NEURAL NETWORK BASED ESTIMATOR AND CONTROLLER FOR SLIP AND TD-SLIP MONOPOD ROBOTS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

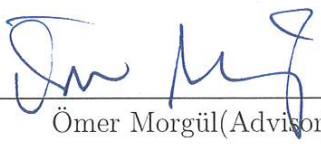
By

Ahmet Safa Öztürk

December 2020

Neural Network Based Estimator and Controller for SLIP and TD-SLIP Monopod Robots
By Ahmet Safa Öztürk
December 2020

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



Ömer Morgül(Advisor)

Hitay Özbay

İsmail Uyanık

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

NEURAL NETWORK BASED ESTIMATOR AND CONTROLLER FOR SLIP AND TD-SLIP MONOPOD ROBOTS

Ahmet Safa Öztürk
M.S. in Electrical and Electronics Engineering
Advisor: Ömer Morgül
December 2020

Using spring loaded inverted pendulum models for legged locomotion, a wide range of applications can be developed for mobile robots. Spring loaded inverted pendulum model brings new challenges of solving the forward and inverse kinematic maps. Although exact solutions of SLIP model cannot be obtained analytically due to nonintegrability of stance dynamics, approximate analytical solutions are proposed to overcome these challenges in many of the previous studies. An alternative to approximate analytical solutions, neural network based forward and inverse kinematic map predictions can also be used. We used neural networks to design estimators and controllers, as forward and inverse kinematic map predictors. Required datasets are generated with existing simulations for spring loaded inverted pendulum models and datasets are constructed by including the determined inputs and outputs for the neural networks. Neural networks are designed by using different architectures and properties. Training results of estimators for predicting the goal state in the apex return map are reported for different configurations. Trained controllers are verified using the simulation and verified controllers are tested under the different run scenarios. By comparing all of the results, potential of the neural network based estimators and controllers is discussed.

Keywords: Spring Loaded Inverted Pendulum, Legged Robots, Legged Locomotion, Neural Networks, SLIP, TD-SLIP.

ÖZET

AKTİF VEYA PASİF YAYLI TERS SARKACA SAHİP TEK BACAKLI ROBOTLAR İÇİN YAPAY SİNİR AĞI TABANLI KESTİRİCİ VE KONTROLCÜLER

Ahmet Safa Öztürk
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Prof. Dr. Ömer Morgül
Aralık 2020

Bacaklı hareket için yaylı ters sarkaç modelleri kullanılarak, mobil robotlar için geniş bir uygulama yelpazesi geliştirilebilir. Yaylı ters sarkaç modeli, ileri ve ters kinematik fonksiyonların çözümünde yeni zorlukları da beraberinde getirir. Önceki çalışmaların çogunda bu zorlukların üstesinden gelmek için yaklaşık analitik çözümler önerilmiştir. Yaklaşık analitik çözümlere alternatif olarak yapay sinir ağı tabanlı ileri ve ters kinematik fonksiyon tahminleri de kullanılabilir. İleri ve ters kinematik fonksiyonları çözmek amacıyla, kestirici ve kontrolcüleri yapay sinir ağlarını kullanarak tasarladık. Yaylı ters sarkaç modelleri için mevcut simülasyonlar ile gerekli veriler üretildi ve yapay sinir ağları için belirlenen girdi ve çıktılar dahil edilerek veri setleri oluşturuldu. Yapay sinir ağları, farklı mimariler ve özellikler kullanılarak tasarlandı. Kestiricilerin apeks dönüş fonksiyonundaki hedef durum tahminindeki öğrenme başarıları, farklı konfigürasyonlar için rapor edildi. Eğitilen kontrolcüler ise simülasyon kullanılarak doğrulandı ve doğrulanan kontrolcüler farklı koşma senaryoları altında test edildi. Tüm sonuçlar karşılaştırılarak, yapay sinir ağı tabanlı kestirici ve kontrolcülerin potansiyeli tartışıldı.

Anahtar sözcükler: Yaylı Ters Sarkaç, Bacaklı Robotlar, Bacaklı Hareket, Yapay Sinir Ağları.

Acknowledgement

Countless valuable people, contributed to my personal and academic development during my studies. First I would like to thank my advisor, Ömer Morgül, for his guidance, visionary insights and priceless support. I would also like to thank the respected members of my jury, Hitay Özbay and İsmail Uyanık, for reviewing and endorsing my work. Additionally, I would like to thank İsmail Uyanık and Uluç Saranlı, for introducing me to the wonders of robotics during the early years of my undergraduate study. I would also like to thank Auke Jan Ijspeert and Mehmet Mutlu, as the advisors of my earliest research project, which I completed in EPFL Biorobotics Laboratory.

Additionally, I would like to thank the brilliant members of Bilkent Control Theory and Robotic Locomotion Group for their assistance to get me more familiar with the robotics research environment. I am grateful to work with Serkan İslamoğlu, Eftun Orhon, Hasan Hamzaçebi, Ali Nail İnal, Bahadır Çatalbaş, Caner Odabaş, Mustafa Gül, Dilan Öztürk and Oğuz Yeğin.

I would like to thank Erdem Aras, Ömer Arol, Abdulsamet Dağaşan, Fatih Konar, Melisa Öntürk, Muzaffer Özbey, Alper Özaslan, Mahmut Can Soydan and Bilal Taşdelen, for the best times outside the school and studies.

Many precious friends also contributed to my personal development during my undergraduate and graduate studies. I would like to thank Ömer Mert Aksoy, Erdem Bıyık, Rahmetullah Çağıl, Zübeyr Furkan Eryılmaz, Gamze Gül, Fatih Karaoğlanoğlu, Sena Kıcıroğlu, Kübra Korkmaz, Deniz Onural, Batuhan Sütbaş, Mehmet Furkan Şahin, Naz Yetimoğlu and İlkay Yıldız, for making Bilkent a greater place to live and study.

I would also like to thank my friends Yasin Bilen and Esat Kalfaoğlu, for making the city of Ankara a hometown for me.

Finally, I would like to send my deepest gratitude to my parents and my brother for their priceless support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis includes neural network designs, developed and utilized on forward and inverse kinematic map predictions for a monopod robot. The monopod robot is modelled with a spring loaded inverted pendulum and running in a simulation. In previous studies, forward and inverse kinematic map predictions are done by using approximate analytical solutions [3, 4, 5]. In our study we aimed to show that, the neural network based designs can become promising replacements for conventional methods. In addition to the success of the neural networks on predicting forward and inverse kinematic maps, behaviors of the neural network based controllers in different scenarios are also demonstrated.

Locomotion is one of the greatest challenges in the field of robotics. From Shakey [6] to Cheetah [7], various methods are proposed for the mobile robot locomotion. Among the different locomotion methods, wheeled or tracked systems are the most common ways among the mobile robots [8]. On the contrary, main inspiration for the mobile robots comes from the nature and the locomotion systems in animals and humans are mainly based on legs. So, for developing a more comprehensive locomotion system, using legs instead of wheels are proposed [1, 8].

Mobile robots with legged locomotion performs better when all of surface types

on the Earth is taken into consideration [8, 9]. Legged morphologies presents the opportunity of locomotion in 3D space, where wheeled robots are only capable of moving on a 2D surface. Legged robots are also suitable for use in plenty of environments, where humans cannot reach or endure [8]. These environments include but not limited to outer space [10, 11, 12]; volcanoes [13]; ocean floor [14]; nuclear power plants [15]; mine fields [16]; and most importantly areas struck by disasters [17].

Legged locomotion also offers a broader range of robot morphologies and dynamics; including **Monopod Robots** like the very first modern monopod built by Matsuoka [18], SLIP Monoped [19], ARL-monopod II [20] or our Monopod Robot built in Bilkent CTRL [21, 22]; **Biped or Humanoid Robots** like HRP-2L [23] or ASIMO [24]; and **Multilegged Robots**: like RHex [25] or Cheetah [7].

Among the proposed models for legged locomotion dynamics, spring loaded inverted pendulum (SLIP) is a successful method to represent the running behavior naturally [1]. Spring loaded inverted pendulum model includes a point mass actuated with a leg equipped with a string. Leg spring stiffness determines the elasticity of the leg while running, corresponding the muscle elasticity in humans or animals. Damping can also be included in order to model the energy loss during running. If there is a damping loss in the system, energy can be recovered by applying a hip torque and this version of SLIP model is called Torque Actuate Dissipative SLIP Model (TD-SLIP). With TD-SLIP model, run mechanism can be explained more correctly, since energy loss is inevitable in real life situations [26]. To illustrate, the spring loaded inverted pendulum model for running behavior is shown in Figure 1.1.

Figure 1.1: Representation of Running with Spring Loaded Inverted Pendulum [1, 2]

Various applications are developed by embedding spring loaded inverted pendulum model. One of the most important applications of SLIP model in legged locomotion is deriving approximate analytical solutions [3, 26, 27, 28]. Another application is system identification on SLIP system [22, 29, 30, 31]. Several other applications are also developed based on SLIP model, like adaptive control [21,32]; experimental validation [4]; energy regulation [5]; gait design [33,34]; motion planning [35]; and designing different morphologies [36].

When we look at the literature most of the proposed methods for SLIP and TD-SLIP models include approximate analytical solutions. Conversely, with the emergence of artificial intelligence methods including neural networks, developing the AI counterparts of conventional methods, became more applicable. Access to wider and bigger datasets and better computational equipment as well as a broad range of methods, facilitates the design and development of neural networks for different tasks. In our study our focus is to develop neural network counterparts of the approximate analytical solutions for SLIP and TD-SLIP models. Reviewing the literature, we observe that neural networks are widely used for robot control [37, 38, 39]; including legged locomotion [40]. Moreover, there are not enough

studies also focusing on neural networks for SLIP model, indeed we have found only one study where neural networks are used to design estimators or controllers for SLIP or TD-SLIP models [41]. In [41] neural networks are used to design controllers for SLIP model. Comparing with [41], we focused on designing neural networks with deeper architectures and more configurations. We also focus on using neural networks to design estimators and controllers for SLIP and TD-SLIP models.

The main contribution of this thesis is, showing the potential of neural networks on SLIP based monopod robots. Our study shows that; neural network based estimators and controllers, brings promising results for the monopod robots modeled with SLIP. We provide the results of neural networks for predicting the forward and inverse kinematic maps, where the monopod robot is modeled with SLIP and TD-SLIP models.

The second contribution is, introducing neural network designs for both SLIP and TD-SLIP models by extending the existing studies. Our results show that, different type of estimators and controllers based on neural networks, can be used to predict the forward and inverse kinematic maps. We provide the success of neural network based estimators and controllers with different configurations and designs.

Final contribution is, testing the designed neural network estimators and controllers under different scenarios. Our tests show that, neural networks can be used to run SLIP and TD-SLIP monopods for different cases. We provide the responses of neural network based estimators and controllers against several scenarios.

Thesis organization for the following chapters are detailed below:

- In **chapter 2**, mathematical foundations of SLIP and TD-SLIP models are detailed. Equations of motions obtained with approximate analytical solutions are given. Usage of SLIP and TD-SLIP simulations are also explained.

- In **chapter 3**, details about the neural network based estimators and controllers developed for SLIP model, are given. Neural network input and output configurations, architectures and properties are explained. Data generation steps with SLIP simulator are clarified. Achieved single stride results on forward and inverse kinematic maps are reported and discussed.

- In **chapter 4**, similar structure with chapter 3 is used to explain the details about the neural network based estimators and controllers developed for TD-SLIP model. Neural network input and output configurations and data generation steps with TD-SLIP simulator are clarified. Achieved single stride results on forward and inverse kinematic maps are reported and discussed.

- In **chapter 5**, trained estimators and controllers are tested against various run scenarios and achieved results are reported. Ideas about different type of controllers are discussed.

- In **chapter 6**, overall discussion about the aimed and achieved results are summarized. Potential ideas about the future work on neural network based estimators and controllers for SLIP and TD-SLIP models, are also mentioned.

Chapter 2

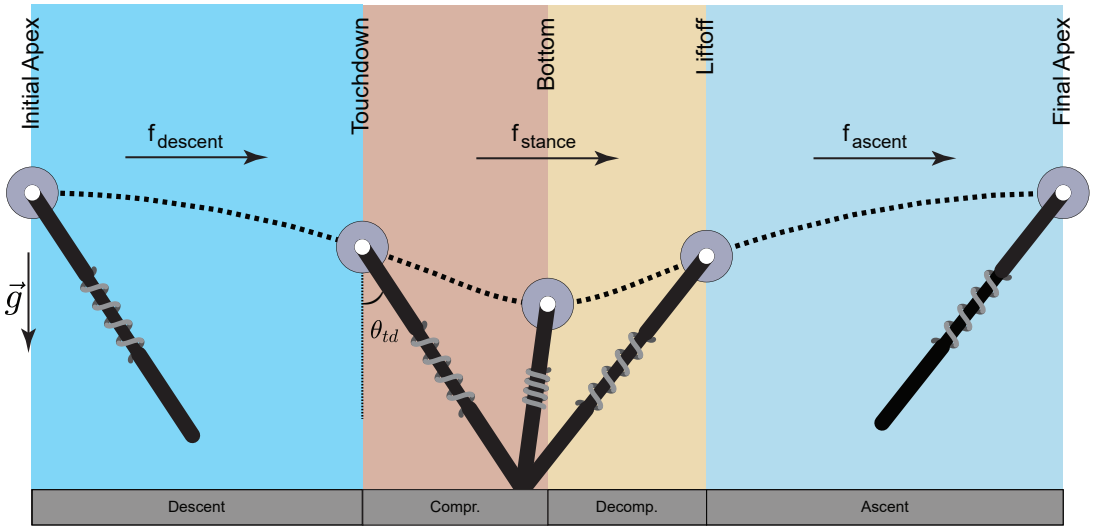
SLIP and TD-SLIP Models

SLIP and TD-SLIP models are based on apex return maps including a single stride of the monopod. Single stride of the monopod consists of four phases and five events pointing the start and the end of these phases. In a single stride movement, monopod starts from an apex point with a horizontal velocity and vertical position which is higher than the maximum leg length. Horizontal position is not important for a single stride but it can be used for multiple consecutive strides. Vertical velocity is zero by the definition of an apex point. Ignoring the air damping, horizontal acceleration is zero and vertical acceleration is equal to the gravitational acceleration of the environment. Starting from this apex point, which is the first event and will be referred as initial apex, monopod follows a ballistic trajectory until the toe of the leg touches the ground. When the toe of the leg touches the ground, touchdown event occurs. The phase between the initial apex and touchdown events is the descent phase. With the touchdown event, leg length starts to decrease because the spring in the leg is being compressed by the gravity. The leg spring is compressed until a point where force applied by gravity becomes equal to the opposite spring force. In this point the leg length has the smallest value and this event is called the bottom. With the bottom event the leg spring starts being decompressed and the leg length starts to increase. When the leg length reaches to its fully decompressed value, the toe of leg lifts off from the ground. This event is called the liftoff and monopod starts

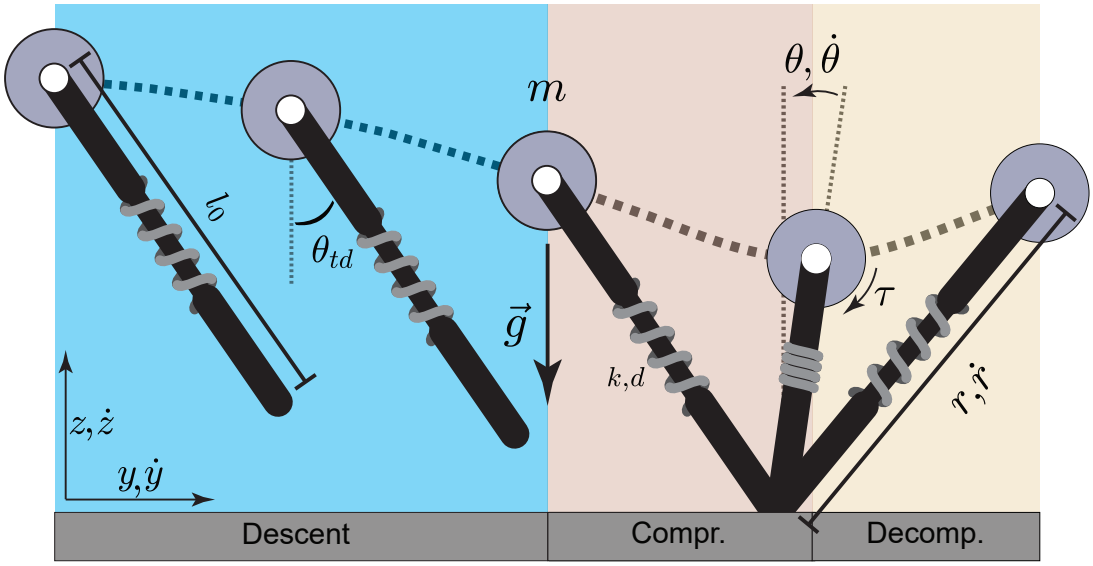
to follow a ballistic trajectory again. When the vertical velocity of the monopod reaches to zero, another apex event occurs and single stride is completed. This event will be called the final apex and the phase between the liftoff and the final apex events, is ascent. In general descent and ascent phases are referred as flight phase, compression and decompression phases are referred as stance phase [3]. In Table 2.1, apex return map variables, control inputs and physical parameters are defined. In Figure 2.1 events and phases of the apex return map is demonstrated.

Table 2.1: State Variables, Stance Positions, Control Inputs and Physical Properties Used in Apex Return Map

	Symbol	Definition
Cartesian Body Positions and Velocities	$X_i : [\dot{y}_i \ z_i]$	Initial Apex State
	$X_{td} : [y_{td} \ \dot{y}_{td} \ z_{td} \ \dot{z}_{td}]$	Touchdown State
	$X_{lo} : [y_{lo} \ \dot{y}_{lo} \ z_{lo} \ \dot{z}_{lo}]$	Liftoff State
	$X_f : [y_f \ \dot{y}_f \ z_f]$	Final Apex State
Polar Coordinates During Stance	$r, \ \dot{r}$	Leg Length and Velocity
	$\theta, \ \dot{\theta}$	Leg Angle and Velocity
Control Inputs	θ_{td}	Touchdown Agnle
	τ_0	Initial Hip Torque
Physical Properties	l_0	Leg Rest Length
	m	Mass of the Monopod
	k	Spring Constant
	d	Damping Coefficient
	g	Gravitational Acceleration



(a) Apex Return Map



(b) Descent and Stance

Figure 2.1: Apex Return Map Events and Monopod Dynamics During Phases. Ascent Phase is not Shown, since Ascent and Descent Dynamics are Equivalent

2.1 Equations of Motion

Monopod states in the apex return map are detailed in Table 2.1 for initial apex, touchdown, liftoff and final apex events. Bottom event is not included since it is not used in the approximate analytical solutions or our study. In Table 2.1, y and z denote the horizontal and vertical positions, $\dot{y}$ and $\dot{z}$ denote the horizontal and vertical velocities. Note that $\dot{z}$ is not included in apex states, since it is zero by the definition.

Flight dynamics are derived with projectile motion and given in Equation 2.1.

$$Flight\,(descent\,\&\,ascend): \begin{bmatrix} \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ -g \end{bmatrix} \tag{2.1}$$

In Equation 2.1; $\ddot{y}$ and $\ddot{z}$ denote the horizontal and vertical accelerations. Since the only force affecting the monopod is gravity, $\ddot{y}$ is zero and $\ddot{z}$ is equal to negative gravitational acceleration [3]. Air damping is neglected for this study. By using Equation 2.1 with the touchdown angle; descent and ascent functions can easily be calculated. In Equation 2.2, descent function that brings the monopod from initial apex to touchdown state and in Equation 2.3, ascent function that brings the monopod from liftoff to final apex state is given.

$$\begin{aligned} X_{td} &= f_{descent}(X_i) \\ y_{td} &= \dot{y}_i t_d \\ \dot{y}_{td} &= \dot{y}_i \\ z_{td} &= l_0 \cos\theta_{td} \\ \dot{z}_{td} &= -g t_d \\ t_d &= \sqrt{2(z_i - z_{td})/g} \end{aligned} \tag{2.2}$$

In Equation 2.2, $f_{descent}$ denotes the descent function that brings the monopod from initial apex to touchdown state. l_0 is the leg rest length and θ_{td} is the touchdown leg angle. t_d is the time passed during the descent phase.

$$
\begin{aligned}
X_f &= f_{ascent}(X_{lo}) \\
y_f &= y_{lo} + \dot{y}_{lo} t_a \\
\dot{y}_f &= \dot{y}_{lo} \\
z_f &= z_{lo} + g {t_a}^2/2 \\
t_a &= \dot{z}_{lo}/g
\end{aligned}
\tag{2.3}
$$

In Equation 2.3, f_{ascent} denotes the ascent function that brings the monopod from liftoff to final apex state. t_a is the time passed during the ascent phase.

Stance dynamics are derived with Lagrangian mechanics and given in Equation 2.4.

$$
\textit{Stance}\,(\textit{compr}\,\&\,\textit{decomp}) : \frac{d}{dt}\begin{bmatrix} m\dot{r} \\ mr^2\dot{\theta} \end{bmatrix} = \begin{bmatrix} mr\dot{\theta}^2 - mg\cos\theta - k(r - r_0) - d\dot{r} \\ mgr\sin\theta + \tau \end{bmatrix} \tag{2.4}
$$

In Equation 2.4, the polar coordinates are utilized instead of Cartesian coordinates, since during the stance phase toe of the leg is stationary. τ is the applied hip torque during the stance [5, 22]. τ is defined by Equation 2.5.

$$
\tau(t_c) \quad = \begin{cases} \tau_0(1 - t_c/t_s), & \text{if } 0 \leq t_c \leq t_s \\ 0, & \text{otherwise} \end{cases} \tag{2.5}
$$

In Equation 2.5, $\tau(t_c)$ denotes the applied torque for the given time, where τ_0 is the initial value of the applied torque. t_c is the duration passed since touchdown and t_s is the total expected duration of the stance phase. This ramp profile is used for τ, since it can be simply integrated in approximate analytical solutions [5, 22].

Difference between SLIP and TD-SLIP models is that; in SLIP model leg damping, d, is assumed to be zero and hip torque, τ, is not used. So for the SLIP model, stance dynamics given in Equation 2.4 are used as $d = 0$ and $\tau = 0$. On

the contrary; in TD-SLIP model, both d and τ can take nonzero values. When d is a nonzero value, total mechanical energy of the monopod decays with time. To compensate for this energy loss, τ is applied to regulate the energy of the monopod robot.

Since the stance dynamics given in Equation 2.4 are non-integrable [42], approximate solutions are used to find the monopod positions during stance. Assuming θ and Δr are very small, approximate variables in Equation 2.6 are defined.

$$\begin{aligned} p_\theta &:= mr^2\dot{\theta} \\ \omega_0 &:= \sqrt{k/m} \\ \omega &:= p_\theta/(mr_0^2) \\ \hat{\omega}_0 &:= \sqrt{\omega_0^2 + 3\omega^2} \\ \xi &:= d/(2m\hat{\omega}_0) \\ \omega_d &:= \hat{\omega}_0\sqrt{1-\xi^2} \\ F &:= -g + r_0\omega_0^2 + 4r_0\omega^2 \end{aligned} \tag{2.6}$$

In Equation 2.6; ω, ω_0, $\hat{\omega}_0$ and ω_d are frequency values, p_θ is the angular momentum, ξ is the damping ratio and F is the force term [5, 22]. Calculating polar coordinates of the monopod body by using the variables from Equation 2.6, is shown in Equation 2.7.

$$\begin{aligned} r_t &:= Me^{-\xi\hat{\omega}_0 t}\cos(\omega_d t + \phi_1) + F/\hat{\omega}_0^2 \\ \dot{r}_t &:= -M\hat{\omega}_0 e^{-\xi\hat{\omega}_0 t}\cos(\omega_d t + \phi_1 + \phi_2) \\ \theta_t &:= \theta_{td} + Xt + Y(e^{-\xi\hat{\omega}_0 t}\cos(\omega_d t + \phi_1 - \phi_2) - \cos(\phi_1 - \phi_2)) \\ \dot{\theta}_t &:= 3\omega - 2\omega F/(r_0\hat{\omega}_0^2) - 2\omega M e^{-\xi\hat{\omega}_0 t}\cos(\omega_d t + \phi_1)/r_0 \end{aligned} \tag{2.7}$$

In Equation 2.7, formulas for r, $\dot{r}$, θ and $\dot{\theta}$ are given. Note that, X is not related to the monopod states. M, X, Y, ϕ_1 and ϕ_2 are defined in Equation 2.8.

$$
\begin{aligned}
A &:= r_0 - F/\hat{\omega}_0^2 \\
B &:= (\dot{r}_{td} + \xi\hat{\omega}_0 A)/\omega_d \\
M &:= \sqrt{A^2 + B^2} \\
X &:= 3\omega - 2\omega F/(r_0\hat{\omega}_0^2) \\
Y &:= 2\omega M/(r_0\hat{\omega}_0) \\
\phi_1 &:= \arctan(-B/A) \\
\phi_2 &:= \arctan(-\sqrt{1-\xi^2})/\xi
\end{aligned}
\tag{2.8}
$$

Radial positions for the touchdown $[r_{td}\ \dot{r}_{td}\ \theta_{td}\ \dot{\theta}_{td}]$ can be calculated from X_{td} : $[y_{td}\ \dot{y}_{td}\ z_{td}\ \dot{z}_{td}]$. After finding the liftoff radial positions, $[r_{lo}\ \dot{r}_{lo}\ \theta_{lo}\ \dot{\theta}_{lo}]$, radial to Cartesian conversion can be use to find X_{lo} : $[y_{lo}\ \dot{y}_{lo}\ z_{lo}\ \dot{z}_{lo}]$. To summarize, apex return map found with approximate analytical solutions can be given as Equation 2.9.

$$
X_f = (f_{ascent} \circ C \circ f_{stance} \circ R \circ f_{descent})(X_i) \tag{2.9}
$$

Note that, in this study approximate analytical solutions for the stance dynamics are not used. Above calculations are only given to reference the existing way of finding the forward kinematic map. Variables defined in Equation 2.6 and Equation 2.8 are not also used in the following chapters. Further details about the flight and stance dynamics can be found in [5,22]. Derivation of the equations can also be found in [3, 34, 43].

2.2 SLIP and TD-SLIP Model Simulations

A monopod simulation based on the flight and stance dynamics given above, is implemented by [3, 34]. Instead of approximate analytical maps, numerical solutions are used in the simulation. Simulation generates apex return maps with given initial conditions, control inputs and physical parameters. By applying the required changes for stance dynamics given in Equation 2.4, we can switch between SLIP and TD-SLIP models. Inputs for the simulation are detailed below:

Physical Parameters:

- m: Total mass of the Monopod
- l_0: Leg length when the spring is resting
- k: Leg spring constant
- d: Leg damping coefficient (if TD-SLIP model is used)
- g: Gravitational acceleration

Initial Conditions:

- $\dot{y}_i$: Initial apex horizontal velocity
- z_i: Initial apex vertical position

Control inputs:

- θ_{td}: Touchdown angle of the leg
- τ_0: Initial hip torque (if TD-SLIP model is used)

By setting the above values, apex return maps can be generated with the simulation. Simulation generates the whole apex return map, but in this study only initial apex, touchdown, liftoff and final apex state variables are used.

Chapter 3

Neural Network Based Estimator and Controller for SLIP Model

For the SLIP model, Neural Networks are trained to predict forward and inverse kinematic maps. Before generating any data, the inputs and outputs of the neural networks are determined. Then, SLIP simulation is used to generate single stride maps with given apex states and control inputs. From the generated single stride maps; initial apex, touchdown, liftoff and final apex states are recorded with control inputs. After constructing dataset from these recorded state variables and parameters, training of neural networks are completed with different network architectures and training parameters. Trained neural networks are tested for training success and used with the SLIP simulation for further examination.

3.1 Input and Output Configuration for Neural Networks

Neural networks are named according to the prediction task on the kinematic maps. Forward Kinematic Map Predictors are called **Estimators** and Inverse Kinematic Map Predictors are called **Controllers**.

3.1.1 Estimator Inputs and Outputs

Two type of Estimators are trained for forward kinematic map. First type of Estimator is trained for complete apex to apex return map and second type of estimator is trained for only stance return map.

3.1.1.1 Apex to Apex Forward Kinematic Map Predictor (a2a-Estimator)

Expected task from Apex to Apex Forward Kinematic Map Predictor is to estimate the next apex state where initial apex state and control inputs are given. In other words Apex to Apex Forward Kinematic Map Predictor is the neural network equivalent of SLIP Simulator. So that the inputs for the Apex to Apex Forward Kinematic Map Predictor will be initial apex state X_i and control inputs u and output will be next apex state X_f.

- *a2a-Estimator Inputs:* $[X_i\ u]$, where $X_i : [y_i\ \dot{y}_i\ z_i\ \dot{z}_i]$ and $u : [\theta_{touchdown}]$. Since $y_i = 0$ and $\dot{z}_i = 0$, X_i becomes $X_i : [\dot{y}_i\ z_i]$ and the input used for a2a-Estimator is $I_{a2a-Estimator} : [\dot{y}_i\ z_i\ \theta_{touchdown}]$.
- *a2a-Estimator Outputs:* $[X_f]$, where $X_f : [y_f\ \dot{y}_f\ z_f\ \dot{z}_f]$. Since $\dot{z}_f = 0$, X_f becomes $X_f : [y_f\ \dot{y}_f\ z_f]$ and the output expected from a2a-Estimator is $O_{a2a-Estimator} : [y_f\ \dot{y}_f\ z_f]$.

3.1.1.2 Touchdown to Liftoff Forward Kinematic Map Predictor (t2l-Estimator)

Since the calculation of touchdown state variables from initial apex state and final apex state variables from liftoff state can be done by ballistic trajectories with absolute theoretical certainty (discussed in **section 2.1**); a second type of estimator can be trained using touchdown and liftoff states, instead of initial and final apex states. Expected task from Touchdown to Liftoff Forward Kinematic

Map Predictor is to estimate the liftoff state where touchdown state and control inputs are given. So that the inputs for the Touchdown to Liftoff Forward Kinematic Map Predictor will be touchdown state X_{td} and control inputs u and output will be liftoff state X_{lo}.

- *t2l-Estimator Inputs:* $[X_{td}\ u]$, where $X_{td} : [y_{td}\ \dot{y}_{td}\ z_{td}\ \dot{z}_{td}]$ and $u : [\theta_{touchdown}]$. So the input used for t2l-Estimator is $I_{t2l-Estimator} : [y_{td}\ \dot{y}_{td}\ z_{td}\ \dot{z}_{td}\ \theta_{touchdown}]$.
- *t2l-Estimator Outputs:* $[X_{lo}]$, where $X_{lo} : [y_{lo}\ \dot{y}_{lo}\ z_{lo}\ \dot{z}_{lo}]$ and the output expected from t2l-Estimator is $O_{t2l-Estimator} : [y_{lo}\ \dot{y}_{lo}\ z_{lo}\ \dot{z}_{lo}]$.

Block diagram demonstrating the process of using a t2l-Estimator to predict Apex to Apex Forward Kinematic Map, is shown in Figure 3.1.

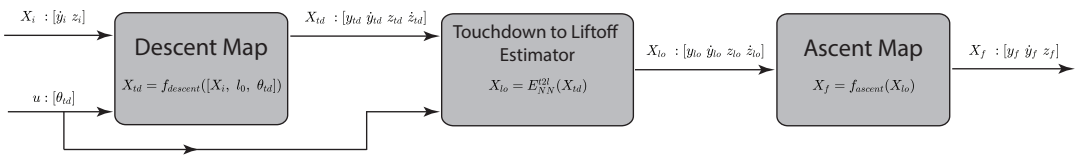


Figure 3.1: Block Diagram Demonstrating the Process of Apex to Apex Forward Kinematic Map Prediction with t2l-Estimator

In Figure 3.1, a trained SLIP t2l-Controller, E^{t2l}_{NN}, is used to predict the final apex state, X_f, from initial apex state, X_i, and control inputs, u. Touchdown state is calculated from X_i with $f_{descent}$. Then liftoff state, X_{lo}, is predicted from X_{td} with E^{t2l}_{NN}. Finally, final apex state, X_f, is calculated from X_{lo} with f_{ascent}. Details about the $f_{descent}$ and f_{ascent} can be revisited in section 2.1. By following this process apex to apex errors can be calculated, in addition to touchdown to liftoff errors.

3.1.2 Controller Inputs and Outputs

Since there is no inverse function for liftoff to final apex trajectory (ie. calculating a unique liftoff state using final apex state is not possible), only Apex to Apex Inverse Kinematic Map Predictor is trained.

3.1.2.1 Apex to Apex Inverse Kinematic Map Predictor (a2a-Controller)

Expected task from Apex to Apex Inverse Kinematic Map Predictor is to predict the control inputs that brings the SLIP monopod from an initial state to a final (desired) apex state. In other words Apex to Apex Inverse Kinematic Map Predictor is a neural network controller that can be used for SLIP simulation. The inputs for the Apex to Apex Inverse Kinematic Map Predictor will be initial and final apex states, X_i and X_f, and the output will be control inputs u.

- *a2a-Controller Inputs:* $[X_i \ X_f]$, where $X_i : [\dot{y}_i \ z_i]$ and $X_f : [y_f \ \dot{y}_f \ z_f]$. So the input used for a2a-Controller is $I_{a2a-Controller} : [\dot{y}_i \ z_i \ y_f \ \dot{y}_f \ z_f]$.
- *a2a-Controller Outputs:* $[u]$, where $u : [\theta_{touchdown}]$. So the output expected from a2a-Controller is $O_{a2a-Controller} : [\theta_{touchdown}]$.

3.2 Data Generation with SLIP Model

Data generation is done with the SLIP Model Simulation with the goal of recording determined inputs and outputs for the neural networks. As explained in **section 2.2**, simulation generates apex return map with starting apex position, control inputs and physical constants. Physical constants are selected from the properties of the monopod robot in our lab, to have more realistic results and measure the usability of the neural networks with physical systems. Having apex

return maps generated, inputs and outputs required for neural networks, i.e. SLIP dataset is complete. Physical constants are also recorded for future reference.

3.2.1 Selected Parameters for Data Generation

For each initial apex state variable and control parameter, a range based on physical limits is selected. A random initial apex state with control inputs used to start the simulation. Since some of these random starting points returns an unrealistic return map, the number of generated apex return maps is increased. Parameters required for data generation are initial apex state variables $X_i : [\dot{y}_i \; z_i]$, control inputs $u : [\theta_{touchdown}]$ and physical constants m, l_0, k, g.

Physical characteristics of the SLIP Monopod Robot:

- $m = 3.005 \; kg$: Total mass of the SLIP Monopod
- $l_0 = 0.205 \; m$: Leg length when the spring is resting
- $k = 9000 \; N/m$: Leg spring constant
- $g = 9.81 \; m/s^2$: Gravitational acceleration

Total mass of the TD-SLIP Monopod, m, and leg rest length, l_0, are the exact measurements taken from the monopod robot in our lab, where leg spring constant, k, is an approximate value calculated based on previous experiments [22].

Initial apex state variables and control inputs are selected as a uniformly distributed random number from the given range:

- $\dot{y}_i$: Horizontal velocity of the SLIP monopod body. $\dot{y}_i$ is selected between 0 m/s and 3 m/s. Velocity is limited to 3 m/s, since in the physical system going faster then this upper limit may result a hazardous situation for the monopod robot and the operators. Lower bound is 0 m/s, since the SLIP monopod goes forward.

- z_i: Vertical position of the SLIP monopod body. z_i is selected from l_0 to $2l_o$. Lower bound is l_0, since the SLIP monopod is at the apex position and the spring is resting and upper bound is $2l_0$ since it is the approximate upper limit the physical SLIP monopod reaches.

- $\theta_{touchdown}$: Angle of the leg with the z axis when touchdown occurs. $\theta_{touchdown}$ is selected from 10° to 40°. Narrower angles than 10° results a back-jump and wider angles than 40° increases the risk of toe sliding during the stance.

3.2.2 Dataset Construction

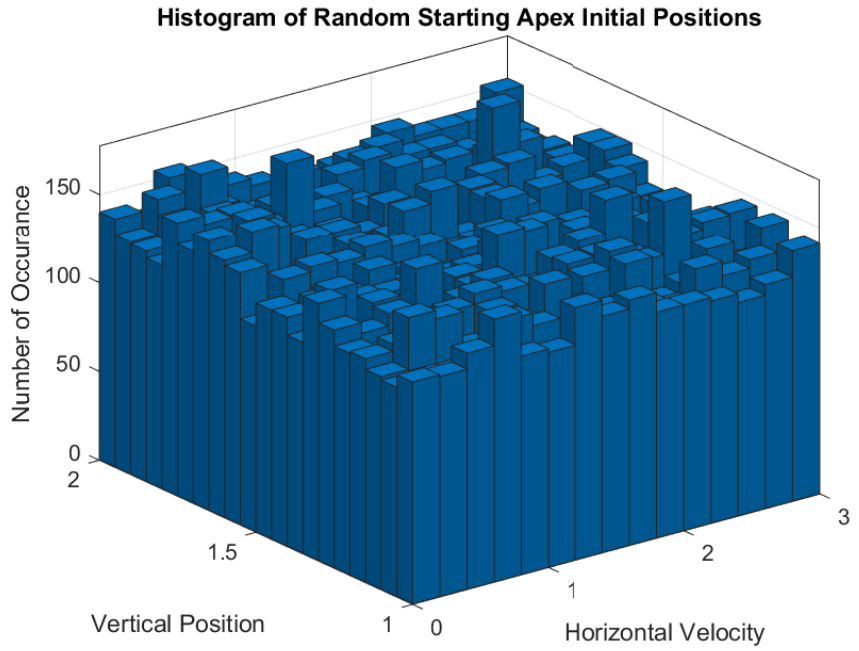


Figure 3.2: Histogram of Random Starting Apex Initial Positions

In [41], dataset size 30000. By referencing this number, SLIP model simulation is used to generate **40000** apex return maps with the selected parameters from the determined range. Histogram of selected initial apex state variables, is shown in **Figure 3.2**. Uniformity tests are not applied over randomly selected parameter

ranges, since some of these parameters will result unrealistic maps and cannot be used for the training. Total number of apex return maps without any back jumps or unrealistic events is **18598**. Histogram of initial apex state variables which result utilizable apex return maps, is shown in **Figure 3.3**. It can be seen that the distribution of initial apex state variables in Figure 3.3 is nonuniform. For the control inputs the most populated bin of Figure 3.3 is observed and shown in **Figure 3.4**. As it can be observed when the the SLIP monopod body is slower in the horizontal direction or the perpendicularity of the SLIP monopod leg to the ground is increased, risk of an unrealistic apex return map increases. Final apex state variables of Figure 3.3 are also shown in **Figure 3.5**. Some final apex positions are also found to be unrealistic in terms of the SLIP body position. So that the apex return maps with final apex vertical positions are higher than 3 l_0 are also excluded from the training. Total number of apex return maps included in SLIP dataset is **17199**.

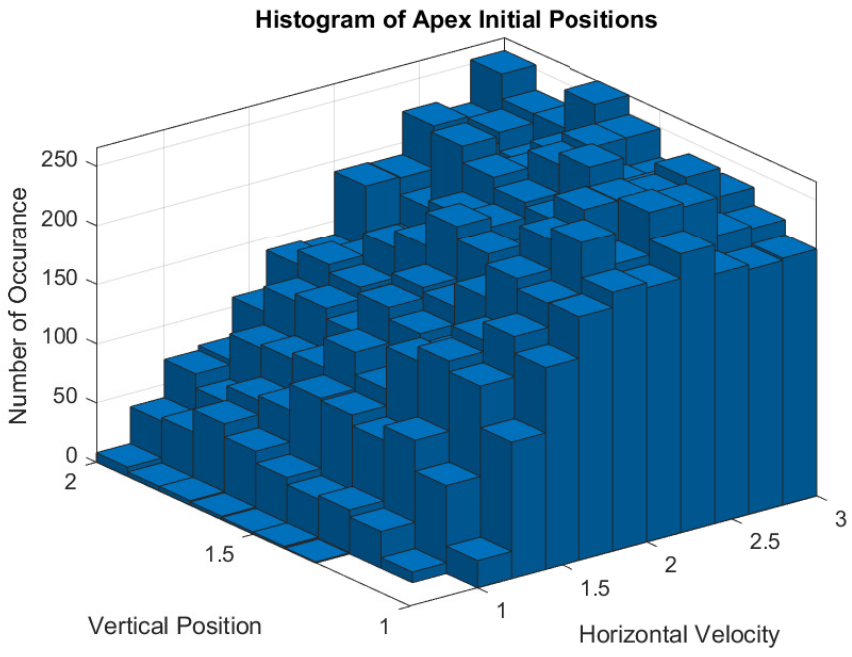


Figure 3.3: Histogram of Apex Initial Positions

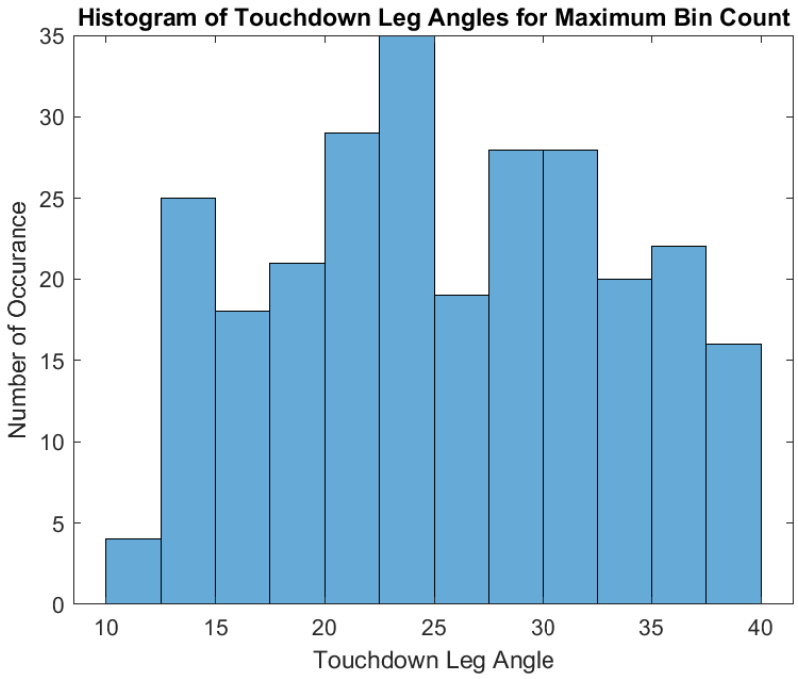


Figure 3.4: Histogram of Touchdown Leg Angles for Maximum Bin Count

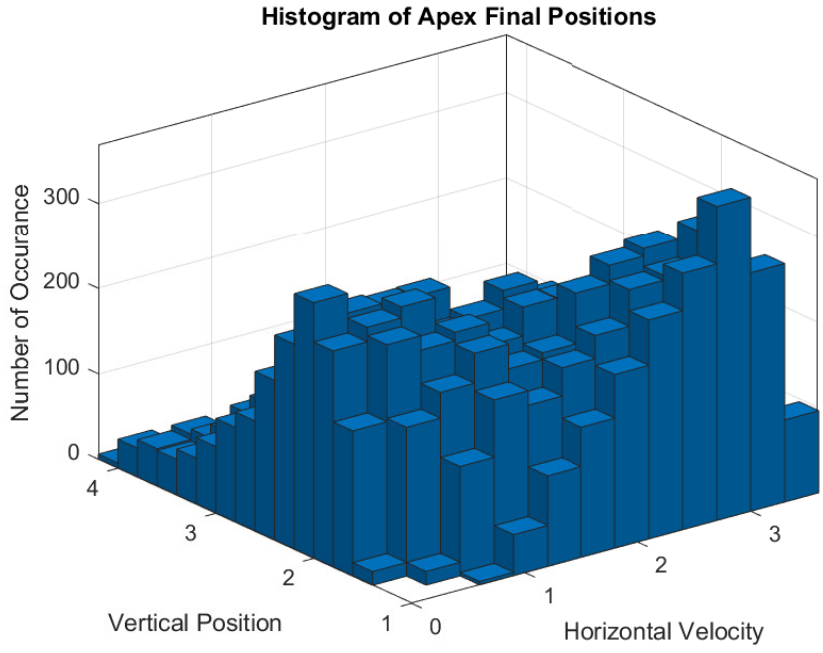


Figure 3.5: Histogram of Apex Final Positions

From the selected apex return maps, required inputs and outputs defined in **section 3.1** are recorded in separate vectors for each apex return map event (initial apex, touchdown, liftoff and final apex). Recorded inputs and outputs are normalized between -1 and 1. Normalization is applied for each state variable and control parameter separately but including all the occurrences of a specific state variable in all of the events. In detail: y is normalized among $[y_{td}\ y_{lo}\ y_f]$, $\dot{y}$ is normalized among $[\dot{y}_i\ \dot{y}_{td}\ \dot{y}_{lo}\ \dot{y}_f]$, z is normalized among $[z_i\ z_{td}\ z_{lo}\ z_f]$ and $\dot{z}$ is normalized among $[\dot{z}_{td}\ \dot{z}_{lo}]$.

After normalization each input and output set is also divided into training, validation and test sets before training. With 3 subset of state variables for 4 events in the apex return map, final dataset includes 12 set of state variables and 3 set of control inputs.

3.3 Neural Networks Architectures and Properties

Using different tools and approaches, several neural network generations are developed. The approaches used to develop neural networks include: building a standalone library from the ground up, using Regression Learner App from MATLAB Statistics and Machine Learning Toolbox [44] and creating complex designs with MATLAB Deep Learning Toolbox [45]. Testing with these approaches showed that neural networks developed with MATLAB Deep Learning Toolbox give more persistent and reliable results. While designing neural networks; from shallow to deeper model architectures are experimented, different type of activation functions and cost functions are also used. For each task some of the parameters are also re-optimized and some of the parameters changed according to the architecture of the neural network.

3.3.1 Layer Structure

All of the tasks for the neural networks includes sequence inputs and require sequence outputs, in other words all the neural networks are regression networks. Input layers are sequence input layers and output layers are regression layers. For the hidden layers fully connected layers are used and the number of neurons in each hidden layer follows a pyramidal sequence: for the initial half of the hidden layers the number of neurons doubles and for the last half of the hidden layers the number of neurons halves. An example of a designed neural network structure is given in **Figure 3.6**. This structure is selected since the number of the initial inputs are limited. Using this structure neural networks are also aimed to function as a feature extractor. Keeping the same pattern for the number of neurons, neural networks with different number of hidden layers are tested.

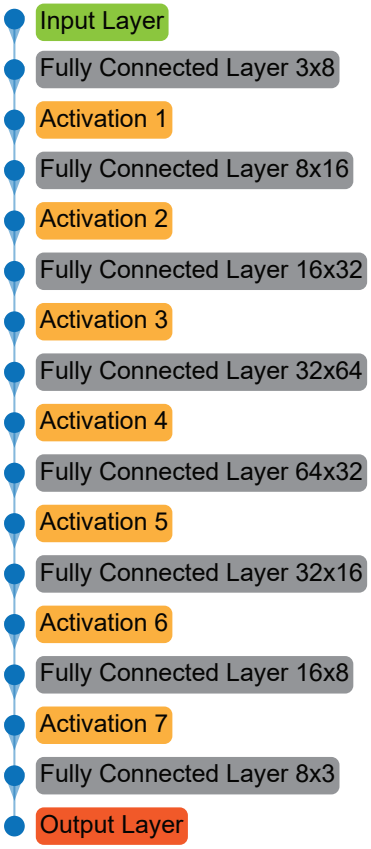


Figure 3.6: Layer Structure of the Used Neural Network Template

3.3.2 Loss Functions Used for Output Layers

Two different loss functions are used in output layers. First one is Half Mean Squared Error or Mean Squared Error and the second one is Mean Absolute Error Ratio. Half Mean Squared Error is used to measure an overall success of the training, since it is a common loss function used for the regression tasks. Conversely, the goal is minimization of the percentage errors on the results. To do that Mean Absolute Error Ratio used as the major indicator of the success for experiments. Regardless of the loss function used in the output layer, Mean Absolute Error Ratio is separately reported for all trained neural networks.

- Formula used to calculate the Half-Mean Squared Error on the outputs of the neural network is given in Equation 3.1 and the partial derivative of the HMSE respect to neural network outputs is given in Equation 3.2. These formulas are not implemented separately since they are already included in the `RegressionLayer` object of MATLAB Deep Learning Toolbox [45].

$$\mathcal{L}_{HMSE} = \frac{1}{2NM} \sum_{n=1}^{N} \sum_{m=1}^{M} (Y_{nm} - T_{nm})^2 \tag{3.1}$$

$$\frac{\partial \mathcal{L}}{\partial Y_{nm}} = \frac{Y_{nm} - T_{nm}}{M} \tag{3.2}$$

  In Equation 3.1 and 3.2: $\mathcal{L}_{HMSE}$ represents the HMSE loss value calculated using all of the output values, while $\frac{\partial \mathcal{L}}{\partial Y_{nm}}$ represents the partial derivative of the loss value, $\mathcal{L}_{HMSE}$, respect to a single output value Y_{nm}, where n is the current training sample and m is the current feature of that training sample. Y represents the neural network outputs, T represents the training targets, N is the number of total training samples and M is the number of features in each training sample.

- Formula used to calculate the Mean Absolute Error Ratio on the outputs of the neural network is given in Equation 3.3 and the partial derivative of the MAER respect to neural network outputs is given in Equation 3.4 and Equation 3.5. These formulas are implemented as a part of a new type of

output layer object and the object is linked to MATLAB Deep Learning Toolbox.

$$\mathcal{L}_{MAER} = \frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M}\left|\frac{Y_{nm}-T_{nm}}{T_{nm}}\right| \tag{3.3}$$

$$\frac{\partial\mathcal{L}}{\partial Y_{nm}} = \frac{Y_{nm}-T_{nm}}{|Y_{nm}-T_{nm}||T_{nm}|M} \tag{3.4}$$

$$\frac{\partial\mathcal{L}}{\partial Y_{nm}} = \frac{sgn(Y_{nm}-T_{nm})}{|T_{nm}|M} \tag{3.5}$$

In Equation 3.3 and 3.4: $\mathcal{L}_{MAER}$ represents the MAER loss value calculated using all of the output values, while $\frac{\partial\mathcal{L}}{\partial Y_{nm}}$ represents the partial derivative of the loss value, $\mathcal{L}_{MAER}$, respect to a single output value Y_{nm}, where n is the current training sample and m is the current feature of that training sample. Y represents the neural network outputs, T represents the training targets, N is the number of total training samples and M is the number of features in each training sample. Equation 3.5 shows the simplified form of the Equation 3.4.

3.3.3 Activation Functions Used for Hidden Layers

Three type of activation functions are used in our neural network designs: Hyperbolic Tangent Function, Exponential Linear Units and Leaky Rectified Linear Units. Formulas for Hyperbolic Tangent Function, Exponential Linear Units and Leaky Rectified Linear Units are given in Equation 3.6, 3.7 and 3.8 respectively. Responses of each activation function are also shown in Figure 3.7, 3.8 and 3.9.

$$a_l = \ f_{tanh}(z_l) = \frac{e^{2z_l}-1}{e^{2z_l}+1} \tag{3.6}$$

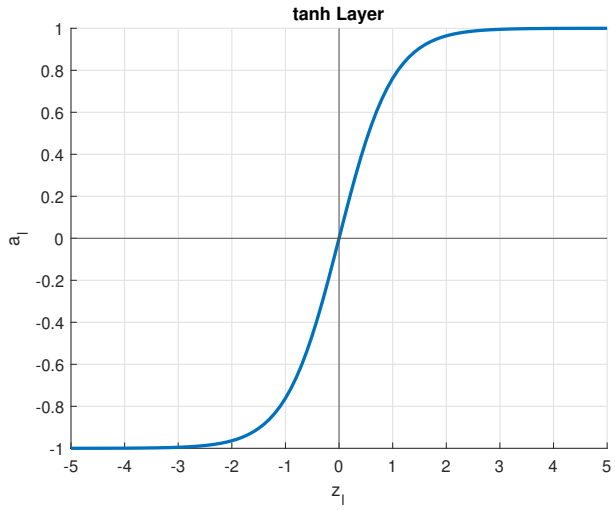


Figure 3.7: Hyperbolic Tangent Activation Function

$$a_l = \ f_{eLU}(z_l) = \begin{cases} z_l, & z_l \geq 0 \\ \alpha(e^x - 1), & z_l < 0 \end{cases} \tag{3.7}$$

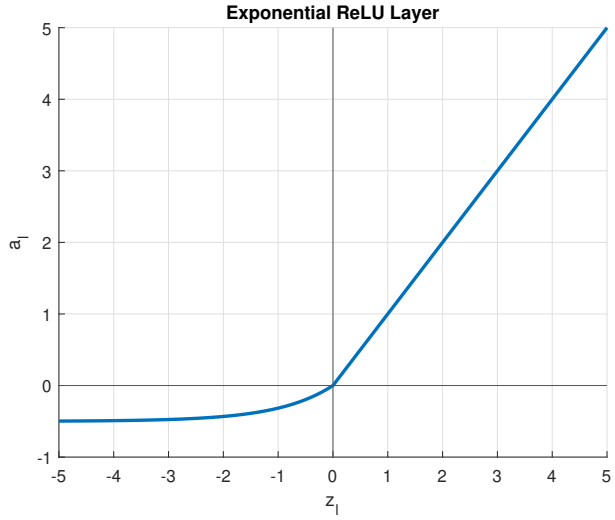


Figure 3.8: Exponential Linear Unit ($\alpha = 0.5$)

$$a_l = \ f_{l-ReLU}(z_l) = \begin{cases} z_l, & z_l \geq 0 \\ \alpha z_l, & z_l < 0 \end{cases} \tag{3.8}$$

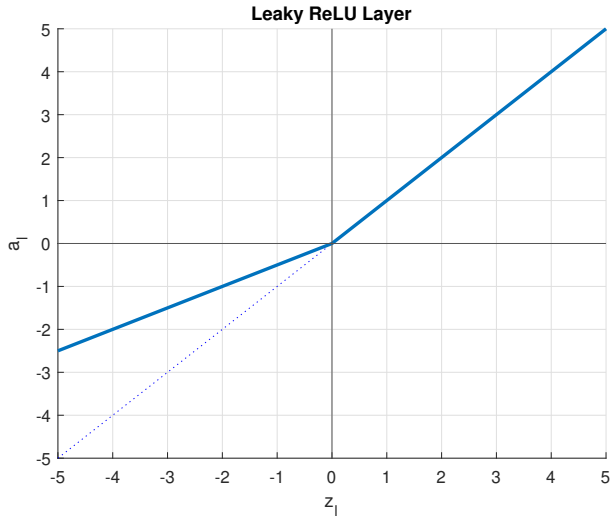


Figure 3.9: Leaky Rectified Linear Unit ($\alpha = 0.5$)

3.4 Training Estimator (Forward Kinematic Map Predictor)

Estimators are trained using different loss functions and activation functions given in subsection 3.3.2 and 3.3.3. For each loss function type, neural networks with three types of activation functions are trained resulting a total of 6 different estimators for each task. Since estimators are trained for apex to apex and touchdown to liftoff forward kinematic map predictions separately, total number trained estimators for SLIP Monopod Model is 12. Each neural network has 12 fully connected layers with 256 neurons in the most populated layer.

3.4.1 Selected Algorithm and Parameters for Training

Stochastic gradient descent algorithm is used for training. Training parameters include maximum number of epochs (ϵ_{max}), initial learn rate (η), learn rate drop period, learn rate drop rate, momentum coefficient (α), regularization coefficient for weights (L_2), mini-batch size and validation frequency. After several tests and

examinations parameters are selected to have the following values:

- Initial learn rate (η) and regularization coefficient (L_2) are optimized before training. Using a smaller neural network with only one or two fully connected layers, an optimization setup is utilized. For η and L_2, 10 different values are generated in logarithmic scale, where η goes from 10^{-8} to 10^{-4} and L_2 goes from 10^{-3} to 1. 10 values each for η and L_2, total 100 neural networks are trained and their final RMSE and loss values on validation set are recorded. For the training results with minimum RMSE and minimum loss, 2 values for η and L_2 are selected as optimal initial learning rates and optimal regularization coefficients. Optimization of η and L_2 are shown in Figure 3.10 and 3.11. In Figure 3.10 and 3.11, RMSE and loss values are shown for both different η values with optimal L_2 value and different L_2 values with optimal η value. Using the optimization results η_{MAER} is selected as 2.1544×10^{-7} and L_2 is selected as 0.0022. For the estimators with HMSE loss, initial learning rate (η) is increased, since forward loss of the output layer is smaller than the optimization results. Forward loss on the output layers are approximately 2 order of magnitude smaller, so the learning rate is multiplied by 100. Thus, the learning rate used for estimators with HMSE loss are $\eta_{HMSE} = 2.1544 \times 10^{-5}$.

- Maximum number of epochs (ϵ_{max}), is determined to be 10000. Changes in the loss and RMSE values becomes insignificant around ϵ_{10000}, so the training is continued until the convergence.

- Learn rate drop period is 1000 epochs and learn rate drop rate is 0.6. To prevent oscillation of the loss and RMSE values, learning rate is decreased during the training of the estimators with MAER loss. Learning rate drop is not applied during the training of the estimators with HMSE loss.

- Momentum coefficient (α) is 0.9 to prevent over fitting.

- Mini-batch size is 10.

- Validation frequency is one for every 100 epochs.

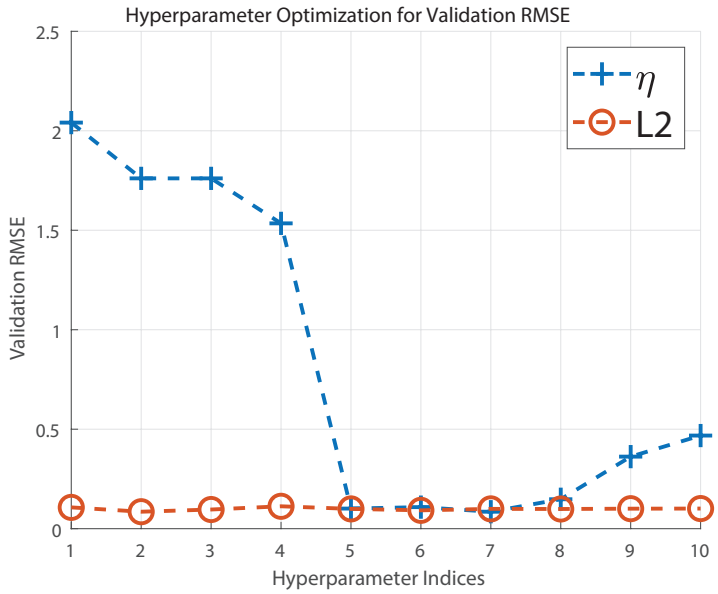


Figure 3.10: Hyperparameter Optimization on Final Validation RMSE

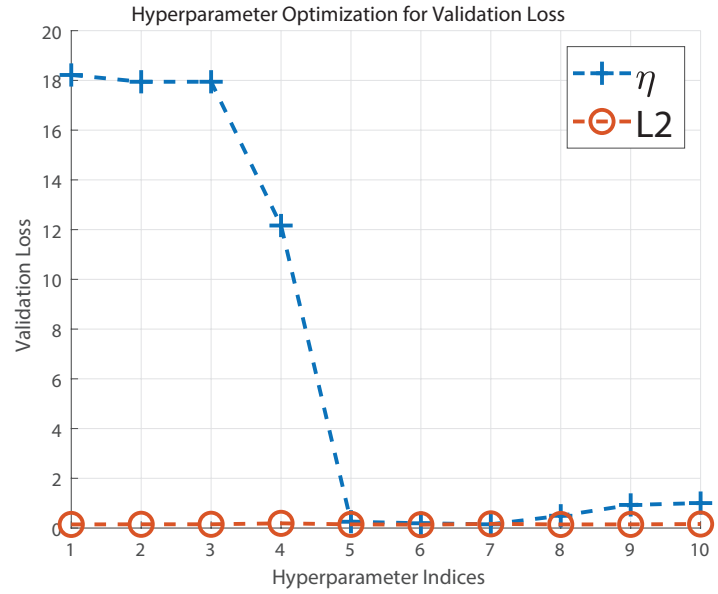


Figure 3.11: Hyperparameter Optimization on Final Validation Loss

3.4.2 Training Results of SLIP Estimators

After developing neural network architectures, setting parameters and selecting the required inputs and outputs from SLIP Dataset (subsection 3.2.2) training is completed for all 12 estimators. Training process and results are reported for each network separately. Success of the Estimators are measured with Mean Absolute Error Ratio and reported for every state variable in training, validation and test subsets. For apex to apex estimators errors for final apex state variables are reported, while for the touchdown to liftoff estimators errors for both liftoff and final apex state variables are reported. Final apex state variables are calculated from the liftoff state variables as described in section 2.1. Also some training parameters are arranged according to the type and design of the estimator.

3.4.2.1 a2a-Estimators with HMSE Loss on the Output

Training parameters are used as stated in subsection 3.4.1, applying the required changes for neural networks with HMSE loss.

Training processes of the apex to apex estimators with HMSE loss on the output are shown in Figure 3.12, 3.13 and 3.14 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 3.1, 3.2 and 3.3 in the same order.

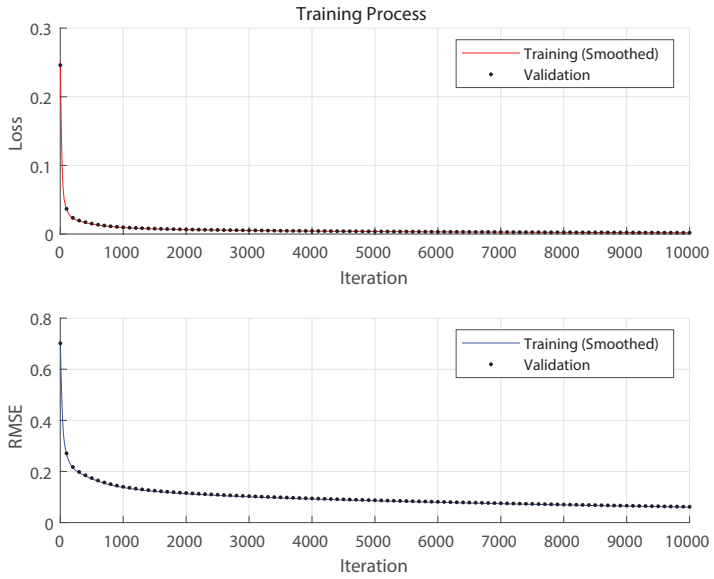


Figure 3.12: Training Process of Apex to Apex Estimator with HMSE and tanh Configuration

Table 3.1: Training Results and Errors over Subsets for Apex to Apex Estimator with HMSE and tanh Configuration

Training Duration:			0:05:38
Final RMSE After Training		Training:	0.06
		Validation:	0.06
Final Loss After Training		Training:	0.0018
		Validation:	0.0019
Mean Absolute Error Ratio over Subsets	Training	y_f	0.27427751
		$\dot{y}_f$	0.14104973
		z_f	0.27280512
	Validation	y_f	0.32957473
		$\dot{y}_f$	0.20526682
		z_f	0.18301858
	Test	y_f	2.3004389
		$\dot{y}_f$	0.42998317
		z_f	0.18383332

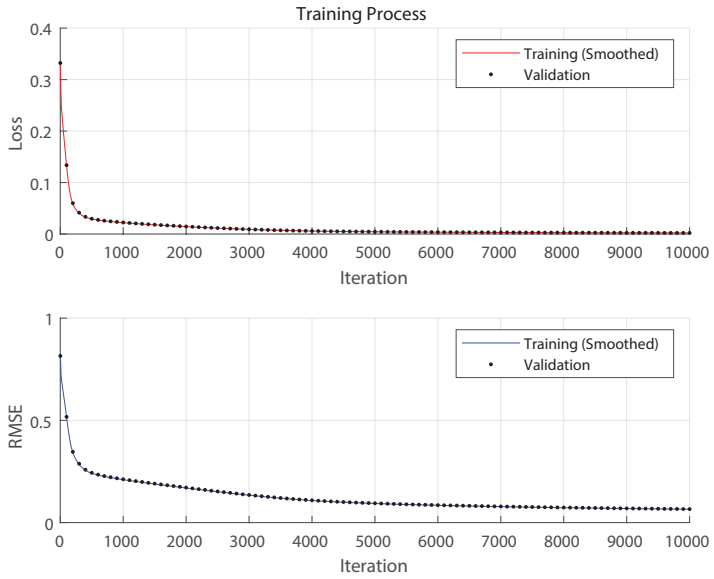


Figure 3.13: Training Process of Apex to Apex Estimator with HMSE and eLU Configuration

Table 3.2: Training Results and Errors over Subsets for Apex to Apex Estimator with HMSE and eLU Configuration

Training Duration:			0:09:40
Final RMSE After Training		Training:	0.07
		Validation:	0.07
Final Loss After Training		Training:	0.0021
		Validation:	0.0022
Mean Absolute Error Ratio over Subsets	Training	y_f	0.28612748
		$\dot{y}_f$	0.147843
		z_f	0.29055986
	Validation	y_f	0.77637196
		$\dot{y}_f$	0.18193865
		z_f	0.19540454
	Test	y_f	1.71656
		$\dot{y}_f$	0.37064141
		z_f	0.23111755

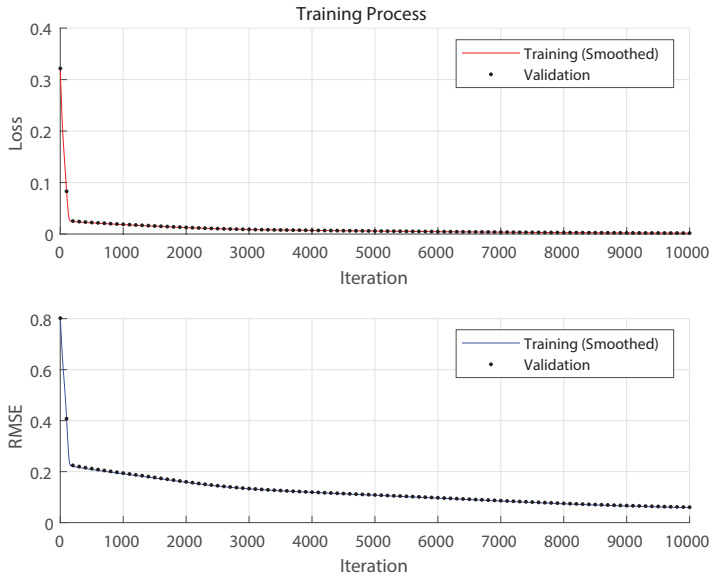


Figure 3.14: Training Process of Apex to Apex Estimator with HMSE and l-ReLU Configuration

Table 3.3: Training Results and Errors over Subsets for Apex to Apex Estimator with HMSE and l-ReLU Configuration

Training Duration:			0:05:22
Final RMSE After Training		Training:	0.06
		Validation:	0.06
Final Loss After Training		Training:	0.0017
		Validation:	0.0018
Mean Absolute Error Ratio over Subsets	Training	y_f	0.26523426
		$\dot{y}_f$	0.20547631
		z_f	0.34576714
	Validation	y_f	0.39390174
		$\dot{y}_f$	0.30839568
		z_f	0.25561053
	Test	y_f	1.3526702
		$\dot{y}_f$	0.55145633
		z_f	0.27822912

3.4.2.2 a2a-Estimators with MAER Loss on the Output

For the apex to apex estimators with MAER loss, parameters for training are used as stated in subsection 3.4.1.

Training processes of the apex to apex estimators with MAER loss on the output are shown in Figure 3.15, 3.16 and 3.17 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 3.4, 3.5 and 3.6 in the same order.

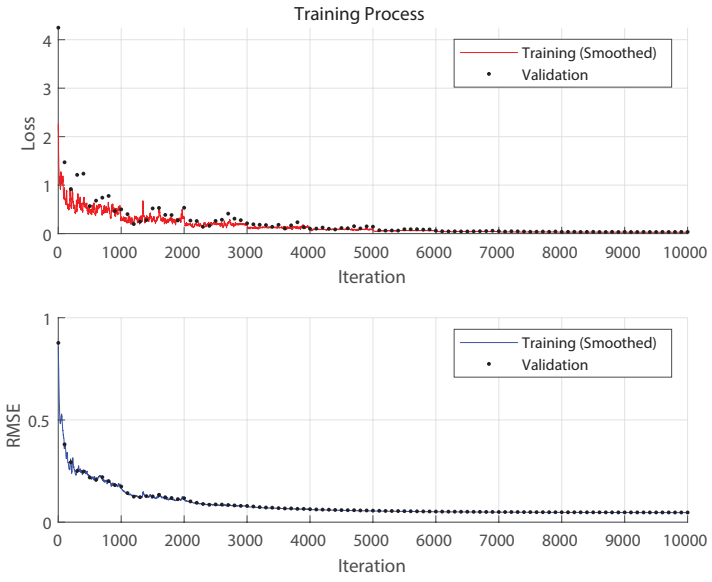


Figure 3.15: Training Process of Apex to Apex Estimator with MAER and tanh Configuration

Table 3.4: Training Results and Errors over Subsets for Apex to Apex Estimator with MAER and tanh Configuration

Training Duration:			0:05:44
Final RMSE After Training		Training:	0.05
		Validation:	0.05
Final Loss After Training		Training:	0.0271
		Validation:	0.0344
Mean Absolute Error Ratio over Subsets	Training	y_f	0.025966102
		$\dot{y}_f$	0.022908749
		z_f	0.032470569
	Validation	y_f	0.038970903
		$\dot{y}_f$	0.031626534
		z_f	0.032745935
	Test	y_f	0.093653627
		$\dot{y}_f$	0.043414772
		z_f	0.033653658

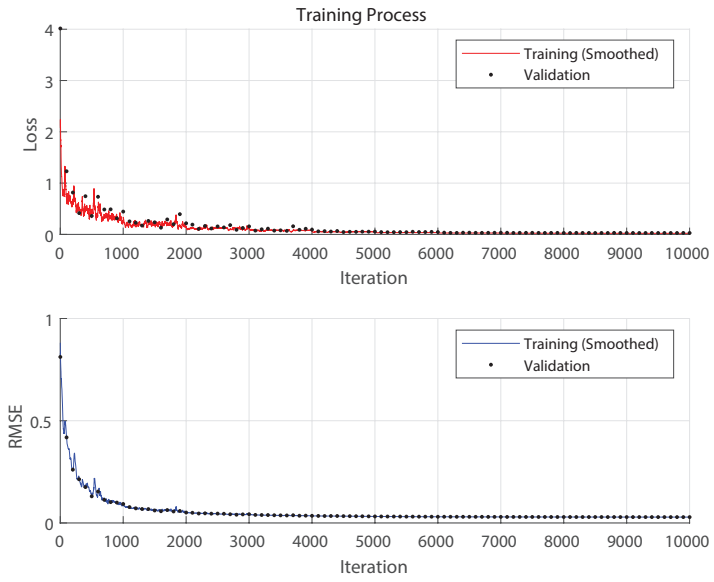


Figure 3.16: Training Process of Apex to Apex Estimator with MAER and eLU Configuration

Table 3.5: Training Results and Errors over Subsets for Apex to Apex Estimator with MAER and eLU Configuration

Training Duration:			0:08:58
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0202
		Validation:	0.0276
Mean Absolute Error Ratio over Subsets	Training	y_f	0.015198419
		$\dot{y}_f$	0.024016932
		z_f	0.021000134
	Validation	y_f	0.024945069
		$\dot{y}_f$	0.033896346
		z_f	0.023895202
	Test	y_f	0.048658121
		$\dot{y}_f$	0.060466051
		z_f	0.0232879

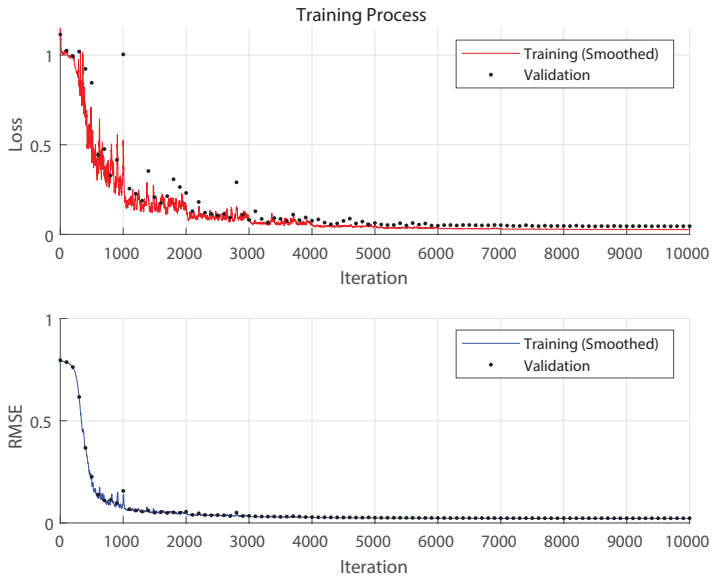


Figure 3.17: Training Process of Apex to Apex Estimator with MAER and l-ReLU Configuration

Table 3.6: Training Results and Errors over Subsets for Apex to Apex Estimator with MAER and l-ReLU Configuration

Training Duration:			0:05:32
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0269
		Validation:	0.046
Mean Absolute Error Ratio over Subsets	Training	y_f	0.029971417
		$\dot{y}_f$	0.023587363
		z_f	0.027603621
	Validation	y_f	0.058489673
		$\dot{y}_f$	0.041033376
		z_f	0.038409829
	Test	y_f	0.20189825
		$\dot{y}_f$	0.087593488
		z_f	0.04068438

3.4.2.3 t2l-Estimators with HMSE Loss on the Output

For the touchdown to liftoff estimators with HMSE loss, same parameters with apex to apex estimators with HMSE loss are used for training.

Training processes of the touchdown to liftoff estimators with HMSE loss on the output are shown in Figure 3.18, 3.19 and 3.20 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 3.7, 3.8 and 3.9 in the same order.

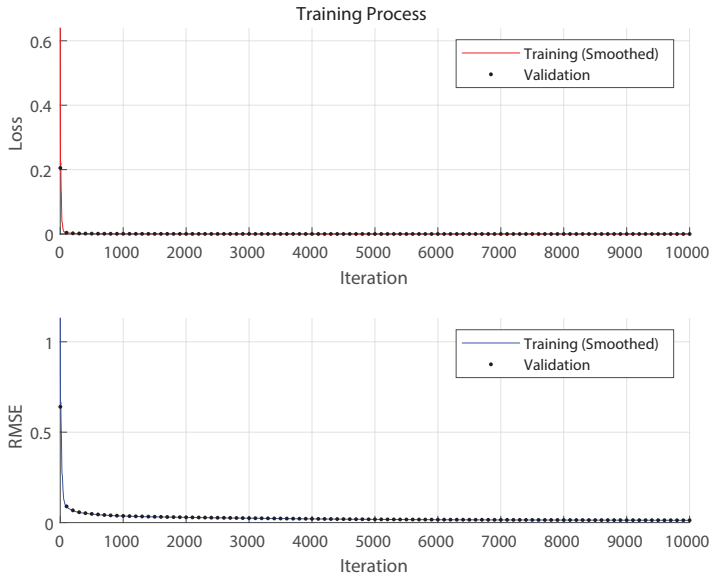


Figure 3.18: Training Process of Touchdown to Liftoff Estimator with HMSE and tanh Configuration

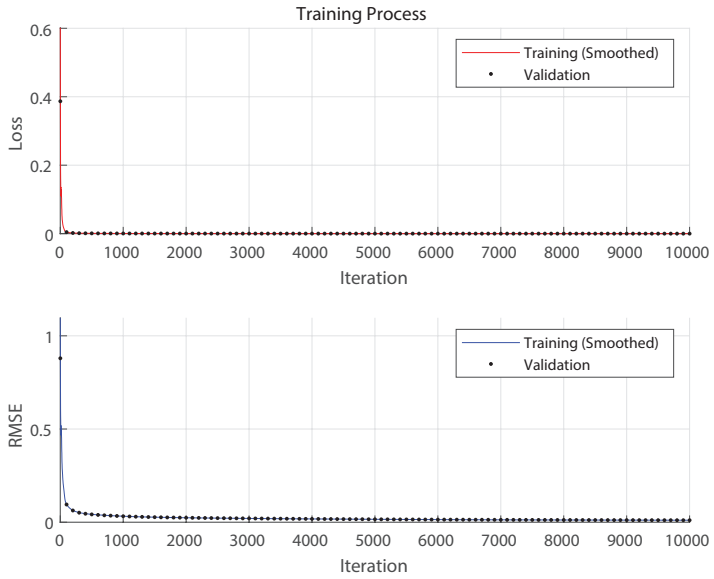


Figure 3.19: Training Process of Touchdown to Liftoff Estimator with HMSE and eLU Configuration

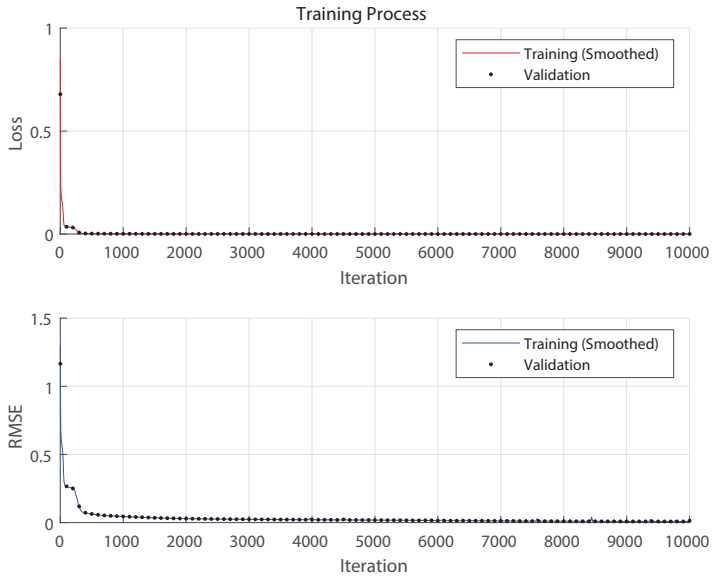


Figure 3.20: Training Process of Touchdown to Liftoff Estimator with HMSE and l-ReLU Configuration

Table 3.7: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with HMSE and tanh Configuration

Training Duration:			0:05:43
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.000078451
		Validation:	0.000082845
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.013072514
		$\dot{y}_{lo}$	0.036961727
		z_{lo}	0.0037391197
		$\dot{z}_{lo}$	0.0089961197
	Validation	y_{lo}	0.01307796
		$\dot{y}_{lo}$	0.048913252
		z_{lo}	0.0037462206
		$\dot{z}_{lo}$	0.011097683
	Test	y_{lo}	0.012556401
		$\dot{y}_{lo}$	0.08634755
		z_{lo}	0.0037918258
		$\dot{z}_{lo}$	0.01111543
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.03872843333
		$\dot{y}_f$	0.03696177939
		z_f	0.1623506166
	Validation	y_f	0.07007919239
		$\dot{y}_f$	0.04891315257
		z_f	0.09177740068
	Test	y_f	0.5971631415
		$\dot{y}_f$	0.0863474767
		z_f	0.1097056626

Table 3.8: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with HMSE and eLU Configuration

Training Duration:			0:09:53
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.000057268
		Validation:	0.000057171
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.0095831482
		$\dot{y}_{lo}$	0.034969117
		z_{lo}	0.0049942206
		$\dot{z}_{lo}$	0.0073638121
	Validation	y_{lo}	0.010252332
		$\dot{y}_{lo}$	0.041334946
		z_{lo}	0.0051382692
		$\dot{z}_{lo}$	0.010184412
	Test	y_{lo}	0.010833418
		$\dot{y}_{lo}$	0.069897465
		z_{lo}	0.0051584486
		$\dot{z}_{lo}$	0.0092805708
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.02994271333
		$\dot{y}_f$	0.03496912025
		z_f	0.1156080774
	Validation	y_f	0.108914016
		$\dot{y}_f$	0.04133493555
		z_f	0.08962312517
	Test	y_f	0.177635517
		$\dot{y}_f$	0.06989777803
		z_f	0.0971383996

Table 3.9: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with HMSE and l-ReLU Configuration

Training Duration:			0:05:15
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.000097837
		Validation:	0.000098396
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.011516994
		$\dot{y}_{lo}$	0.073189609
		z_{lo}	0.0026829632
		$\dot{z}_{lo}$	0.0087224245
	Validation	y_{lo}	0.0114194
		$\dot{y}_{lo}$	0.091287486
		z_{lo}	0.002758234
		$\dot{z}_{lo}$	0.010164133
	Test	y_{lo}	0.0104634
		$\dot{y}_{lo}$	0.14116484
		z_{lo}	0.0027540347
		$\dot{z}_{lo}$	0.010337484
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.06689160637
		$\dot{y}_f$	0.07318968602
		z_f	0.146839258
	Validation	y_f	0.1703415005
		$\dot{y}_f$	0.09128731102
		z_f	0.10785834
	Test	y_f	1.750324008
		$\dot{y}_f$	0.1411644516
		z_f	0.1020152102

3.4.2.4 t2l-Estimators with MAER Loss on the Output

For the touchdown to liftoff estimators with MAER loss, same parameters with apex to apex estimators with MAER loss are used for training.

Training processes of the touchdown to liftoff estimators with MAER loss on the output are shown in Figure 3.21, 3.22 and 3.23 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 3.10, 3.11 and 3.12 in the same order.

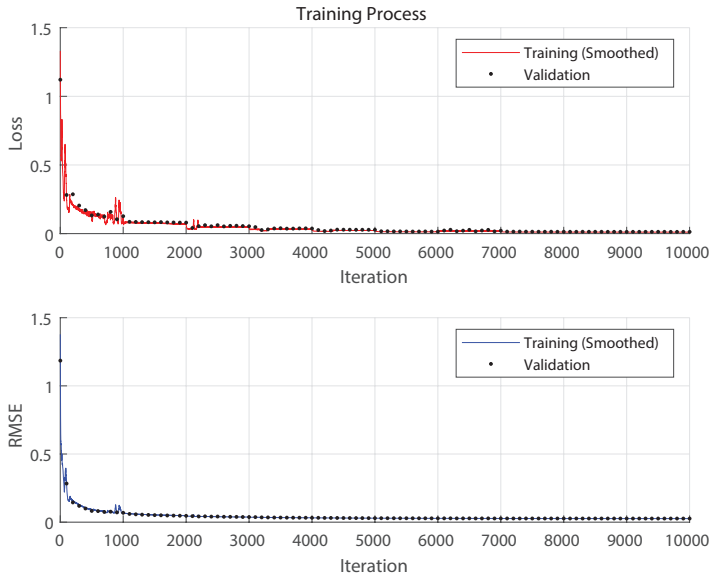


Figure 3.21: Training Process of Touchdown to Liftoff Estimator with MAER and tanh Configuration

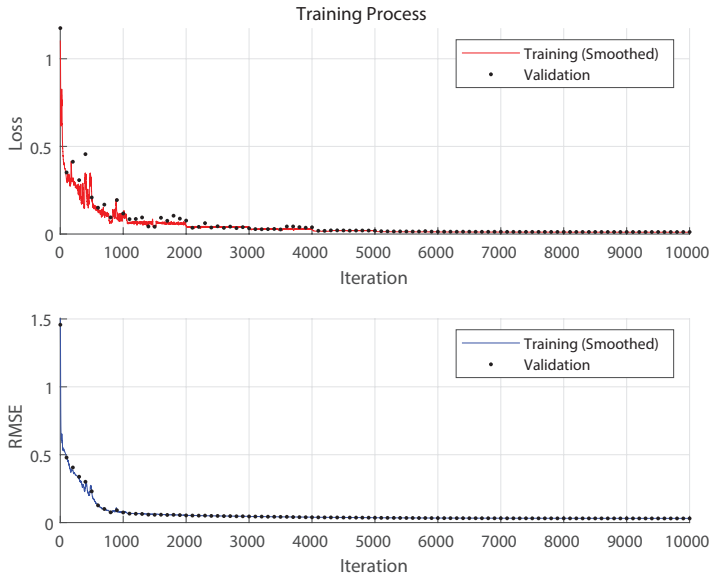


Figure 3.22: Training Process of Touchdown to Liftoff Estimator with MAER and eLU Configuration

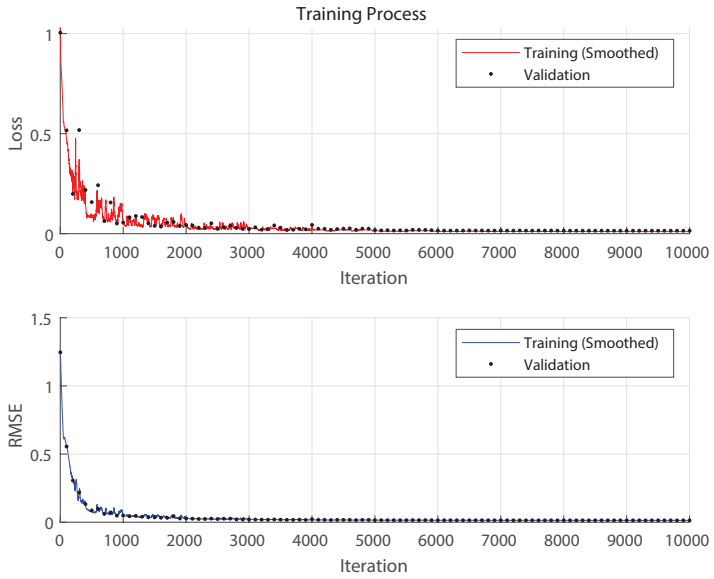


Figure 3.23: Training Process of Touchdown to Liftoff Estimator with MAER and l-ReLU Configuration

Table 3.10: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with MAER and tanh Configuration

Training Duration:			0:06:09
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0106
		Validation:	0.0128
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.0086987698
		$\dot{y}_{lo}$	0.019737339
		z_{lo}	0.0039977655
		$\dot{z}_{lo}$	0.0098327631
	Validation	y_{lo}	0.0094041377
		$\dot{y}_{lo}$	0.024151271
		z_{lo}	0.0041896584
		$\dot{z}_{lo}$	0.013600123
	Test	y_{lo}	0.0094064027
		$\dot{y}_{lo}$	0.030879913
		z_{lo}	0.0042452398
		$\dot{z}_{lo}$	0.011482082
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.04736462599
		$\dot{y}_f$	0.01973732526
		z_f	0.1557312105
	Validation	y_f	0.1001245625
		$\dot{y}_f$	0.02415115877
		z_f	0.1033695035
	Test	y_f	0.08400823589
		$\dot{y}_f$	0.03087997813
		z_f	0.1240034348

Table 3.11: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with MAER and eLU Configuration

Training Duration:			0:09:24
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0102
		Validation:	0.0116
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.0072461097
		$\dot{y}_{lo}$	0.02630941
		z_{lo}	0.0027725541
		$\dot{z}_{lo}$	0.0044854586
	Validation	y_{lo}	0.0080978274
		$\dot{y}_{lo}$	0.029334776
		z_{lo}	0.0029257322
		$\dot{z}_{lo}$	0.0058555705
	Test	y_{lo}	0.0077012228
		$\dot{y}_{lo}$	0.037863888
		z_{lo}	0.0028159837
		$\dot{z}_{lo}$	0.0059818854
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.04663942735
		$\dot{y}_f$	0.02630944138
		z_f	0.0778213073
	Validation	y_f	0.09218919148
		$\dot{y}_f$	0.02933470583
		z_f	0.06320033357
	Test	y_f	0.4612702886
		$\dot{y}_f$	0.03786377802
		z_f	0.06878992874

Table 3.12: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with MAER and l-ReLU Configuration

Training Duration:			0:05:34
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.0101
		Validation:	0.0147
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.0072040134
		$\dot{y}_{lo}$	0.023221785
		z_{lo}	0.0025972957
		$\dot{z}_{lo}$	0.0080413865
	Validation	y_{lo}	0.0090649826
		$\dot{y}_{lo}$	0.036609091
		z_{lo}	0.002657166
		$\dot{z}_{lo}$	0.01033284
	Test	y_{lo}	0.0090093603
		$\dot{y}_{lo}$	0.051619589
		z_{lo}	0.002633296
		$\dot{z}_{lo}$	0.0090776943
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.05013236874
		$\dot{y}_f$	0.0232218158
		z_f	0.1302107761
	Validation	y_f	0.09638822046
		$\dot{y}_f$	0.03660900955
		z_f	0.1056479726
	Test	y_f	0.2875423119
		$\dot{y}_f$	0.05161982143
		z_f	0.1112442706

3.4.3 Comparison of SLIP Estimators

In section 3.1 it is stated that a2a-estimators are designed to be the neural network equivalent of SLIP simulation. To understand the success of the estimators on this task, Mean Absolute Error Ratios over subsets are taken into consideration.

By observing the error ratios, one can say that, a2a-estimators with HMSE loss are less successful than a2a-estimators with MAER loss. Same pattern is also correct for t2l estimators, if apex to apex errors are taken into consideration. These differences can be explained by the compatibility of the expected errors with used loss function. Estimators with MAER loss is optimized exactly for the expected success measure. Thus, it can be observed that both estimator types with MAER loss are more suitable to replace the SLIP simulation.

After selecting the estimators with MAER loss; success rates of different types can also be compared by using apex to apex results. Between a2a-estimators and t2l-estimators, it can be said that a2a-estimators produce more correct predictions for the final apex, while t2l estimators produce more correct result for the determined outputs.

Apex to apex errors of t2l-estimators are higher than touchdown to liftoff errors because of error accumulation. While calculating the final apex state using the liftoff state, equations in section 2.1 are used. In these equations final apex positions are calculated from integration of liftoff velocities. So, errors in liftoff velocities projects higher errors for final apex positions.

3.5 Training Controller (Inverse Kinematic Map Predictor)

Training approach for the controllers is similar to the training approach of the estimators. Controllers are trained using different loss functions and activation functions given in subsection 3.3.2 and 3.3.3. For each loss function type, neural networks with three types of activation functions are trained resulting a total of 6 different controllers. As also mentioned in subsection 3.1.2; unlike estimators, controllers are trained only for apex to apex inverse kinematic map prediction. Layer structure for the neural networks is also kept same with the estimator, so each neural network has 12 fully connected layers with 256 neurons in the most populated layer.

3.5.1 Selected Algorithm and Parameters for Training

Stochastic gradient descent algorithm is used for training. Training parameters include maximum number of epochs (ϵ_{max}), initial learn rate (η), learn rate drop period, learn rate drop rate, momentum coefficient (α), regularization coefficient for weights (L_2), mini-batch size and validation frequency.

- Using the same process detailed in subsection 3.4.1, initial learn rate (η) and regularization coefficient (L_2) are optimized before training. Optimization of η and L_2 are shown in Figure 3.24 and 3.25. Using the optimization results η_{MAER} is selected as 2.1544×10^{-7} and L_2 is selected as 0.0464. η_{HMSE} is increased with the same reasons detailed in subsection 3.4.1. Learning rate used for controllers with HMSE loss are $\eta_{HMSE} = 2.1544 \times 10^{-5}$.
- Maximum number of epochs (ϵ_{max}), is determined to be 10000. Changes in the loss and RMSE values becomes insignificant around ϵ_{10000}, so the training is continued until the convergence.
- Learn rate drop period is 1000 epochs and learn rate drop rate is 0.6. To

prevent oscillation of the loss and RMSE values, learning rate is decreased during the training of the estimators with MAER loss. Learning rate drop is not applied during the training of the estimators with HMSE loss.

- Momentum coefficient (α) is 0.9 to prevent over fitting.
- Mini-batch size is 10.
- Validation frequency is one for every 100 epochs.

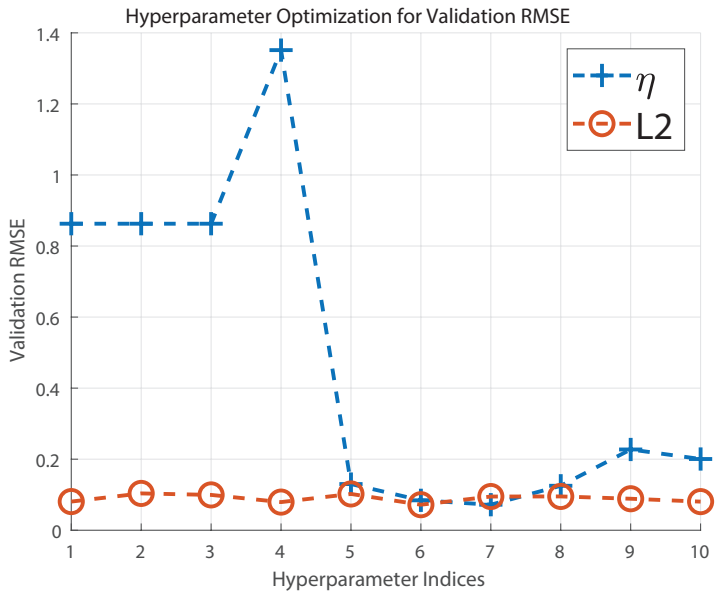


Figure 3.24: Hyperparameter Optimization on Final Validation RMSE

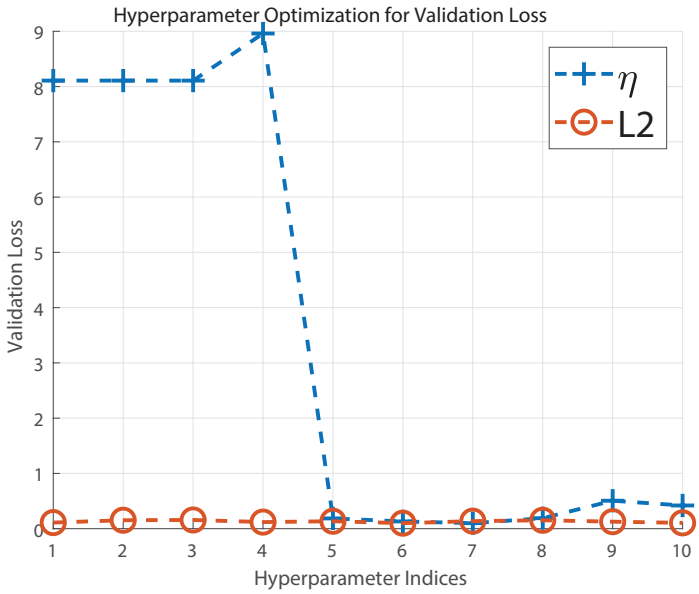


Figure 3.25: Hyperparameter Optimization on Final Validation Loss

3.5.2 Training Results of SLIP Controllers

Trained controllers are tested with SLIP simulation and errors for reaching goal state variables are also reported, on top of the success measures used to report estimator results. This additional step of success rate measurement, stands as the verification of the trained controllers. Block diagram explaining the test structure is given in Figure 3.26.

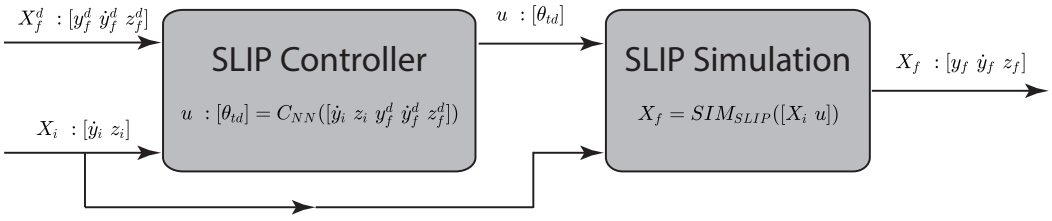


Figure 3.26: Block Diagram of Testing Controllers Against SLIP Simulation

In Figure 3.26, a trained SLIP Controller, C_{NN}, is used to predict the control inputs, u, that bring the monopod from initial apex state, X_i, to a desired final apex state, X_f^d. Than the predicted control inputs are used with SLIP Simulation, SIM_{SLIP}, to compute the achieved final apex state, X_f. Success of the trained controller is measured by calculating the error of X_f respect to X_f^d with MAER formula.

3.5.2.1 a2a-Controllers with HMSE Loss on the Output

Training parameters are used as stated in subsection 3.5.1, applying the required changes for neural networks with HMSE loss.

Training processes of the apex to apex controllers with HMSE loss on the output are shown in Figure 3.27, 3.28 and 3.29 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 3.13, 3.14 and 3.15 in the same order.

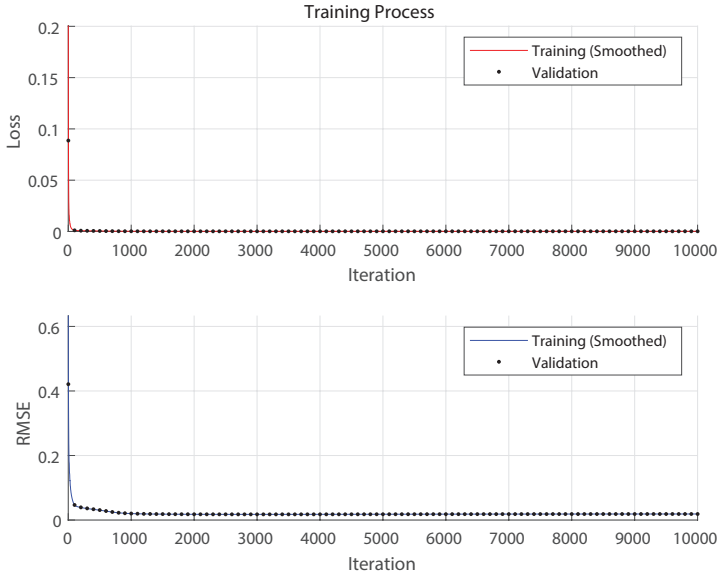


Figure 3.27: Training Process of Controller with HMSE and tanh Configuration

Table 3.13: Training Results and Errors over Subsets for Controller with HMSE and tanh Configuration

Training Duration:			0:05:29
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0002
		Validation:	0.0002
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.18039799
	Validation	θ_{td}	0.08784312
	Test	θ_{td}	0.10628206
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.007565884577
		$\dot{y}_f$	0.06653273019
		z_f	0.006164176415
	Validation	y_f	0.007347677733
		$\dot{y}_f$	0.08739774472
		z_f	0.006310926519
	Test	y_f	0.007394841226
		$\dot{y}_f$	0.1185440422
		z_f	0.006324647842

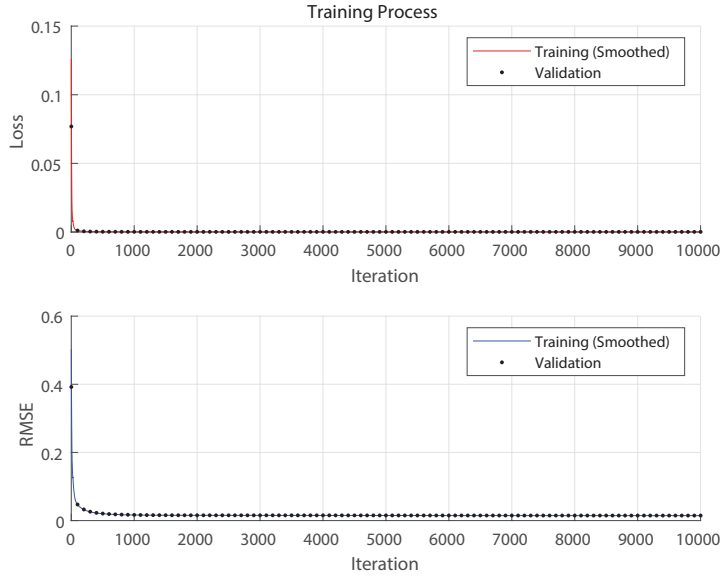


Figure 3.28: Training Process of Controller with HMSE and eLU Configuration

Table 3.14: Training Results and Errors over Subsets for Controller with HMSE and eLU Configuration

Training Duration:			0:09:54
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.0001
		Validation:	0.0001
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.11081291
	Validation	θ_{td}	0.056017235
	Test	θ_{td}	0.067150854
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.004989205307
		$\dot{y}_f$	0.0335245992
		z_f	0.004511152757
	Validation	y_f	0.005034165128
		$\dot{y}_f$	0.07753253355
		z_f	0.004623326405
	Test	y_f	0.005158071069
		$\dot{y}_f$	0.08844888611
		z_f	0.004696636529

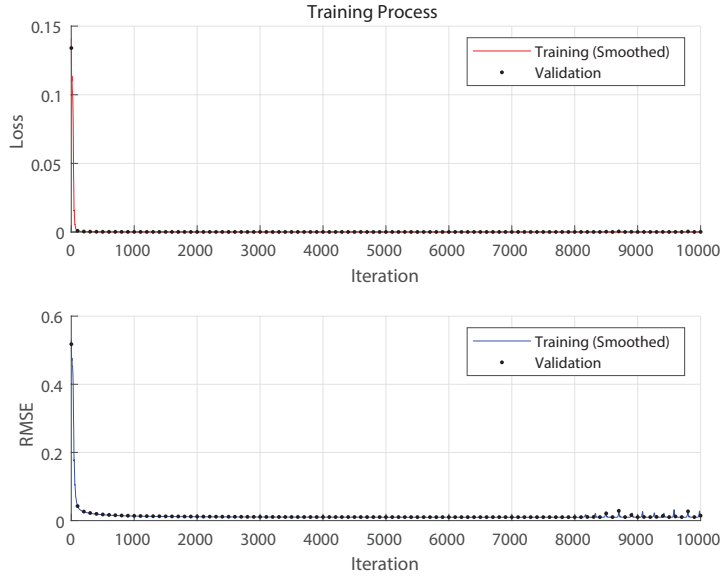


Figure 3.29: Training Process of Controller with HMSE and l-ReLU Configuration

Table 3.15: Training Results and Errors over Subsets for Controller with HMSE and l-ReLU Configuration

Training Duration:			0:05:26
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.0001
		Validation:	0.0001
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.23093496
	Validation	θ_{td}	0.10782152
	Test	θ_{td}	0.14862889
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.00626657002
		$\dot{y}_f$	0.05922734103
		z_f	0.005910456486
	Validation	y_f	0.006208278016
		$\dot{y}_f$	0.1378509833
		z_f	0.006015975533
	Test	y_f	0.006389456168
		$\dot{y}_f$	0.1214414959
		z_f	0.006137695138

3.5.2.2 a2a-Controllers with MAER Loss on the Output

For the apex to apex controllers with MAER loss, parameters for training are used as stated in subsection 3.5.1.

Training processes of the apex to apex controllers with MAER loss on the output are shown in Figure 3.30, 3.31 and 3.32 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 3.16, 3.17 and 3.18 in the same order.

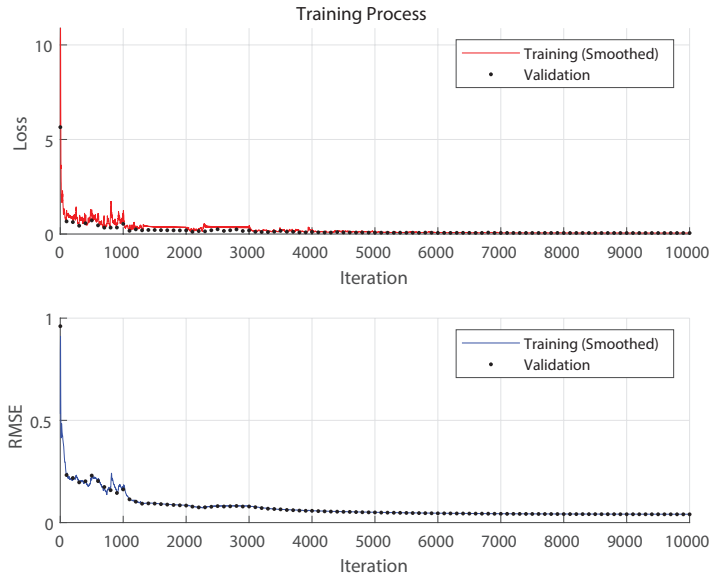


Figure 3.30: Training Process of Controller with MAER and tanh Configuration

Table 3.16: Training Results and Errors over Subsets for Controller with MAER and tanh Configuration

Training Duration:			0:05:50
Final RMSE After Training		Training:	0.04
		Validation:	0.04
Final Loss After Training		Training:	0.054
		Validation:	0.0529
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.054044675
	Validation	θ_{td}	0.052910306
	Test	θ_{td}	0.055578336
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.01401148171
		$\dot{y}_f$	0.1388383548
		z_f	0.009779385429
	Validation	y_f	0.01362957291
		$\dot{y}_f$	0.18784321
		z_f	0.01023939503
	Test	y_f	0.01335662723
		$\dot{y}_f$	0.1940003945
		z_f	0.009978365507

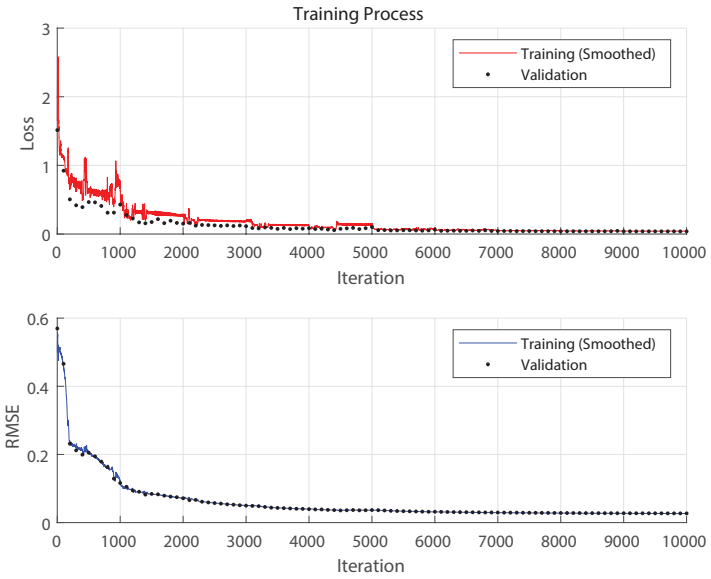


Figure 3.31: Training Process of Controller with MAER and eLU Configuration

Table 3.17: Training Results and Errors over Subsets for Controller with MAER and eLU Configuration

Training Duration:			0:09:46
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.042
		Validation:	0.0385
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.045250472
	Validation	θ_{td}	0.0384924
	Test	θ_{td}	0.041862395
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.007665931992
		$\dot{y}_f$	0.051051983
		z_f	0.00749852659
	Validation	y_f	0.007668520416
		$\dot{y}_f$	0.0952896154
		z_f	0.007782904613
	Test	y_f	0.007947051824
		$\dot{y}_f$	0.1189035424
		z_f	0.007940039277

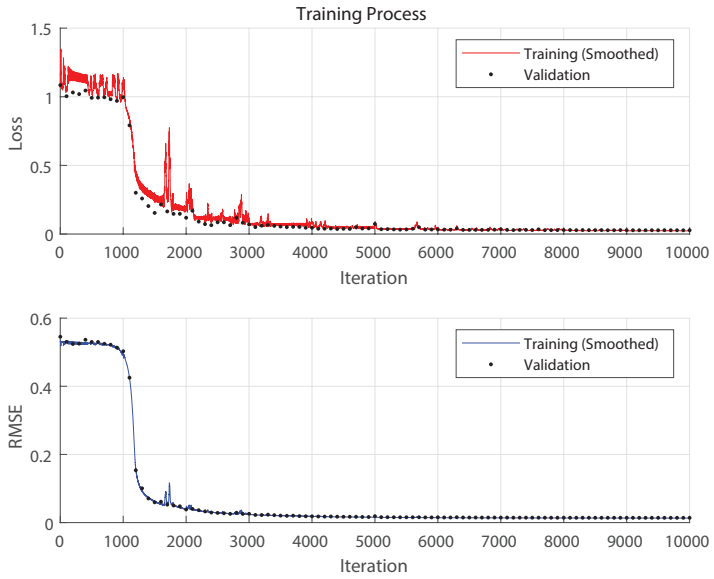


Figure 3.32: Training Process of Controller with MAER and l-ReLU Configuration

Table 3.18: Training Results and Errors over Subsets for Controller with MAER and l-ReLU Configuration

Training Duration:			0:05:35
Final RMSE After Training		Training:	0.01
		Validation:	0.01
Final Loss After Training		Training:	0.0247
		Validation:	0.0274
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.02308565
	Validation	θ_{td}	0.027394641
	Test	θ_{td}	0.030097278
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.004620474909
		$\dot{y}_f$	0.03910755248
		z_f	0.003827411423
	Validation	y_f	0.004792954873
		$\dot{y}_f$	0.05950410064
		z_f	0.004144985851
	Test	y_f	0.004925945488
		$\dot{y}_f$	0.1159105306
		z_f	0.004198467319

3.5.3 Comparison of SLIP Controllers

In section 3.1 it is stated that a2a-controllers are designed to be neural network based controllers for SLIP model. To understand the success of the controllers on this task, verification results on SLIP simulation are taken into consideration.

By observing the error ratios over subsets, success of the controller for reaching desired control inputs can be compared. From the error ratios over subsets, it is observable that a2a-controllers with HMSE loss are less successful than a2a-controllers with MAER loss. Controllers with MAER loss are optimized for predicting the control inputs used in that apex return map, so resulting control inputs are more accurate than the control inputs resulted from controllers with HMSE loss.

On the contrary, by observing the verification results on SLIP simulation, controllers with HMSE and MAER losses results closer success rates for reaching the desired apex state. Since there are ranges of control inputs that can be used to reach desired apex states, error margins of the control inputs projects a smaller error margins on the final apex state.

Chapter 4

Neural Network Based Estimator and Controller for TD-SLIP Model

Neural Networks for the TD-SLIP model are developed by applying a similar methodology of developing neural networks for the SLIP model (**chapter 3**). Main difference between SLIP and TD-SLIP models is that TD-SLIP model is a more accurate representation of the behaviors of a monopod robot. In TD-SLIP model mechanical energy decays over time as a result of damping loss and in order to compensate this loss, a hip torque is applied during stance to inject mechanical energy to the system. Thus, physical parameters of the TD-SLIP model also include a damping coefficient and the control inputs of the TD-SLIP model additionally include the hip torque. Neural networks trained to predict forward and inverse kinematic maps. Inputs and outputs of the neural networks are configured according to these changes and TD-SLIP simulation is used to generate single stride maps. From the generated single stride maps; initial apex, touchdown, liftoff and final apex states are recorded with control inputs. After constructing dataset form these recorded state variables and parameters, training of neural networks are completed with different network architectures and training parameters. Trained networks are tested for training success and used with the TD-SLIP simulation for further examination.

4.1 Input and Output Configuration for Neural Networks

Neural networks are named according to the prediction task on the kinematic maps. Forward Kinematic Map Predictors are called **Estimators** and Inverse Kinematic Map Predictors are called **Controllers**.

4.1.1 Estimator Inputs and Outputs

Two type of Estimators are trained for forward kinematic map. First type of Estimator is trained for complete apex to apex return map and second type of estimator is trained for only stance return map.

4.1.1.1 Apex to Apex Forward Kinematic Map Predictor (a2a-Estimator)

Expected task from Apex to Apex Forward Kinematic Map Predictor is to estimate the next apex state where initial apex state and control inputs are given. In other words Apex to Apex Forward Kinematic Map Predictor is the neural network equivalent of TD-SLIP Simulator. So that the inputs for the Apex to Apex Forward Kinematic Map Predictor will be initial apex state X_i and control inputs u and output will be next apex state, X_f.

- *a2a-Estimator Inputs:* $[X_i\ u]$, where $X_i : [y_i\ \dot{y}_i\ z_i\ \dot{z}_i]$ and $u : [\theta_{touchdown}\ \tau_0]$. Since $y_i = 0$ and $\dot{z}_i = 0$, X_i becomes $X_i : [\dot{y}_i\ z_i]$ and the input used for a2a-Estimator is $I_{a2a-Estimator} : [\dot{y}_i\ z_i\ \theta_{touchdown}\ \tau_0]$.
- *a2a-Estimator Outputs:* $[X_f]$, where $X_f : [y_f\ \dot{y}_f\ z_f\ \dot{z}_f]$. Since $\dot{z}_f = 0$, X_f becomes $X_f : [y_f\ \dot{y}_f\ z_f]$ and the output expected from a2a-Estimator is $O_{a2a-Estimator} : [y_f\ \dot{y}_f\ z_f]$.

4.1.1.2 Touchdown to Liftoff Forward Kinematic Map Predictor (t2l-Estimator)

Since the calculation of touchdown state variables from initial apex state and final apex state variables from liftoff state can be done by ballistic trajectories with absolute theoretical certainty (discussed in **section 2.1**); a second type of estimator can be trained using touchdown and liftoff states, instead of initial and final apex states. Expected task from Touchdown to Liftoff Forward Kinematic Map Predictor is to estimate the liftoff state where touchdown state and control inputs are given. So that the inputs for the Touchdown to Liftoff Forward Kinematic Map Predictor will be touchdown state X_{td} and control inputs u and output will be liftoff state, X_{lo}.

- *t2l-Estimator Inputs:* $[X_{td}\ u]$, where $X_{td} : [y_{td}\ \dot{y}_{td}\ z_{td}\ \dot{z}_{td}]$ and $u : [\theta_{touchdown}\ \tau_0]$. So the input used for t2l-Estimator is $I_{t2l-Estimator} : [y_{td}\ \dot{y}_{td}\ z_{td}\ \dot{z}_{td}\ \theta_{touchdown}\ \tau_0]$.
- *t2l-Estimator Outputs:* $[X_{lo}]$, where $X_{lo} : [y_{lo}\ \dot{y}_{lo}\ z_{lo}\ \dot{z}_{lo}]$ and the output expected from t2l-Estimator is $O_{t2l-Estimator} : [y_{lo}\ \dot{y}_{lo}\ z_{lo}\ \dot{z}_{lo}]$.

Same process demonstrated in Figure 3.1, is followed for using a t2l-Estimator to predict the apex return map, for also TD-SLIP Model.

4.1.2 Controller Inputs and Outputs

Calculating a unique liftoff state using final apex state is not possible, since there is no inverse function for liftoff to final apex trajectory. So, only Apex to Apex Inverse Kinematic Map Predictor is trained.

4.1.2.1 Apex to Apex Inverse Kinematic Map Predictor (a2a-Controller)

Expected task from Apex to Apex Inverse Kinematic Map Predictor is to predict the control inputs that brings the TD-SLIP monopod from an initial state to a final (desired) apex state. In other words Apex to Apex Inverse Kinematic Map Predictor is a neural network controller that can be used for TD-SLIP simulation. The inputs for the Apex to Apex Inverse Kinematic Map Predictor will be initial and final apex states, X_i and X_f, and the output will be control inputs, u.

- *a2a-Controller Inputs:* $[X_i \ X_f]$, where $X_i : [\dot{y}_i \ z_i]$ and $X_f : [y_f \ \dot{y}_f \ z_f]$. So the input used for a2a-Controller is $I_{a2a-Controller} : [\dot{y}_i \ z_i \ y_f \ \dot{y}_f \ z_f]$.
- *a2a-Controller Outputs:* $[u]$, where $u : [\theta_{touchdown} \ \tau_0]$. So the output expected from a2a-Controller is $O_{a2a-Controller} : [\theta_{touchdown} \ \tau_0]$.

4.2 Data Generation with TD-SLIP Model

Data generation is done with the TD-SLIP Model Simulation with the goal of recording determined inputs and outputs for the neural networks. As explained in **section 2.2**, simulation generates apex return map with starting apex position, control inputs and physical constants. Physical constants are selected from the properties of the actual monopod robot to have more realistic results and measure the usability of the neural networks with physical systems. Having apex return maps, generated inputs and outputs required for neural networks, dataset is complete. Physical constants are also recorded for future reference.

4.2.1 Selected Parameters for Data Generation

For each initial apex state variable and control parameter, a range based on physical limits is selected. A random initial apex state with control inputs used

to start the simulation. Since some of these random starting points returns an unrealistic return map, the number of generated apex return maps is increased. Parameters required for data generation are initial apex state variables $X_i : [\dot{y}_i \; z_i]$, control inputs $u : [\theta_{touchdown} \; \tau_0]$ and physical constants m, l_0, k, d, g.

Physical characteristics of the TD-SLIP Monopod Robot:

- $m = 3.005 \; kg$: Total mass of the TD-SLIP Monopod
- $l_0 = 0.205 \; m$: Leg length when the spring is resting
- $k = 9000 \; N/m$: Leg spring constant
- $d = 10 \; Ns/m$: Leg damping coefficient
- $g = 9.81 \; m/s^2$: Gravitational acceleration

Total mass of the TD-SLIP Monopod, m, and leg rest length, l_0, are the exact measurements taken from the monopod robot in our lab, where leg spring constant, k, and leg damping coefficient, d, are approximate values calculated based on previous experiments [22].

Initial apex state variables and control inputs are selected as a uniformly distributed random number from the given range:

- $\dot{y}_i$: Horizontal velocity of the TD-SLIP monopod body. $\dot{y}_i$ is selected between 0 m/s and 3 m/s. Velocity is limited to 3 m/s, since in the physical system going faster then this upper limit may result a hazardous situation for the monopod robot and the operators. Lower bound is 0 m/s, since the TD-SLIP monopod goes forward.
- z_i: Vertical position of the TD-SLIP monopod body. z_i is selected from l_0 to $2l_o$. Lower bound is l_0, since the TD-SLIP monopod is at the apex position and the spring is resting and upper bound is $2l_0$ since it is the approximate upper limit the physical TD-SLIP monopod reaches.

- $\theta_{touchdown}$: Angle of the leg with the z axis when touchdown occurs. $\theta_{touchdown}$ is selected from 10° to 40°. Narrower angles than 10° results a back-jump and wider angles than 40° increases the risk of toe sliding during the stance.

- τ_0: Initial value of the applied torque during stance. τ_0 is selected from $-10\ Nm$ to $10\ Nm$. These region is selected from the realistic capacity of the monopod robot. Both negative and positive torques are applied, so that the energy of the system can be regulated in both ways.

4.2.2 Dataset Construction

TD-SLIP model simulation is used to generate **60000** apex return maps with the selected parameters from the determined range. Histogram of selected initial apex state variables, is shown in **Figure 4.1** and histogram of selected control inputs, is shown in **Figure 4.2**. Uniformity tests are not applied over randomly selected parameter ranges, since some of these parameters will result unrealistic maps and cannot be used for the training.

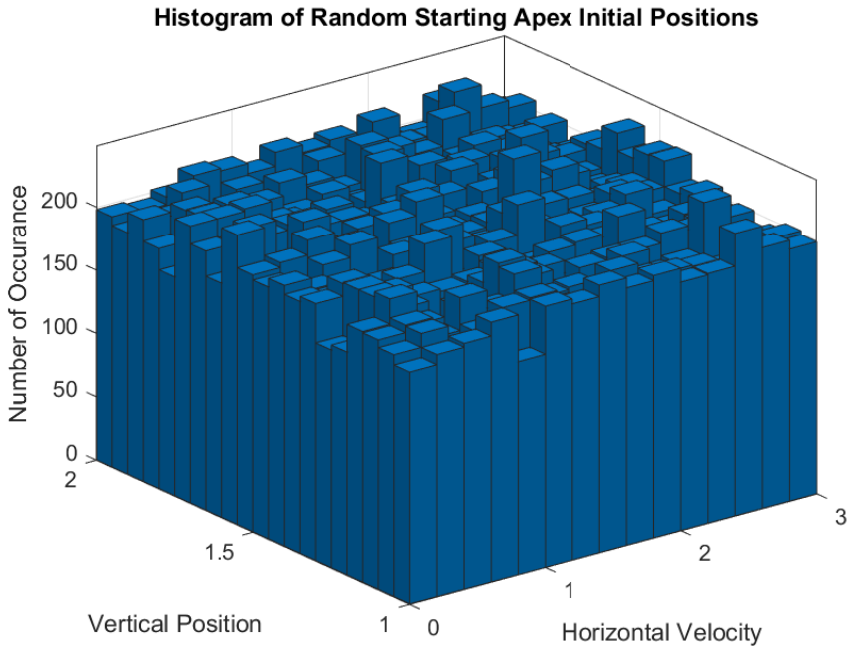


Figure 4.1: Histogram of Random Starting Apex Initial Positions

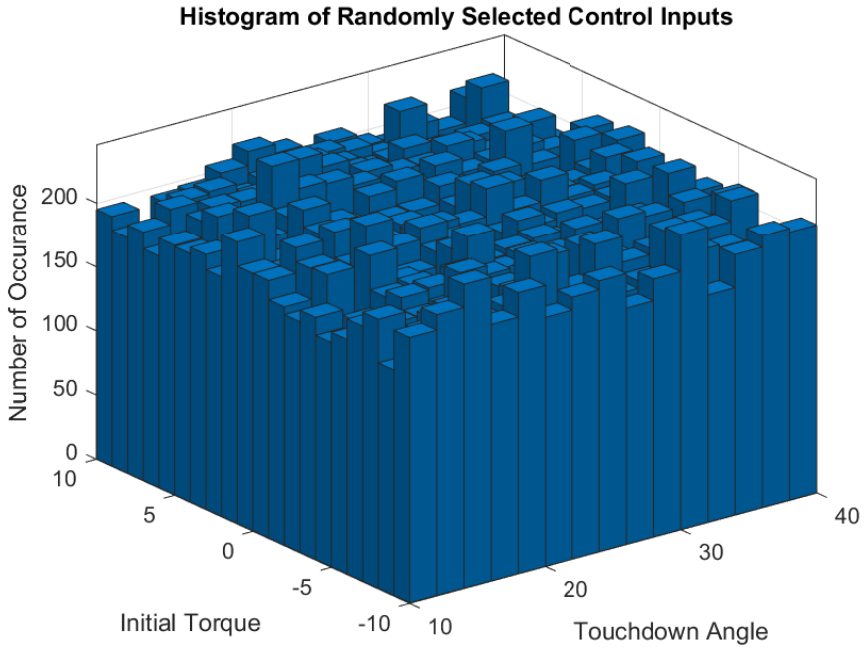


Figure 4.2: Histogram of Randomly Selected Control Inputs

Total number of apex return maps without any back jumps or unrealistic events is **27682**. Histogram of initial apex state variables which result utilizable apex return maps, is shown in **Figure 4.3**. It can be seen that the distribution of initial apex state variables in Figure 4.3 is nonuniform. Histogram of control inputs which result utilizable apex return maps, is also shown in shown in **Figure 4.4**. As it can be observed when the the TD-SLIP monopod body is slower in the horizontal direction or the perpendicularity of the TD-SLIP monopod leg to the ground is increased, risk of an unrealistic apex return map increases. Final apex state variables of Figure 4.3 are also shown in **Figure 4.5**. Some final apex positions are also found to be unrealistic in terms of the TD-SLIP body position. So that the apex return maps with final apex vertical positions are higher than 3 l_0 are also excluded from the training. Total number of apex return maps included in TD-SLIP dataset is **26887**.

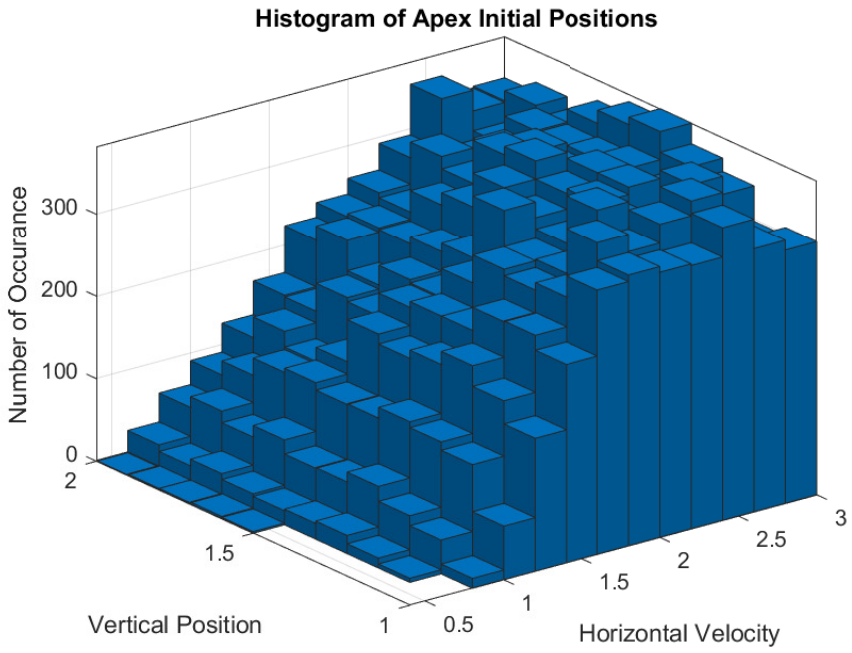


Figure 4.3: Histogram of Apex Initial Positions

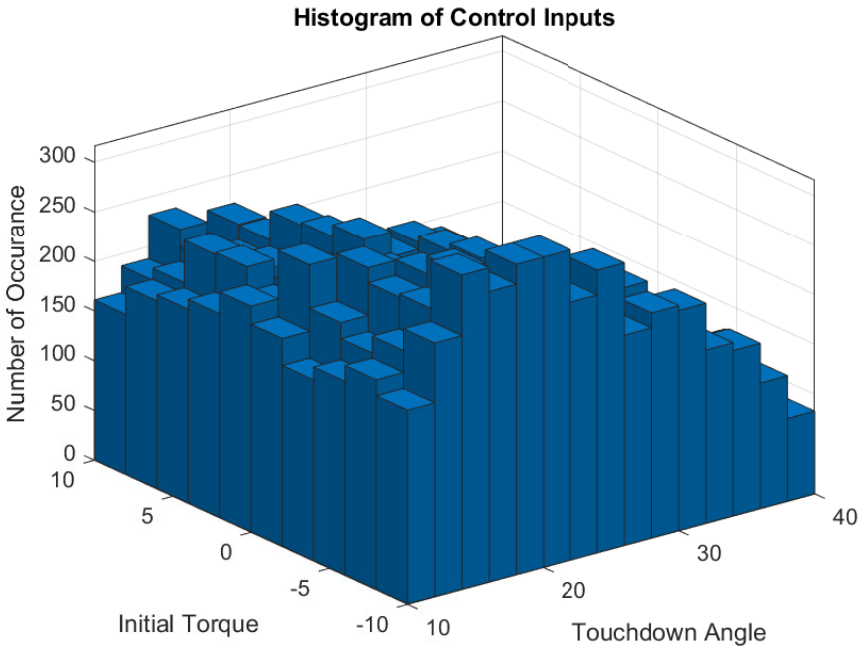


Figure 4.4: Histogram of Control Inputs

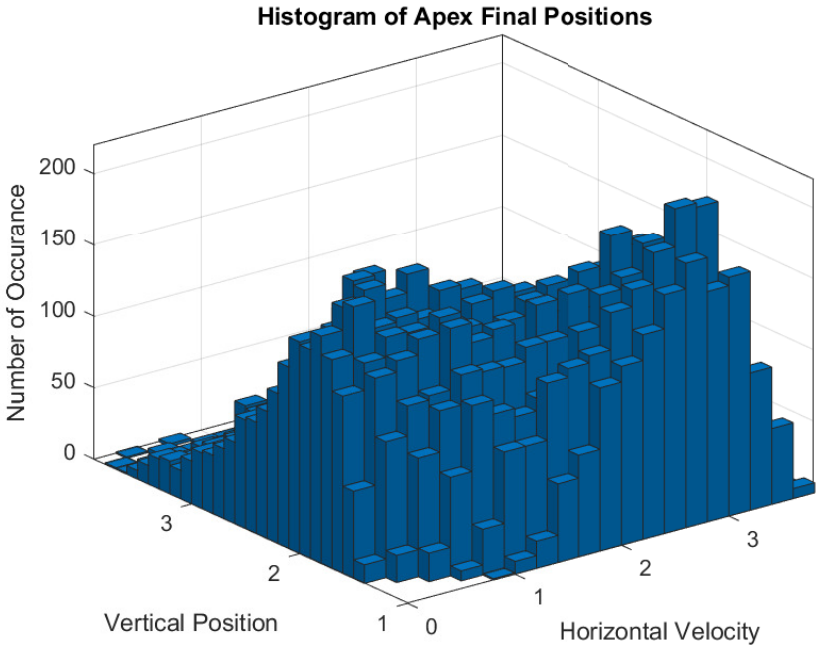


Figure 4.5: Histogram of Apex Final Positions

4.3 Neural Networks Architectures and Properties

As the neural networks has showed promising results in **chapter 3**, MATLAB Deep Learning Toolbox is used to develop neural networks also for TD-SLIP model. While designing neural networks; different type of activation functions and cost functions are used. For each task some of the parameters are also re-optimized and some of the parameters changed according to the architecture of the neural network.

For layer structure, loss functions and activation functions same designs used with **section 3.3**. Details about the layer structure, loss functions and activation functions can be revisited in **subsection 3.3.1**, **3.3.2** and **3.3.3**.

4.4 Training Estimator (Forward Kinematic Map Predictor)

Estimators are trained using different loss functions and activation functions given in subsection 3.3.2 and 3.3.3. For each loss function type, neural networks with three types of activation functions are trained resulting a total of 6 different estimators for each task. Since estimators are trained for apex to apex and touchdown to liftoff forward kinematic map predictions separately, total number trained estimators for TD-SLIP Monopod Model is 12. Each neural network has 12 fully connected layers with 256 neurons in the most populated layer.

4.4.1 Selected Algorithm and Parameters for Training

Stochastic gradient descent algorithm is used for training. Training parameters include maximum number of epochs (ϵ_{max}), initial learn rate (η), learn rate drop

period, learn rate drop rate, momentum coefficient (α), regularization coefficient for weights (L_2), mini-batch size and validation frequency.

- Using the same process detailed in subsection 3.4.1, initial learn rate (η) and regularization coefficient (L_2) are optimized before training. Optimization of η and L_2 are shown in Figure 4.6 and 4.7. Using the optimization results η_{MAER} is selected as 2.1544×10^{-7} and L_2 is selected as 0.4642. η_{HMSE} is increased with the same reasons detailed in subsection 3.4.1. Learning rate used for estimators with HMSE loss are $\eta_{HMSE} = 2.1544 \times 10^{-5}$.
- Maximum number of epochs (ϵ_{max}), is determined to be 10000. Changes in the loss and RMSE values becomes insignificant around ϵ_{10000}, so the training is continued until the convergence.
- Learn rate drop period is 1000 epochs and learn rate drop rate is 0.6. To prevent oscillation of the loss and RMSE values, learning rate is decreased during the training of the estimators with MAER loss. Learning rate drop is not applied during the training of the estimators with HMSE loss.
- Momentum coefficient (α) is 0.9 to prevent over fitting.
- Mini-batch size is 10.
- Validation frequency is one for every 100 epochs.

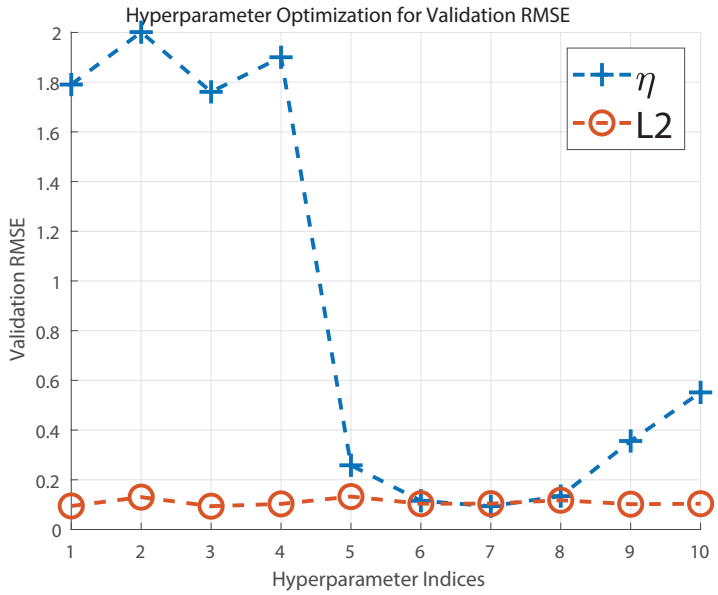


Figure 4.6: Hyperparameter Optimization on Final Validation RMSE

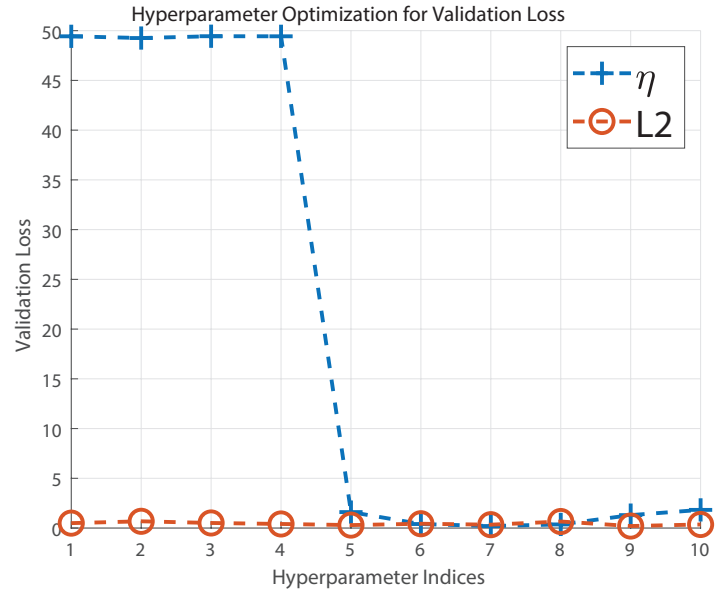


Figure 4.7: Hyperparameter Optimization on Final Validation Loss

4.4.2 Training Results of TD-SLIP Estimators

After developing neural network architectures, setting parameters and selecting the required inputs and outputs from TD-SLIP Dataset (section 4.2) training is completed for all 12 estimators. Training process and results are reported for each network separately. Success of the Estimators are measured with Mean Absolute Error Ratio and reported for every state variable in training, validation and test subsets. For apex to apex estimators errors for final apex state variables are reported, while for the touchdown to liftoff estimators errors for both liftoff and final apex state variables are reported. Final apex state variables are calculated from the liftoff state variables as described in section 2.1. Also some training parameters are arranged according to the type and design of the estimator.

4.4.2.1 a2a-Estimators with HMSE Loss on the Output

Training parameters are used as stated in subsection 4.4.1, applying the required changes for neural networks with HMSE loss.

Training processes of the apex to apex estimators with HMSE loss on the output are shown in Figure 4.8, 4.9 and 4.10 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 4.1, 4.2 and 4.3 in the same order.

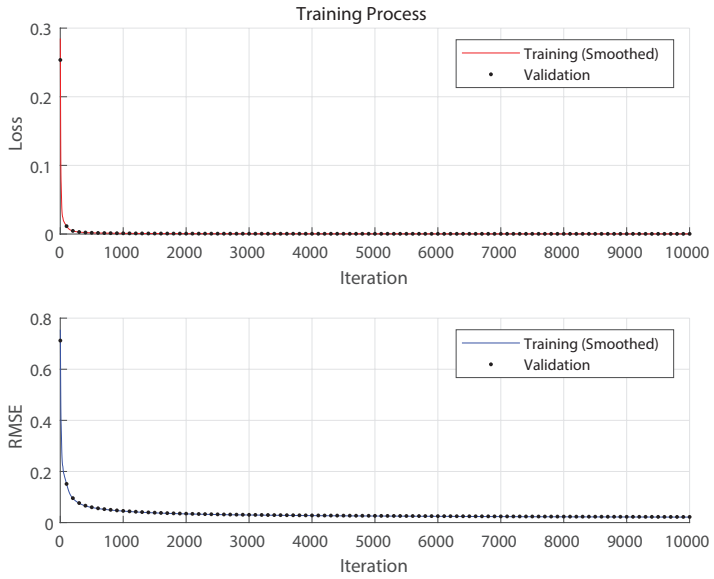


Figure 4.8: Training Process of Apex to Apex Estimator with HMSE and tanh Configuration

Table 4.1: Training Results and Errors over Subsets for Apex to Apex Estimator with HMSE and tanh Configuration

Training Duration:			0:07:57
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0002
		Validation:	0.0002
Mean Absolute Error Ratio over Subsets	Training	y_f	0.1499055
		$\dot{y}_f$	0.074676149
		z_f	0.079763159
	Validation	y_f	0.099935725
		$\dot{y}_f$	0.084168807
		z_f	0.099447072
	Test	y_f	0.113172
		$\dot{y}_f$	0.1293059
		z_f	0.10626145

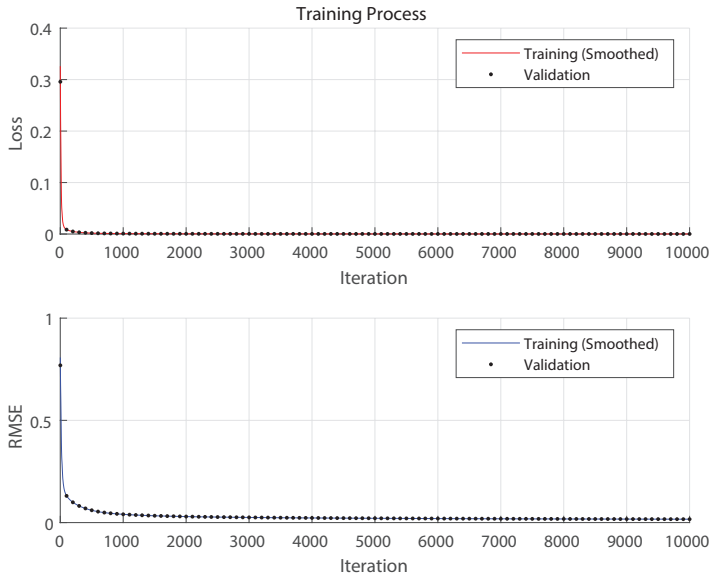


Figure 4.9: Training Process of Apex to Apex Estimator with HMSE and eLU Configuration

Table 4.2: Training Results and Errors over Subsets for Apex to Apex Estimator with HMSE and eLU Configuration

Training Duration:			0:14:47
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0001
		Validation:	0.0001
Mean Absolute Error Ratio over Subsets	Training	y_f	0.12573208
		$\dot{y}_f$	0.054059822
		z_f	0.09036269
	Validation	y_f	0.090949468
		$\dot{y}_f$	0.081733242
		z_f	0.11004723
	Test	y_f	0.11030825
		$\dot{y}_f$	0.072162688
		z_f	0.093498595

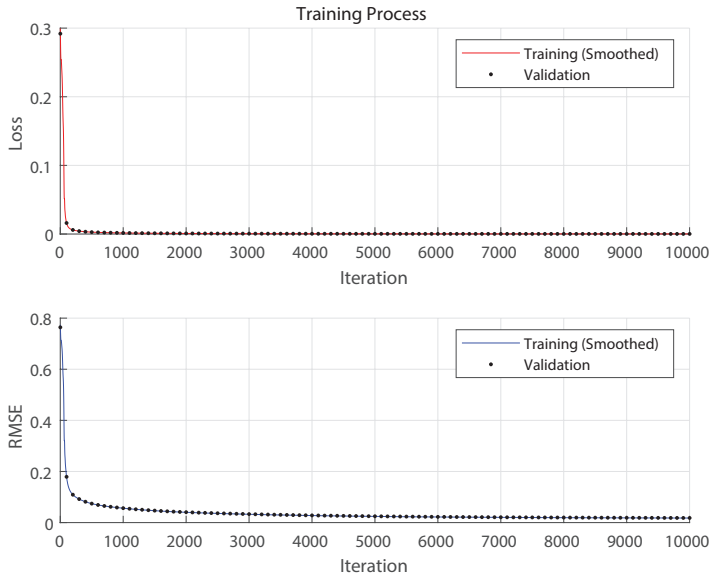


Figure 4.10: Training Process of Apex to Apex Estimator with HMSE and l-ReLU Configuration

Table 4.3: Training Results and Errors over Subsets for Apex to Apex Estimator with HMSE and l-ReLU Configuration

Training Duration:			0:07:39
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0002
		Validation:	0.0002
Mean Absolute Error Ratio over Subsets	Training	y_f	0.2821061
		$\dot{y}_f$	0.10314602
		z_f	0.13896321
	Validation	y_f	0.16145411
		$\dot{y}_f$	0.21014743
		z_f	0.20746098
	Test	y_f	0.20393127
		$\dot{y}_f$	0.21183619
		z_f	0.22344483

4.4.2.2 a2a-Estimators with MAER Loss on the Output

For the apex to apex estimators with MAER loss, parameters for training are used as stated in subsection 4.4.1.

Training processes of the apex to apex estimators with MAER loss on the output are shown in Figure 4.11, 4.12 and 4.13 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 4.4, 4.5 and 4.6 in the same order.

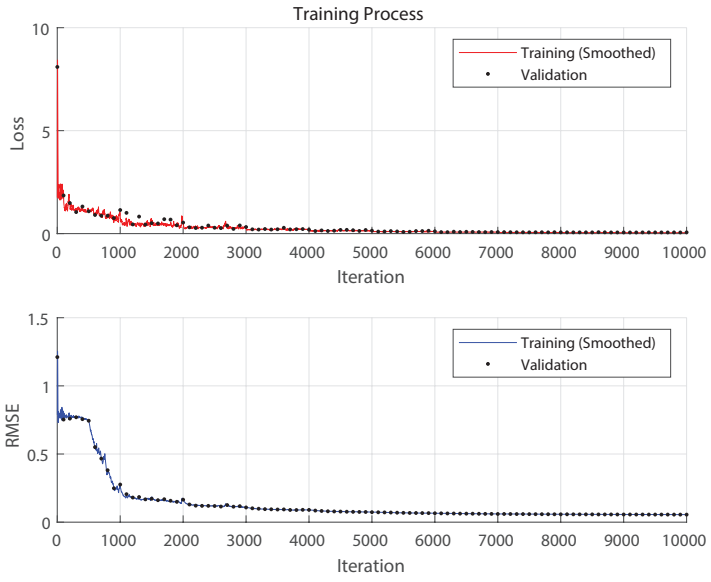


Figure 4.11: Training Process of Apex to Apex Estimator with MAER and tanh Configuration

Table 4.4: Training Results and Errors over Subsets for Apex to Apex Estimator with MAER and tanh Configuration

Training Duration:			0:07:11
Final RMSE After Training		Training:	0.06
		Validation:	0.06
Final Loss After Training		Training:	0.0433
		Validation:	0.0604
Mean Absolute Error Ratio over Subsets	Training	y_f	0.045725554
		$\dot{y}_f$	0.038786817
		z_f	0.055051904
	Validation	y_f	0.057119701
		$\dot{y}_f$	0.060070053
		z_f	0.063900255
	Test	y_f	0.054333657
		$\dot{y}_f$	0.048390649
		z_f	0.062488392

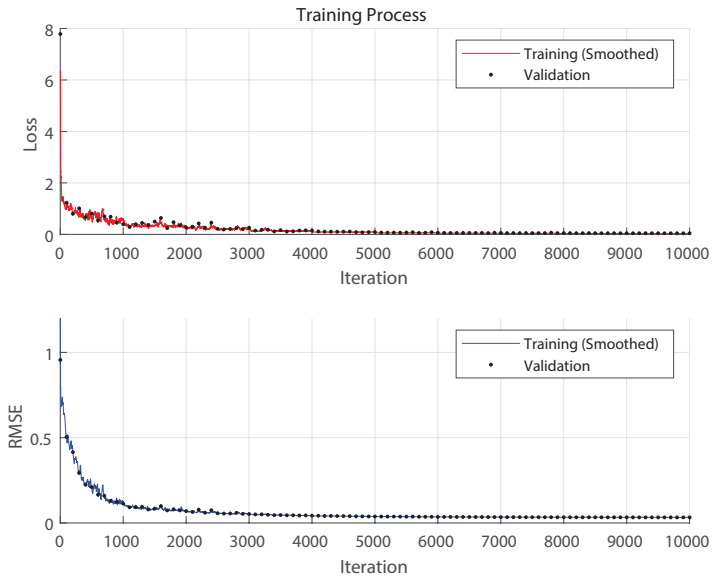


Figure 4.12: Training Process of Apex to Apex Estimator with MAER and eLU Configuration

Table 4.5: Training Results and Errors over Subsets for Apex to Apex Estimator with MAER and eLU Configuration

Training Duration:			0:14:39
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0309
		Validation:	0.0442
Mean Absolute Error Ratio over Subsets	Training	y_f	0.032311376
		$\dot{y}_f$	0.029075585
		z_f	0.02801046
	Validation	y_f	0.040646754
		$\dot{y}_f$	0.054048151
		z_f	0.037870739
	Test	y_f	0.045694817
		$\dot{y}_f$	0.038462657
		z_f	0.036747638

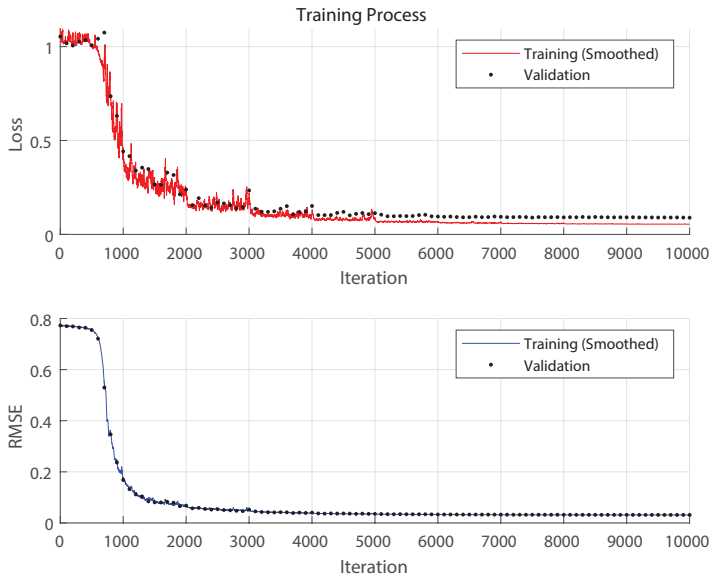


Figure 4.13: Training Process of Apex to Apex Estimator with MAER and l-ReLU Configuration

Table 4.6: Training Results and Errors over Subsets for Apex to Apex Estimator with MAER and l-ReLU Configuration

Training Duration:			0:07:33
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0559
		Validation:	0.0888
Mean Absolute Error Ratio over Subsets	Training	y_f	0.070871644
		$\dot{y}_f$	0.044692017
		z_f	0.050765365
	Validation	y_f	0.090767302
		$\dot{y}_f$	0.067389876
		z_f	0.10822126
	Test	y_f	0.095806599
		$\dot{y}_f$	0.13097343
		z_f	0.093624689

4.4.2.3 t2l-Estimators with HMSE Loss on the Output

For the touchdown to liftoff estimators with HMSE loss, same parameters with apex to apex estimators with HMSE loss are used for training.

Training processes of the touchdown to liftoff estimators with HMSE loss on the output are shown in Figure 4.14, 4.15 and 4.16 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 4.7, 4.8 and 4.9 in the same order.

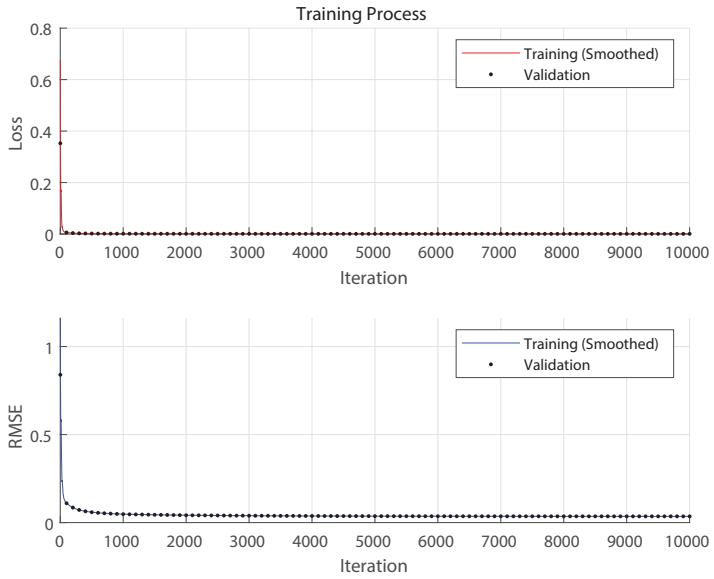


Figure 4.14: Training Process of Touchdown to Liftoff Estimator with HMSE and tanh Configuration

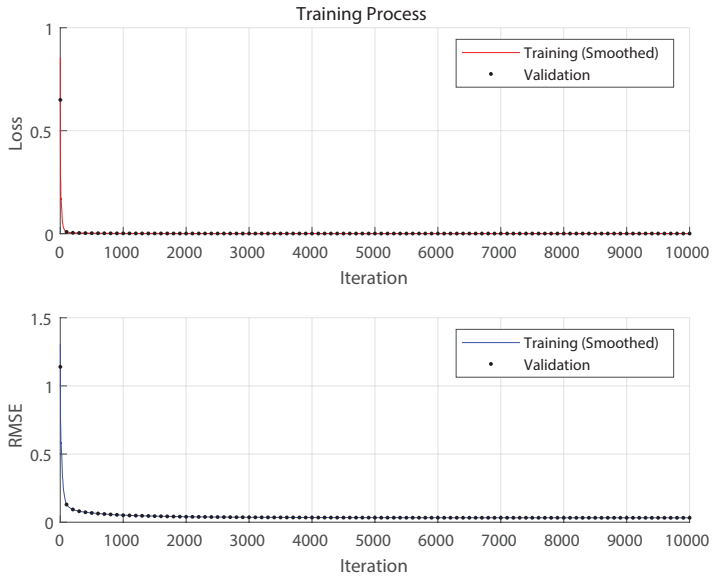


Figure 4.15: Training Process of Touchdown to Liftoff Estimator with HMSE and eLU Configuration

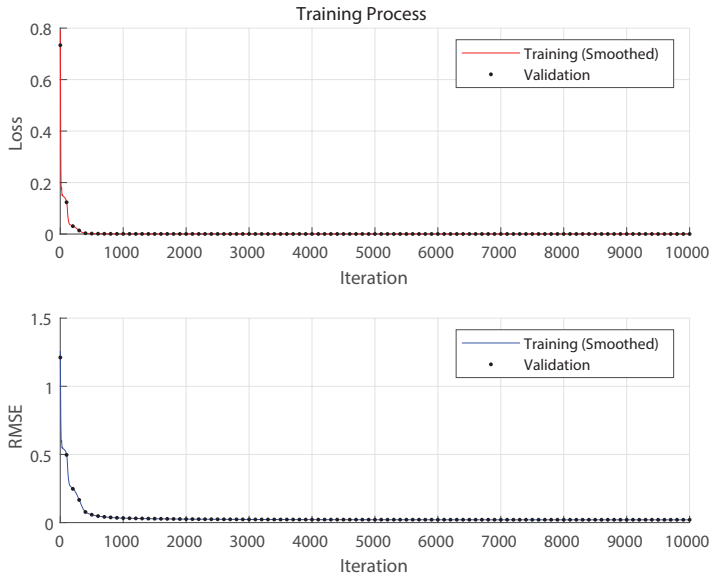


Figure 4.16: Training Process of Touchdown to Liftoff Estimator with HMSE and l-ReLU Configuration

Table 4.7: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with HMSE and tanh Configuration

Training Duration:			0:07:42
Final RMSE After Training		Training:	0.04
		Validation:	0.04
Final Loss After Training		Training:	0.0006
		Validation:	0.0006
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.070023276
		$\dot{y}_{lo}$	0.093282647
		z_{lo}	0.0053657368
		$\dot{z}_{lo}$	0.036420479
	Validation	y_{lo}	0.092519298
		$\dot{y}_{lo}$	0.15426439
		z_{lo}	0.0054085609
		$\dot{z}_{lo}$	0.044550512
	Test	y_{lo}	0.072782233
		$\dot{y}_{lo}$	0.14526787
		z_{lo}	0.0054154838
		$\dot{z}_{lo}$	0.036254548
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.2155526519
		$\dot{y}_f$	0.09328270196
		z_f	0.4606097847
	Validation	y_f	0.1595953365
		$\dot{y}_f$	0.1542646826
		z_f	0.7071452197
	Test	y_f	0.1785873588
		$\dot{y}_f$	0.1452681036
		z_f	0.5797536789

Table 4.8: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with HMSE and eLU Configuration

Training Duration:			0:12:46
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0005
		Validation:	0.0005
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.047593098
		$\dot{y}_{lo}$	0.17834765
		z_{lo}	0.0078592598
		$\dot{z}_{lo}$	0.025210859
	Validation	y_{lo}	0.064227425
		$\dot{y}_{lo}$	0.28134772
		z_{lo}	0.0078570079
		$\dot{z}_{lo}$	0.031179596
	Test	y_{lo}	0.065333121
		$\dot{y}_{lo}$	0.28948408
		z_{lo}	0.0077750152
		$\dot{z}_{lo}$	0.026506398
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.1555738742
		$\dot{y}_f$	0.1783476745
		z_f	0.3986409249
	Validation	y_f	0.1278926352
		$\dot{y}_f$	0.2813481467
		z_f	0.5973791686
	Test	y_f	0.1421757952
		$\dot{y}_f$	0.289483873
		z_f	0.4632449052

Table 4.9: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with HMSE and l-ReLU Configuration

Training Duration:			0:07:13
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0002
		Validation:	0.0002
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.05288158
		$\dot{y}_{lo}$	0.12146632
		z_{lo}	0.0059151901
		$\dot{z}_{lo}$	0.023351405
	Validation	y_{lo}	0.051459059
		$\dot{y}_{lo}$	0.25345391
		z_{lo}	0.0059398408
		$\dot{z}_{lo}$	0.030199373
	Test	y_{lo}	0.048328709
		$\dot{y}_{lo}$	0.20930524
		z_{lo}	0.0059008365
		$\dot{z}_{lo}$	0.024307845
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.2583442341
		$\dot{y}_f$	0.1214663195
		z_f	0.2355163933
	Validation	y_f	0.172411991
		$\dot{y}_f$	0.2534534073
		z_f	0.340378992
	Test	y_f	0.199325015
		$\dot{y}_f$	0.2093051096
		z_f	0.2491242738

4.4.2.4 t2l-Estimators with MAER Loss on the Output

For the touchdown to liftoff estimators with MAER loss, same parameters with apex to apex estimators with MAER loss are used for training.

Training processes of the touchdown to liftoff estimators with MAER loss on the output are shown in Figure 4.17, 4.18 and 4.19 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the estimators are also reported in Table 4.10, 4.11 and 4.12 in the same order.

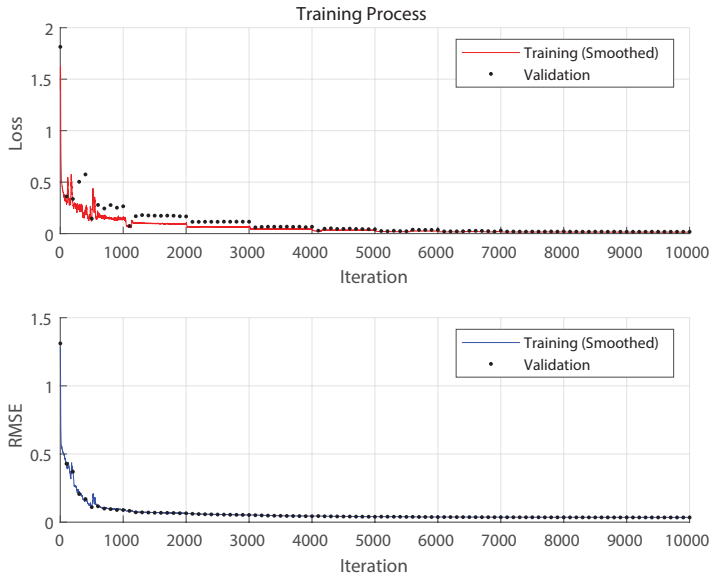


Figure 4.17: Training Process of Touchdown to Liftoff Estimator with MAER and tanh Configuration

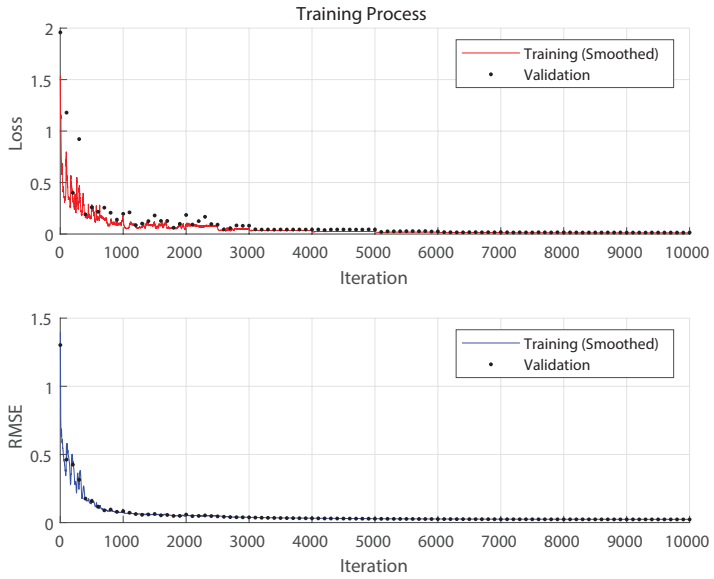


Figure 4.18: Training Process of Touchdown to Liftoff Estimator with MAER and eLU Configuration

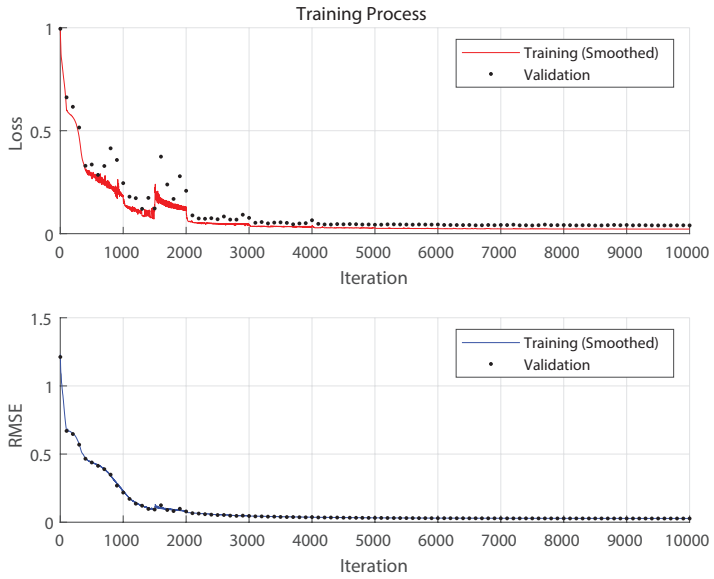


Figure 4.19: Training Process of Touchdown to Liftoff Estimator with MAER and l-ReLU Configuration

Table 4.10: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with MAER and tanh Configuration

Training Duration:			0:07:58
Final RMSE After Training		Training:	0.04
		Validation:	0.03
Final Loss After Training		Training:	0.0142
		Validation:	0.0177
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.010376887
		$\dot{y}_{lo}$	0.032012198
		z_{lo}	0.0030134849
		$\dot{z}_{lo}$	0.01110398
	Validation	y_{lo}	0.01183226
		$\dot{y}_{lo}$	0.044174764
		z_{lo}	0.0030273211
		$\dot{z}_{lo}$	0.01175295
	Test	y_{lo}	0.011536852
		$\dot{y}_{lo}$	0.044983875
		z_{lo}	0.0030011409
		$\dot{z}_{lo}$	0.012986336
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.07490286986
		$\dot{y}_f$	0.03201218596
		z_f	0.2230533843
	Validation	y_f	0.08118762892
		$\dot{y}_f$	0.04417505292
		z_f	0.1971142834
	Test	y_f	0.09185778668
		$\dot{y}_f$	0.04498373907
		z_f	0.1764716739

Table 4.11: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with MAER and eLU Configuration

Training Duration:			0:14:50
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0114
		Validation:	0.0143
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.0092361383
		$\dot{y}_{lo}$	0.025824154
		z_{lo}	0.0030221627
		$\dot{z}_{lo}$	0.007697403
	Validation	y_{lo}	0.010500723
		$\dot{y}_{lo}$	0.035714909
		z_{lo}	0.0030178917
		$\dot{z}_{lo}$	0.0078362636
	Test	y_{lo}	0.011432099
		$\dot{y}_{lo}$	0.033160407
		z_{lo}	0.0030282068
		$\dot{z}_{lo}$	0.0079803476
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.07498900793
		$\dot{y}_f$	0.0258241907
		z_f	0.1626951463
	Validation	y_f	0.06927385845
		$\dot{y}_f$	0.03571529077
		z_f	0.1086664074
	Test	y_f	0.08410808409
		$\dot{y}_f$	0.03316038162
		z_f	0.1400018458

Table 4.12: Training Results and Errors over Subsets for Touchdown to Liftoff Estimator with MAER and l-ReLU Configuration

Training Duration:			0:07:02
Final RMSE After Training		Training:	0.03
		Validation:	0.03
Final Loss After Training		Training:	0.0222
		Validation:	0.0401
Mean Absolute Error Ratio over Subsets (Touchdown to Liftoff)	Training	y_{lo}	0.018420162
		$\dot{y}_{lo}$	0.049523044
		z_{lo}	0.0053307214
		$\dot{z}_{lo}$	0.015730143
	Validation	y_{lo}	0.026945788
		$\dot{y}_{lo}$	0.11049977
		z_{lo}	0.005295618
		$\dot{z}_{lo}$	0.017599767
	Test	y_{lo}	0.025681499
		$\dot{y}_{lo}$	0.093659751
		z_{lo}	0.0052562659
		$\dot{z}_{lo}$	0.018259948
Mean Absolute Error Ratio over Subsets (Apex to Apex)	Training	y_f	0.1692703621
		$\dot{y}_f$	0.0495231163
		z_f	0.3700238415
	Validation	y_f	0.2140308759
		$\dot{y}_f$	0.1104996375
		z_f	0.3415178563
	Test	y_f	0.1815207558
		$\dot{y}_f$	0.09365976898
		z_f	0.3742885071

4.4.3 Comparison of TD-SLIP Estimators

Comparison of TD-SLIP Estimators forms a parallel result with the comparison of SLIP Estimators, discussed in subsection 3.4.3. In section 4.1 it is stated that a2a-estimators are designed to be the neural network equivalent of TD-SLIP simulation. To understand the success of the estimators on this task, Mean Absolute Error Ratios over subsets are taken into consideration.

By observing the error ratios, one can say that, a2a-estimators with HMSE loss are less successful than a2a-estimators with MAER loss. Same pattern is also correct for t2l estimators, if apex to apex errors are taken into consideration. These differences can be explained by the compatibility of the expected errors with used loss function. Estimators with MAER loss is optimized exactly for the expected success measure. Thus, it can be observed that both estimator types with MAER loss are more suitable to replace the TD-SLIP simulation.

After selecting the estimators with MAER loss; success rates of different types can also be compared by using apex to apex results. Between a2a-estimators and t2l-estimators, it can be said that a2a-estimators produce more correct predictions for the final apex, while t2l estimators produce more correct result for the determined outputs.

Apex to apex errors of t2l-estimators are higher than touchdown to liftoff errors because of error accumulation. While calculating the final apex state using the liftoff state, equations in section 2.1 are used. In these equations final apex positions are calculated from integration of liftoff velocities. So errors in liftoff velocities projects higher errors for final apex positions.

4.5 Training Controller (Inverse Kinematic Map Predictor)

Training approach for the controllers is similar to the training approach of the estimators. Controllers are trained using different loss functions and activation functions given in subsection 3.3.2 and 3.3.3. For each loss function type, neural networks with three types of activation functions are trained resulting a total of 6 different controllers. As also mentioned in subsection 4.1.2; unlike estimators, controllers are trained only for apex to apex inverse kinematic map prediction. Layer structure for the neural networks is also kept same with the estimator, so each neural network has 12 fully connected layers with 256 neurons in the most populated layer.

4.5.1 Selected Algorithm and Parameters for Training

Stochastic gradient descent algorithm is used for training. Training parameters include maximum number of epochs (ϵ_{max}), initial learn rate (η), learn rate drop period, learn rate drop rate, momentum coefficient (α), regularization coefficient for weights (L_2), mini-batch size and validation frequency.

- Using the same process detailed in subsection 3.4.1, initial learn rate (η) and regularization coefficient (L_2) are optimized before training. Optimization of η and L_2 are shown in Figure 4.20 and 4.21. Using the optimization results η_{MAER} is selected as 2.1544×10^{-7} and L_2 is selected as 0.2154. η_{HMSE} is increased with the same reasons detailed in subsection 3.4.1. Learning rate used for controllers with HMSE loss are $\eta_{HMSE} = 2.1544 \times 10^{-6}$.
- Maximum number of epochs (ϵ_{max}), is determined to be 15000. Changes in the loss and RMSE values becomes insignificant around ϵ_{15000}, so the training is continued until the convergence.
- Learn rate drop period is 1000 epochs and learn rate drop rate is 0.6. To

prevent oscillation of the loss and RMSE values, learning rate is decreased during the training of the controllers with MAER loss. Learning rate drop is not applied during the training of the controllers with HMSE loss.

- Momentum coefficient (α) is 0.9 to prevent over fitting.
- Mini-batch size is 10.
- Validation frequency is one for every 100 epochs.

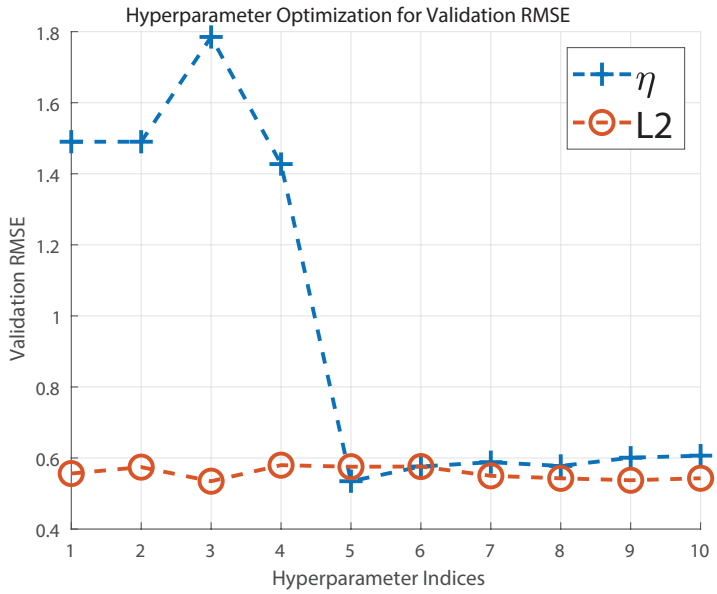


Figure 4.20: Hyperparameter Optimization on Final Validation RMSE

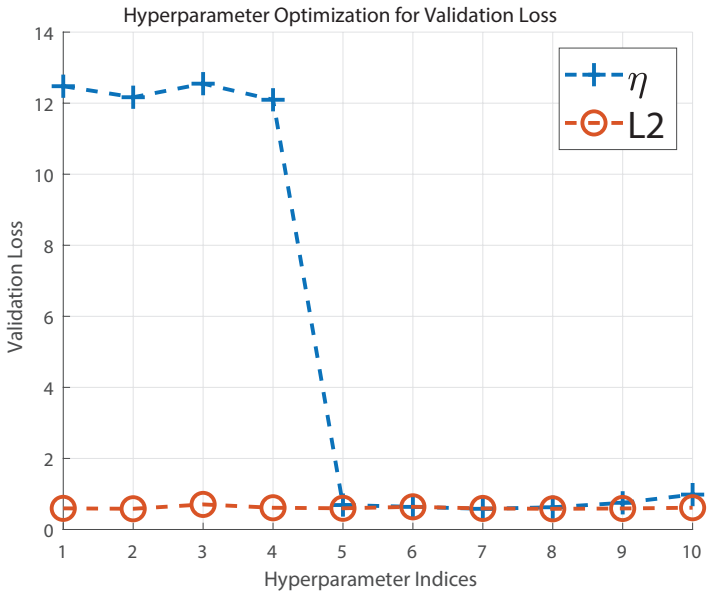


Figure 4.21: Hyperparameter Optimization on Final Validation Loss

4.5.2 Training Results of TD-SLIP Controllers

Trained controllers are tested with TD-SLIP simulation and errors for reaching goal state variables are also reported, on top of the success measures used to report estimator results. This additional step of success rate measurement, stands as the verification of the trained controllers. Block diagram explaining the test structure is given in Figure 4.22.

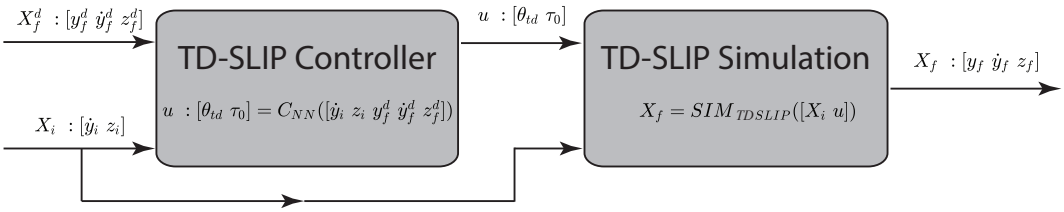


Figure 4.22: Block Diagram of Testing Controllers Against TD-SLIP Simulation

In Figure 4.22, a trained TD-SLIP Controller, C_{NN}, is used to predict the control inputs, u, that bring the monopod from initial apex state, X_i, to a desired final apex state, X_f^d. Than the predicted control inputs are used with TD-SLIP Simulation, $SIM_{TD-SLIP}$, to compute the achieved final apex state, X_f. Success of the trained controller is measured by calculating the error of X_f respect to X_f^d with MAER formula.

4.5.2.1 a2a-Controllers with HMSE Loss on the Output

Training parameters are used as stated in subsection 4.5.1, applying the required changes for neural networks with HMSE loss.

Training processes of the apex to apex controllers with HMSE loss on the output are shown in Figure 4.23, 4.24 and 4.25 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the controllers are also reported in Table 4.13, 4.14 and 4.15 in the same order.

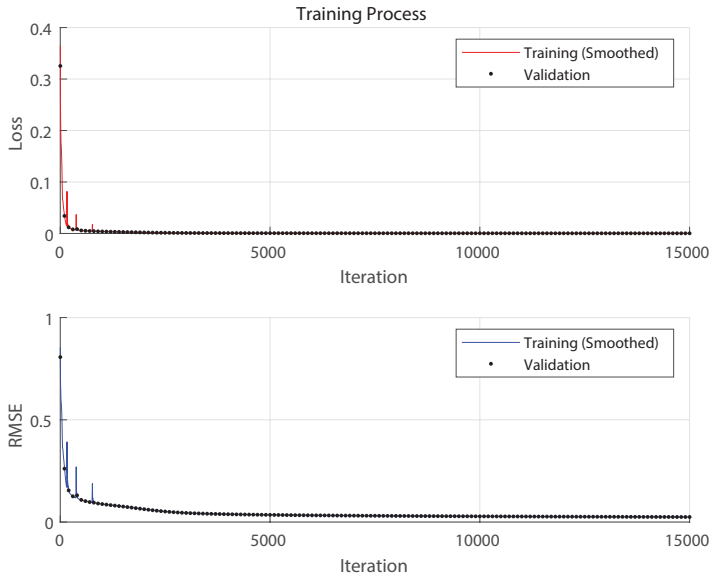


Figure 4.23: Training Process of Controller with HMSE and tanh Configuration

Table 4.13: Training Results and Errors over Subsets for Controller with HMSE and tanh Configuration

Training Duration:			0:11:30
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0003
		Validation:	0.0003
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.092082366
		τ_0	0.084488876
	Validation	θ_{td}	0.086861119
		τ_0	0.075495683
	Test	θ_{td}	0.10032888
		τ_0	0.089022219
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.007864830412
		$\dot{y}_f$	0.1610192738
		z_f	0.004423687926
	Validation	y_f	0.007600699928
		$\dot{y}_f$	0.216714842
		z_f	0.004448371909
	Test	y_f	0.007898092969
		$\dot{y}_f$	0.1243604703
		z_f	0.004421095532

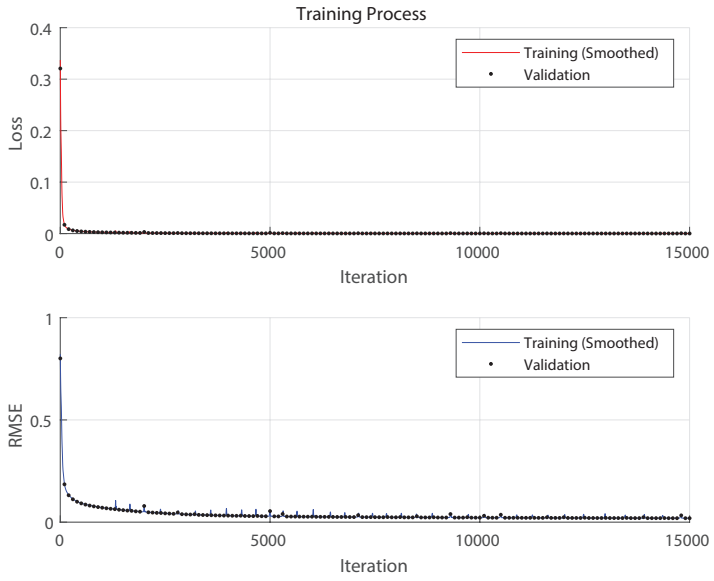


Figure 4.24: Training Process of Controller with HMSE and eLU Configuration

Table 4.14: Training Results and Errors over Subsets for Controller with HMSE and eLU Configuration

Training Duration:			0:19:05
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0002
		Validation:	0.0002
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.073457628
		τ_0	0.079166196
	Validation	θ_{td}	0.074321017
		τ_0	0.08695399
	Test	θ_{td}	0.084820814
		τ_0	0.075128056
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.005556958442
		$\dot{y}_f$	0.06299860602
		z_f	0.004189524075
	Validation	y_f	0.005590963279
		$\dot{y}_f$	0.1159493969
		z_f	0.004250579744
	Test	y_f	0.005697877638
		$\dot{y}_f$	0.06578167092
		z_f	0.004213411558

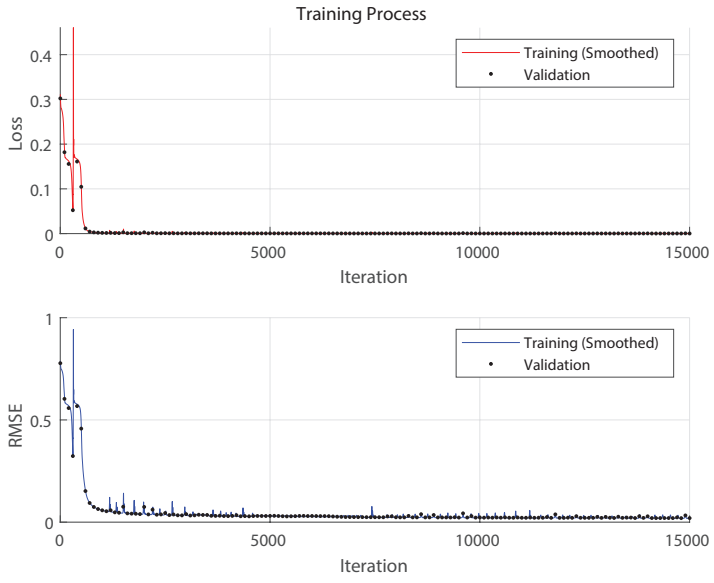


Figure 4.25: Training Process of Controller with HMSE and l-ReLU Configuration

Table 4.15: Training Results and Errors over Subsets for Controller with HMSE and l-ReLU Configuration

Training Duration:			0:10:35
Final RMSE After Training		Training:	0.02
		Validation:	0.02
Final Loss After Training		Training:	0.0002
		Validation:	0.0002
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.09147983
		τ_0	0.15082973
	Validation	θ_{td}	0.092706181
		τ_0	0.1059919
	Test	θ_{td}	0.1080826
		τ_0	0.11375234
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.004741280895
		$\dot{y}_f$	0.06293848536
		z_f	0.003992698182
	Validation	y_f	0.004939844198
		$\dot{y}_f$	0.0766367157
		z_f	0.00417794162
	Test	y_f	0.00495592435
		$\dot{y}_f$	0.04971637752
		z_f	0.004212706617

4.5.2.2 a2a-Controllers with MAER Loss on the Output

For the apex to apex controllers with MAER loss, parameters for training are used as stated in subsection 4.5.1.

Training processes of the apex to apex controllers with MAER loss on the output are shown in Figure 4.26, 4.27 and 4.28 while using tanh, eLU or l-ReLU activation functions respectively. Success rates for the controllers are also reported in Table 4.16, 4.17 and 4.18 in the same order.

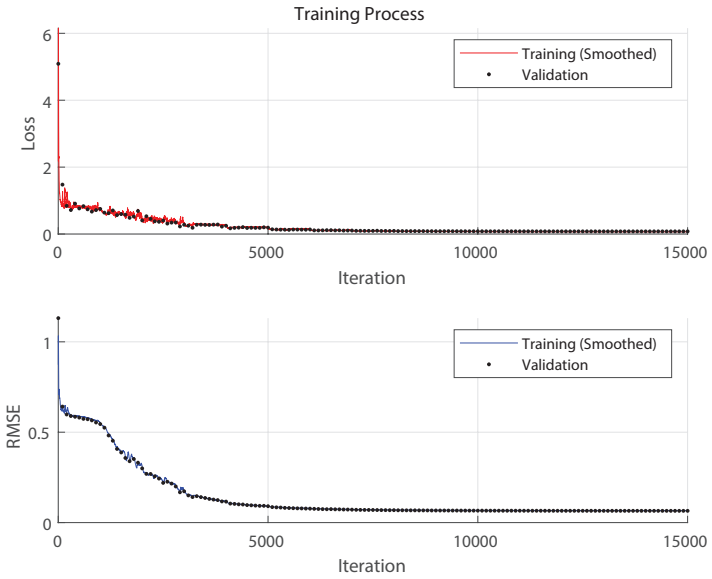


Figure 4.26: Training Process of Controller with MAER and tanh Configuration

Table 4.16: Training Results and Errors over Subsets for Controller with MAER and tanh Configuration

Training Duration:			0:10:56
Final RMSE After Training		Training:	0.07
		Validation:	0.07
Final Loss After Training		Training:	0.0664
		Validation:	0.0791
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.041129548
		τ_0	0.091348998
	Validation	θ_{td}	0.05029276
		τ_0	0.10784642
	Test	θ_{td}	0.049211826
		τ_0	0.10581311
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.0150751874
		$\dot{y}_f$	0.1864403087
		z_f	0.008382313309
	Validation	y_f	0.01496703379
		$\dot{y}_f$	0.3320137866
		z_f	0.008380317781
	Test	y_f	0.01515987422
		$\dot{y}_f$	0.1799003101
		z_f	0.008319847984

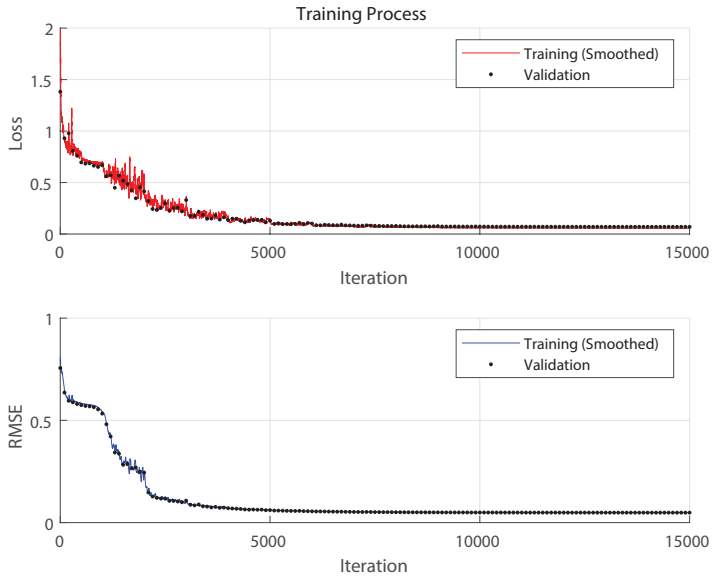


Figure 4.27: Training Process of Controller with MAER and tanh Configuration

Table 4.17: Training Results and Errors over Subsets for Controller with MAER and eLU Configuration

Training Duration:			0:21:51
Final RMSE After Training		Training:	0.05
		Validation:	0.05
Final Loss After Training		Training:	0.055
		Validation:	0.07
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.030297361
		τ_0	0.079672217
	Validation	θ_{td}	0.037977144
		τ_0	0.10196318
	Test	θ_{td}	0.038895614
		τ_0	0.094883986
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.009545137821
		$\dot{y}_f$	0.09111402193
		z_f	0.005380703908
	Validation	y_f	0.009245282887
		$\dot{y}_f$	0.244706984
		z_f	0.005296058266
	Test	y_f	0.00936800824
		$\dot{y}_f$	0.1123735711
		z_f	0.005276510141

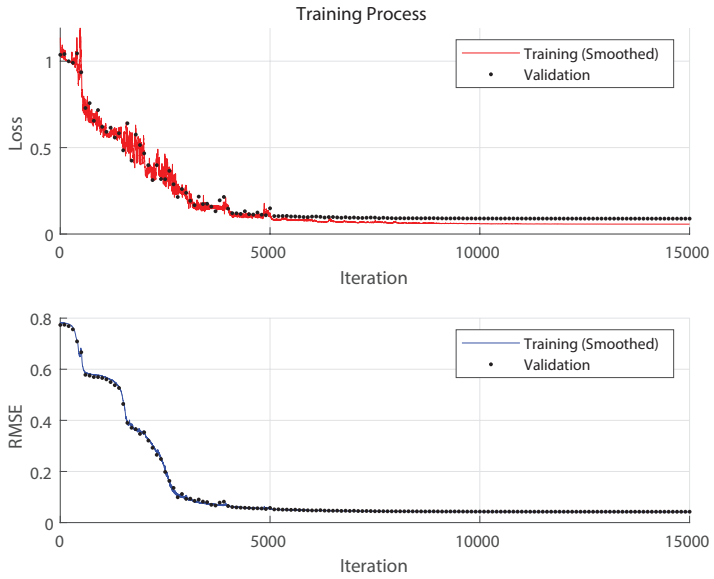


Figure 4.28: Training Process of Controller with MAER and tanh Configuration

Table 4.18: Training Results and Errors over Subsets for Controller with MAER and l-ReLU Configuration

Training Duration:			0:11:01
Final RMSE After Training		Training:	0.04
		Validation:	0.04
Final Loss After Training		Training:	0.0571
		Validation:	0.089
Mean Absolute Error Ratio over Subsets	Training	θ_{td}	0.039622389
		τ_0	0.07467062
	Validation	θ_{td}	0.058714863
		τ_0	0.11928506
	Test	θ_{td}	0.058842149
		τ_0	0.11190754
Mean Absolute Error Ratio over Subsets (Verifying Controller with Simulation)	Training	y_f	0.009437561631
		$\dot{y}_f$	0.1106543274
		z_f	0.00582569163
	Validation	y_f	0.009368892232
		$\dot{y}_f$	0.2815500122
		z_f	0.006052692472
	Test	y_f	0.009306000168
		$\dot{y}_f$	0.1488300369
		z_f	0.005926188151

4.5.3 Comparison of TD-SLIP Controllers

Comparison of TD-SLIP Controllers forms a parallel result with the comparison of SLIP Controllers, discussed in subsection 3.5.3. In section 4.1 it is stated that a2a-controllers are designed to be neural network based controllers for TD-SLIP model. To understand the success of the controllers on this task, verification results on TD-SLIP simulation are taken into consideration.

By observing the error ratios over subsets, success of the controller for reaching desired control inputs can be compared. From the error ratios over subsets, it is observable that a2a-controllers with HMSE loss are less successful than a2a-controllers with MAER loss. Controllers with MAER loss are optimized for predicting the control inputs used in that apex return map, so resulting control inputs are more accurate than the control inputs resulted from controllers with HMSE loss.

On the contrary, by observing the verification results on SLIP simulation, controllers with HMSE and MAER losses results closer success rates for reaching the desired apex state. Since there are ranges of control parameters that can be used to reach desired apex states, error margins of the control parameters projects a smaller error margins on the final apex state.

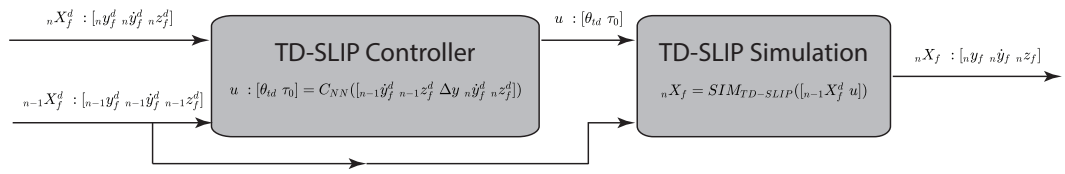
Chapter 5

Achieved Results with Neural Network Based Estimators and Controllers for SLIP and TD-SLIP Model

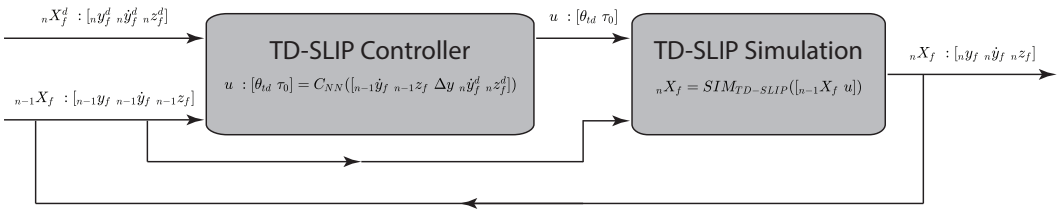
Having estimators and controllers trained for SLIP and TD-SLIP models, we can use estimators and controllers to run SLIP and TD-SLIP models. By running SLIP and TD-SLIP monopod robots with trained controllers, the success of controllers can be observed under different scenarios. Trained SLIP and TD-SLIP controllers are used both in open loop and closed loop runs. In this chapter different wave forms are tested for final apex vertical position. A new type of controller based on trained estimators is also briefly tested for comparison.

Block diagrams demonstrating the process of Open Loop and Closed Loop runs, are shown in Figure 5.1.

In Figure 5.1, a trained TD-SLIP Controller, C_{NN}, is used to predict the control inputs, u, that bring the monopod from the previous final apex state, ${}_{n-1}X_f$, to the next desired final apex state, ${}_nX_f^d$. Than the predicted control inputs are



(a) Open Loop



(b) Closed Loop

Figure 5.1: Block Diagrams Demonstrating the Process of Open Loop and Closed Loop Runs

used with TD-SLIP Simulation, $SIM_{TD-SLIP}$, to compute the next achieved final apex state, ${}_nX_f$. Success of the trained controller during the run is measured by calculating the average percentage error of X_f respect to X_f^d with MAER formula. Revisiting subsection 4.1.2 it can be checked that controllers are trained with the horizontal position of only desired final apex state, y_f, since the horizontal position of the initial apex state, y_i, is assumed as zero. To adopt the controller for the runs, instead of ${}_ny_f^d$; horizontal displacement, ${}_n\Delta y =_n y_f^d -_{n-1} y_f$, is used in controller inputs. In closed loop runs, monopod is expected to reach the next desired final apex state from the previous achieved apex state; where in open loop runs, the previously desired final apex state is assumed as achieved. So, in closed loop runs ${}_{n-1}X_f$ is set to ${}_nX_f$ after each apex and in open loop runs ${}_{n-1}X_f$ is equal to ${}_{n-1}X_f^d$. Switch between SLIP and TD-SLIP models is done by changing the simulation and controller types.

5.1 Run Scenarios

For desired final apex vertical position, z_f, four scenarios with different wave forms are created. By using different wave forms, controller characteristics can be observed under different conditions. Controllers are tested in the following scenarios:

- z_f Should be Constant Value: Desired vertical positions of final apex state are aimed to be always constant, ${}_n z_f^d = c_z$.
- z_f Should Follow a Sine Trajectory: Desired vertical positions of final apex state are aimed to form a sine wave with period of T apexes and amplitude of A: ${}_n z_f^d = c_z + A \sin 2\pi \frac{n}{T}$.
- z_f Should Follow a Square Trajectory: Desired vertical positions of final apex state are aimed to form a square wave with 50% duty cycle, period of T apexes and amplitude of A: ${}_n z_f^d = c_z + Asgn(\sin 2\pi \frac{n}{T})$.
- z_f Should Follow a Sawtooth Trajectory: Desired vertical positions of final apex state are aimed to form a sawtooth wave a period of T apexes and amplitude of A: ${}_n z_f^d = c_z + A \mod \frac{n}{T}$.

Desired horizontal displacements and velocities for final apex states are kept constant in all of the scenarios (${}_n \Delta y^d = c_y$ and ${}_n \dot{y}_f^d = c_{\dot{y}}$) to prevent unstable outputs like monopod back-jumps, falls and other unrealistic events. Desired trajectories for final apex vertical positions are shown in Figure 5.2.

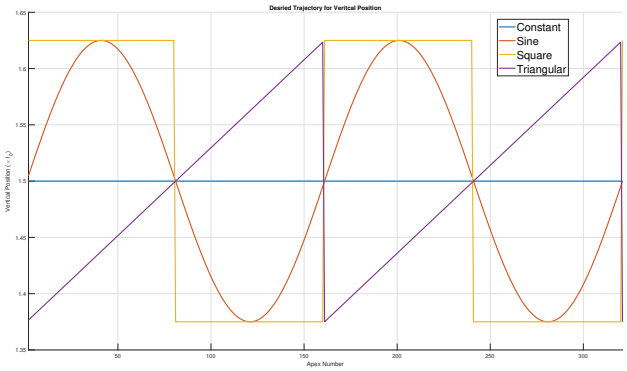


Figure 5.2: Desired Final Apex Vertical Position Trajectories

Expected behavior from the monopod is shown in Figure 5.3, including both the desired trajectory for final apex state vertical positions and the potential positions of the monopod during the run is demonstrated.

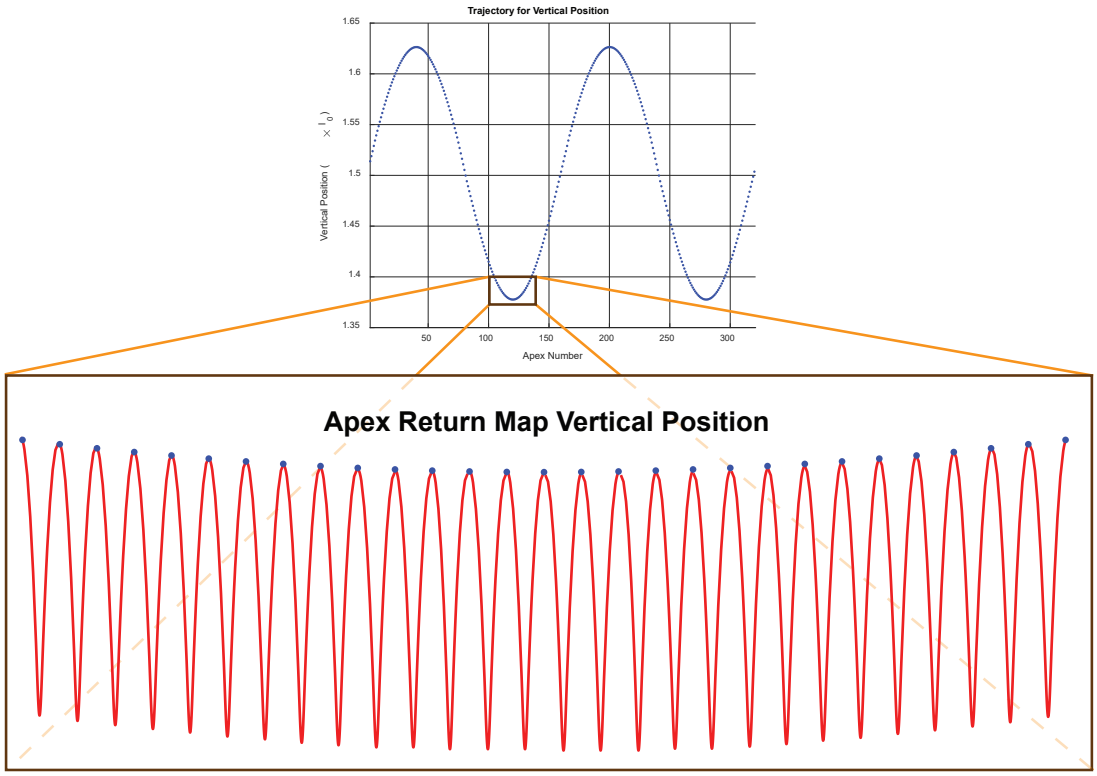
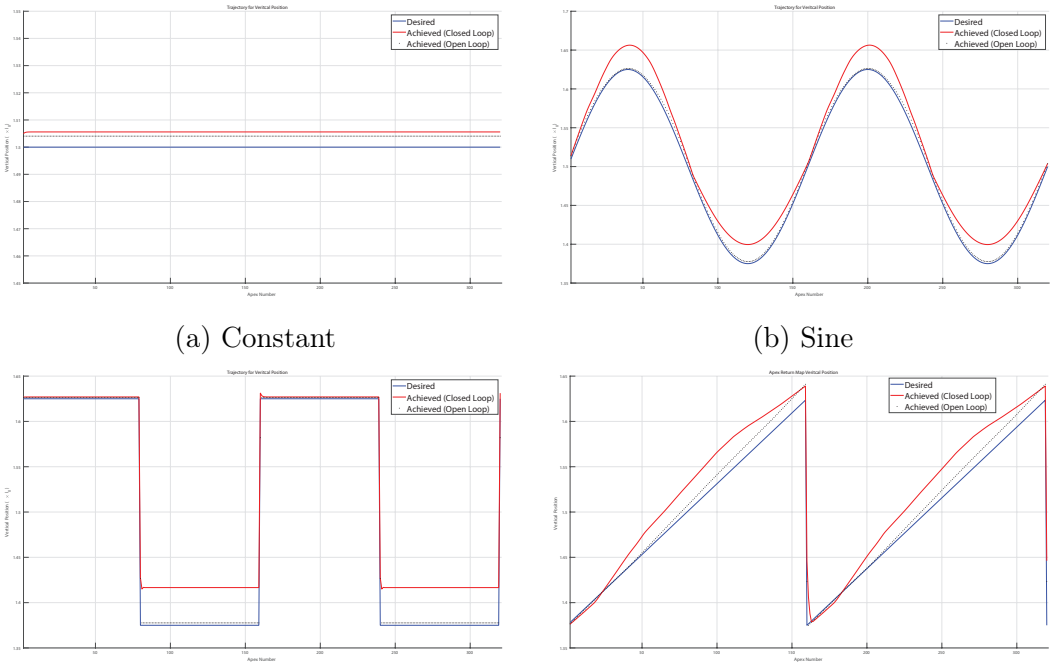


Figure 5.3: Desired Final Apex Vertical Position Trajectory with Sine Wave and Corresponding Vertical Position of the Monopod Body During Run

5.2 Testing SLIP Controllers

SLIP Controller with MAER and l-ReLU configuration is tested against the scenarios described in section 5.1. Achieved trajectories with open loop and closed loop runs are shown in Figure 5.4 for the scenarios with constant, sine, square and sawtooth waves. Maximum and average percentage errors for the achieved trajectories are also reported in Table 5.1.



(a) Constant

(b) Sine

(c) Square

(d) Sawtooh

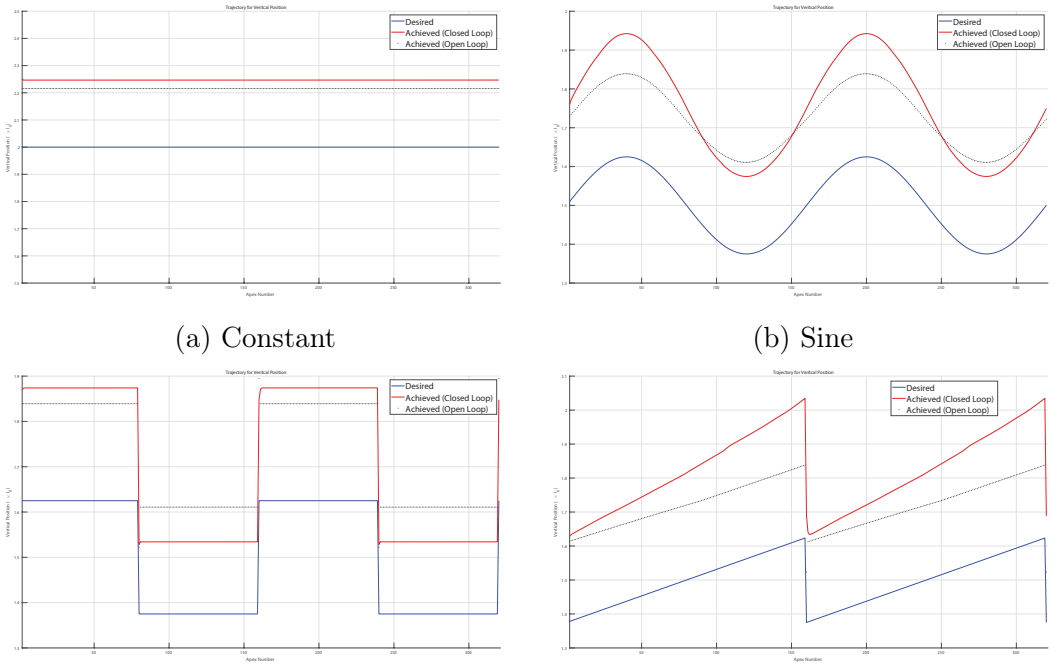
Figure 5.4: Desired and Achieved Final Apex Vertical Position Trajectories with SLIP Controller for Open Loop and Closed Loop Runs

Table 5.1: Errors for Achieved Final Apex Horizontal Position Trajectories with SLIP Controller for Open Loop and Close Loop Runs

	Run Tpye:		Open Loop	Closed Loop
Desired Trajectory Wave Form:	Constant	e_{max}	0.27 %	0.37 %
		e_{avg}	0.27 %	0.37 %
	Sine	e_{max}	0.54 %	1.97 %
		e_{avg}	0.23 %	1.24 %
	Square	e_{max}	3.77 %	3.81 %
		e_{avg}	0.18 %	1.58 %
	Sawtooh	e_{max}	2.97 %	4.72 %
		e_{avg}	0.65 %	1.01 %

5.3 Testing TD-SLIP Controllers

TD-SLIP Controller with HMSE and l-ReLU configuration is tested against the scenarios described in section 5.1. Achieved trajectories with open loop and closed loop runs are shown in Figure 5.5 for the scenarios with constant, sine, square and sawtooth waves. Maximum and average percentage errors for the achieved trajectories are also reported in Table 5.2.



(a) Constant

(b) Sine

(c) Square

(d) Sawtooh

Figure 5.5: Desired and Achieved Final Apex Vertical Position Trajectories with TD-SLIP Controller for Open Loop and Closed Loop Runs

Table 5.2: Errors for Achieved Final Apex Horizontal Position Trajectories with TD-SLIP Controller for Open Loop and Close Loop Runs

	Run Tpye:		Open Loop	Closed Loop
Desired Trajectory Wave Form:	Constant	e_{max}	10.8 %	12.57 %
		e_{avg}	10.8 %	12.33 %
	Sine	e_{max}	17.15 %	19.53 %
		e_{avg}	14.96 %	16.79 %
	Square	e_{max}	17.15 %	15.32 %
		e_{avg}	15.14 %	13.43 %
	Sawtooh	e_{max}	17.15 %	25.33 %
		e_{avg}	14.89 %	21.39 %

5.4 Comparison of SLIP and TD-SLIP Controllers

By observing the achieved results with SLIP and TD-SLIP controllers; one can say that open loop runs result lower percentage errors than the results of closed loop runs. This difference is expected; since in open loop runs, monopod is assumed to reach previous desired position. In detail, controller tries to bring the monopod to next desired position from previous desired position regardless of previously achieved position. On the contrary; for closed loop runs, controller tries to bring the monopod to next desired position from previous achieved position. So when there is an error between the previous desired and achieved positions, further achieved positions are affected from this error. This effect can be observed better by comparing the trajectory results and single stride results. By revisiting controller verification results in subsection 3.5.2 and 4.5.2, it is clear that controllers perform better on single stride inverse kinematic map predictions rather than a continuous run.

By comparing SLIP and TD-SLIP controllers, it can be observed that TD-SLIP controllers suffer from an overshoot in general. Since TD-SLIP controllers also regulates the mechanical energy of the monopod; when there is an error between the previous desired and achieved positions, overshooting becomes more critical.

5.5 Estimator Based Controller Design for SLIP and TD-SLIP Model

Another way to design the controller is to use trained estimators. Estimators generates the final apex position when the current apex position is given with control inputs. To build a controller based on an estimator, estimator can be tested with different control inputs. Among the tested control inputs, an optimal control input value can be selected by comparing generated and desired final apex positions. This problem can be formalised in the following way:

$$
\begin{aligned}
\text{minimize} \quad & \tfrac{1}{M}\sum_{m=1}^{M}\left|\tfrac{X_f^m - X_f^{m\,target}}{X_f^{m\,target}}\right| \\
\text{subject to} \quad & {X_f}^{target} : [{y_f}^{target}\ \dot{y}_f^{target}\ {z_f}^{target}] \\
& X_f : [y_f\ \dot{y}_f\ z_f] = E_{NN}^{a2a}(I_{a2a-Estimator}) \\
& I_{a2a-Estimator} = [X_i\ u^*] \\
& X_i = [\dot{y}_i\ z_i] \\
& u^* = \begin{cases} [\theta_{td}^*], & \text{if model is SLIP} \\ [\theta_{td}^*\ \tau_0^*], & \text{if model is TD-SLIP} \end{cases} \\
& \theta_{td}^{min} \leq \theta_{td}^* \leq \theta_{td}^{max} \\
& \tau_0^{min} \leq \tau_0^* \leq \tau_0^{max}
\end{aligned}
$$

where, ${X_f}^{target}$ and X_f represent the target and resulting final apex states from X_i the initial apex state and u^* is the candidate control inputs. $Estimator_{a2a}$ represents the trained estimator and $I_{a2a-Estimator}$ is the inputs for the trained estimator.

This optimization problem can be solved by using neural network based estimator with including the candidate control inputs selected from a determined range. Estimator based controller works by finding the optimal control inputs using this optimization. Same methodology can also be used to develop a controller based on touchdown to liftoff estimator, instead of apex to apex estimator.

Block diagram demonstrating the process of using a SLIP a2a-Estimator to predict Apex to Apex Inverse Kinematic Map, is shown in Figure 5.6.

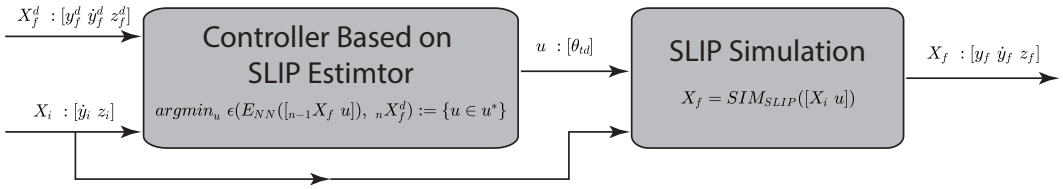
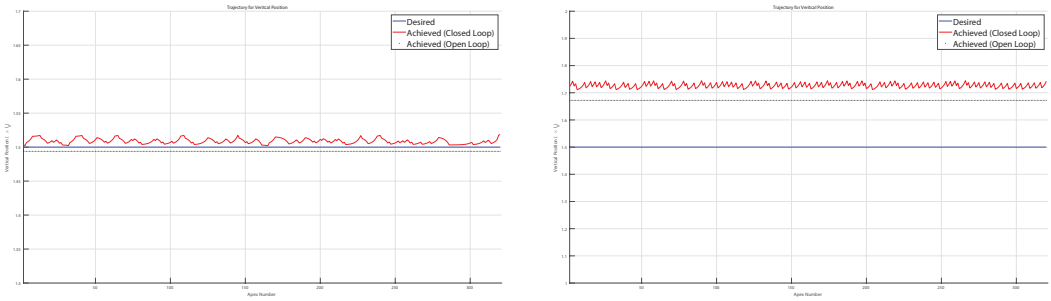


Figure 5.6: Block Diagram Demonstrating the Process of Using a SLIP a2a-Estimator to Predict Apex to Apex Inverse Kinematic Map with Open Loop Run

In Figure 5.6, a trained SLIP Estimator, E_{NN}, is used to predict the control inputs, u, that bring the monopod from the previous final apex state, $_{n-1}X_f$, to the next desired final apex state, $_nX_f^d$. E_{NN} is used to find u, by trying candidate control inputs, u^* with $_{n-1}X_f$. u has the optimal values among u^*, that minimize the mean absolute error of next predicted final apex state respect to $_nX_f^d$. Than u and $_{n-1}X_f$ are used to compute the next achieved final apex state, $_nX_f$. Switch between SLIP and TD-SLIP models, is done by changing the simulation and estimator types. Switch between Open Loop and Close Loop runs, is done by configuring the run process by applying the difference demonstrated in Figure 5.1. Instead of using a2a-Estimator, a t2l-Estimator can also be used in the same process. Using t2l-Estimator to predict Apex to Apex Forward Kinematic Map, is demonstrated in Figure 3.1.

Results of estimator based controller for keeping the final apex vertical position at a constant value is, shown in Figure 5.7 for SLIP and TD-SLIP models. Maximum and average percentage errors for the achieved trajectories are also reported in Table 5.3;

Comparison of Estimator Based Controllers forms a parallel result with the comparison of SLIP and TD-SLIP Controllers, discussed in section 5.4. Estimator Based Controllers are also more successful in open loop runs. In addition to lower



(a) SLIP (b) TD-SLIP

Figure 5.7: Desired and Achieved Final Apex Vertical Position Trajectories with Controller Based on SLIP and TD-SLIP Estimators for Open Loop and Closed Loop Runs, when the Desired Trajectory is Constant

Table 5.3: Errors for Achieved Final Apex Vertical Position Trajectories with Controller Based on SLIP and TD-SLIP Estimators for Open Loop and Closed Loop Runs, when the Desired Trajectory is Constant

	Run Tpye:		Open Loop	Closed Loop
Model:	SLIP	e_{max}	0.41 %	1.26 %
		e_{avg}	0.41 %	0.65 %
	TD-SLIP	e_{max}	11.48 %	16.29 %
		e_{avg}	11.48 %	11.51 %

percentage errors, open loop runs also result better forms for the desired trajectory. Oscillations observed in a closed loop runs, as a result of using optimization for generating control inputs. This oscillations is a result of selecting neighboring values from control input ranges. Selecting a neighboring control input results ripples in the final apex vertical position.

Comparing Estimator Based Controllers with SLIP and TD-SLIP Controllers, one can say that using Estimator Based Controllers decreases percentage errors especially for TD-SLIP model but achieved trajectories are less successful in wave forms. Different controller types can also be designed by using neural network based Estimators and Controllers for SLIP and TD-SLIP models. Further examination of controller design is kept out of the scope of this thesis.

Chapter 6

Conclusion, Discussion and Future Work

In this thesis we study on neural network based estimators and controllers for SLIP and TD-SLIP monopods. Prior to designing and training the neural networks, current solutions for SLIP and TD-SLIP models are investigated. After observing that virtually all of the solutions for SLIP and TD-SLIP models are based on analytical approximations, we proposed that neural network based solutions could be a successful alternative to the existing solutions. We designed and tested our neural networks with the aim of bringing the existing solutions to a higher standard.

To design and test neural networks for more realistic input-output relationships, flight and stance dynamics of SLIP and TD-SLIP models are revisited. By understanding the theory of SLIP and TD-SLIP simulations, required apex return maps are generated. From the generated apex return maps, inputs and outputs for the neural networks, i.e., datasets for training SLIP and TD-SLIP estimators and controllers, are constructed.

By using SLIP and TD-SLIP datasets, neural networks are designed and tested with different configurations. Several error measures are reported to evaluate

the designed estimators and controllers and to compare different designs within themselves as well as with existing solutions in the literature. Trained controllers are used to run the monopod in the simulation environment including different scenarios. These simulation runs demonstrate a better support on the potential of neural network based controllers for SLIP and TD-SLIP models.

We conclude by our extensive experiments that neural network based estimators are promising alternatives to the existing solutions. Instead of solving the estimation problem for each new system separately, neural networks are capable of creating forward kinematic maps at once. We can also observe that neural network based controllers for SLIP model, can be successful counterparts for the existing controllers. Furthermore, creating of neural network controllers in this case is straightforward by only changing the input-output configuration. This method of designing controllers can also be further tested and utilized for mobile robots with similar or different dynamics and morphologies. Meanwhile, experiments with neural network based controllers for TD-SLIP model show promising results, yet more improvements should be made before neural network based controllers for TD-SLIP model become successful counterparts of the existing solutions.

A promising improvement could be changing neural network architectures to more complex and suitable designs. For instance, using a recurrent neural network instead may lead to more successful simulation runs consisting of multiple strides. Another potential improvement is using the trained estimators to generate the optimal control inputs for reaching a desired apex state, as discussed in chapter 5. Last but not least, hybrid multi-modal models including both analytical and data driven solutions can be used to design more advanced estimators and controllers. Possible details of the above proposed improvements and new implementations are given below:

1. **Using Estimators to Find Optimal Control Inputs:** As some details are given in chapter 5, Estimators can be used to find the control inputs that minimize the error between achieved and desired final apex states. An improvement can be made by changing the optimization algorithm or completely changing the training process of Controllers. In this study Controllers are trained to predict the control inputs that were used in a specific apex return map but same final apex state can be reached from the same initial apex state with different control inputs. So, instead of predicting the exact control inputs in an apex return amp, Controller can be trained by changing the loss function on the output layer. The new loss function would be defined as the error between the desired final apex state and achieved final apex state that is predicted with an Estimator. In all these processes both a2a and t2l Estimators can be used. Using a t2l Estimator to predict the full apex return map is shown in subsection 3.1.1.2.

2. **Error Correction for Closed Loop Runs:** A simple error correction can be used in the closed loop run process by adding a PID block after the controller block. The PID block would be used to correct the predicted control inputs by using the error between previous desired and achieved final apex states. With an error correction block overshoot observed in section 5.3 can be eliminated. Instead of a PID block different controller blocks can also be utilized.

3. **Recurrent Neural Networks (RNNs):** In this study, our aim was to train Estimators and Controllers for a single stride. On the contrary, the monopod robots are used in runs consisting of consecutive strides. As an alternative to our approach, Estimators and Controllers can be trained by utilizing a complete run by including several strides as inputs and outputs of a recurrent neural network. Gated RNNs or Long Short-Term Memory (LSTM) Networks can be designed as well as RNNs.

4. **t2l-Controllers with Motion Planning:** As mentioned in subsection 3.1.1.2, there is no inverse function for f_{ascent}; since calculating a single liftoff state from the final apex state is not possible. There are probable liftoff states reaching the same final apex state. This probability can be

used as an advantage to design t2l-Controllers. Since the touchdown state is single, a reachable liftoff state region will be selected among the probable liftoff states. Then a t2l-Controller can be trained by using the touchdown state and the reachable liftoff states. Since this training process results several control inputs to bring the monopod from the touchdown state to the liftoff state, a motion planning approach could be utilized to reach the desired final apex state.

Bibliography

[1] R. Alexander *et al.*, "Three uses for springs in legged locomotion," *International Journal of Robotics Research*, vol. 9, no. 2, pp. 53–61, 1990.

[2] www.flaticon.com, *Running Human Icon*, 2020 (accessed December 6, 2020).

[3] U. Saranlı, Ö. Arslan, M. M. Ankaralı, and Ö. Morgül, "Approximate analytic solutions to non-symmetric stance trajectories of the passive spring-loaded inverted pendulum with damping," *Nonlinear Dynamics*, vol. 62, no. 4, pp. 729–742, 2010.

[4] İ. Uyanık, Ö. Morgül, and U. Saranli, "Experimental validation of a feedforward predictor for the spring-loaded inverted pendulum template," *IEEE Transactions on robotics*, vol. 31, no. 1, pp. 208–216, 2015.

[5] 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.

[6] N. J. Nilsson, C. A. Rosen, and B. Raphael, *Application of intelligent automata to reconnaissance*. No. 5953, Rome Air Development Center, 1969.

[7] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[8] M. F. Silva and J. Tenreiro Machado, "A historical perspective of legged robots," *Journal of Vibration and Control*, vol. 13, no. 9-10, pp. 1447–1486, 2007.

[9] V. Fort Eustis, "Us army transportation combat developments agency,"," *Logistical Vehicle Off-Road Mobility," Project TCCO*, pp. 62–5, 1967.

[10] B. Kennedy, A. Okon, H. Aghazarian, M. Garrett, T. Huntsberger, L. Magnone, M. Robinson, and J. Townsend, "The lemur ii-class robots for inspection and maintenance of orbital structures: A system description," in *Climbing and Walking Robots*, pp. 1069–1076, Springer, 2006.

[11] T. Kubota and H. Takahashi, "Micro walking robot design for planetary exploration," in *Proceedings CLAWAR'2003–6th International Conference on Climbing and Walking Robots*, pp. 357–364, 2003.

[12] P. Fiorini, "Ground mobility systems for planetary exploration," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 908–913, IEEE, 2000.

[13] D. Wettergreen, C. Thorpe, and R. Whittaker, "Exploring mount erebus by walking robot," *Robotics and Autonomous Systems*, vol. 11, no. 3-4, pp. 171–185, 1993.

[14] J. Ayers and J. Witting, "Biomimetic approaches to the control of underwater walking machines," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 365, no. 1850, pp. 273–295, 2007.

[15] M. Konaka, "National project on advanced robot technology in japan," in *Fifth International Conference on Advanced Robotics' Robots in Unstructured Environments*, pp. 24–31, IEEE, 1991.

[16] E. García Armada, J. Estremera, and P. González de Santos, "A control architecture for humanitarian-demining legged robots," in *Proceedings CLAWAR'2003–6th International Conference on Climbing and Walking Robots*, pp. 383–390, 2003.

[17] F. Kikuchi, Y. Ota, and S. Hirose, "Basic performance experiments for jumping quadruped," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 4, pp. 3378–3383, IEEE, 2003.

[18] K. Matsuoka, "A model of repetitive hopping movements in man," in *Proceedings of 5 th World Congress of Theory of Machines and Mechanisms*, vol. 2, pp. 1168–1171, 1979.

[19] W. J. Schwind and D. E. Koditschek, "Characterization of monoped equilibrium gaits," in *Proceedings of International Conference on Robotics and Automation*, vol. 3, pp. 1986–1992, IEEE, 1997.

[20] M. Ahmadi and M. Buehler, "Controlled passive dynamic running experiments with the arl-monopod ii," *IEEE Transactions on Robotics*, vol. 22, no. 5, pp. 974–986, 2006.

[21] I. Uyanık, *Adaptive control of a one-legged hopping robot through dynamically embedded spring-loaded inverted pendulum template*. PhD thesis, bilkent university, 2011.

[22] H. E. Orhon, "Model-based identification and control of a one-legged hopping robot," *arXiv preprint arXiv:1802.09634*, 2018.

[23] T. Nagasaki, S. Kajita, K. Yokoi, K. Kaneko, and K. Tanie, "Running pattern generation and its evaluation using a realistic humanoid model," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 1, pp. 1336–1342, IEEE, 2003.

[24] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, "The intelligent asimo: System overview and integration," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 3, pp. 2478–2483, IEEE, 2002.

[25] U. Saranli, 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.

[26] M. M. Ankarali, N. J. Cowan, and U. Saranli, "Td-slip: A better predictive model for human running," *Proceedings of Dynamic Walking*, 2012.

[27] M. Ankarali, O. Arslan, and U. Saranli, "An analytical solution to the stance dynamics of passive spring-loaded inverted pendulum with damping," in *Mobile Robotics: Solutions and Challenges*, pp. 693–700, World Scientific, 2010.

[28] M. Shahbazi, U. Saranlı, R. Babuška, and G. A. Lopes, "Approximate analytical solutions to the double-stance dynamics of the lossy spring-loaded inverted pendulum," *Bioinspiration & Biomimetics*, vol. 12, no. 1, p. 016003, 2016.

[29] I. Uyanık, M. M. Ankaralı, N. J. Cowan, Ö. Morgül, and U. Saranlı, "Identification of a hybrid spring mass damper via harmonic transfer functions as a step towards data-driven models for legged locomotion," *arXiv preprint arXiv:1501.05628*, 2015.

[30] 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.

[31] I. Uyanik, "Identification of legged locomotion via model-based and data-driven approaches," *arXiv preprint arXiv:1710.04275*, 2017.

[32] 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.

[33] H. Geyer, U. Saranli, A. Goswami, and P. Vadakkepat, "Gait based on the spring-loaded inverted pendulum," *Humanoid Robotics: A Reference, A. Goswami and P. Vadakkepat, Eds. Springer Netherlands*, pp. 1–25, 2018.

[34] H. Hamzaçebi and Ö. Morgül, "On the periodic gait stability of a multi-actuated spring-mass hopper model via partial feedback linearization," *Nonlinear Dynamics*, vol. 88, no. 2, pp. 1237–1256, 2017.

[35] Ö. Arslan, U. Saranli, and Ö. Morgül, "Reactive footstep planning for a planar spring mass hopper," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 160–166, IEEE, 2009.

[36] H. E. Orhon, C. Odabaş, I. Uyanık, Ö. Morgül, and U. Saranlı, "Extending the lossy spring-loaded inverted pendulum model with a slider-crank mechanism," in *2015 International Conference on Advanced Robotics (ICAR)*, pp. 99–104, IEEE, 2015.

[37] J. Velagic, N. Osmic, and B. Lacevic, "Neural network controller for mobile robot motion control," *World Academy of Science, Engineering and Technology*, vol. 47, pp. 193–198, 2008.

[38] Y. H. Kim and F. L. Lewis, "Optimal design of cmac neural-network controller for robot manipulators," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 1, pp. 22–31, 2000.

[39] M. K. Singh and D. R. Parhi, "Path optimisation of a mobile robot using an artificial neural network controller," *International Journal of Systems Science*, vol. 42, no. 1, pp. 107–120, 2011.

[40] H. J. Chiel, R. D. Beer, R. D. Quinn, and K. S. Espenschied, "Robustness of a distributed neural network controller for locomotion in a hexapod robot," *IEEE Transactions on robotics and automation*, vol. 8, no. 3, pp. 293–303, 1992.

[41] S. Xin, B. Delhaisse, Y. You, C. Zhou, M. Shahbazi, and N. Tsagarakis, "Neural-network-controlled spring mass template for humanoid running," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1725–1731, IEEE, 2018.

[42] P. Holmes, "Poincaré, celestial mechanics, dynamical-systems theory and "chaos"," *Physics Reports*, vol. 193, no. 3, pp. 137–163, 1990.

[43] H. Geyer, A. Seyfarth, and R. Blickhan, "Spring-mass running: simple approximate solution and application to gait stability," *Journal of theoretical biology*, vol. 232, no. 3, pp. 315–328, 2005.

[44] MATLAB, *MATLAB: Statistics and Machine Learning Toolbox*. Natick, Massachusetts: The MathWorks Inc., 2020.

[45] MATLAB, *MATLAB: Deep Learning Toolbox*. Natick, Massachusetts: The MathWorks Inc., 2020.