 10^{-6}$	$8.73 \cdot 10^{-6}$	$8.11 \cdot 10^{-6}$
0.0001	$4.34 \cdot 10^{-6}$	$8.75 \cdot 10^{-6}$	$8.13 \cdot 10^{-6}$
0.001	$4.53 \cdot 10^{-6}$	$8.82 \cdot 10^{-6}$	$8.3 \cdot 10^{-6}$
0.01	$4.7 \cdot 10^{-6}$	$8.9 \cdot 10^{-6}$	$8.45 \cdot 10^{-6}$

3.3.3 Discussion

In this study, the walking data set was generated using the biped robot model and central pattern generator. Then, the data set was divided into training and testing data sets. This separation was performed to prevent the over-fitting issue that degrades machine learning algorithms' performance on unseen data. In addition to these, we employed 5-fold cross-validation to minimize the possibility of trapping to local minima. In detail, neural network weights were initialized randomly, and training set patterns were divided into five, and one of them was used as the validation set, and the remainings constitute the training set for each fold. All reported results were obtained with neural network weights that gave the lowest MSE rates on validation set patterns throughout the training process in this chapter.

The utilization of one and two recurrent layers became helpful in reducing MSE rates on all data sets and each system identification model. However, the

contribution of a higher number of recurrent layers was either limited or negative. Furthermore, the addition of feedforward layers might help minimize error rates. Still, their optimum positions in the neural network architectures showed differences between parallel and series-parallel models in our studies. In addition, fully feedforward neural networks cannot be employed in an end-to-end manner in the parallel system identification model due to the lack of series connection in the model. One solution way is to add a block with memory to the system identification scheme, such as in parallel neural network Wiener or Hammerstein models, see [59]. With similar motivation, we proposed utilizing recurrent layers due to their memory property. Thus, their usage lets us use neural networks end-to-end in the parallel model.

Data set generation is one of the crucial steps in training neural networks because the generalization ability of neural networks mostly depends on the inclusivity of the data set due to the nature of the supervised learning algorithm that we utilized in this study. For this reason, it is beneficial to collect more movement data for the biped robot model, so the data set should be enlarged by the addition of not only successful but also failing walking patterns. In addition to this, we limited feedforward layer sizes to 70 to force this layer to extract features in data with a similar motivation to the autoencoder usage. Moreover, we had to determine a constant layer size to be comparable between different neural network architectures. For this reason, the use of combinations of recurrent and feedforward layers requires further research using different feedforward layer neuron sizes. Besides applied layer types in this study, there are different types of layers, such as convolutional layers. Their performance should be evaluated to find an optimum neural network-based system identification scheme for our problem. Another critical topic is the effect of applied regularization techniques. L_2 and dropout regularization with small regularization constants reduced the error rate of the testing set in the series-parallel model. Unfortunately, we cannot obtain any error rate decrease with L_2 in the parallel model. This situation may be related to the insufficient number of neural network parameters for increased problem difficulty due to the inexistence of the series connection from the robot model in the parallel model. Inevitably this topic requires further research, but

we had to limit our analysis to a similar parameter number with series-parallel model due to obtaining comparable results and our limited computation power for training large neural networks.

To sum up, it is concluded that combinations of recurrent and feedforward layers promise the potential for system identification of hybrid dynamical systems in an end-to-end manner in terms of results obtained in this study.

3.4 Conclusion

In this Chapter, we proposed the use of neural networks for system identification of the biped robot model by using parallel and series-parallel identification schemes. We utilized feedforward and recurrent layers in the proposed neural network architectures. The proposed system identification neural networks estimate robot model state variables and derivatives of these state variables with respect to inputs and previous time step state information of the robot model.

These neural networks were trained with varying training options. Later on, we evaluated and compared their identification performance in the simulation environment. As another contribution, the generalization abilities of proposed SINN architectures were tested by investigating training properties that may affect generalization performance. As a result, dropout regularization was the most effective regularization technique in the training of networks to minimize mean squared error rates on the testing data set.

Finally, data set generation, over-fitting issues, and differences in system identification models were discussed in detail. Later on, possible development ways were proposed related to data set expansion, neural network architecture enlargements, and different neuron models. Apart from these, adding a recurrent layer allowed the neural network to estimate the next time step of robot states with lower error rates, and it made end-to-end neural network utilization in a parallel identification scheme possible. Moreover, using combinations of feedforward and

recurrent layers was exemplified in the system identification of hybrid dynamical systems by a biped robot model.

Chapter 4

Conclusion and Future Works

Legged locomotion is a popular area of research due to its agile mobility with a wide range of motion achieved thanks to its structural similarity to its biological counterparts. To achieve the targeted performance, legged robots must be controlled with well-designed locomotion controllers, and this requires a good understanding of the characteristics of the robotic system. Unfortunately, legged robotic systems contain nonlinear dynamics which vary depending on the foot's contact with the ground. For this reason, control and system identification of them are complex research problems. This dissertation proposes the utilization of recurrent neural networks to overcome the complexity of nonlinear dynamics of the biped robot model in the control and system identification.

In the first part of this dissertation, we began by summarizing the literature on the legged robotic systems, stability analysis methods for legged locomotion, classical controllers, reflex-based gait controllers inspired by nature, neural networks, and system identification. While doing that, we criticized some of these studies and suggested possible ways to improve. Later on, we presented the motivation and contributions of this dissertation.

The second part presents our studies on the NNBC design problem for two-legged robot motion control. To this end, we benefited from a biped robot

model and CPG to produce walking data for the training and testing of proposed NNBCs. In this study, we utilized the LSTM layer as a hidden layer and feedforward regression layer in the proposed neural network architectures. The proposed neural controllers were employed in the feedback and feedforward paths to control bipedal locomotion so that they produced either joint torques or limb angles to achieve stable walking. Moreover, we investigated the effects of varying training options such as mini-batch sizes and regularization methods. Finally, the stable walking generation capacity of proposed NNBCs is evaluated and compared in the simulation environment for varying ground conditions and robot model dynamics. Our results showed that recurrent layer utilization allowed the tracking of legged locomotion phase changes and increased the control robustness. Hence, NNBCs showed more robust performance than CPG and PID type controllers in the varying walking ground roughness conditions, robot weight changes, and joint torque limitations. In addition, we analyzed the characteristics of NNBCs with well-known stability analysis methods. Finally, the generation of the data sets, the emergence of over-fitting issues, and the limitations of the LSTM neuron model are discussed in detail.

In the third part, we focused on system identification of the biped robot model using neural networks in an end-to-end manner. To this end, we utilized the same biped robot model and CPG to form data sets that were employed in the training and testing of proposed neural network architectures. Throughout this study, we benefited from parallel and series-parallel system identification models. First, we searched for suitable neuron models and found that LSTM and LeakyReLU performed the best among recurrent and feedforward neuron types, respectively. Moreover, the addition of a recurrent layer was beneficial in reducing the identification error metric in parallel and series-parallel models. Furthermore, the combination of two recurrent layers, one feedforward layer, and a regression layer reached the lowest error rates among the proposed network architectures. Later on, we determined that regularization techniques helped minimize error rates, especially in the series-parallel model. Finally, we discussed the importance of the representability of the robot model through the generated data set in the

system identification, and we proposed data set enlargement methods. In addition, we suggested expanding evaluated neural network architectures with bigger feedforward layers and different neuron models.

As a future work, proposed neural network-based control and system identification techniques in this dissertation will be tested with a physical biped robot platform in our laboratory. In the scope of 120E104 coded TÜBİTAK project, we built a test platform that includes a biped robot with seven degrees of freedom. In this way, we aim to validate proposed algorithms under real-world conditions using this robot platform developed throughout the project.

In the near future, we will examine the generalization abilities of the proposed controllers at varying ground stiffness in the simulation environment. Subsequently, we will explore possible gains by widening the proposed neural controller with parallel neural layers, rather than serial flow as in this study. In addition to these, we plan to modify the internal structure of the LSTM neuron model to be more appropriate for the control tasks.

Another possible future research direction is to enhance the representativity of the system identification neural network, by increasing the number and diversity of robot motion patterns in the data set. To achieve this, we will use different walking controllers in the short term and reinforcement learning to collect more movement data from the biped robot model in the medium term.

As a continuation of these studies, we plan to include robot platform dynamics in the neural network training process within the scope of 120E104 coded TÜBİTAK project. To yield this, we will employ a model reference-based neural network adaptive control scheme via the addition of recurrent layers in both system identification and controller blocks. Thus, we want to design adaptable neural controllers which can adapt to the differences between the theoretical robot model and the physical robot platform, resulting from aging, carrying a load, and unmodeled dynamics of the robotic system. Finally, the performance of the proposed model reference-based neural network adaptive control scheme

will be investigated and evaluated using both the biped robot model in the simulation environment and the physical robot platform built in our laboratory as a comparative study.

Bibliography

[1] Ba. Çatalbaş and Ö. Morgül, "Two-legged robot motion control with recurrent neural networks," *Journal of Intelligent & Robotic Systems*, vol. 104, no. 4, pp. 1–30, 2022.

[2] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim, "Optimization based controller design and implementation for the atlas robot in the darpa robotics challenge finals," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 1028–1035, IEEE, 2015.

[3] E. Guizzo, "By leaps and bounds: An exclusive look at how boston dynamics is redefining robot agility," *IEEE Spectrum*, vol. 56, no. 12, pp. 34–39, 2019.

[4] P. Holmes, R. J. Full, D. Koditschek, and J. Guckenheimer, "The dynamics of legged locomotion: Models, analyses, and challenges," *SIAM review*, vol. 48, no. 2, pp. 207–304, 2006.

[5] U. Saranlı, M. Buehler, and D. E. Koditschek, "Rhex: A simple and highly mobile hexapod robot," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 616–631, 2001.

[6] İ. Uyanık, U. Saranlı, and Ö. Morgül, "Adaptive control of a spring-mass hopper," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2138–2143, IEEE, 2011.

[7] M. Chignoli, D. Kim, E. Stanger-Jones, and S. Kim, "The mit humanoid robot: Design, motion planning, and control for acrobatic behaviors," *arXiv preprint arXiv:2104.09025*, 2021.

[8] W. J. Schwind, *Spring loaded inverted pendulum running: A plant model*. PhD thesis, University of Michigan, USA, 1998.

[9] I. Uyanık, M. M. Ankaralı, N. J. Cowan, U. Saranlı, and Ö. Morgül, "Identification of a vertical hopping robot model via harmonic transfer functions," *Transactions of the Institute of Measurement and Control*, vol. 38, no. 5, pp. 501–511, 2016.

[10] C. L. Shih, "Ascending and descending stairs for a biped robot," *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans*, vol. 29, no. 3, pp. 255 – 268, 1999.

[11] K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired zmp," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2684 – 2689, 2002.

[12] A. Goswami, "Foot rotation indicator (fri) point: A new gait planning tool to evaluate postural stability of biped robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 1, pp. 47–52, IEEE, 1999.

[13] M. B. Popovic, A. Goswami, and H. Herr, "Ground reference points in legged locomotion: Definitions, biological trajectories and control implications," *The international journal of robotics research*, vol. 24, no. 12, pp. 1013–1032, 2005.

[14] M. M. Ankarali and U. Saranli, "Stride-to-stride energy regulation for robust self-stability of a torque-actuated dissipative spring-mass hopper," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 20, no. 3, p. 033121, 2010.

[15] D. Kerimoğlu, Ö. Morgül, and U. Saranli, "Stability and control of planar compass gait walking with series-elastic ankle actuation," *Transactions of the Institute of Measurement and Control*, vol. 39, no. 3, pp. 312–323, 2017.

[16] A. Spröwitz, A. Tuleu, M. Vespignani, M. Ajallooeian, E. Badri, and A. J. Ijspeert, "Towards dynamic trot gait locomotion: Design, control, and experiments with cheetah-cub, a compliant quadruped robot," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 932–950, 2013.

[17] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 423–443, 2008.

[18] A. Crespi and A. J. Ijspeert, "Online optimization of swimming and crawling in an amphibious snake robot," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 75–87, 2008.

[19] S. Aoi and K. Tsuchiya, "Stability analysis of a simple walking model driven by an oscillator with a phase reset using sensory feedback," *IEEE Transactions on Robotics*, vol. 22, pp. 391–397, April 2006.

[20] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008.

[21] J. André, C. Teixeira, C. P. Santos, and L. Costa, "Adapting biped locomotion to sloped environments," *Journal of Intelligent & Robotic Systems*, vol. 80, no. 3-4, pp. 625–640, 2015.

[22] C. P. Santos, N. Alves, and J. C. Moreno, "Biped locomotion control through a biomimetic cpg-based controller," *Journal of Intelligent & Robotic Systems*, vol. 85, no. 1, pp. 47–70, 2017.

[23] C. Liu, J. Yang, K. An, and Q. Chen, "Rhythmic-reflex hybrid adaptive walking control of biped robot," *Journal of Intelligent & Robotic Systems*, vol. 94, no. 3-4, pp. 603–619, 2019.

[24] L. Righetti and A. J. Ijspeert, "Pattern generators with sensory feedback for the control of quadruped locomotion," in *2008 IEEE International Conference on Robotics and Automation*, pp. 819–824, IEEE, 2008.

[25] A. J. Ijspeert and J. Kodjabachian, "Evolution and development of a central pattern generator for the swimming of a lamprey," *Artificial life*, vol. 5, no. 3, pp. 247–269, 1999.

[26] Y. Nakamura, T. Mori, M. A. Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a cpg-actor-critic method," *Neural Networks*, vol. 20, no. 6, pp. 723–735, 2007.

[27] K. Matsuoka, "Mechanisms of frequency and pattern control in the neural rhythm generators," *Biological cybernetics*, vol. 56, no. 5, pp. 345–353, 1987.

[28] Y. Maeda, A. Ito, and H. Ito, "Central pattern generator and its learning via simultaneous perturbation method," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1 – 6, 2012.

[29] C.-S. Park, Y.-D. Hong, and J.-H. Kim, "Evolutionary-optimized central pattern generator for stable modifiable bipedal walking," *IEEE/ASME Transactions on Mechatronics*, vol. 19, no. 4, pp. 1374–1383, 2013.

[30] G. Taga, Y. Yamaguchi, and H. Shimizu, "Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment," *Biological cybernetics*, vol. 65, no. 3, pp. 147–159, 1991.

[31] Q. Lu and J. Tian, "Research on walking gait of biped robot based on a modified cpg model," *Mathematical Problems in Engineering*, vol. 2015, p. 9, 2015.

[32] F. L. Lewis, S. Jagannathan, and A. Yesildirak, *Neural network control of robot manipulators and non-linear systems*. CRC Press, 1998.

[33] J. M. Zurada, *Introduction to artificial neural systems*, vol. 8. West publishing company St. Paul, 1992.

[34] S. Haykin, *Neural Networks and Learning Machines, 3/E*. Pearson Education India, 2010.

[35] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Transactions on Neural networks*, vol. 6, no. 5, pp. 1212–1228, 1995.

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

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[38] Bu. Çatalbaş, Ba. Çatalbaş, and Ö. Morgül, "Human activity recognition with different artificial neural network based classifiers," in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2017.

[39] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, July 2017.

[40] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding," in *Interspeech*, pp. 3771 – 3775, 2013.

[41] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning*, pp. 1017–1024, 2011.

[42] P. Hénaff, V. Scesa, F. B. Ouezdou, and O. Bruneau, "Real time implementation of ctrnn and bptt algorithm to learn on-line biped robot balance: Experiments on the standing posture," *Control Engineering Practice*, vol. 19, no. 1, pp. 89–99, 2011.

[43] Ba. Çatalbaş, "Recurrent neural network learning with an application to the control of legged locomotion," Master's thesis, Bilkent University, 2015.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[46] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[47] Bu. Çatalbaş and Ö. Morgül, "A new learning algorithm: Sinadamax," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2019.

[48] Bu. Çatalbaş, "Improved artificial neural network training with advanced methods," Master's thesis, Bilkent University, 2018.

[49] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[51] Bu. Çatalbaş, Ba. Çatalbaş, and Ö. Morgül, "A new initialization method for artificial neural networks: Laplacian," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2018.

[52] H. Rostro-Gonzalez, P. A. Cerna-Garcia, G. Trejo-Caballero, C. H. Garcia-Capulin, M. A. Ibarra-Manzano, J. G. Avina-Cervantes, and C. Torres-Huitzil, "A cpg system based on spiking neurons for hexapod robot locomotion," *Neurocomputing*, vol. 170, pp. 47–54, 2015.

[53] U. Jaramillo-Avila, H. Rostro-Gonzalez, L. A. Camuñas-Mesa, R. d. J. Romero-Troncoso, and B. Linares-Barranco, "An address event representation-based processing system for a biped robot," *International Journal of Advanced Robotic Systems*, vol. 13, no. 1, p. 39, 2016.

[54] E. I. Guerra-Hernandez, A. Espinal, P. Batres-Mendoza, C. H. Garcia-Capulin, R. D. J. Romero-Troncoso, and H. Rostro-Gonzalez, "A fpga-based neuromorphic locomotion system for multi-legged robots," *IEEE Access*, vol. 5, pp. 8301–8312, 2017.

[55] D. Gutierrez-Galan, J. P. Dominguez-Morales, F. Perez-Peña, A. Jimenez-Fernandez, and A. Linares-Barranco, "Neuropod: a real-time neuromorphic spiking cpg applied to robotics," *Neurocomputing*, vol. 381, pp. 10–19, 2020.

[56] J. Wright and I. Jordanov, "Intelligent approaches in locomotion-a review," *Journal of Intelligent & Robotic Systems*, vol. 80, no. 2, pp. 255–277, 2015.

[57] S. Auddy, S. Magg, and S. Wermter, "Hierarchical control for bipedal locomotion using central pattern generators and neural networks," in *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, (Oslo, Norway), pp. 13–18, IEEE, 2019.

[58] R. K. Mandava and P. R. Vundavilli, "An adaptive pid control algorithm for the two-legged robot walking on a slope," *Neural Computing and Applications*, vol. 32, pp. 3407–3421, 2020.

[59] A. Janczak, *Identification of nonlinear systems using neural networks and polynomial models: a block-oriented approach*, vol. 310. Springer Science & Business Media, 2004.

[60] Ba. Çatalbaş, Bu. Çatalbaş, and Ö. Morgül, "Two-legged robot system identification with artificial neural networks," in *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2020.

[61] H. Choi, C. Crump, C. Duriez, A. Elmquist, G. Hager, D. Han, F. Hearl, J. Hodgins, A. Jain, F. Leve, *et al.*, "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *Proceedings of the National Academy of Sciences*, vol. 118, no. 1, 2021.

[62] A. D. Olaru, S. A. Olaru, N. F. Mihai, and N. M. Smidova, "Animation in robotics with labview instrumentation," *Int. J. Modeling Optim*, vol. 9, pp. 34–40, 2019.

[63] B. Zhang and P. Liu, "Control and benchmarking of a 7-dof robotic arm using gazebo and ros," *PeerJ Computer Science*, vol. 7, p. e383, 2021.

[64] T. Schulz, J. Torresen, and J. Herstad, "Animation techniques in human-robot interaction user studies: A systematic literature review," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 8, no. 2, pp. 1–22, 2019.

[65] C. K. Liu and D. Negrut, "The role of physics-based simulators in robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, 2020.

[66] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[67] T. Xu, D. An, Y. Jia, and Y. Yue, "A review: Point cloud-based 3d human joints estimation," *Sensors*, vol. 21, no. 5, p. 1684, 2021.

[68] Z. Makhataeva and H. A. Varol, "Augmented reality for robotics: a review," *Robotics*, vol. 9, no. 2, p. 21, 2020.

[69] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2245–2252, IEEE, 2018.

[70] M. Focchi, A. Del Prete, I. Havoutis, R. Featherstone, D. G. Caldwell, and C. Semini, "High-slope terrain locomotion for torque-controlled quadruped robots," *Autonomous Robots*, vol. 41, no. 1, pp. 259–272, 2017.

[71] Q. Nguyen, M. J. Powell, B. Katz, J. Di Carlo, and S. Kim, "Optimized jumping on the mit cheetah 3 robot," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 7448–7454, IEEE, 2019.

[72] J. Li and Q. Nguyen, "Force-and-moment-based model predictive control for achieving highly dynamic locomotion on bipedal robots," *arXiv preprint arXiv:2104.00065*, 2021.

[73] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, IEEE, 2018.

[74] F. Plestan, J. W. Grizzle, E. R. Westervelt, and G. Abba, "Stable walking of a 7-dof biped robot," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, pp. 653–668, 2003.

[75] J. Vazquez and M. Velasco-Villa, "Numerical analysis of the sliding effects of a 5-dof biped robot," in *2011 8th International Conference on Electrical Engineering, Computing Science and Automatic Control*, pp. 1–6, IEEE, 2011.

[76] Z. You and Z. Zhang, "An overview of the underactuated biped robots," in *2011 IEEE International Conference on Information and Automation*, pp. 772–776, IEEE, 2011.

[77] J. H. Barron-Zambrano and C. Torres-Huitzil, "Cpg implementations for robot locomotion: Analysis and design," in *Robotic Systems-Applications, Control and Programming*, IntechOpen, 2012.

[78] M. Ö. Efe, "Neural network assisted computationally simple pid control of a quadrotor uav," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 354–361, 2011.

[79] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical review letters*, vol. 59, no. 19, p. 2229, 1987.

[80] C. Olah, "Understanding lstm networks." `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015. Accessed: 2019-11-17.

[81] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[82] A. Gomez, "Backpropogating an lstm: A numerical example." `https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9`, April 2016. Accessed: 2019-11-17.

[83] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, pp. 950–957, 1992.

[84] D. G. Hobbelen and M. Wisse, "Limit cycle walking," in *Humanoid Robots, Human-like Machines*, IntechOpen, 2007.

[85] H. Hamzaçebi, *Analysis and control of periodic gaits in legged robots*. PhD thesis, Bilkent University, 2017.

[86] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[87] R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen, "Ablation studies in artificial neural networks," *arXiv preprint arXiv:1901.08644*, 2019.

[88] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[89] Y. Wang, "A new concept using lstm neural networks for dynamic system identification," in *2017 American control conference (ACC)*, pp. 5324–5329, IEEE, 2017.

[90] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[91] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[92] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[93] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[94] PyTorch, "Hardshrink." `https://pytorch.org/docs/stable/generated/torch.nn.Hardshrink.html#torch.nn.Hardshrink`. Accessed: 2022-05-23.

[95] PyTorch, "Hardsigmoid." `https://pytorch.org/docs/stable/generated/torch.nn.Hardsigmoid.html#torch.nn.Hardsigmoid`. Accessed: 2022-05-23.

[96] PyTorch, "Hardtanh." `https://pytorch.org/docs/stable/generated/torch.nn.Hardtanh.html#torch.nn.Hardtanh`. Accessed: 2022-05-23.

[97] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

[98] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, Citeseer, 2013.

[99] PyTorch, "Logsigmoid." `https://pytorch.org/docs/stable/generated/torch.nn.LogSigmoid.html#torch.nn.LogSigmoid`. Accessed: 2022-05-23.

[100] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[101] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[102] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[103] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[104] J. T. Barron, "Continuously differentiable exponential linear units," *arXiv preprint arXiv:1704.07483*, 2017.

[105] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[106] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2018.

[107] D. Misra, "Mish: A self regularized non-monotonic activation function," *arXiv preprint arXiv:1908.08681*, 2019.

[108] PyTorch, "Softplus." `https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html#torch.nn.Softplus`. Accessed: 2022-05-23.

[109] PyTorch, "Softshrink." `https://pytorch.org/docs/stable/generated/torch.nn.Softshrink.html#torch.nn.Softshrink`. Accessed: 2022-05-23.

[110] PyTorch, "Softsign." `https://pytorch.org/docs/stable/generated/torch.nn.Softsign.html#torch.nn.Softsign`. Accessed: 2022-05-23.

Appendix A

Equations of Biped Robot Model

Following equations are taken from Taga et al. [30] and they are added to appendix for the sake of completeness. They are employed to generate data sets and test the locomotion control ability of trained neural networks in the paper. In addition to these, x_5, x_8, x_{11} and x_{14} are referred as θ_1, θ_2, θ_3 and θ_4 for the sake of simplicity throughout the paper, respectively. For further details see [30].

System Parameters:

$M = 48,\ m_1 = 7,\ m_2 = 4,\ l_1 = 0.5,\ l_2 = 0.6$

$I_1 = \dfrac{m_1 l_1^2}{12},\ I_2 = \dfrac{m_2 l_2^2}{12},\ b_1 = 10,\ b_2 = 10,\ g = 9.8,$

$b_k = 1000,\ k_k = 10000,\ k_g = 10000,\ b_g = 1000,$

Initial Conditions:

$x_1 = 0.0,\ x_2 = 1.09,\ x_5 = 0.45\pi,\ x_8 = 0.57\pi,$

$x_3 = x_1 + \frac{l_1}{2}\cos x_5,\ x_4 = x_2 - \frac{l_1}{2}\sin x_5,\ x_{11} = 0.45\pi,$

$x_6 = x_1 + \frac{l_1}{2}\cos x_8,\ x_7 = x_2 - \frac{l_1}{2}\sin x_8,\ x_{14} = 0.57\pi,$

$x_9 = l_1\cos x_5 + \frac{l_2}{2}\cos x_{11},\ x_{12} = l_1\cos x_8 + \frac{l_2}{2}\cos x_{14},$

$x_{10} = x_2 - l_1\sin x5 - \frac{l_2}{2}\sin x_{11},$

$x_{13} = x_2 - l_1\sin x8 - \frac{l_2}{2}\sin x_{14},$

$\dot{x}_i = 0,\ (i = 1, 2, .., 14),\ \dot{u}_i,\ \dot{v}_i = 0,\ (i = 1, 2, .., 12)$

Equations of Kinematic Constraints:

$x_1 = x_3 - \frac{l_1}{2}\cos x_5 = x_6 - \frac{l_1}{2}\cos x_8$

$x_2 = x_4 - \frac{l_1}{2}\sin x_5 = x_7 - \frac{l_1}{2}\sin x_8$

$x_3 + \frac{l_1}{2}\cos x_5 = x_9 - \frac{l_2}{2}\cos x_{11}$

$x_4 - \frac{l_1}{2}\sin x_5 = x_{10} + \frac{l_2}{2}\sin x_{11}$

$x_6 + \frac{l_1}{2}\cos x_8 = x_{12} - \frac{l_2}{2}\cos x_{14}$

$x_7 + \frac{l_1}{2}\sin x_8 = x_{13} + \frac{l_2}{2}\sin x_{14}$

$(x_r, y_r) = (x_9 + \frac{l_2}{2}\cos x_{11}, x_{10} - \frac{l_2}{2}\sin x_{11})$

$(x_l, y_l) = (x_{12} + \frac{l_2}{2}\cos x_{14}, x_{13} - \frac{l_2}{2}\sin x_{14})$

Feedback pathway:

$$
\begin{aligned}
a_1 = a_3 = a_4 = a_6 =& a_8 = 1.5,\ a_2 = 1.0,\ a_5 = a_7 = 3.0 \\
Feed_1 \ = -Feed_2 \ =& a_3(x_{11} - \pi/2)h(F_{g2}) + a_4 h(F_{g4}) \\
& + a_1(x_5 - \pi/2) - a_2(x_8 - \pi/2) \\
Feed_3 \ = -Feed_4 \ =& a_3(x_{14} - \pi/2)h(F_{g4}) + a_4 h(F_{g2}) \\
& + a_1(x_8 - \pi/2) - a_2(x_5 - \pi/2) \\
Feed_5 \ = -Feed_6 \ =& a_5(\pi/2 - x_{14})h(F_{g4}) \\
Feed_7 \ = -Feed_8 \ =& a_5(\pi/2 - x_{11})h(F_{g2}) \\
Feed_9 \ = -Feed_{10} =& (a_6(\pi/2 - x_{11}) - a_8\dot{x}_{11})h(F_{g2}) \\
& + a_7(\pi/2 - x_{14})h(F_{g4}) \\
Feed_{11} = -Feed_{12} =& (a_6(\pi/2 - x_{14}) - a_8\dot{x}_{14})h(F_{g4}) \\
& + a_7(\pi/2 - x_{11})h(F_{g2}) -
\end{aligned}
$$

Equations of Motion:

$$
\begin{aligned}
M\ddot{x}_1 =& F_1 + F_3,\ M\ddot{x}_2 = F_2 + F_4 - Mg \\
m_1\ddot{x}_3 =& -F_1 + F_5,\ m_1\ddot{x}_4 = -F_2 + F_6 - m_1 g \\
I_1\ddot{x}_5 =& -F_1\frac{l_1}{2}\sin x_5 - F_2\frac{l_1}{2}\cos x_5 - F_5\frac{l_1}{2}\sin x_5 \\
& - F_6\frac{l_1}{2}\cos x_5 - b_1|x_5 - \frac{\pi}{2}|\dot{x}_5 - k_k h(x_5 - x_{11}) \\
& - (b_2 + b_k f(x_5 - x_{11}))(\dot{x}_5 - \dot{x}_{11}) + T_{r1} + T_{r3} \\
I_1\ddot{x}_8 =& -F_3\frac{l_1}{2}\sin x_8 - F_4\frac{l_1}{2}\cos x_8 - F_7\frac{l_1}{2}\sin x_8 \\
& - F_8\frac{l_1}{2}\cos x_8 - b_1|x_8 - \frac{\pi}{2}|\dot{x}_8 - k_k h(x_8 - x_{14}) \\
& - (b_2 + b_k f(x_8 - x_{14}))(\dot{x}_8 - \dot{x}_{14}) + T_{r2} + T_{r4} \\
I_2\ddot{x}_{11} =& -F_5\frac{l_2}{2}\sin x_{11} - F_6\frac{l_2}{2}\cos x_{11} - F_{g1}\frac{l_2}{2}\sin x_{11} \\
& - F_{g2}\frac{l_2}{2}\cos x_{11} + k_k h(x_5 - x_{11}) - T_{r3} - T_{r5} \\
& - (b_2 + b_k f(x_5 - x_{11}))(\dot{x}_{11} - \dot{x}_5) \\
I_2\ddot{x}_{14} =& -F_7\frac{l_2}{2}\sin x_{14} - F_8\frac{l_2}{2}\cos x_{14} - F_{g3}\frac{l_2}{2}\sin x_{14} \\
& - F_{g4}\frac{l_2}{2}\cos x_{14} + k_k h(x_8 - x_{14}) - T_{r4} - T_{r6} \\
& - (b_2 + b_k f(x_8 - x_{14}))(\dot{x}_{14} - \dot{x}_8)
\end{aligned}
$$

$$
f(x) = max(0, x),\ h(x) = \begin{cases} 0 & \text{for } x \leqslant 0 \\ 1 & \text{for } x > 0 \end{cases}
$$

$$
F_{g1} = \begin{cases} -k_g(x_r - x_{r0}) - b_g\dot{x}_r & \text{for } y_r - y_g(x_r) < 0 \\ 0 & otherwise \end{cases}
$$

$$
F_{g2} = \begin{cases} k_g(y_{r0} - y_r) - b_g f(-\dot{y}_r) & \text{for } y_r - y_g(x_r) < 0 \\ 0 & otherwise \end{cases}
$$

$$
F_{g3} = \begin{cases} -k_g(x_l - x_{l0}) - b_g\dot{x}_l) & \text{for } y_l - y_g(x_l) < 0 \\ 0 & otherwise \end{cases}
$$

$$
F_{g4} = \begin{cases} -k_g(y_l - y_{l0}) - b_g\dot{y}_l) & \text{for } y_l - y_g(x_l) < 0 \\ 0 & otherwise \end{cases}
$$