6-27-2022

Predicting Joint Mechanics using sEMG and Deep Neural Networks

Heath James Staley Boyea
Rose-Hulman Institute of Technology

Predicting Joint Mechanics using sEMG and Deep Neural Networks

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

Heath James Staley Boyea

In Partial Fulfillment of the requirements for the degree

of

Master of Science in Mechanical Engineering

June 2022

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Final Examination Report

Heath Boyea — Name

Mechanical Engineering — Graduate Major

Thesis Title: Predicting Joint Mechanics Using sEMG and Deep Neural Networks

DATE OF EXAM: June 27, 2022

EXAMINATION COMMITTEE:

	Thesis Advisory Committee	Department
Thesis Advisor:	Joel Canino	ME
	Cheuk Ming Lui	ME
	Rebecca Bercich	ME
	Alan Chiu	BBE

PASSED X **FAILED** _____

ABSTRACT

Boyea, Heath James Staley

M.S.M.E.

Rose-Hulman Institute of Technology

June 2022

Predicting Joint Mechanics using sEMG and Deep Neural Networks

Thesis Advisor: Dr. J. Miles Canino

According to the Center for Disease Control (CDC), 1 in 7 American adults are affected by disabilities that affect mobility [1]. Assistive devices, such as exoskeletons, may be able to assist affected individuals. Though these devices may be helpful, constant assistance can create dependence on the device. We hypothesize that an assist-as-needed control system can be developed to aid the wearer only when necessary. To develop an assist-as-needed control system, it needs to be determined when assistance is needed and the magnitude of this assistance. We hypothesize that an artificial neural network (ANN) can predict future normal torque outputs of joints given past joint angles, torques, and sEMG data of the governing muscles. Actual torque outputs can then be extrapolated from sensor data and the difference in these torque values can be determined to be the required assistance magnitude. To evaluate this method, a dataset of 10 subjects walking at 7 speeds was ascertained [3]. This dataset included pelvis, hip, knee, and ankle torque and angle about the 3 major axes across 3 strides at each walking speed. The surface electromyography data (sEMG) of the Tibialis Anterior (TA), Gastrocnemius Lateralis (GAL),

Biceps Femoris (BF) and Vastus Lateralis (VL) were also collected for each subject during each stride. The first step in this investigation was to evaluate this method on a single and multi-joint system while considering the sEMG activity of the pertinent muscles. A multi-joint system could then be evaluated when not all governing sEMG data can be measured. Finally, a principal component analysis could be performed on this multi-joint system to evaluate the contribution of each feature in the training outcomes of the ANNs. This allowed the researcher to reduce the order of this system while still maintaining prediction accuracy. Features that contribute the most to prediction accuracy could be retained and the others could be removed to simplify the network, reducing training time, testing time, and hardware complexity. These results of this research will be used to inform future development of assist-as-needed devices.

ACKNOWLEDGEMENTS

I would like to thank my committee members, Dr. Lui, Dr. Bercich, and Dr. Chiu, who contributed their diverse and valuable knowledge to help make this interdisciplinary research possible.

I would also like to thank Dr. Canino, my advisor, for his role in advising, inspiring, and guiding, me on my journey through academia. Since advising me through my undergraduate degree, he has been the largest contributor to my pursuit of higher learning. Without his influence, I would not be on my current path.

Finally, I would like to thank my mother for always encouraging me to follow my dreams and do what I feel is right, not what is easy. From a young age, she embraced and encouraged my curiosity, creativity, and interest in everything mechanical. From lawn mowers to decommissioned fighter jets, she showed me the systems that would continue to drive my engineering interest throughout the rest of my life. Her guidance has helped forge my personal ethos and has made me the man I am today.

TABLE OF CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Center for Disease Control-CDC

Parkinson's Disease-PD

Cerebral Palsy-CP

Amyotrophic Lateral Sclerosis-ALS

Essential Tremors-ET

Huntington's Disease-HD

Muscular Dystrophy-MD

Electromyography-EMG

Surface Electromyography-sEMG

artificial neural network- ANN

Nonlinear Input-Output Time Series Neural Network -NIOTSNN

Root mean square error- RMSE

Analysis of variance- ANOVA

Tibialis Anterior- TA

Gastrocnemius Lateralis-GAL

Biceps Femoris-BF

Vastus Lateralis-VL

Ground Reaction Forces-GRF

Center of Pressure-CoP

Force Platforms Moments-FPMs

Principal Component Analysis-PCA

Wavelet Neural Network-WNN

1. INTRODUCTION

According to the Center for Disease Control (CDC), 1 in 7 American adults are affected by disabilities that affect mobility [1]. Some of these conditions deal with mechanical limitations such as Arthritis, tendon damage, and bone fractures while others, such as Parkinson's Disease (PD), Cerebral Palsy (CP), Amyotrophic Lateral Sclerosis (ALS), Essential Tremors (ET), Huntington's Disease (HD), and Muscular Dystrophy (MD) deals with an uncontrolled, involuntary movement, or Dyskinesia. In recent history, the field of Rehabilitation Robotics has sought to develop devices to rehabilitate or permanently assist individuals afflicted by these conditions. Of these technologies, exoskeletons are among the most prolific.

Assistive exoskeletons often come as one of two types, hard or soft. Hard exoskeletons are rigid devices that provide both structure and actuation for the user [4]. These devices have been shown to be useful in hospitals and rehabilitation clinics. These devices can help facilitate specific rehabilitation routines that have been shown to produce favorable rehabilitation outcomes [6]. These systems can also be useful in cases where the user cannot provide any, very little, or voluntary muscular input, such as paralysis from spinal cord injury or stroke and MD [5]. They are often heavy and lack biomechanical compliance, though in some cases this can be favorable. This increases the inertia of the limb making the devices often cumbersome. This trend can make hard exoskeletons unrealistic for everyday use for many individuals who maintain limited, voluntary muscle control.

Soft exoskeletons, on the other hand, are compliant devices that provide little structural support while providing assistive actuation to the user [4]. Like their hard counterparts, soft exoskeletons can assist in performing desired movements for rehabilitation purposes or daily tasks. Since soft exoskeletons do not require a rigid structure, they can be comprised of flexible

materials such as textiles and flexible polymers. Actuators can also be moved off from the device to a remote location such as a backpack. Actuation can be performed through remote lines such as Bowden cables [7] or pneumatic actuators [12]. This results in soft exoskeletons adding significantly less inertia to the affected limb and maintaining significantly higher biomechanical compliance. These two factors result in soft exoskeletons allowing a much more natural range of motion and movement dynamics.

Apart from the design of exoskeleton hardware, Rehabilitation Robotics is highly focused on the control systems of these exoskeleton systems. The control paradigm for these systems depends wholly on their intended use. Many assistive devices require the ability to anticipate a user's intended movement. This results in a large sector of Rehabilitation Robotics research being centered around intent recognition. Much of this is necessary for paralyzed individuals or those missing limbs entirely. On the other hand, A user's intent may or may not need to be taken into consideration. For example, if a user can move voluntarily but with a limited capacity their movement may only need to be identified for assistance to be provided. For many users that do not suffer from Paralysis or Dyskinesia, this is the case.

Assist-as-needed control paradigms can be preferential over constant assistance in some applications. This control paradigm allows for a system to assist only when and at the amplitude necessary for the user to complete the intended action. This encourages rehabilitation, discourages dependence on the device, and reduces power consumption and required force output. There is a multitude of philosophies used while attempting to tackle this problem. The leaps in machine learning over recent years have given scientist and engineers the ability to create control systems for assistive devices that can learn and reproduce or assist user's natural

movements [13]. These systems tend to be complex and current methods don't seem to produce a distinct solution.

We hypothesize that artificial neural networks can be used to predict a user's intended torque output about a joint, their actual torque output could then be measured. The difference between these two values can then be calculated and applied as the required assistance. This method will allow the system to be simple as it can be trained on healthy joint movement and the governing Electromyography (EMG) signals from the surrounding muscles. The system can then predict future joint mechanics based on this training data. This type of system does not require a complex model and is capable of scaling to multiple joints. This torque prediction method can then become the basis for a unique assist-as-needed control system in an assistive exoskeleton.

.

2. BACKGROUND

It has been shown that future joint angles can be predicted by observing current joint angles, torques, and sEMG data of pertinent muscles through the use of a Nonlinear Input-Output Time Series Neural Network (NIOTSNN). Coker et al. sought to assess the performance of a supervised learning artificial neural network (ANN) algorithm trained with both knee flexion angle and knee muscle EMG signals during walking to predict knee flexion angle during walking at various amounts of time into the future. They hypothesized that as the prediction horizon increased, prediction accuracy would decrease and as the number of algorithm training trials increased, the prediction accuracy would also increase [2].

Ten subjects were recruited, comprised of five males and five females with an average age of 21.5 +/- 2 years, weight of 64.5 +/- 9.8 kg, and height of 166.9 +/- 14.5 cm. These subjects reported no history of chronic pain in the back or lower extremities in the previous six months before the study. Twelve surface Electromyography(sEMG) electrodes were placed on each subject's left and right tensor fasciae latae, rectus femoris, vastus medialis, vastus lateralis, bicep femoris, and semitendinosus (Figure 2.1). Raw sEMG data was collected at 1111Hz and passed through a Butterworth filter with a band from 20Hz to 500Hz to remove motion artifacts and high-frequency aliasing. The envelope of each sEMG channel was then determined using the Root Mean Square (RMS) value of the signal with a 300 ms movable window as described by Farfan et al. [8]. All readings were then presented as percent maximum voluntary contraction (%MVC). A ten-camera motion capture system was used to track 79 retroreflective markers on each subject (Figure 2.1). Marker position was captured at 120Hz and passed through a 15Hz low-pass Butterworth filter to remove noise. Body segments were created using marker positions and were used to calculate knee flexion.

Figure 2.1: Coker et al. [2] sEMG Sensor and Retroreflective Marker Placement

Each subject performed 15 walking trials over a distance of 20 feet at a self-regulated pace to maintain natural motion. Data from ten trials were used for training the ANN and five trials were kept out to test the algorithm's accuracy. The ANN to be used was MATLABS's NIOTSNN using Bayesian Regularization with a single hidden layer of ten nodes and feedback delay set to two timesteps. Feedback delay refers to the memory provided to each neuron so that it can look back N timesteps. This is synonymous with lagging inputs by N timesteps (Lopes et al. [9]). There were to be seven input features comprised of the six sEMG channels and calculated knee angle of one leg. The output was to be knee angle predicted at 50, 100, 150, and 200 ms in the future. Root mean square error (RMSE) was then calculated between the ANN's

predictions and the actually calculated knee angle at each prediction horizon. Prediction accuracy can be seen in Table 2.1.

Table 2.1: Coker et al. [2] Knee Angle Prediction Error

Prediction Horizon (ms)	Mean RMSE (degrees)
50	0.68
100	2.04
150	3.38
200	4.61

Results from Coker et al. show that joint angle can be accurately predicted by a NIOTSNN to increase metabolic economy while using a lower limb exoskeleton. Using this precedent, we hypothesize this method could be expanded to predict joint torque to inform an assist-as-needed control system. To test this hypothesis, this study must be augmented to include joint torque calculation and prediction, expanding the NIOTSNN's input and output composition. To adequately predict joint mechanics of everyday use, walking speed would also need to be varied as different everyday tasks will necessitate a different walking. Varying walking speed will also be a more rigorous test of the algorithm and should produce a more robust network.

We acquired a suitable dataset from Moriera et al. [3]. This dataset was comprised of lower body sEMG and joint kinematic data of 15 subjects. Eight sEMG sensors were placed on each participant on the left and right Tibialis Anterior (TA), Gastrocnemius Lateralis (GAL),

Biceps Femoris (BF) and Vastus Lateralis (VL) (Figure 2.1). Twenty-four retroreflective sensors were also placed on the lower body to facilitate motion capture (Figure 2.2).

Figure 2.2: Moreira et al. [2] sEMG Sensor Placement

Figure 2.3: Moreira et al.[2] Retroreflective Sensor Placement

sEMG data was captured at 2000 Hz and bandpass filtered by a 4th order, zero-lag, Butterworth Filter with cutoff frequencies of 20Hz and 450Hz. Motion capture was captured at 200 Hz by a twelve-camera motion capture system to calculate Joint angles and torques about the x-, y-, and z-axis. Joint angles were then filtered by a low-pass 4th order, zero-lag, Butterworth Filter with a cutoff frequency of 6 Hz. Six force platforms, placed in the floor captured ground reaction forces (GRF), center of pressure (CoP), and Force Platforms Moments (FPMs) at 200 Hz.

Each subject performed 70 walking trials. Each trial was comprised of three strides, stepping off with the right foot, then continuing on the left foot, then the right foot, at a fixed

pace, guided by a metronome. This course, along with an exaggerated demonstration stride, can be seen in Figure 2.4. The subjects each began performing ten trials at 1.0 km/h and then rested for five minutes. This process was then repeated at 1.5, 2.0, 2.5, 3.0, 3.5, and 4.0 km/h. Heel strike detection was used to determine stride times and split data into gait cycles. Each stride was then normalized to 1001 datapoints.

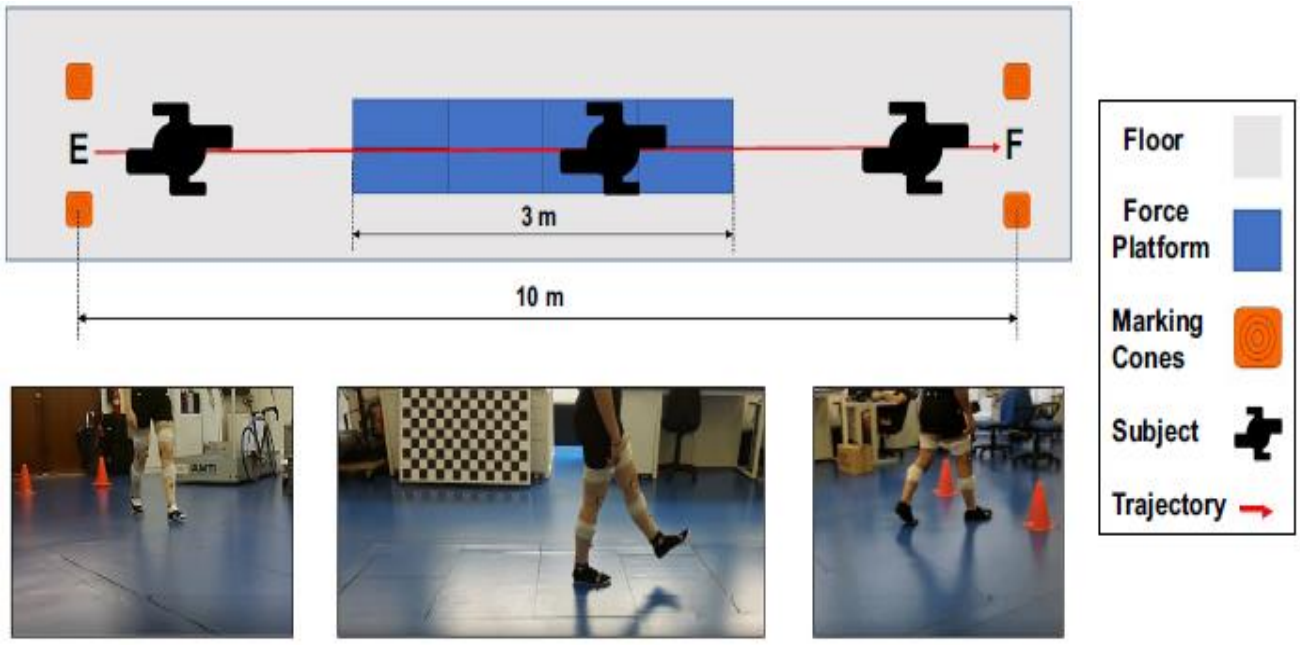

Figure 2.4: Moreira et al. [2] Experimental gait course

This dataset supplies the researcher with 6300 strides (15 subjects x 2 legs/ subject x 7 walking speeds x 10 trials/ walking speed x 3 strides/ trial). Due to data irregularities, not all trials were retained by researchers to be post-processed. This resulted in the researcher retaining ten subjects with complete datasets to comprise the cohort for this analysis. The ten-subject cohort supplied 4200 strides (10 subjects x 2 legs/ subject x 7 walking speeds x 10 trials/ walking speed x 3 strides/ trial) to train and test the NIOTSNN.

3. SINGLE-JOINT SYSTEM

3.1 Introduction

We hypothesized that a NIOTSNN could be used to accurately predict future joint torque at various movement speeds. We began by replicating Coker et al.'s methods with the addition of joint torque prediction, utilizing the data set supplied by Moreira et al. This required the researcher to train a NIOTSNN to predict knee joint angle and torque at varying prediction horizons. We then calculated the error and compared it to the results from Coker et al.

3.2 Methods

Like Coker et al. we utilized MATLABS's NIOTSNN. This Network was comprised of a single hidden layer of ten nodes with a sigmoid activation function in the hidden layer and a linear activation function in the output layer (Figure 3.2.1). Feedback delay was set to two timesteps. According to Surakhi et al. [10], there are many methods to optimize input lag and none are optimal for all prediction models but experimentation is valid for feed-forward neural networks. This is the method we will utilize. We began by initializing input lag to two timesteps as this was the value used by Coker et al. Finally, Bayesian Regularization was used.

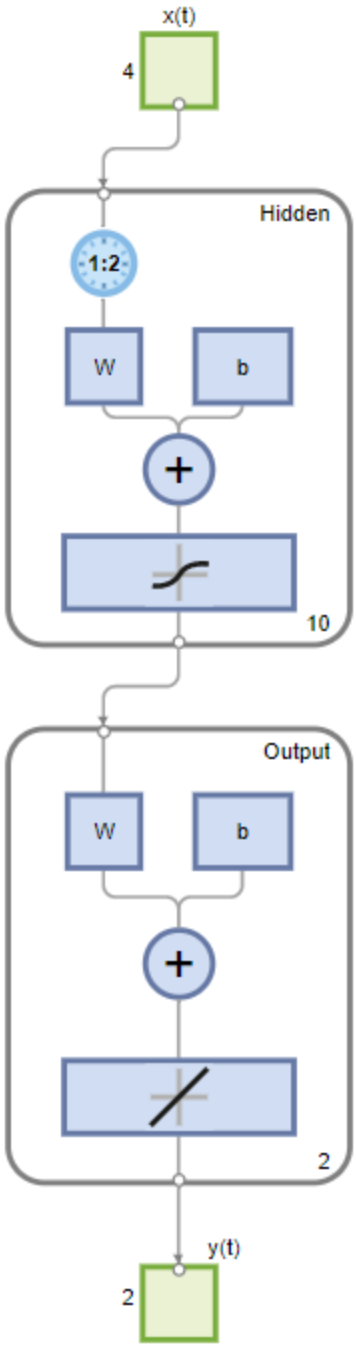

Figure 3.2.1: Knee-Only Nonlinear Input-Output Time Series Neural Network

The network used 70% train, 15% test, and 15% validation data split. When training, MATLAB's fixed stop conditions were used. These stop conditions can be seen in Table 3.2.1.

Table 3.2.1: NIOTSNN Stop Conditions

Unit	Stop Value
Epoch	1000
Elapsed Time	-
Performance	0
Gradient	$7x10^{-7}$
Mu	$1x10^{10}$
Effective # Param	0
Sum Squared Param	0

The input to this network was current joint angle and extrapolated torque about the knee flexion-extension (FE) axis and the corresponding sEMG data of the correlated muscles. For the knee, these muscles include the Bicep Femoris (BF) and the Vastus Lateralis (VL) (Figure 3.2). The VL muscle contributes to knee extension while the BF muscle contributes to knee flexion. Though torque about the knee is caused by multiple muscles, the activities of these muscles should correlate well and sufficiently inform our network.

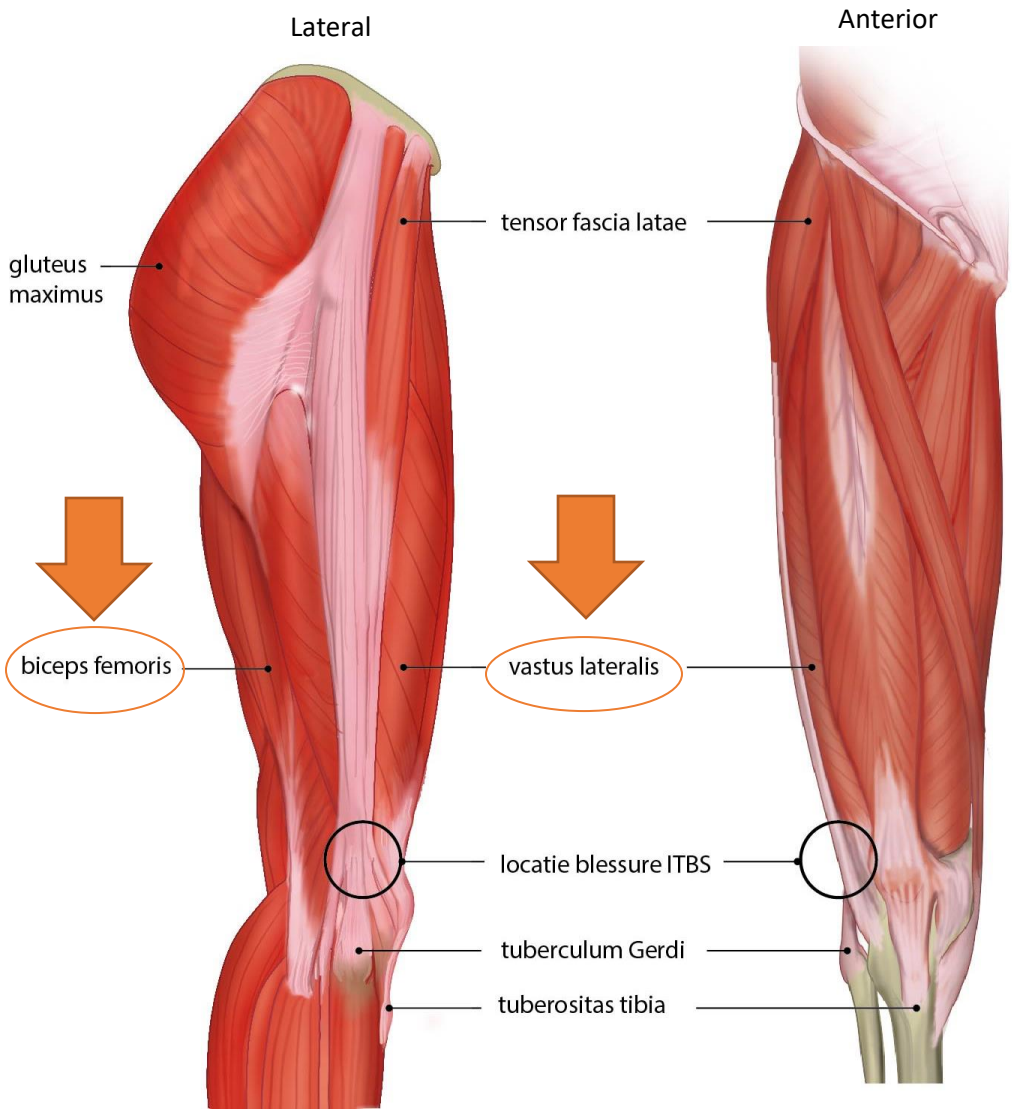

Figure 3.2.2: Bicep Femoris and Vastus Lateralis Location (anatomytool.org)

The target for the networks was time-shifted joint angle and torque data. This allowed the network to correlate current kinematic and EMG data with future joint kinematic data. This allowed the network to accurately predict future joint mechanics once trained. For this analysis,

there were four input features, BF sEMG data, VL sEMG data, knee angle, and knee torque. There were two output features, future knee angle and future knee torque. To time shift this data, our prediction horizon needed to be converted to a number of data points or entry offset. Since each stride was regularized to 1001 datapoints, this was essentially a ratio of our prediction horizon to total stride time multiplied by the total number of points in the stride (Equation 3.2.1). This entry offset could then be removed from the beginning of the data matrix that we wish to shift into the future. To maintain data synchronicity, the same number of points must also be removed from the rear of the current input data. This will produce two matrices of equal length with one phase shifted from the other by the prediction horizon (Figure 3.2.3). Increasing the number of points being removed as the prediction horizon increases will reduce the portion of the stride being considered by the neural network. This will likely contribute to a decrease in prediction accuracy as the prediction horizon increases but is to be expected in a time-delay neural network.

$$entry\ offset = \frac{prediction\ horizon(ms) * 1001(\frac{points}{stride})}{stride\ time(ms)}$$

Equation 3.2.1: Prediction Horizon to Entry Offset Conversion

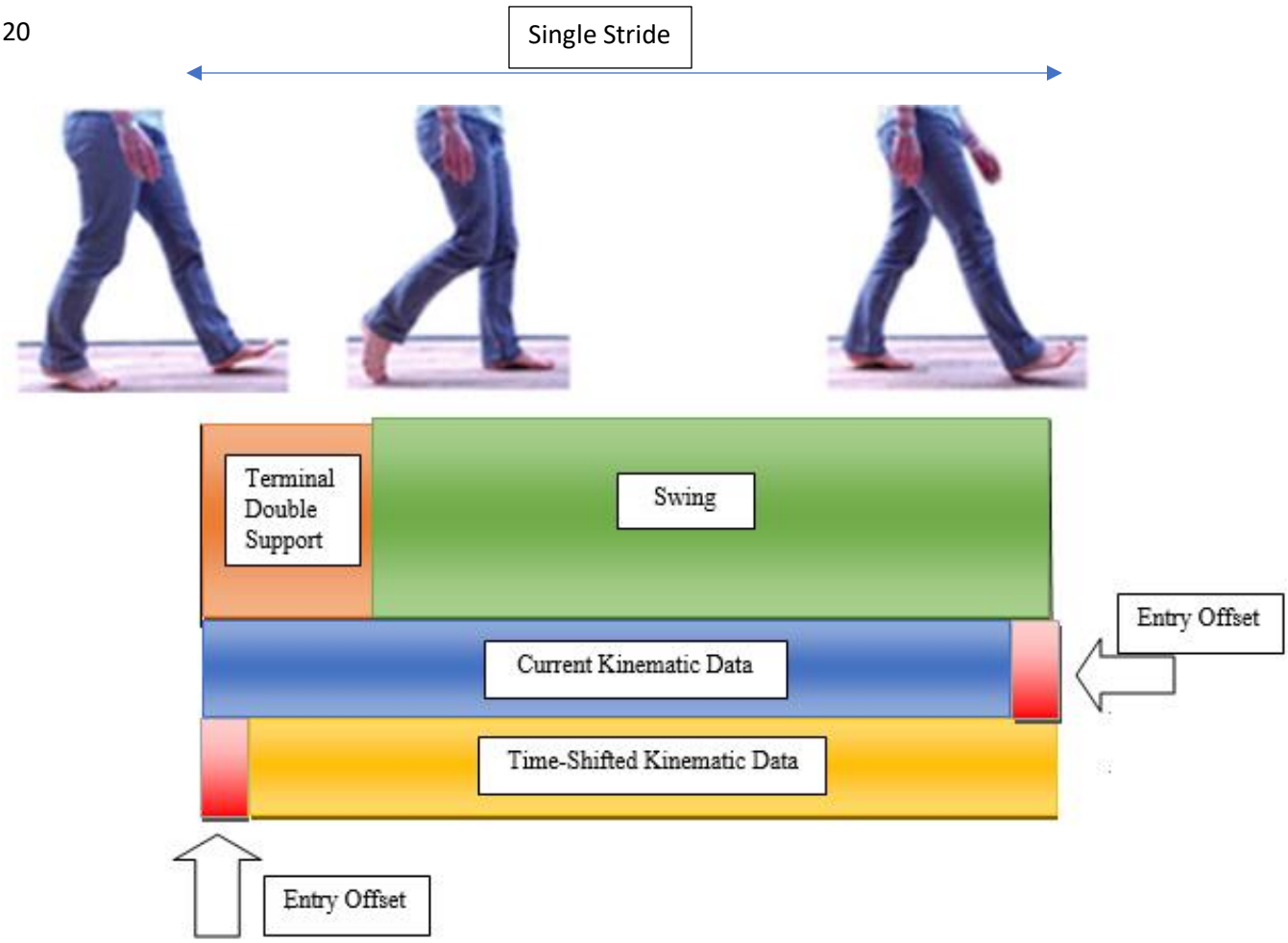

Figure 3.2.3: Time-Shifted Stride Data Illustration

One network was to be trained for each leg of each subject at prediction horizons of 50 ms, 100 ms, and 150 ms, totaling 60 total networks (10 subjects x 2 legs x 3 prediction horizons). This allowed the researcher to compare our results to Coker et al. as they share the same prediction horizons and will also allow the researcher to compare training results between legs and subjects. Since each trial consisted of two right strides and one left stride, we had twice as much data on the subjects' right legs as we did their left.

Each subject performed 10 trials at each speed. Nine trials were used for training while one trial was left out to test each resulting model. Since each speed was covered in each trial, this method evaluated the model's ability to predict joint kinematics at all covered speeds. Since each

trial consisted of two right strides and one left stride, left leg networks were trained on 630 strides (10 subjects x 7 walking speeds/ trial x 9 trials x 1stride/ trial) while right leg networks were trained on 1260 strides (10 subjects x 7 walking speeds/ trial x 9 trials x 2 strides/ trial) at each prediction horizon. To extract this data and prepare it for the network, the knee data extraction code (Appendix 2.1) was written in MATLAB to scour the dataset and extract all knee data for a specified subject and leg at a specified prediction horizon. This script located, collected, time-shifted, concatenate, and returned training input, training target, testing input, and testing target matrices.

A pilot training session was performed on the first network to optimize input delay. Input delay was varied around our initial value of 2. When input delay was decreased, network prediction accuracy was diminished. When input lag was increased, negligible error accuracy change was observed. In addition, the network's input-error cross-correlation graph shows error correlation at all lags that fell within the confidence limit (Figure 3.2.4). This proves that an input lag of 2 timesteps is satisfactory. The network was then trained using the training input and training target matrices until a stop condition was reached as seen in Table 3.2.1. The model was then exported and saved to be evaluated. The resulting trained model was then evaluated using the returned testing data. The testing data was input into the model and a prediction was produced. The RMSE between the prediction and the time-shifted test data was then calculated.

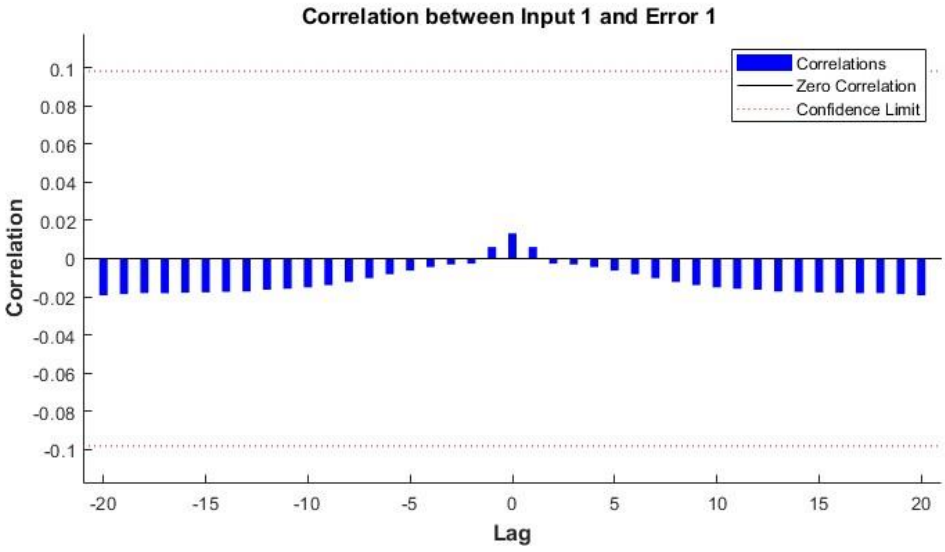

Figure 3.2.4: Input-Error Cross-Correlation with an Input Lag of 2 Timesteps

3.3 Results

All 60 networks were evaluated with the withheld trial data for their corresponding subject and leg. This trial was comprised of 1 stride at each walking speed for left leg networks and 2 strides at each walking speed for right leg networks. The average training time was 3 minutes and 47 seconds. The average prediction time was 72 ms. This means that with current hardware, it would not be possible to compare to actual kinematic output at the 50 ms prediction horizon. With dedicated hardware, this prediction time could possibly be decreased below the 50ms threshold.

The RMSE between ground truth and knee angle (Table 3.3.1), and knee torque (Table 3.3.2**)** was then calculated.

Table 3.3.1: Knee-Only Network Knee Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.5723	1.5558	2.9068	3.6888	4.6483	5.7382
2	1.8917	1.2998	4.5225	3.0163	5.3133	4.2396
3	1.6449	1.6382	2.8223	4.0235	5.3942	5.1816
8	2.5928	1.1799	3.9188	2.9471	4.8075	3.2575
9	3.0557	1.9482	3.7829	2.3473	5.0898	4.2418
10	1.3906	1.2560	2.6572	2.8876	4.3221	4.7159
11	2.1677	1.7392	4.0028	4.5065	5.9282	5.8187
12	2.4537	1.4667	3.9001	2.9656	3.7900	3.9796
13	1.6091	1.6069	3.3417	3.6715	4.9152	5.3089
14	1.9170	2.0128	3.6188	3.8472	7.9161	5.0264
Average	2.0296	1.5703	3.5474	3.3901	5.2125	4.7508

Table 3.3.2: Knee-only Network Knee Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.7792	2.0688	5.3553	4.2574	6.9726	5.4493
2	1.5133	1.6564	3.5815	3.4091	4.6302	4.9098
3	1.5841	1.3654	2.3614	2.7925	3.4637	3.4984
8	2.5430	1.9137	7.4359	3.9798	6.0974	4.7125
9	4.0249	2.0522	4.0976	4.0205	6.2085	6.7611
10	2.5104	2.1561	6.2781	4.3789	7.4020	5.8490
11	1.7385	1.8065	3.4298	3.7127	4.8572	4.8258
12	2.5305	1.8174	4.1997	3.6902	4.9613	5.0910
13	1.7003	1.7429	3.5792	3.3310	4.2254	4.1212
14	2.9095	2.1733	4.1475	3.8146	5.5512	4.6357
Average	2.3834	1.8753	4.4466	3.7387	5.4369	4.9854

When the predictions from the neural network were mapped against actual future data at 50 ms (Figure 3.3.1), 100 ms (Figure 3.3.2), and 150 ms (Figure 3.3.3), it can be seen how well the actual joint mechanics could be reproduced with this method.

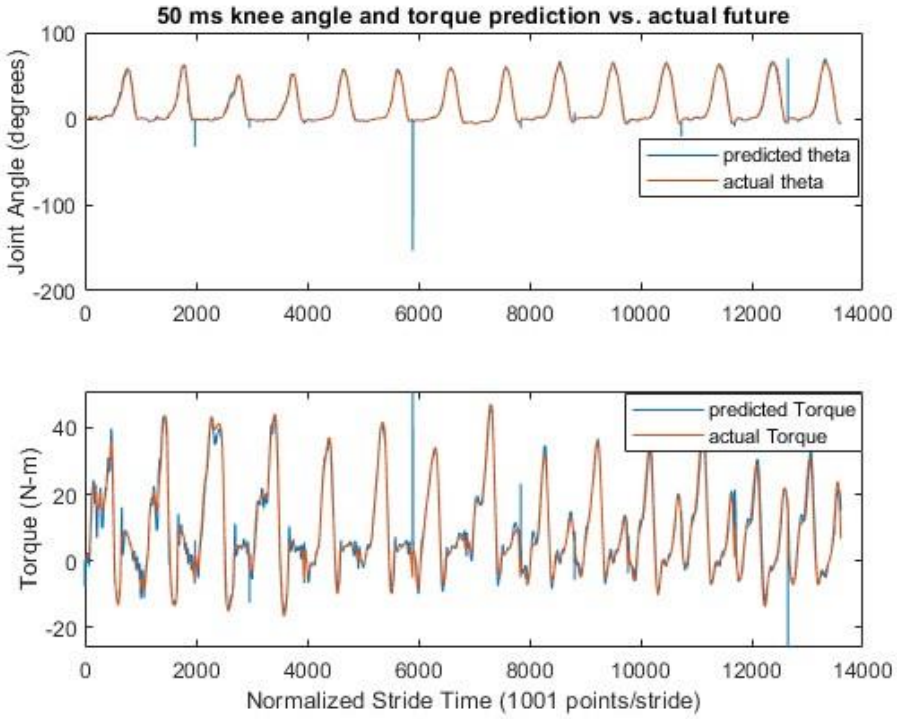

Figure 3.3.1: 50 ms Knee-Only Network Prediction vs. Actual Future Data

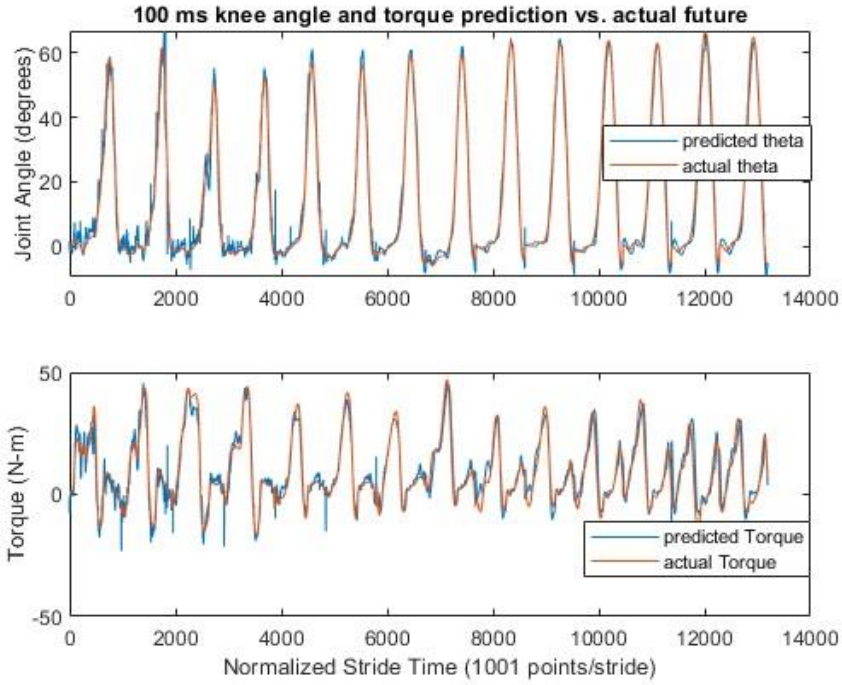

Figure 3.3.2: 100 ms Knee-Only Network Prediction vs. Actual Future Data

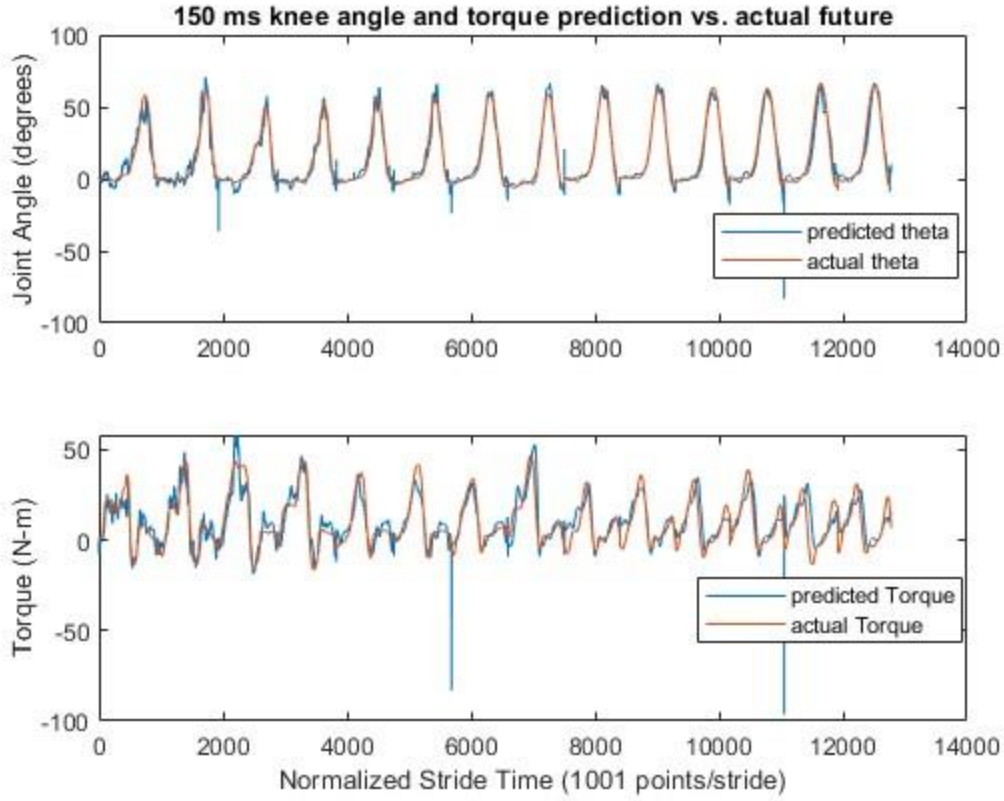

Figure 3.3.3: 150 ms Knee-Only Network Prediction vs. Actual Future Data

Though RMSE is an accurate metric for average error, actual predictions show that error occurs at significantly different rates in certain areas of the waveform over others. To accurately show the areas of higher error, the error regions must be mapped over the true waveform. To accomplish this, testing gait data and network prediction for each subject were separately grand averaged. The average error was then determined by subtracting these two waveforms. The average error was then added to the actual gait data to produce an average error waveform. The difference was then shaded to show regions where the error is more likely. 50 ms, 100 ms, and 150 ms error region graphs can be seen in Figure 3.3.4, Figure 3.3.5, and Figure 3.3.6, respectively.

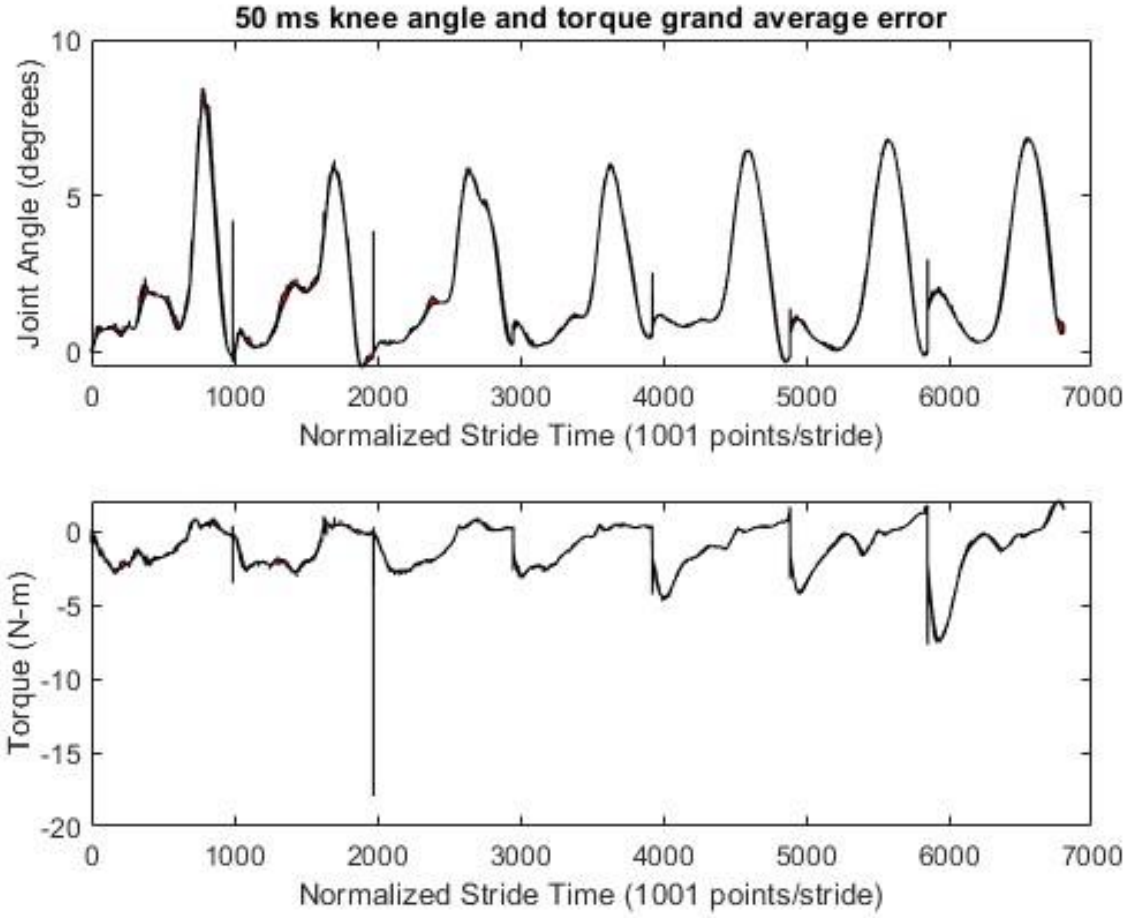

Figure 3.3.4: 50 ms Knee-Only Network Grand Average Prediction Error

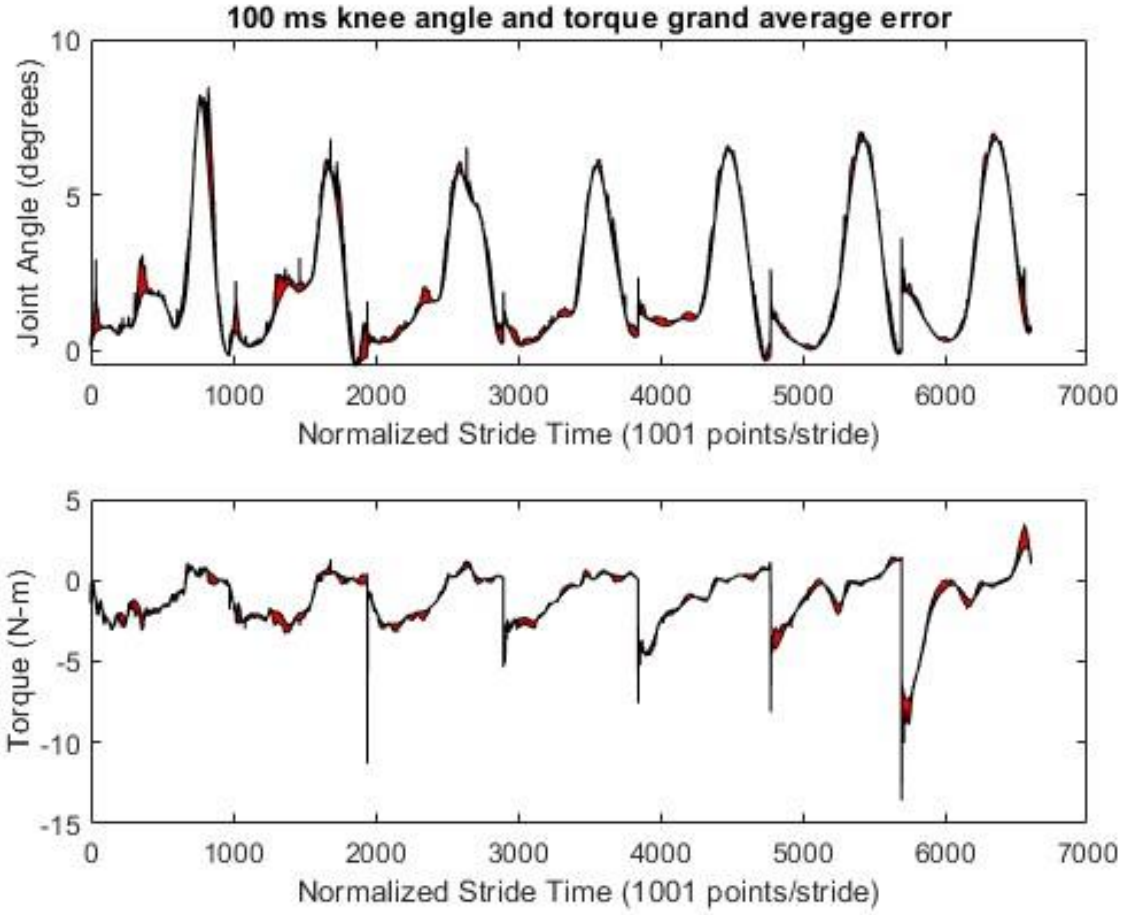

Figure 3.3.5: 100 ms Knee-Only Network Grand Average Prediction Error

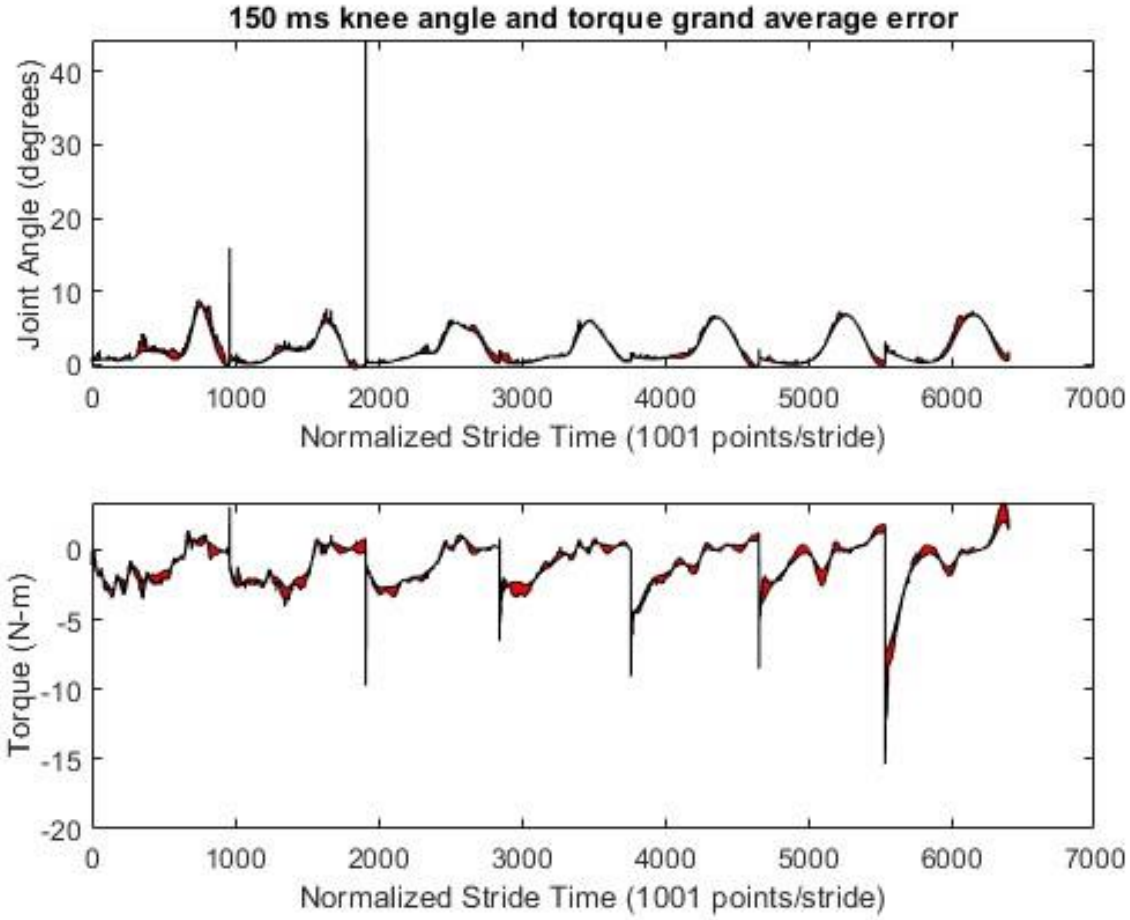

Figure 3.3.6: 150 ms Knee-Only Network Grand Average Prediction Error

3.4 Discussion

When comparing this analysis to that done by Coker et. Al., most parameters were held constant other than the addition of variable walking speed and torque prediction. When comparing the resulting error, it can be seen in Table 3.4.1 that the addition of variable walking speed significantly increases the error. We can also observe numerous error spikes. These could be removed with low pass filtering. Unfortunately, since each stride is regularized to 1001 points/ stride and each stride varies in duration, a constant sampling frequency cannot be established to inform a digital filter, making this method impossible. Though these error spikes have a large magnitude, they have a short duration and should therefore have little effect on performance.

Prediction horizon(ms)	Prediction RMSE (Degrees)	Coker et al. RMSE (Degrees)	% Difference
50	1.80	0.68	62.22
100	3.47	2.04	41.19
150	4.98	3.38	32.15

Table 3.4.1: Knee Angle Prediction RMSE vs Coker et al. Knee Angle Prediction RMSE

Though our prediction error is significantly higher than those from Coker et al., they are in line with findings from other similar studies. Ma et al. [10] utilized a long short-term memory (LSTM) neural network with a time advance feature to predict knee angles from sEMG of governing muscles at the same timestep of individuals walking at a fixed pace. Ma et al. were able to achieve a prediction error RMSE of 2.6164 degrees to 4.1744 degrees. This is consistent

with our findings though their prediction was wholly based on sEMG and predicting current mechanics rather than future mechanics.

Ardestani et al. [11] compared the effectiveness of a wavelet neural network (WNN) to a more traditional three-layer feed-forward artificial neural network (FFANN) in predicting lower body joint torques of walking subjects unilaterally implanted with knee prostheses. These neural networks utilized Ground Reactive forces(GRF) and sEMG of eight lower body muscles to predict current joint torque. Their findings showed that the FFANN was able to predict knee flexion-extension torque with a normalized root mean squared error (NRMSE) of 10% while their WNN could predict knee flexion-extension torque with an NRMSE of 5%. When we normalize our knee flexion-extension torque prediction to the torque range of each subject, we find that our average NRMSE ranges between 3.21% at 50 ms prediction horizon to 8% at 150 ms prediction horizon. From this, we can see that torque prediction capabilities of this method perform similarly with adjacent works. Model predictions (Figures 3.3.4-3.3.6) also show that the waveforms produced by the networks closely mimic the natural gate waveform of the subjects.

It can be seen in Figures 3.3.4-3.3.6 that error most likely occurs at peaks and troughs in the gate cycle. This error appears to decrease as the prediction progresses. This may signify that a larger test dataset, like what would be seen in a real-world application, could significantly improve performance.

This analysis has shown that though, an average error has increased from the results of Coker et al., this method can still predict joint angle and torque accurately and is consistent with other similar studies. This error also appears to improve as the network continues to make predictions.

This supports the validity of this method and further analysis of this method should be performed.

4. MULTI-JOINT SYSTEM

4.1 Introduction

With the NIOTSNN validated on a single-joint system, the next step was to expand the network to a multi-joint system. In the case of this dataset, this entails the prediction of angle and torque about the ankle and hip. The first task is to incorporate the ankle as the muscles that directly govern its plantar flexion and dorsiflexion are monitored in this dataset. This allows the researcher to analyze the performance changes of this network as we add more joints when directly observing the governing muscles for that joint. The next task is to predict the mechanics of the hip. Muscles governing rotation of the hip were not recorded in this dataset. Predicting mechanics of the hip will display the performance loss of the network when sEMG cannot be monitored or when it is difficult to consistently monitor sEMG of the governing muscles of a joint.

4.2 Methods

For this section, the same network parameters were used as in Section 3. This includes the stopping conditions outlined in Table 3.2.1, Bayesian Regularization, and the structure of the network, except for the size of the input and output matrices. To predict knee and ankle mechanics, the input matrix incorporated eight features, four current sEMG channels, current ankle angle, current ankle torque, current knee angle, and current knee torque. The output matrix was comprised of four features, predicted knee angle, predicted knee torque, predicted ankle angle, and predicted ankle torque (Figure 4.2.1).

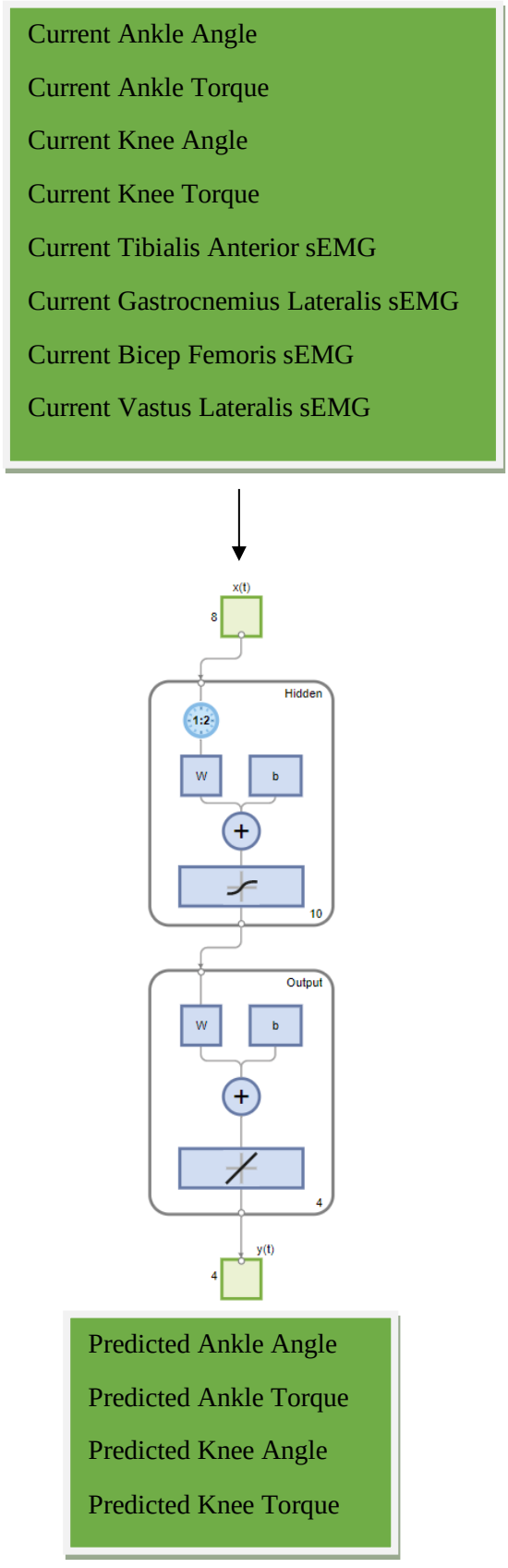

Figure 4.2.1: Knee-Ankle Nonlinear Input-Output Time Series Neural Network

As seen in Figure 4.2.2, sEMG recorded at the Tibialis Anterior (TA) and Gastrocnemius Lateralis (GAL) was also added to the input matrix of the NIOTSNN. The TA muscle contributes to dorsiflexion about the ankle and the GAL muscle contributes to plantar flexion about the ankle. These two movements were our main concern as they comprise the movement of the foot about the ankle in the sagittal plane.

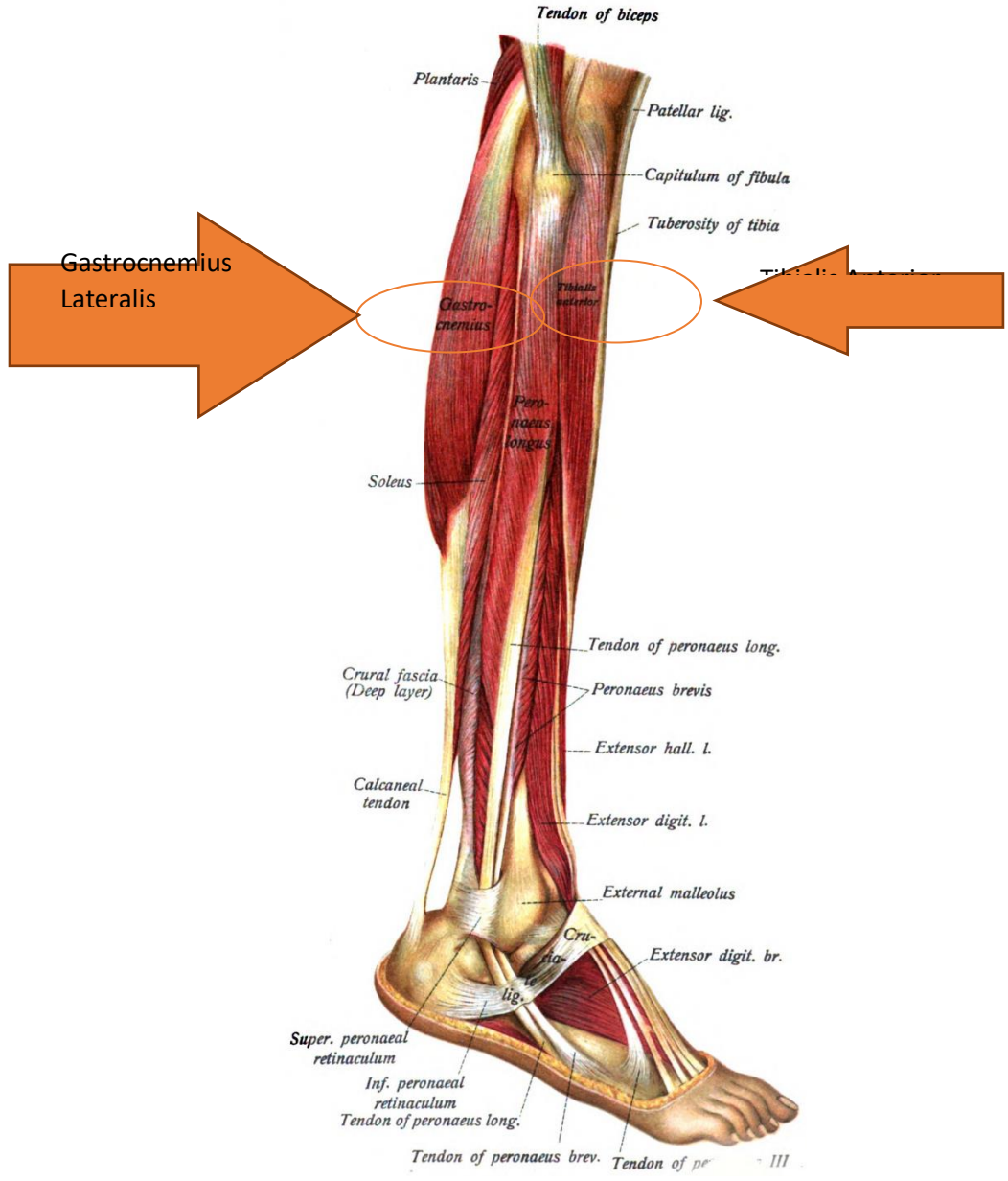

Figure 4.2.2: Gastrocnemius Lateralis and Tibialis Anterior Location (anatomytool.org)

To construct these matrices, a MATLAB script (Appendix 2.2) was written to perform all preprocessing steps described in Section 3. The only exception to this was data for two joints were processed and concatenated. Like Section 3, one network was trained for each leg of each subject at prediction horizons of 50 ms, 100 ms, and 150 ms, totaling 60 total networks (10 subjects x 2 legs x 3 prediction horizons). Nine trials were used for training while one trial was left out to test each resulting model. This allowed the researcher to compare our results for each joint to those produced in Section 3. The network was then trained using the training input and training target matrices until a stop condition was reached. Each model was then exported and saved to be evaluated. The resulting trained model was then evaluated using the returned testing data. The testing data was input into the model and a prediction was produced. The RMSE between the prediction and the time-shifted test data was then calculated.

This process was then repeated with the addition of hip mechanics. This resulted in an input matrix comprised of ten features, four current sEMG channels, current ankle angle, current ankle torque, current knee angle, current knee torque, current hip angle, and current hip torque. The output matrix was comprised of four features, predicted knee angle, predicted knee torque, predicted ankle angle, and predicted ankle torque (Figure 4.2.3).

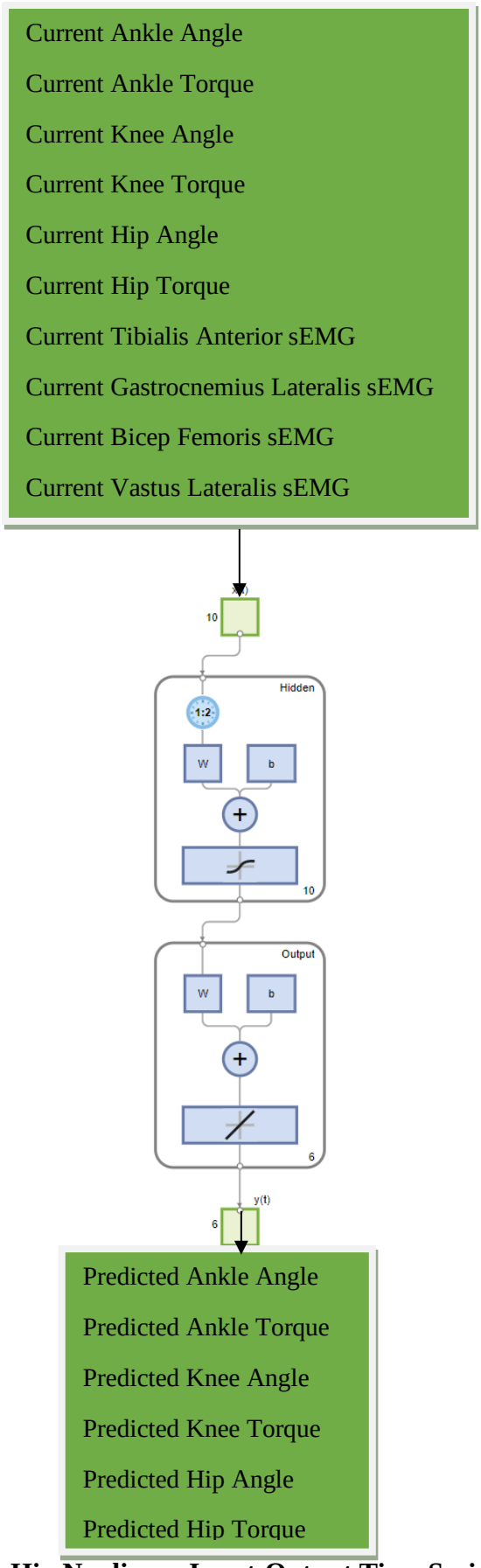

Figure 4.2.3:Knee-Ankle-Hip Nonlinear Input-Output Time Series Neural Network

4.3 Knee-Ankle Results

All 60 networks were evaluated with the withheld trial data for their corresponding subject and leg. The average training time was 7 minutes and 15 seconds. The average prediction time was 82 ms. This means that with current hardware, it would not be possible to compare to actual kinematic output at the 50 ms prediction horizon. With dedicated hardware, this prediction time could, potentially, be decreased below the 50ms threshold.

With all 60 knee-ankle networks evaluated, knee angle (Table 4. 3.1), knee torque (Table 4.3.2), ankle angle (Table 4. 3.3), and ankle torque (Table 4. 3.4) RMSE was produced.

Table 4.3.1: Knee-Ankle Network Knee Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.5565	1.8229	3.2953	3.4397	5.7466	4.9531
2	1.9908	1.8364	3.7466	2.7004	6.9236	4.0358
3	1.6267	1.4641	3.7544	3.1570	5.6813	11.3054
8	2.0899	1.3490	2.5504	2.9549	3.3841	5.7754
9	1.4476	1.3102	2.6217	2.3588	4.1538	3.1312
10	1.7523	1.5450	3.3853	2.9473	4.2220	6.0362
11	1.8991	1.8190	4.2584	5.0798	5.4472	5.2822
12	1.8367	1.9767	3.3967	3.3953	4.2370	4.7924
13	1.3541	1.5723	3.0941	3.1061	4.9581	3.9866
14	2.7763	1.4855	4.8703	4.1681	5.7393	5.1175
Average	1.8330	1.6181	3.4973	3.3307	5.0493	5.4416

Table 4.3.2: Knee-Ankle Network Knee Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.8286	2.2483	5.7604	4.6551	6.5505	5.2216
2	2.0730	1.8697	3.2389	3.4281	8.4997	4.4597
3	1.4659	1.3568	3.5705	2.6696	3.3034	4.1673
8	2.4703	1.9158	3.8472	3.6785	5.5682	5.0481
9	2.1068	1.6736	3.9500	3.6059	5.3333	5.5642
10	2.4061	2.2193	4.6913	4.3576	13.0596	5.6427
11	1.6183	1.7172	3.2210	3.8895	4.6381	4.2319
12	2.4983	1.7901	4.0161	3.6709	5.3368	4.8612
13	1.6994	1.7183	3.6544	3.0599	4.3762	3.9265
14	2.1968	1.7784	4.1783	3.6518	5.3249	4.5374
Average	2.1363	1.8287	4.0128	3.6667	6.1991	4.7661

Table 4.3.3: Knee-Ankle Network Ankle Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.0619	1.1948	2.1301	2.0881	2.9426	3.3683
2	0.9315	1.0976	1.9346	1.8208	3.7885	2.9187
3	1.3986	1.0410	2.6811	2.0486	3.3621	2.4415
8	1.0228	1.0817	2.1019	1.6810	2.8253	2.5822
9	1.1253	0.9577	1.9525	2.0621	3.1351	2.6070
10	1.0544	1.0857	2.1916	2.3925	3.6224	3.6352
11	1.0505	1.0845	2.3851	2.3132	3.2148	4.0360
12	1.1426	1.1689	2.2250	2.3588	2.5468	3.6927
13	0.9255	0.8685	1.9762	1.8289	3.2196	2.5038
14	1.3524	1.1254	1.9039	1.8434	2.5657	2.9509
Average	1.1065	1.0706	2.1482	2.0437	3.1223	3.0736

Table 4.3.4: Knee-Ankle Network Ankle Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.5401	3.0070	5.1807	5.9812	8.0286	9.0205
2	2.1462	2.9271	4.1330	4.1993	9.7637	5.4439
3	1.7139	2.3306	7.9429	3.3904	5.6473	5.1726
8	1.8353	1.7842	3.7474	4.1483	5.9430	5.5256
9	3.4798	2.5183	5.3988	4.5305	8.8019	7.0884
10	2.7194	2.6634	5.1280	6.5691	9.3357	12.7670
11	1.5829	1.6574	3.4410	2.8969	5.1465	7.4659
12	2.1493	2.0414	4.5901	4.6688	7.6451	8.3086
13	1.5988	1.8561	8.6869	4.0227	12.1785	5.7136
14	2.0874	1.9005	4.2694	3.7768	6.0536	6.1854
Average	2.1853	2.2686	5.2518	4.4184	7.8544	7.2691

When the predictions from the neural network were mapped against actual future data at 50 ms torque (Figure 4.3.5) and angle (Figure 4.3.6), 100 ms torque (Figure 4.3.7) and angle (Figure 4.3.8), and 150 ms torque (Figure 4.3.9) and angle (Figure 4.3.10), it can be seen just how well the actual joint mechanics can be reproduced with this method.

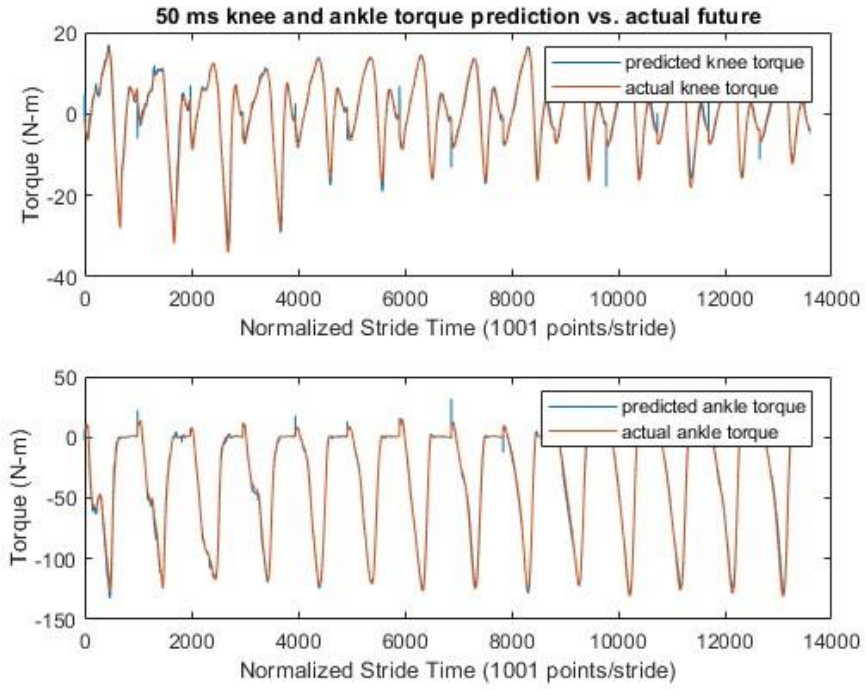

Figure 4.3.1: 50 ms Knee-Ankle Network Predicted vs. Actual Knee and Ankle Torque

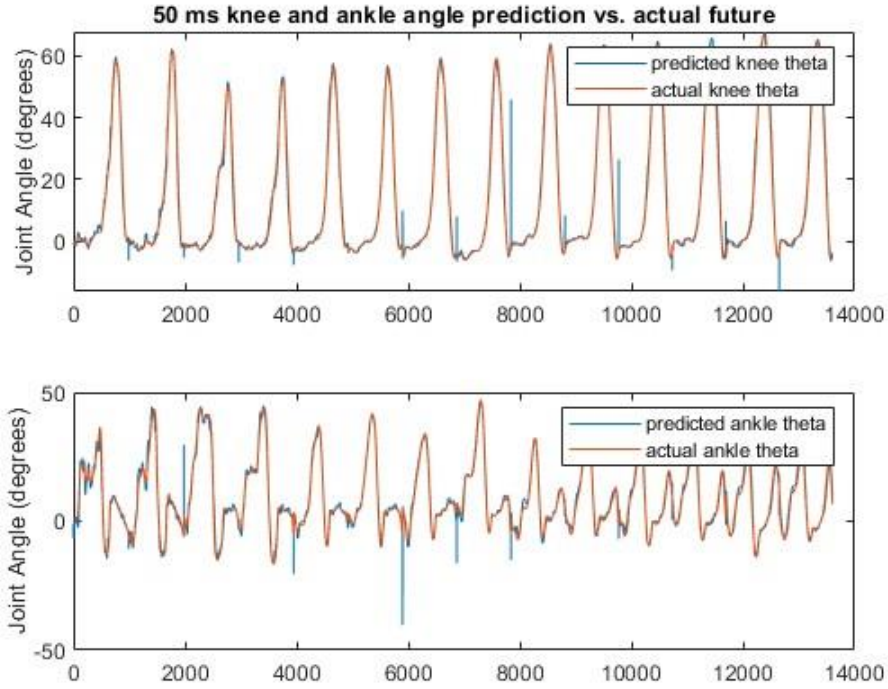

Figure 4.3.2: 50 ms Knee-Ankle Network Predicted vs. Actual Knee and Ankle Angle

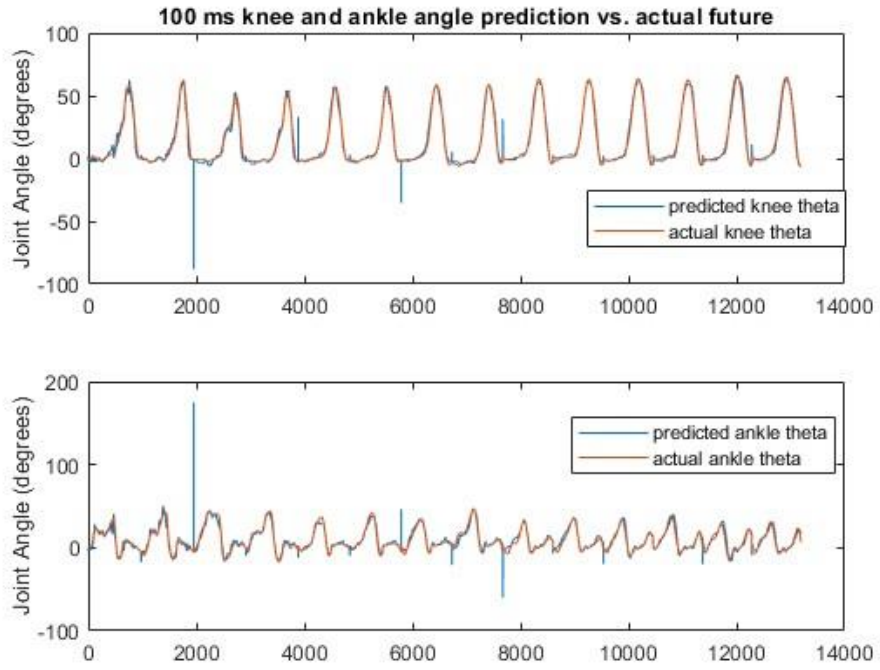

Figure 4.3.3: 100 ms Knee-Ankle Network Predicted vs. Actual Knee and Ankle Angle

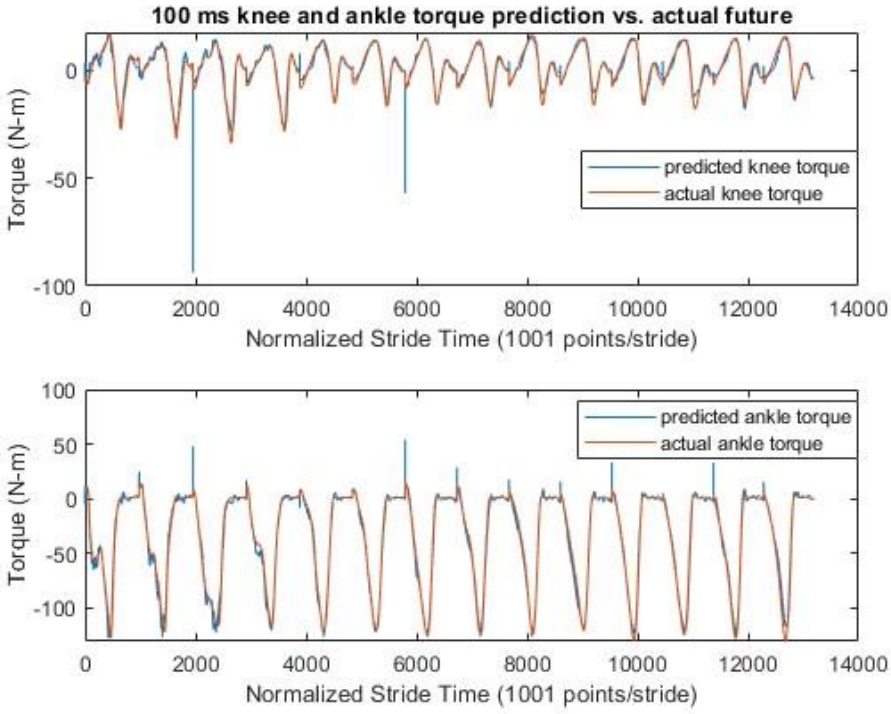

Figure 4.3.4: 100 ms Knee-Ankle Network Predicted vs. Actual Knee and Ankle Torque

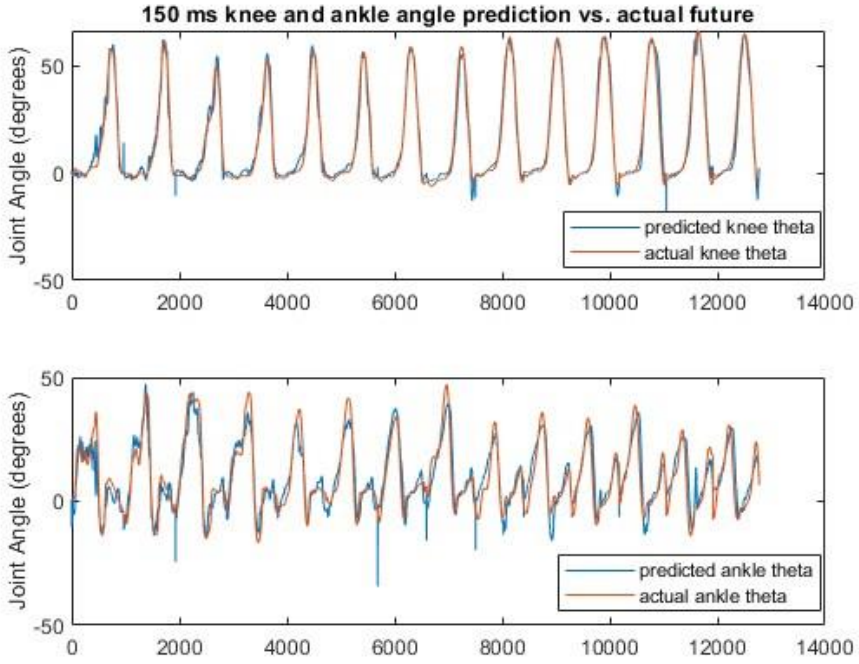

Figure 4.3.5: 150 ms Knee-Ankle Network Predicted vs. Actual Knee and Ankle Angle

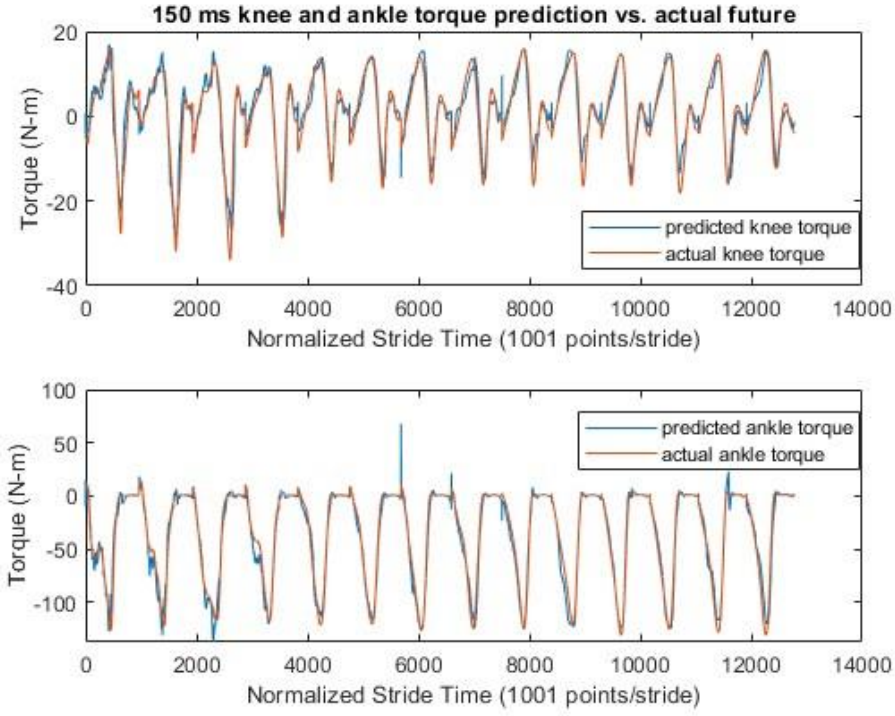

Figure 4.3.6: 150 ms Knee-Ankle Network Predicted vs. Actual Knee and Ankle Torque

Though RMSE is an accurate metric for average error, actual predictions show that error occurs at significantly different rates in certain areas of the waveform over others. To accurately show the areas of higher error, the error regions must be mapped over the true waveform. To accomplish this, testing gait data and network prediction for each subject were separately grand averaged. The average error was then determined by subtracting these two waveforms. The average error was then added to the actual gait data to produce an average error waveform. The difference was then shaded to show regions where the error is more likely. 50 ms, 100 ms, and 150 ms error region graphs can be seen in Figure 4.3.7, Figure 4.3.8, and Figure 4.3.9, respectively

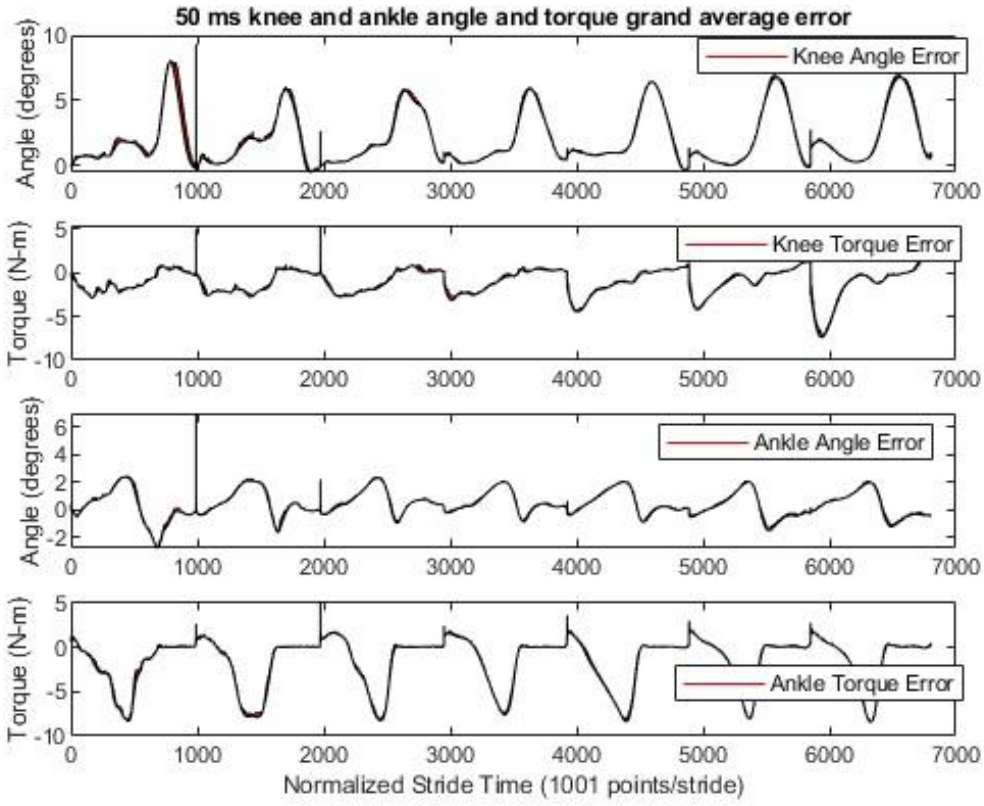

Figure 4.3.7: 50 ms Knee-Ankle Network Grand Average Prediction Error

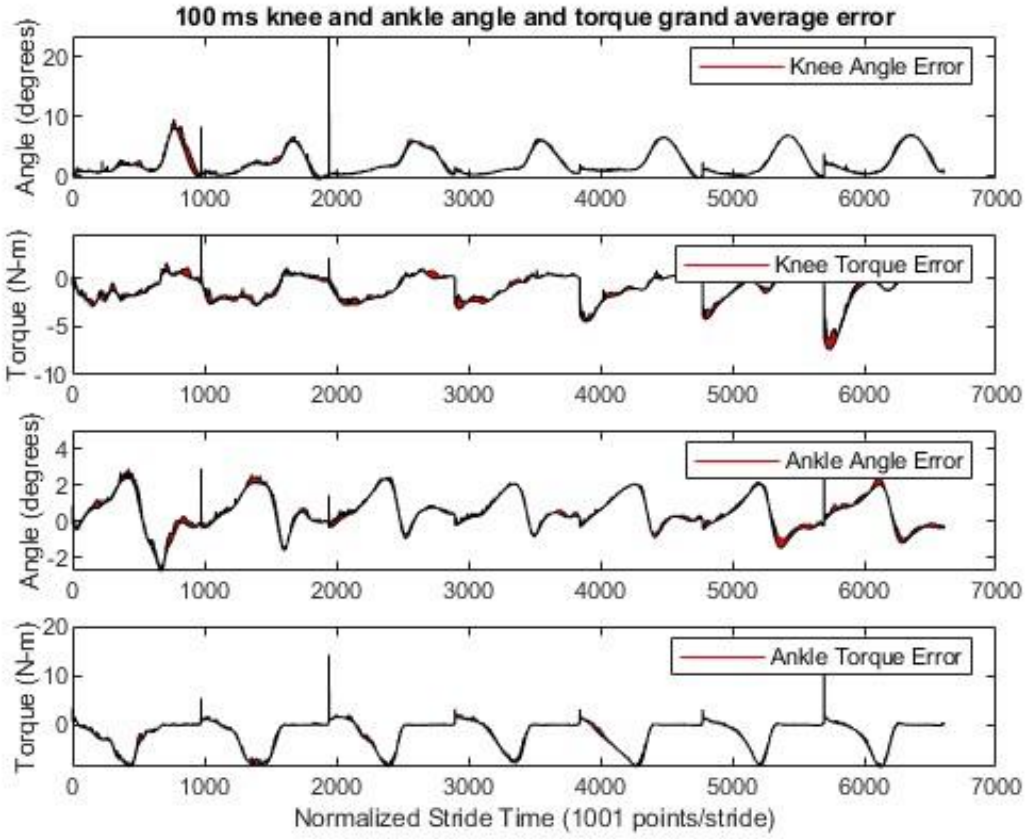

Figure 4.3.8: 100 ms Knee-Ankle Network Grand Average Prediction Error

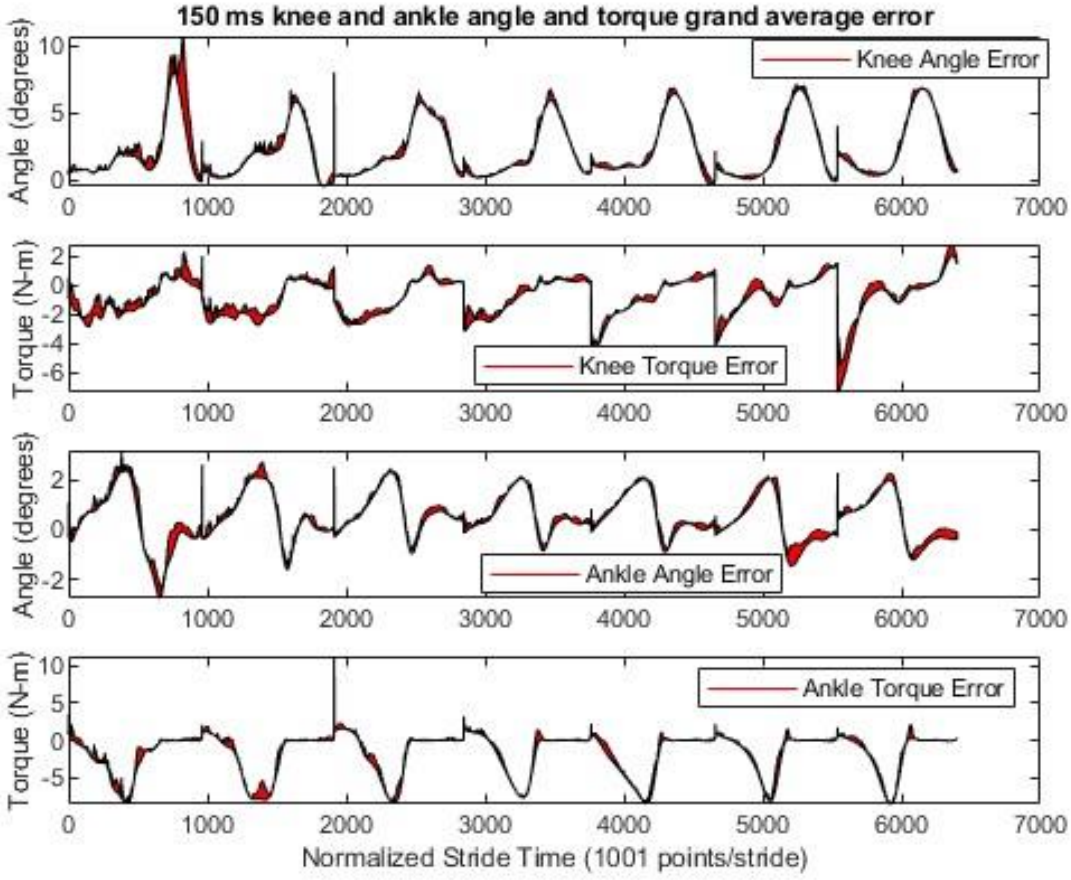

Figure 4.3.9: 150 ms Knee-Ankle Network Grand Average Prediction Error

4.4 Knee-Ankle-Hip Results

All 60 networks were evaluated with the withheld trial data for their corresponding subject and leg. The average training time was 15 minutes. The average prediction time was 155 ms. This means that with current hardware, it would not be possible to compare to actual kinematic output at any of our prediction horizons. With dedicated hardware, this prediction time could possibly be decreased below one or more of our prediction thresholds.

With all 60 knee-ankle-hip networks evaluated, knee torque (Table 4.4.1), knee angle (Table 4.4.2), ankle torque (Table 4.4.3), ankle angle (Table 4.4.4), hip torque (Table 4.4.3), and hip angle (Table 4.4.4) RMSE was produced.

Table 4.4.1: Knee-Ankle-Hip Network Knee Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.7578	2.0927	4.7975	3.7092	5.7072	4.9844
2	1.6578	1.6457	3.1700	3.4580	4.5116	4.3697
3	1.2771	1.2749	2.4565	2.3877	4.4716	6.9818
8	2.0096	2.1698	3.5031	3.1717	4.9612	4.1411
9	1.6443	1.7417	3.6458	3.0911	5.1629	4.5554
10	2.5803	2.2097	4.4653	4.1168	6.8915	5.5210
11	1.4173	1.6118	2.8443	3.2102	3.8890	3.9469
12	2.2411	1.8310	3.8901	3.1921	4.5988	4.4605
13	1.5985	1.3731	3.3709	3.0129	4.4825	3.5867
14	1.8119	1.7762	3.6514	3.6310	6.4946	4.8485
Average	1.8996	1.7727	3.5795	3.2981	5.1171	4.7396

Table 4.4.2: Knee-Ankle-Hip Network Knee Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.5999	1.6390	3.2112	3.3788	3.4434	5.0442
2	1.1760	4.4719	2.5843	2.3315	3.1710	3.2332
3	1.1106	1.0183	2.4778	3.0700	3.8966	16.4296
8	1.3104	1.0972	2.2488	2.0662	3.2845	3.0426
9	1.1182	1.2432	2.2259	1.9820	3.2454	3.2682
10	1.5213	1.4453	2.2450	3.1204	4.1063	3.5174
11	1.3499	1.4788	3.0278	2.7258	4.4876	4.0540
12	1.2272	1.1593	2.2846	1.9803	3.5053	2.7832
13	1.4969	1.1883	3.2525	2.4635	4.7354	4.1321
14	2.6966	1.7076	4.5054	2.7607	5.5013	4.7586
Average	1.4607	1.6449	2.8063	2.5879	3.9377	5.0263

Table 4.4.3: Knee-Ankle-Hip Network Ankle Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.2850	2.6887	4.2812	5.6200	6.1865	8.3997
2	1.5866	8.0674	4.5808	3.1883	5.2511	4.7318
3	1.7189	1.6034	3.7840	3.8302	26.9370	9.7799
8	1.9247	2.1975	3.7030	3.4635	5.7382	5.1747
9	2.5389	2.3470	4.2890	4.4364	7.3498	6.8997
10	1.9632	2.4106	4.6067	5.9261	6.5713	7.1248
11	1.5275	1.3957	2.6083	2.5840	5.1002	4.0929
12	1.8019	1.8446	4.1332	3.8167	7.2350	5.5378
13	1.7324	1.6180	3.6062	3.8859	5.4107	5.5897
14	3.6970	1.7732	7.1096	3.3815	18.8788	6.2148
Average	2.0776	2.5946	4.2702	4.0133	9.4659	6.3546

Table 4.4.5: Knee-Ankle-Hip Network Ankle Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	0.9178	1.0455	1.9376	2.0880	2.4983	3.1398
2	0.8733	2.6270	1.9756	1.8229	2.5203	2.9089
3	1.2179	0.8545	2.2982	1.6846	7.1504	4.6692
8	0.9438	0.9623	1.7501	1.6603	2.2980	2.5158
9	0.9146	1.0015	2.0284	2.0740	2.9802	2.7273
10	1.0921	1.2089	2.4410	2.3044	3.0817	3.8540
11	0.9524	0.9553	2.0087	2.2824	3.1523	3.1715
12	1.0834	1.0972	2.3467	2.5472	3.0115	3.2699
13	0.8870	0.7685	1.4893	1.7418	2.1950	2.0634
14	1.1537	0.9410	2.5053	1.5884	5.5151	3.1657
Average	1.0036	1.1462	2.0781	1.9794	3.4403	3.1486

Table 4.4.6: Knee-Ankle-Hip Network Hip Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.7555	2.4394	3.1910	3.8735	4.0336	5.4029
2	1.3806	3.5726	2.6267	2.5903	3.0806	4.4198
3	1.4600	1.4272	2.8151	2.9303	17.8767	9.7598
8	1.9995	1.9952	3.5752	3.2044	4.7538	4.3656
9	2.4728	2.2511	3.9468	3.9309	5.1502	5.3298
10	2.2781	2.8277	4.2466	5.3166	5.4004	5.4697
11	1.4323	1.6389	2.4672	2.7413	3.3335	3.6920
12	1.9434	2.2243	2.8231	3.1515	4.0492	3.9284
13	2.4318	1.9063	4.2149	3.3440	5.3705	4.0440
14	2.1520	1.8820	4.4416	3.5122	11.8854	5.2701
Average	1.9306	2.2165	3.4348	3.4595	6.4934	5.1682

Table 4.4.7: Knee-Ankle-Hip Network Hip Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	0.8304	0.9150	1.4610	1.9110	2.2324	2.6191
2	0.8653	0.8455	1.5757	1.6855	2.1372	2.5263
3	1.0429	1.0745	1.5164	1.7840	2.5877	2.5224
8	0.7227	0.7368	1.7284	1.4170	2.1813	1.9524
9	0.9326	0.8208	1.8448	1.5543	2.1933	2.2540
10	1.0240	1.0038	2.0732	1.9791	2.6256	2.6358
11	0.8562	0.9157	1.6164	1.7907	2.4437	3.2444
12	0.8340	0.9150	1.2257	1.2308	2.1668	2.2476
13	0.7841	0.7208	1.6217	1.5085	2.3662	2.0898
14	0.9129	0.9210	1.9926	1.8187	3.1768	2.6858
Average	0.8805	0.8869	1.6656	1.6680	2.4111	2.4778

When the predictions from the neural network are mapped against actual future data at 50 ms torque (Figure 4.4.1) and angle (Figure 4.4.2), 100 ms torque (Figure 4.4.3) and angle (Figure 4.4.4), and 150 ms torque (Figure 4.4.5) and angle (Figure 4.4.6), it can be seen just how well the actual joint mechanics can be reproduced with this method.

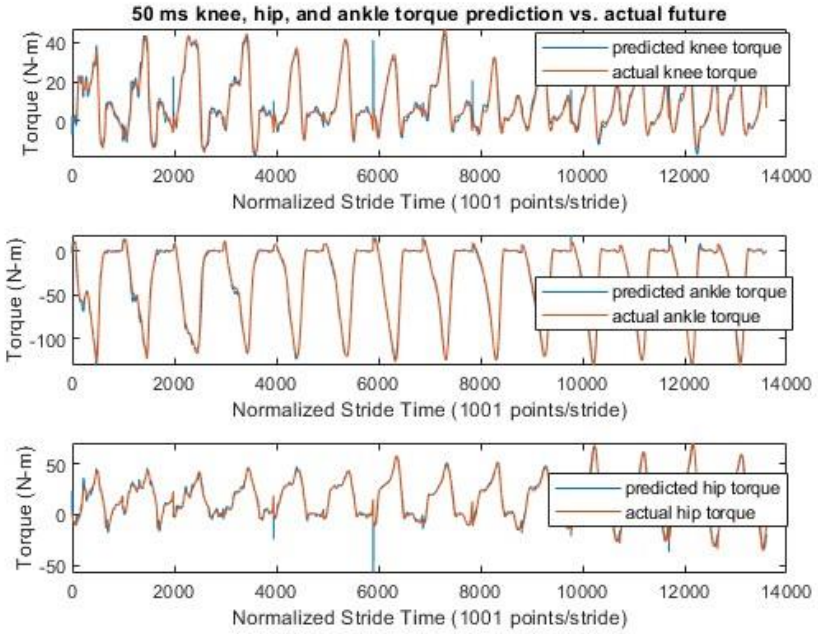

Figure 4.4.1: 50 ms Knee-Ankle-Hip Network Predicted vs Actual Knee, Ankle, and Hip Torque

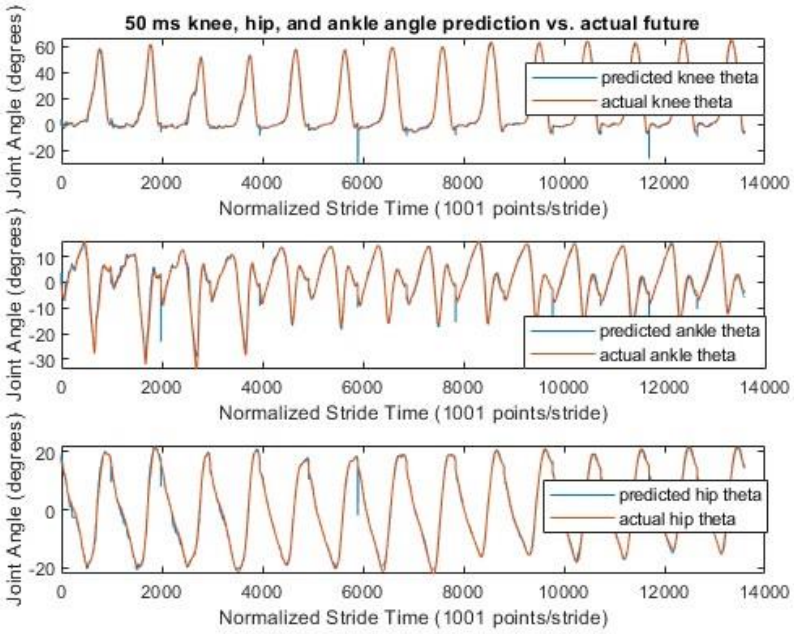

Figure 4.4.2: 50 ms Knee-Ankle-Hip Network Predicted vs Actual Knee, Ankle, and Hip Angle

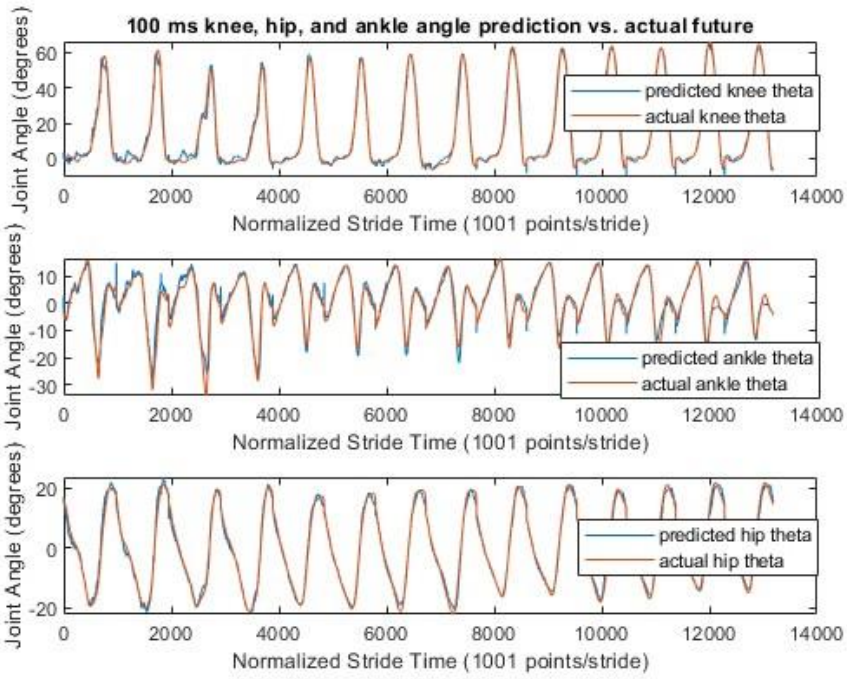

Figure 4.4.3: 100 ms Knee-Ankle-Hip Network Predicted vs Actual Knee, Hip, and Ankle Angle

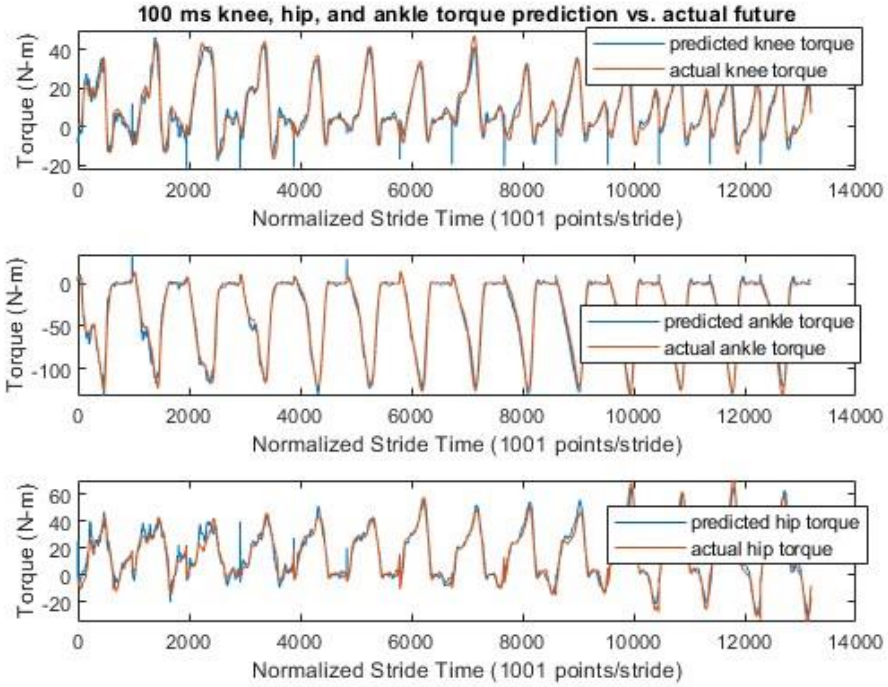

Figure 4.4.4:100 ms Knee-Ankle-Hip Network Predicted vs Actual Knee, Hip, and Ankle Torque

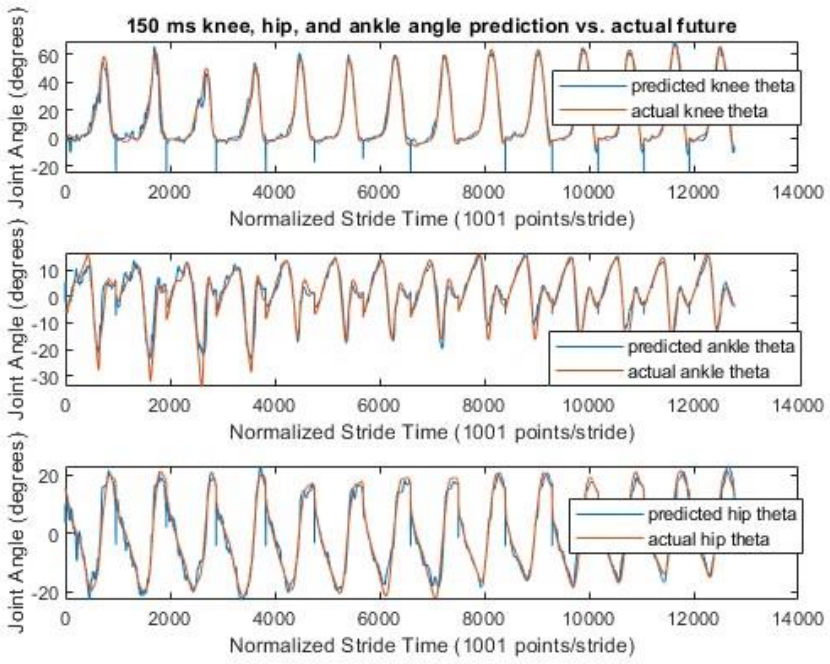

Figure 4.4.5: 150 ms Knee-Ankle-Hip Network Predicted vs Actual Knee, Hip, and Ankle Angle

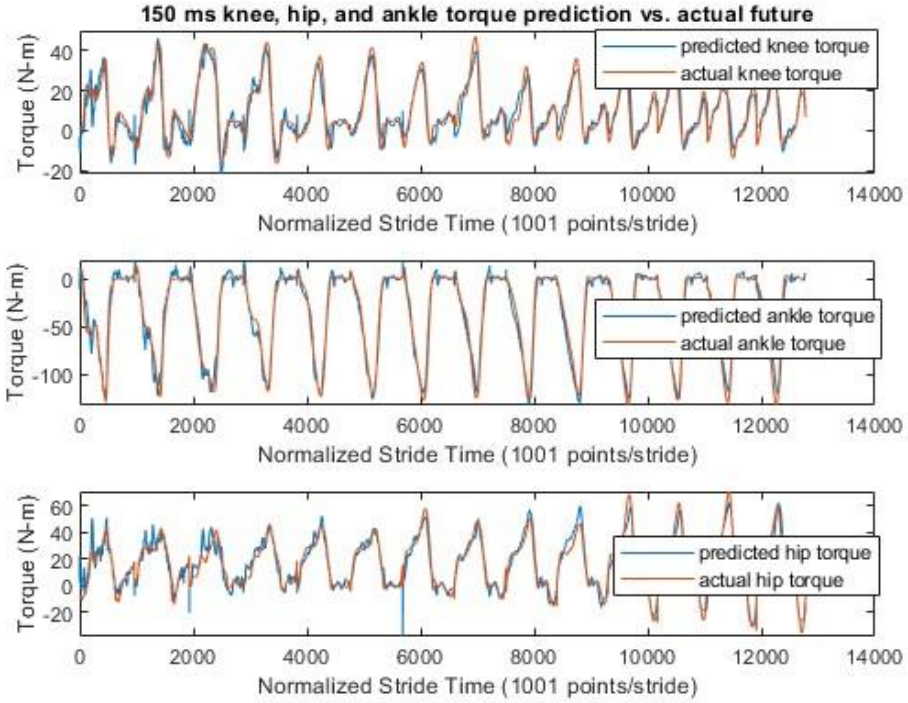

Figure 4.4.6:150 ms Knee-Ankle-Hip Network Predicted vs Actual Knee, Hip, and Ankle Torque

Though RMSE is an accurate metric for average error, actual predictions show that error occurs at significantly different rates in certain areas of the waveform over others. To accurately show the areas of higher error, the error regions must be mapped over the true waveform. Error region graphs were once again produced for this data. 50 ms, 100 ms, and 150 ms error region graphs can be seen in Figure 4.4.7, Figure 4.4.8, and Figure 4.4.9, respectively

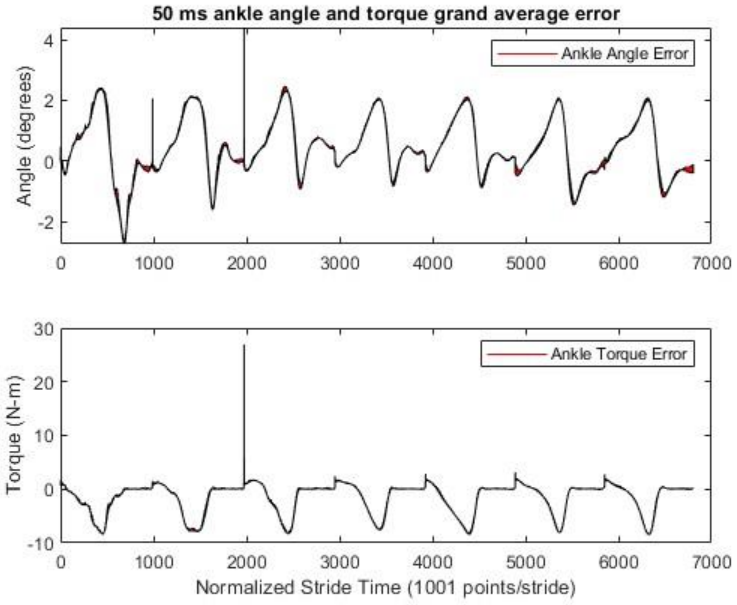

Figure 4.4.7.1: 50 ms Knee-Ankle-Hip Network Ankle Prediction Grand Average Error

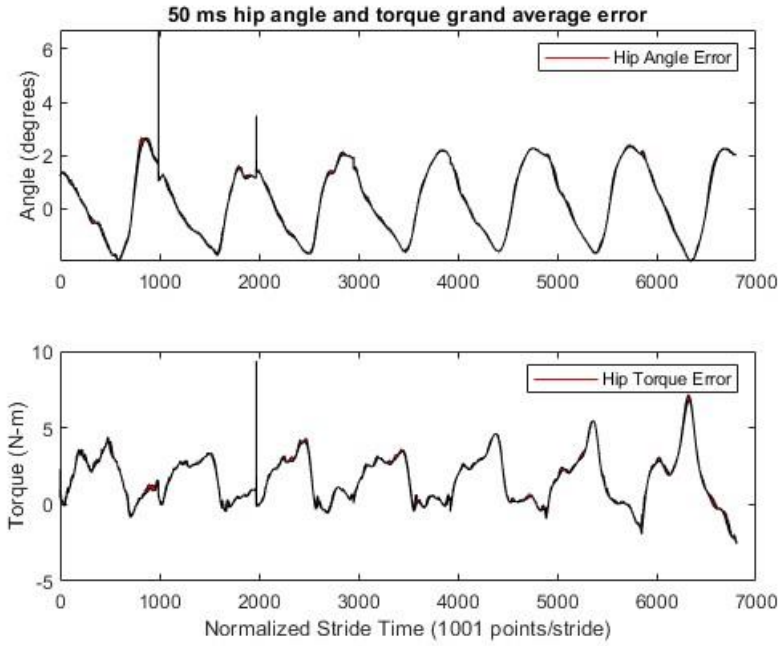

Figure 4.4.7.2: 50 ms Knee-Ankle-Hip Network Hip Prediction Grand Average Error

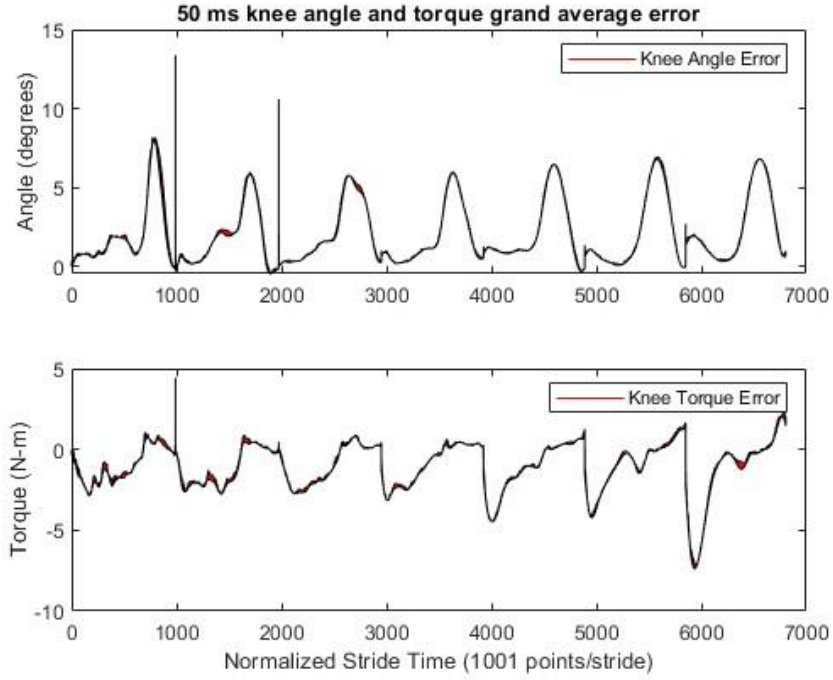

Figure 4.4.7.3: 50 ms Knee-Ankle-Hip Network Knee Prediction Grand Average Error

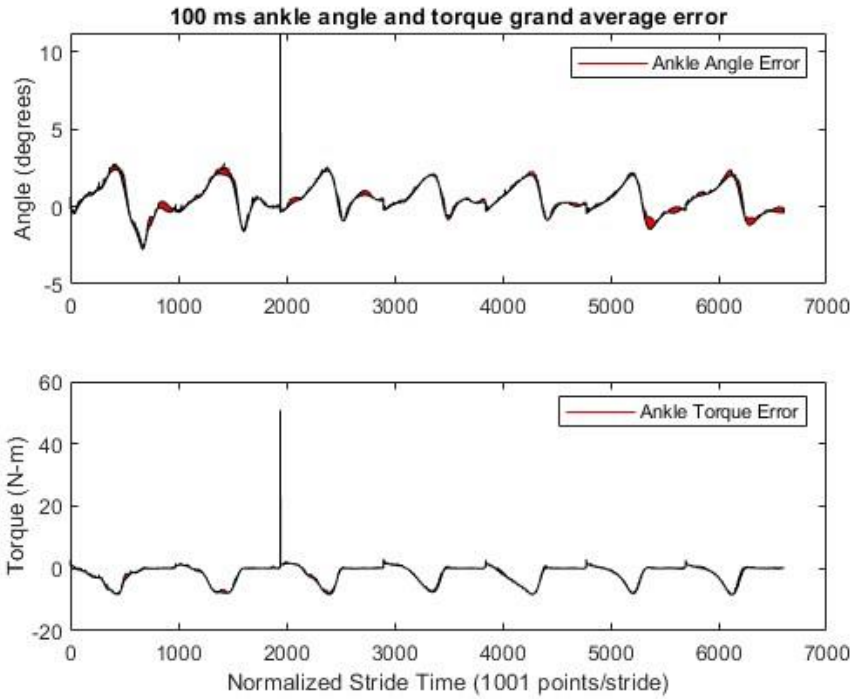

Figure 4.4.8.1: 100 ms Knee-Ankle-Hip Network Ankle Prediction Grand Average Error

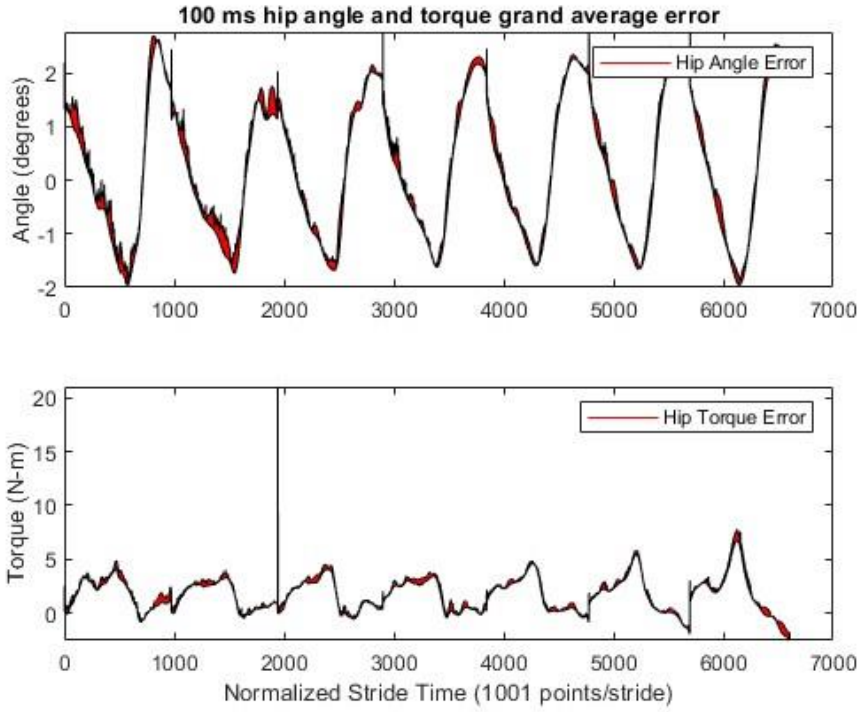

Figure 4.4.8.2: 100 ms Knee-Ankle-Hip Network Hip Prediction Grand Average Error

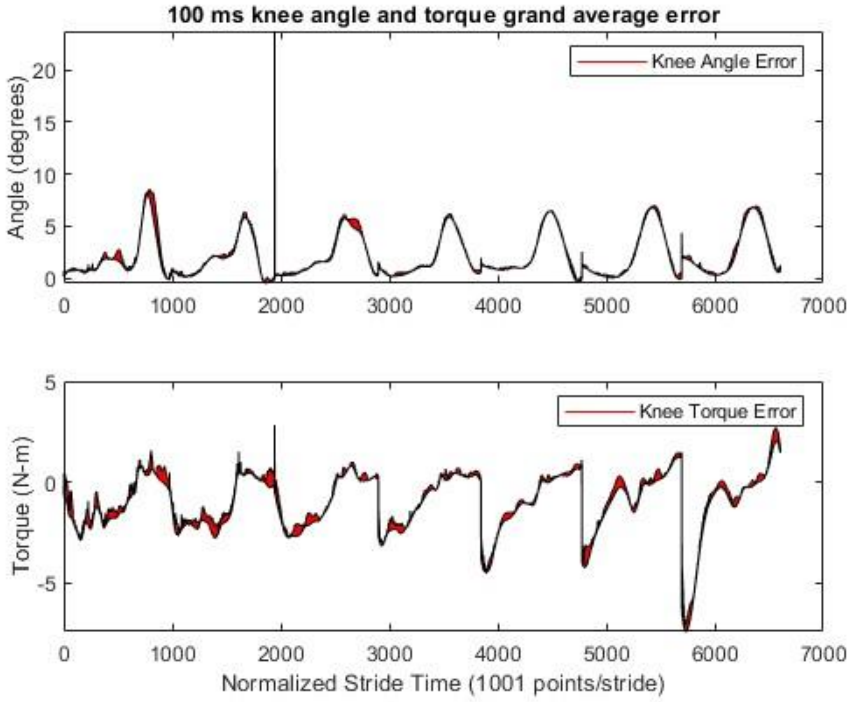

Figure 4.4.8.3: 100 ms Knee-Ankle-Hip Network Knee Prediction Grand Average Error

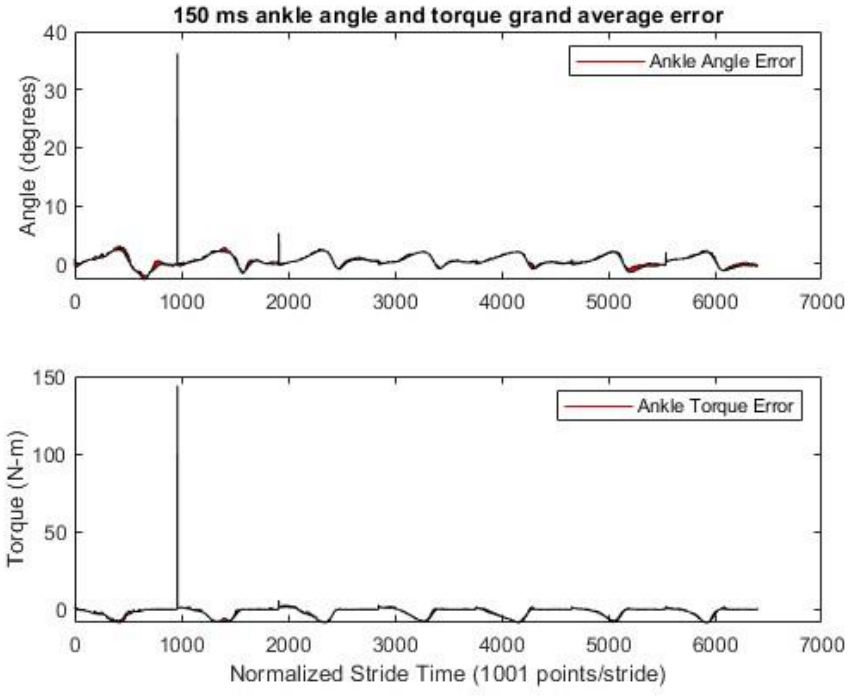

Figure 4.4.9.1: 150 ms Knee-Ankle-Hip Network Ankle Prediction Grand Average Error

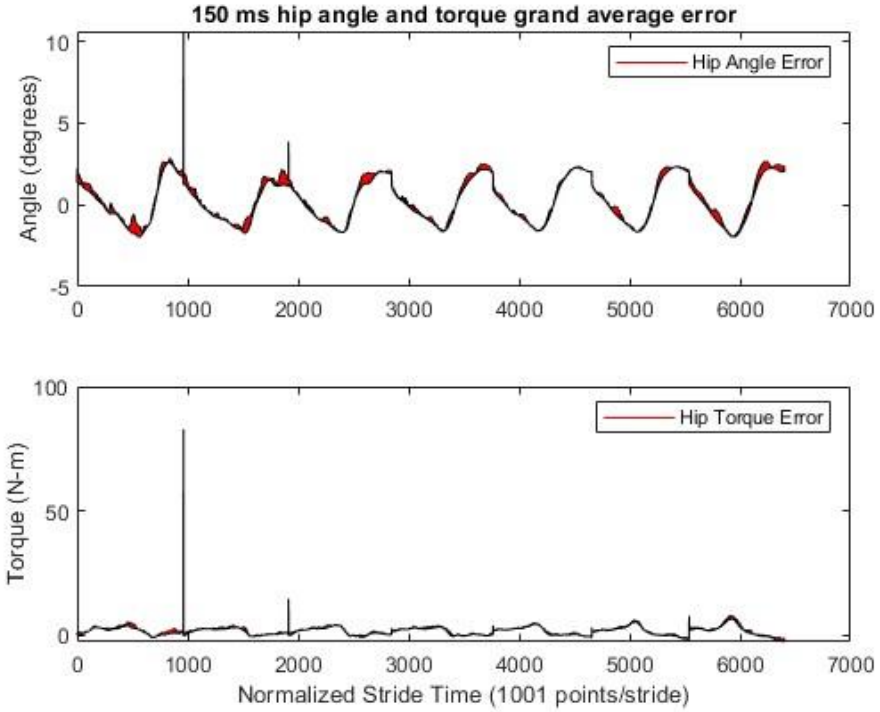

Figure 4.4.9.2: 150 ms Knee-Ankle-Hip Network Hip Prediction Grand Average Error

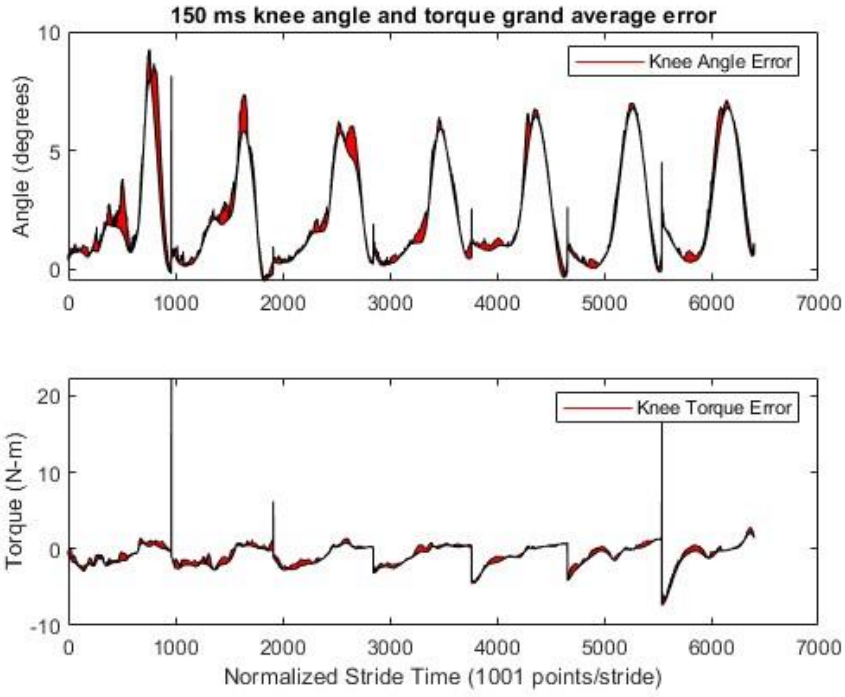

Figure 4.4.9.3: 150 ms Knee-Ankle-Hip Network Knee Prediction Grand Average Error

4.5 Discussion

This section evaluated the performance changes as additional joint predictions were added to the network. Between the single-joint prediction (Table 3.3.2 and Table 3.3.3) and the knee-ankle-hip prediction (Table 4.4.1 and Table 4.4.2), knee angle RMSE decreased by an average of 14.4% with a standard deviation of 15.4 degrees and knee torque RMSE decreased by an average of 13.5% with a standard deviation of 11.1 N-m. Knee mechanics predictions are the best metric for this as they appear in all network configurations. This performance improvement is likely a result of knee mechanics being dependent on the added components. Though the sEMG of the GAL and TA muscles may not govern knee flexion and extension, hip and ankle rotation are dependent on knee rotation. This is due to gait being relatively cyclical and dependent on the coordinated movement of all joints involved. It appears that adding mechanics of correlated joints may behave similarly sEMG data of governing muscles as they also impact the movement of the joint being predicted. This is exemplified by the addition of the hip joint. No muscles governing hip movement were observed but the addition of hip mechanics improved the prediction performance of other joints of the leg.

The other point evaluated in this section is the performance of a joint prediction when no governing sEMG data was considered by the neural network. It can be seen in Table 4.4.6 and Table 4.4.7 that the error of hip mechanics predictions were lower than those of any other joint. This was not anticipated as no sEMG data for the hip was considered. This raises the question of why this occurred. Hip mechanics may be more cyclical and inherently easier to predict for the network. Whatever the reason, the contribution of sEMG in joint mechanics predictions must be analyzed.

5. NETWORK SIMPLIFICATION AND REDUCTION OF ORDER

5.1 Introduction

Section 4 displayed the prediction accuracy improvement resulting from more kinematic and sEMG inputs. Though the network appears to be very capable of making accurate predictions, the complexity has made prediction time too slow to be valuable at current prediction horizons. To alleviate this, the network needs to be simplified so that it is not training on values that contribute minimally to its ultimate performance. In Section 4, it was observed that though the hip joint had no sEMG input, it outperformed all other joints. This could signify that sEMG contributes very little to prediction accuracy, so we began by eliminating sEMG from the network entirely and evaluated performance. We then performed principal component analysis (PCA) to further identify what elements contribute the most to prediction accuracy to further reduce the complexity of the network.

5.2 Methods

For this section, the same network parameters were used as Sections 3 and 4. This includes the stopping conditions outlined in Table 3.2.1, Bayesian Regularization, and the structure of the network, with the exception of the size of the input and output matrices. To predict knee, ankle, and hip mechanics, the input matrix incorporated six features, current ankle angle, current ankle torque, current knee angle, current knee torque, current hip angle, and current hip torque. The output matrix was comprised of six features, predicted knee angle, predicted knee torque, predicted ankle angle, predicted ankle torque, predicted hip angle, and predicted hip torque (Figure 5.2.1).

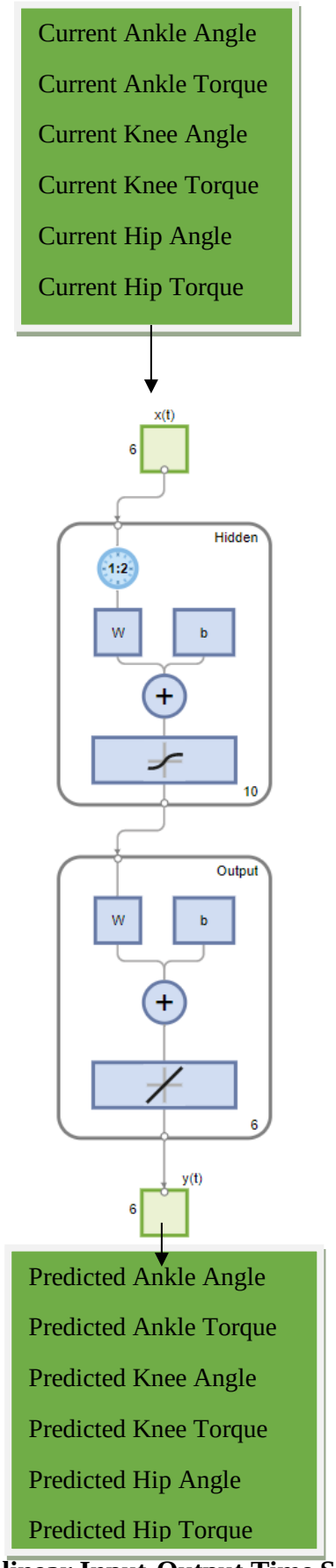

Figure 5.2.1: sEMG-Free Nonlinear Input-Output Time Series Neural Network

To construct these matrices, a MATLAB script (Appendix 2.4) was written to perform all preprocessing steps described in Section 3. The only exception to this was data for only joint mechanics was processed and concatenated with no sEMG data. Like Sections 3 and 4, one network was trained for each leg of each subject at prediction horizons of 50ms, 100ms, and 150ms, totaling 60 total networks (10 subjects x 2 legs x 3 prediction horizons). Nine trials were used for training while one trial was left out to test each resulting model. This allowed the researcher to compare our results for each joint to those produced in Section 4. Each network was then trained using the training input and training target matrices until a stop condition was reached. Each model was then exported and saved to be evaluated. The resulting trained model was then evaluated using the returned testing data. The testing data was entered into the model and a prediction was produced. The RMSE between the prediction and the time-shifted test data was then calculated.

5.3 Results

All 60 networks were evaluated with the withheld trial data for their corresponding subject and leg. The average training time was 10 minutes and 41 seconds. The average prediction time was 80 ms. This means that with current hardware, it would not be possible to compare to actual kinematic output at the 50 ms prediction horizon. With dedicated hardware, this prediction time could possibly be decreased below the 50ms threshold.

With all 60 sEMG-free knee-ankle-hip networks evaluated, knee torque (Table 5.3.1), knee angle (Table 5.3.2), ankle torque (Table 5.3.3), ankle angle (Table 5.3.4), hip torque (Table 5.3.5), and hip angle (Table 5.3.6) RMSE was produced.

Table 5.3.1: sEMG-Free Network Knee Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.7602	1.9675	4.9162	3.8821	5.4795	5.2097
2	19.3325	1.6052	3.2286	3.1162	4.8616	4.0622
3	1.5105	1.3429	2.7460	2.4114	2.9106	3.1518
8	1.8275	1.6664	3.5262	3.5204	4.7791	4.6906
9	1.5966	1.7961	3.8558	3.0796	5.1413	4.1098
10	2.4805	2.3434	4.5796	4.5029	6.4710	6.0515
11	1.3959	1.4452	3.0581	3.1161	4.0627	3.6911
12	2.1377	1.7914	3.9714	3.1972	5.0499	4.3504
13	1.5588	1.5090	3.2982	2.8707	3.7560	3.4750
14	2.0877	1.6774	3.3133	3.4543	5.1868	4.5651
Average	3.6688	1.7145	3.6493	3.3151	4.7699	4.3357

Table 5.3.2: sEMG-Free Network Knee Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.4467	1.6149	3.0351	3.3995	3.7910	8.9675
2	1.1922	1.2886	2.2002	2.1707	3.8718	3.3680
3	1.5642	1.3950	2.6580	2.7351	4.2596	3.6359
8	1.3118	1.0792	2.1322	2.1224	3.4639	3.4366
9	1.0777	1.0716	2.4523	2.1766	3.9867	2.6628
10	1.5857	2.1407	2.7206	2.7728	3.3568	4.0126
11	1.3702	1.5714	2.7133	2.8139	3.6415	4.0623
12	1.3345	1.0936	2.1815	2.1407	2.6993	3.4000
13	1.7439	1.5225	3.5760	2.9840	4.4212	4.0448
14	1.8332	1.5946	3.1315	2.8807	4.8912	4.4406
Average	1.4460	1.4372	2.6801	2.6196	3.8383	4.2031

Table 5.3.3: sEMG-Free Network Ankle Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	2.2838	2.5090	4.3589	5.5841	6.4202	13.3242
2	4.2734	1.5808	3.6806	3.4992	5.7301	5.1582
3	1.7982	1.6415	3.9528	3.4537	5.5525	5.1016
8	1.7207	2.0287	3.8950	3.6765	5.4733	5.7749
9	2.3020	2.5086	4.8238	4.6826	6.2927	6.4807
10	2.0364	4.8582	4.2606	5.9320	6.9038	13.9427
11	1.3840	1.5042	2.9177	2.8550	3.8260	4.0915
12	1.9162	1.7239	4.0744	4.2556	6.5875	5.9311
13	1.8444	2.0140	4.0856	4.1077	6.1210	5.6089
14	2.1788	1.7908	3.9688	3.6669	7.8406	5.7838
Average	2.1738	2.2160	4.0018	4.1713	6.0748	7.1198

Table 5.3.4: sEMG-Free Network Ankle Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	0.9528	1.0343	1.8794	2.3748	2.3294	4.6948
2	1.9889	0.8973	1.7910	2.0442	2.5427	3.2734
3	1.1493	0.8778	2.3156	1.6822	3.3269	2.6742
8	0.8865	0.9755	1.9690	1.6916	2.6626	2.4025
9	1.0407	1.1259	1.9097	2.0382	2.9050	2.9243
10	1.0090	1.6250	2.4784	2.7948	3.3465	3.8174
11	0.9086	1.0776	2.0812	2.3193	2.7464	3.2906
12	1.0116	1.1077	2.1323	2.6064	2.9258	3.4726
13	0.7748	0.8331	1.9072	1.8353	2.4912	2.8193
14	1.1017	0.8580	1.9021	2.0224	3.2066	2.8014
Average	1.0824	1.0412	2.0366	2.1409	2.8483	3.2171

Table 5.3.5: sEMG-Free Network Hip Torque Prediction RMSE (N-m)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	1.7872	2.3199	3.2055	4.1370	4.0034	6.7180
2	12.1127	1.2703	2.5234	2.8126	3.3168	3.4820
3	1.4587	1.4866	2.7063	2.6903	3.9125	3.5146
8	1.9548	1.9072	3.3899	3.4036	4.8392	4.5940
9	2.1696	2.4360	3.8426	3.8146	2.1571	5.2844
10	2.2894	3.1596	2.8072	4.7833	6.1985	8.8827
11	1.5453	1.6225	2.4746	2.7191	3.6124	3.3796
12	1.9799	2.0558	2.8075	3.4416	4.0497	3.9372
13	2.3722	2.0302	4.1749	3.3107	5.0671	4.1409
14	2.1455	1.9803	4.1921	3.3659	5.8247	4.8692
Average	2.9815	2.0268	3.2124	3.4479	4.2981	4.8803

Table 5.3.6: sEMG-Free Network Hip Angle Prediction RMSE (Degrees)

	50 ms		100 ms		150 ms	
subject	left	right	left	right	left	right
1	0.8317	0.9342	1.5301	1.7091	2.1853	3.2125
2	0.8057	0.8256	1.5199	1.6674	2.4452	2.5251
3	0.9846	1.0544	1.4318	1.8280	2.3058	2.5861
8	0.7213	0.7516	1.6335	1.4233	2.1115	2.1040
9	0.8331	0.7854	1.6467	1.6713	2.6871	1.9454
10	0.9292	1.1967	1.9303	1.9565	2.7580	2.6450
11	0.8073	0.9661	1.6706	1.9899	2.1438	2.3089
12	0.7516	0.7471	1.3193	1.6718	1.9994	2.1765
13	0.8165	0.7071	1.6618	1.5137	2.4402	2.2484
14	1.0627	0.8897	2.0268	1.9433	2.5550	2.3162
Average	0.8544	0.8858	1.6371	1.7374	2.3631	2.4068

It can be seen that the classification accuracy of the sEMG-free network minimally varies from that of the networks that consider sEMG. This is likely since sEMG governs movement so joint mechanics and sEMG may be somewhat redundant. To evaluate this, principal component analysis (PCA) had to be performed. To accomplish this, the MATLAB PCA function was used to quantify the contribution of each feature to the total variance of the dataset. PCA was performed on the concatenated input data for the knee-ankle-hip network. This produced ten principal components whose coefficient matrix can be seen in Table 5.3.7. This resulted in data variance dependence of each component seen in Table 5.3.8. It can be seen that over 90% of the variance of the data set can be described by four principal components, PC1, PC2, PC3, and PC4.

Table 5.3.7: Principal Component Coefficients for Knee-Ankle-Hip Data

	VL sEMG	BF sEMG	Knee Angle	Knee Torque	TA sEMG	GAL sEMG	Ankle Angle	Ankle Torque	Hip Angle	Hip Torque
PC1	0.0798	-0.3781	-0.1473	0.0664	0.1674	0.0142	-0.0426	0.8414	-0.2169	0.1988
PC2	0.0326	-0.1185	0.0627	0.3331	-0.3382	-0.4000	0.7637	0.0691	0.0811	-0.0302
PC3	0.1181	0.8333	-0.2386	0.3559	-0.0131	0.1575	0.0433	0.2815	-0.0172	0.0411
PC4	0.0805	0.2507	0.8756	-0.1009	0.2691	-0.1502	0.0519	0.1558	-0.1190	0.1330
PC5	0.1806	-0.1458	-0.0960	0.5578	0.6958	-0.1989	-0.0359	-0.2629	-0.0785	-0.1597
PC6	-0.0126	-0.0377	-0.0286	0.1341	0.0647	-0.0934	-0.1038	-0.0858	0.5740	0.7869
PC7	-0.1799	-0.0767	0.0126	0.1060	0.0578	0.5324	0.3504	-0.2658	-0.5284	0.4370
PC8	0.8451	0.0364	-0.1851	-0.3840	0.0231	-0.0890	0.1511	-0.1404	-0.1473	0.1736
PC9	0.2626	-0.1593	0.1858	0.0623	0.1134	0.6649	0.2447	0.1071	0.5172	-0.2710
PC10	-0.3519	0.1744	-0.2679	-0.5061	0.5290	-0.1038	0.4390	0.0742	0.1669	-0.0324

Table 5.3.8: Principal Component Variance Contribution for Knee-Ankle-Hip Data

PC1	67.6488
PC2	17.3013
PC3	9.0621
PC4	3.6098
PC5	1.78
PC6	0.5977
PC7	0.0003
PC8	0
PC9	0
PC10	0

5.4 Discussion

This section has demonstrated the networks' prediction ability has little dependence on many of the inputs to the network. This raises the question of which input variables are significantly contributing to the network's classification accuracy. Upon performing PCA on the entire dataset used by the neural networks, it can be seen in Table 5.3.8 that over 90% of the variance of the

data set can be described by four principal components, PC1, PC2, PC3, and PC4. These four principal components correlate closely with ankle torque, ankle angle, BF sEMG, and knee angle (Table 5.3.8). This suggests that this system can be mostly predicted through ankle torque, ankle angle, BF sEMG, and knee angle.

To evaluate this claim, each network seen in the past sections was retrained using these four components as input features and the six output features from the knee-ankle-hip network were set as targets. These networks took an average of 7 minutes and 31 seconds to train and took an average of 76 ms to make a prediction. Average classification RMSE across all subjects of each prediction target can be seen in Table 5.4.1.

Table 5.4.1: Simplified Network Average RMSE Across All Subjects

	50 ms		100ms		150ms	
	left	right	left	right	left	right
Knee Angle(deg.)	2.3569	2.2971	3.9639	4.1032	5.5108	5.5237
Knee Torque (N-M)	4.3521	3.7443	5.6158	5.4028	6.2207	5.3446
Ankle Angle(deg.)	1.7509	1.6804	2.5199	2.7725	3.2620	3.5212
Ankle Torque (N-M)	3.0569	2.9268	5.3494	5.2486	7.4317	7.8372
Hip Angle(deg.)	3.4883	3.4103	3.6834	3.6960	3.9275	4.2034
Hip torque (N-M)	5.3240	5.4032	5.8746	5.9312	6.5839	6.3378

When comparing these results to those of the full knee-ankle-hip network with 10 features, the average training and prediction time was reduced by 50%. This reduction of prediction time makes this simplified network capable of making predictions within the 100ms and 150ms prediction horizon on current hardware. On average, each parameter has a 67% higher prediction error when compared to the full network. Though this is a significant performance loss, it still outperforms similar systems.

Ardestani et al. [11] compared the effectiveness of a wavelet neural network (WNN) to a more traditional three-layer feed-forward artificial neural network (FFANN) in predicting lower body joint torques of walking subjects unilaterally implanted with knee prostheses. They were able to achieve torque prediction NRMSE for hip flexion-extension, knee flexion-extension, and ankle dorsiflexion-plantarflexion seen in Table 5.4.2. To compare our result to these findings, we normalized our torque RMSE to the total torque range of the subject, leg, and joint being evaluated. We then averaged this torque NRMSE between legs at each prediction horizon. Ardestani et al. predicted current joint torques from sEMG and GRF but the similarity in these methods provides a good baseline to compare with (Table 5.4.2).

Table 5.4.2: Ardestani et al. [11] Torque Prediction with WNN and FFANN vs. NIOTSNN Torque Prediction

	WNN(%)	FFANN(%)	NIOTSNN 50 ms (%)	NIOTSNN 100 ms(%)	NIOTSNN 150 ms(%)
Hip	9.00	14.00	6.80	8.10	9.20
Knee	5.00	10.00	2.80	5.00	7.30
Ankle	8.00	12.00	6.70	7.30	8.00

It can be seen in Table 5.4.2 that our NIOTSNN outperforms both the WNN and FFANN at prediction horizons up to 150ms, where it performs similarly to the WNN. This displays that even though our network is heavily simplified, it can still perform similarly to adjacent systems in terms of accuracy while maintaining low prediction and training times relative to our more complex networks.

6. CONCLUSION

We hypothesized that artificial neural networks can be used to predict an individual's joint torque and angle when past joint torque, angle, and sEMG activity of pertinent muscles were known. In this thesis, we have displayed this is true. We have also established that increasing the number of observed joints can greatly improve the prediction accuracy at each joint. This was a result of biomechanical motion being heavily dependent on certain factors over others. In the example presented here, over 90% of the variance of the data set was a result of four principal components that varied directly with four inputs. When this was evaluated, it was shown that when the network was simplified to these four inputs, acceptable prediction accuracy and speed were maintained.

These findings show that this method would be viable for the originally proposed assistive exoskeleton control system. The extent of the reduction of order that was performed here displays that this method could function with a less complex data collection system than originally proposed.

7. FUTURE WORK

With the effectiveness of this method shown, it can be expanded in the future. The main goal of this joint mechanics prediction method is to inform an assist-as-needed control system for assistive devices. To integrate this method into a real-world system, it will need to first be evaluated with real-time data. In this paper, all data post-processed and all strides were regularized to 1001 points. This would be impossible to do in real-time as the actual duration of a stride is unknown.

Other types of movement should also be evaluated on various body parts. In this paper, only walking was evaluated. Walking is very cyclical making it much easier to predict than other motions or changing between motions such as sitting or climbing. Upper body movements would also be a good candidate for this evaluation. With these motions being much less cyclical, they will likely not reduce dimensionality as extreme of an extent. These applications may benefit from sEMG consideration to a higher degree.

With the findings from this paper, this method appears to be a valid method to inform assistive device control systems.

APPENDICIES

Appendix 1: Data Extraction Function

Appendix 1.1: Ankle Data Extraction Function:

```
function[Input,Target,validation,validationTarget,time]=ankleExtract(delay, leg, sub)
    FileName   = 'Processed_Data.mat';
    for  i = sub;
        delay=delay
        Ltrainshin=[];
        Ltraincalf=[];
        LtrainAtheta=[];
        LtrainAtorque=[];
        Rtrainshin=[];
        Rtraincalf=[];
        RtrainAtheta=[];
        RtrainAtorque=[];


        Ltestshin=[];
        Ltestcalf=[];
        LtestAtheta=[];
        LtestAtorque=[];
        Rtestshin=[];
        Rtestcalf=[];
        RtestAtheta=[];
        RtestAtorque=[];


        Rvalcalf=[];
        Rvalshin=[];
        RvalAtheta=[];
        RvalAtorque=[];

        Lvalcalf=[];
        Lvalshin=[];
        LvalAtheta=[];
        LvalAtorque=[];

        RtargAtheta=[];
        LtargAtheta=[];

        RtargAtorque=[];
        LtargAtorque=[];

        Ltargcalf=[];
        Ltargshin=[];



        FolderName = sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\knee angle emg\\Participant%i\\',i);
        File = fullfile(FolderName, FileName);
        participantData = load(File);
```

```
participantData = struct2cell(participantData);
velocities = participantData{1};
velocities=struct2cell(velocities);
for v = 1:length(velocities) %all walking velocities
    v1=velocities{v}; %current walking velocity
    v1=struct2cell(v1);
    R=v1{1};
    R=struct2cell(R);
    R_emg=R{3};
    R_stridetime=R{5};

    R_emg_trials=R_emg(:,1);
    Rtrials=[];
    for rT = 1:length(R_emg_trials); % right emg data indicies
        rT;
        trialnum=R_emg_trials(rT);
        trialnum=(trialnum{1,1}{1,1});
        k = strsplit(trialnum,':');
        %k=cell2mat(k);
        k=(k(2));
        k=cell2mat(k);
        k=strtrim(k);
        k=str2num(k);
        Rtrials=[Rtrials, k]; % right emg data indicies array
    end

    R_theta=R{1};
    R_torque=R{8};
    for rtrial = Rtrials;

        stridetime1=R_stridetime(rtrial,2);
        stridetime1=table2array(stridetime1);

        stridetime2=R_stridetime(rtrial,3);
        stridetime2=table2array(stridetime2);

        removeentries1=round((delay/stridetime1));
        removeentries2=round(delay/stridetime2);

        theta=R_theta(rtrial,2);
        theta=table2array(theta{1});

        theta1=theta(:,19);
        theta1=theta1';

        traintheta1=theta1(1:end-removeentries1);
        testtheta1=theta1(removeentries1+1:end);

        theta2=theta(:,22);
        theta2=theta2';

        traintheta2=theta2(1:end-removeentries2);
        testtheta2=theta2(removeentries2+1:end);

        if rtrial == Rtrials(end)
```

```
    for element =1:length(traintheta1);
        RvalAtheta(end+1)=traintheta1(element);
    end
    for element =1:length(traintheta2);
        RvalAtheta(end+1)=traintheta2(element);
    end
    for element =1:length(testtheta1);
        RtargAtheta(end+1)=testtheta1(element);
    end
    for element =1:length(testtheta2);
        RtargAtheta(end+1)=testtheta2(element);
    end
else
    for element =1:length(traintheta1);
        RtrainAtheta(end+1)=traintheta1(element);
    end
    for element =1:length(traintheta2);
        RtrainAtheta(end+1)=traintheta2(element);
    end
    for element =1:length(testtheta1);
        RtestAtheta(end+1)=testtheta1(element);
    end
    for element =1:length(testtheta2);
        RtestAtheta(end+1)=testtheta2(element);
    end
end


torque=R_torque(rtrial,2);
torque=table2array(torque{1});

torque1=torque(:,19);
torque1=torque1';

traintorque1=torque1(1:end-removeentries1);
testtorque1=torque1(removeentries1+1:end);

torque2=torque(:,22);
torque2=torque2';

traintorque2=torque2(1:end-removeentries2);
testtorque2=torque2(removeentries2+1:end);
if rtrial == Rtrials(end)
    for element =1:length(traintorque1);
        RvalAtorque(end+1)=traintorque1(element);
    end
    for element =1:length(traintorque2);
        RvalAtorque(end+1)=traintorque2(element);
    end
    for element =1:length(testtorque1);
        RtargAtorque(end+1)=testtorque1(element);
    end
    for element =1:length(testtorque2);
        RtargAtorque(end+1)=testtorque2(element);
    end
```

```
        else
            for element =1:length(traintorque1);
                RtrainAtorque(end+1)=traintorque1(element);
            end
            for element =1:length(traintorque2);
                RtrainAtorque(end+1)=traintorque2(element);
            end
            for element =1:length(testtorque1);
                RtestAtorque(end+1)=testtorque1(element);
            end
            for element =1:length(testtorque2);
                RtestAtorque(end+1)=testtorque2(element);
            end
        end

    end
    for Rtrial = 1:length(R_emg); %collect emg data for current trial
         stridetime1=R_stridetime(Rtrials(Rtrial),2);
         stridetime1=table2array(stridetime1);

         stridetime2=R_stridetime(Rtrials(Rtrial),3);
         stridetime2=table2array(stridetime2);

         removeentries1=round((delay/stridetime1));
         removeentries2=round(delay/stridetime2);

         EMG=R_emg(Rtrial,2);
         EMG=EMG{1};

         shinEMG1=EMG(:,5);
         shinEMG1=table2array(shinEMG1)';

         calfEMG1=EMG(:,7);
         calfEMG1=table2array(calfEMG1)';

         shinEMG2=EMG(:,6);
         shinEMG2=table2array(shinEMG2)';

         calfEMG2=EMG(:,8);
         calfEMG2=table2array(calfEMG2)';

         trainshinEMG1=shinEMG1(1:end-removeentries1);

         traincalfEMG1=calfEMG1(1:end-removeentries1);

         trainshinEMG2=shinEMG2(1:end-removeentries2);

         traincalfEMG2=calfEMG2(1:end-removeentries2);


         if Rtrial== length(R_emg)
            for element =1:length(trainshinEMG1);
                Rvalshin(end+1)=trainshinEMG1(element);
            end
            for element =1:length(trainshinEMG2);
```

```
                    Rvalshin(end+1)=trainshinEMG2(element);
                end
                for element =1:length(traincalfEMG1);
                    Rvalcalf(end+1)=traincalfEMG1(element);
                end
                for element =1:length(traincalfEMG2);
                    Rvalcalf(end+1)=traincalfEMG2(element);
                end
            else
                for element =1:length(trainshinEMG1);
                    Rtrainshin(end+1)=trainshinEMG1(element);
                end
                for element =1:length(trainshinEMG2);
                    Rtrainshin(end+1)=trainshinEMG2(element);
                end
                for element =1:length(traincalfEMG1);
                    Rtraincalf(end+1)=traincalfEMG1(element);
                end
                for element =1:length(traincalfEMG2);
                    Rtraincalf(end+1)=traincalfEMG2(element);
                end
            end

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        L=v1{2};
        L=struct2cell(L);
        L_emg=L{3};
        L_stridetime=L{5};

        L_emg_trials=L_emg(:,1);
        Ltrials=[];
        for lT = 1:length(L_emg_trials); % right emg data indicies
            lT;
            trialnum=L_emg_trials(lT);
            trialnum=(trialnum{1,1}{1,1});
            k = strsplit(trialnum,':');
            %k=cell2mat(k);
            k=(k(2));
            k=cell2mat(k);
            k=strtrim(k);
            k=str2num(k);
            Ltrials=[Ltrials, k] ;% right emg data indicies array
        end

        L_theta=L{1};
        L_torque=L{8};
        for ltrial = Ltrials;

            stridetime1=L_stridetime(ltrial,2);
            stridetime1=table2array(stridetime1);

            removeentries1=round((delay/stridetime1));
```

```
        theta=L_theta(ltrial,2);
        theta=table2array(theta{1});

        theta1=theta(:,7);
        theta1=theta1';

        traintheta1=theta1(1:end-removeentries1);
        testtheta1=theta1(removeentries1+1:end);
        if ltrial== Ltrials(end)

            for element =1:length(traintheta1);
                LvalAtheta(end+1)=traintheta1(element);
            end
            for element =1:length(testtheta1);
                LtargAtheta(end+1)=testtheta1(element);
            end
        else
            for element =1:length(traintheta1);
                LtrainAtheta(end+1)=traintheta1(element);
            end
            for element =1:length(testtheta1);
                LtestAtheta(end+1)=testtheta1(element);
            end
        end

        torque=L_torque(ltrial,2);
        torque=table2array(torque{1});

        torque1=torque(:,7);
        torque1=torque1';

        traintorque1=torque1(1:end-removeentries1);
        testtorque1=torque1(removeentries1+1:end);

        if ltrial== Ltrials(end)
            for element =1:length(traintorque1);
                LvalAtorque(end+1)=traintorque1(element);
            end
            for element =1:length(testtorque1)
                LtargAtorque(end+1)=testtorque1(element);
            end
        else
            for element =1:length(traintorque1);
                LtrainAtorque(end+1)=traintorque1(element);
            end
            for element =1:length(testtorque1);
                LtestAtorque(end+1)=testtorque1(element);
            end
        end

    end
    for Ltrial = 1:length(L_emg); %collect emg data for current trial
         stridetime1=L_stridetime(Ltrials(Ltrial),2);
         stridetime1=table2array(stridetime1);
```

```
                removeentries1=round((delay/stridetime1));

                EMG=L_emg(Ltrial,2);
                EMG=EMG{1};

                shinEMG1=EMG(:,3);
                shinEMG1=table2array(shinEMG1)';

                calfEMG1=EMG(:,4);
                calfEMG1=table2array(calfEMG1)';


                trainshinEMG1=shinEMG1(1:end-removeentries1);


                traincalfEMG1=calfEMG1(1:end-removeentries1);

                testshinEMG1=shinEMG1(removeentries1+1:end);


                testcalfEMG1=calfEMG1(removeentries1+1:end);



                if Ltrial== length(L_emg)
                   for element =1:length(trainshinEMG1);
                       Lvalshin(end+1)=trainshinEMG1(element);
                   end
                   for element =1:length(traincalfEMG1);
                       Lvalcalf(end+1)=traincalfEMG1(element);
                   end
                   for element =1:length(testshinEMG1);
                       Ltargshin(end+1)=testshinEMG1(element);
                   end
                   for element =1:length(testcalfEMG1);
                       Ltargcalf(end+1)=testcalfEMG1(element);
                   end
                else
                   for element =1:length(trainshinEMG1);
                       Ltrainshin(end+1)=trainshinEMG1(element);
                   end
                   for element =1:length(traincalfEMG1);
                       Ltraincalf(end+1)=traincalfEMG1(element);
                   end
                   for element =1:length(testshinEMG1);
                       Ltestshin(end+1)=testshinEMG1(element);
                   end
                   for element =1:length(testcalfEMG1);
                       Ltestcalf(end+1)=testcalfEMG1(element);
                   end
                end


            end
        end
```

```
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    LEG=leg
    if LEG== "L"
        Input=[Ltrainshin;Ltraincalf;LtrainAtheta;LtrainAtorque];
        Target=[LtestAtheta;LtestAtorque];
        validation=[Lvalshin;Lvalcalf;LvalAtheta;LvalAtorque];
        validationTarget=[LtargAtheta;LtargAtorque];
        time=[1:length(validation)];
    else
        Input=[Rtrainshin;Rtraincalf;RtrainAtheta;RtrainAtorque];
        Target=[RtestAtheta;RtestAtorque];
        validation=[Rvalshin;Rvalcalf;RvalAtheta;RvalAtorque];
        validationTarget=[RtargAtheta;RtargAtorque];
        time=[1:length(validation)];
    end
```

Appendix 1.2: Knee Data Extraction Function:

```
function[Input,Target,validation,validationTarget,time]=kneeExtract(delay, leg,sub)
FileName   = 'Processed_Data.mat';
for  i = sub;
    delay=delay;
    Ltrainquad=[];
    Ltrainham=[];
    Ltraintheta=[];
    Ltraintorque=[];
    Rtrainquad=[];
    Rtrainham=[];
    Rtraintheta=[];
    Rtraintorque=[];
    Ltestquad=[];
    Ltestham=[];
    Ltesttheta=[];
    Ltesttorque=[];
    Rtestquad=[];
    Rtestham=[];
    Rtesttheta=[];
    Rtesttorque=[];


    Rvalham=[];
    Rvalquad=[];
    Rvaltheta=[];
    Rvaltorque=[];

    Lvalham=[];
    Lvalquad=[];
    Lvaltheta=[];
    Lvaltorque=[];

    Rtargtheta=[];
    Ltargtheta=[];
```

```
    Rtargtorque=[];
    Ltargtorque=[];

    Ltargham=[];
    Ltargquad=[];



    FolderName = sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\knee angle emg\\Participant%i\\',i);
    File = fullfile(FolderName, FileName);
    participantData = load(File);
    participantData = struct2cell(participantData);
    velocities = participantData{1};
    velocities=struct2cell(velocities);
    for v = 1:length(velocities) %all walking velocities
        v1=velocities{v}; %current walking velocity
        v1=struct2cell(v1);
        R=v1{1};
        R=struct2cell(R);
        R_emg=R{3};
        R_stridetime=R{5};

        R_emg_trials=R_emg(:,1);
        Rtrials=[];
        for rT = 1:length(R_emg_trials); % right emg data indicies
            rT;
            trialnum=R_emg_trials(rT);
            trialnum=(trialnum{1,1}{1,1});
            k = strsplit(trialnum,':');
            %k=cell2mat(k);;
            k=(k(2));
            k=cell2mat(k);
            k=strtrim(k);
            k=str2num(k);
            Rtrials=[Rtrials, k]; % right emg data indicies array
        end
        Rtrials;

        R_theta=R{1};
        R_torque=R{8};
        for rtrial = Rtrials;

            stridetime1=R_stridetime(rtrial,2);
            stridetime1=table2array(stridetime1);

            stridetime2=R_stridetime(rtrial,3);
            stridetime2=table2array(stridetime2);

            removeentries1=round((delay/stridetime1));
            removeentries2=round(delay/stridetime2);

            theta=R_theta(rtrial,2);
            theta=table2array(theta{1});
```

```
theta1=theta(:,13);
theta1=theta1';

traintheta1=theta1(1:end-removeentries1);
testtheta1=theta1(removeentries1+1:end);

theta2=theta(:,16);
theta2=theta2';

traintheta2=theta2(1:end-removeentries2);
testtheta2=theta2(removeentries2+1:end);
if rtrial == Rtrials(end)
    for element =1:length(traintheta1);
        Rvaltheta(end+1)=traintheta1(element);
    end
    for element =1:length(traintheta2);
        Rvaltheta(end+1)=traintheta2(element);
    end
    for element =1:length(testtheta1);
        Rtargtheta(end+1)=testtheta1(element);
    end
    for element =1:length(testtheta2);
        Rtargtheta(end+1)=testtheta2(element);
    end
else
    for element =1:length(traintheta1);
        Rtraintheta(end+1)=traintheta1(element);
    end
    for element =1:length(traintheta2);
        Rtraintheta(end+1)=traintheta2(element);
    end
    for element =1:length(testtheta1);
        Rtesttheta(end+1)=testtheta1(element);
    end
    for element =1:length(testtheta2);
        Rtesttheta(end+1)=testtheta2(element);
    end
end


torque=R_torque(rtrial,2);
torque=table2array(torque{1});

torque1=torque(:,13);
torque1=torque1';

traintorque1=torque1(1:end-removeentries1);
testtorque1=torque1(removeentries1+1:end);

torque2=torque(:,16);
torque2=torque2';

traintorque2=torque2(1:end-removeentries2);
testtorque2=torque2(removeentries2+1:end);
```

```
        if rtrial == Rtrials(end)
            for element =1:length(traintorque1);
                Rvaltorque(end+1)=traintorque1(element);
            end
            for element =1:length(traintorque2);
                Rvaltorque(end+1)=traintorque2(element);
            end
            for element =1:length(testtorque1);
                Rtargtorque(end+1)=testtorque1(element);
            end
            for element =1:length(testtorque2);
                Rtargtorque(end+1)=testtorque2(element);
            end
        else
            for element =1:length(traintorque1);
                Rtraintorque(end+1)=traintorque1(element);
            end
            for element =1:length(traintorque2);
                Rtraintorque(end+1)=traintorque2(element);
            end
            for element =1:length(testtorque1);
                Rtesttorque(end+1)=testtorque1(element);
            end
            for element =1:length(testtorque2);
                Rtesttorque(end+1)=testtorque2(element);
            end
        end

    end



    for Rtrial = 1:length(R_emg); %collect emg data for current trial
         stridetime1=R_stridetime(Rtrials(Rtrial),2);
         stridetime1=table2array(stridetime1);

         stridetime2=R_stridetime(Rtrials(Rtrial),3);
         stridetime2=table2array(stridetime2);

         removeentries1=round((delay/stridetime1));
         removeentries2=round(delay/stridetime2);

         EMG=R_emg(Rtrial,2);
         EMG=EMG{1};

         quadEMG1=EMG(:,1);
         quadEMG1=table2array(quadEMG1)';

         hamEMG1=EMG(:,3);
         hamEMG1=table2array(hamEMG1)';

         quadEMG2=EMG(:,2);
         quadEMG2=table2array(quadEMG2)';
```

```
            hamEMG2=EMG(:,4);
            hamEMG2=table2array(hamEMG2)';

            trainquadEMG1=quadEMG1(1:end-removeentries1);

            trainhamEMG1=hamEMG1(1:end-removeentries1);

            trainquadEMG2=quadEMG2(1:end-removeentries2);

            trainhamEMG2=hamEMG2(1:end-removeentries2);


            if Rtrial== length(R_emg)
               for element =1:length(trainquadEMG1);
                   Rvalquad(end+1)=trainquadEMG1(element);
               end
               for element =1:length(trainquadEMG2);
                   Rvalquad(end+1)=trainquadEMG2(element);
               end
               for element =1:length(trainhamEMG1);
                   Rvalham(end+1)=trainhamEMG1(element);
               end
               for element =1:length(trainhamEMG2);
                   Rvalham(end+1)=trainhamEMG2(element);
               end
            else
               for element =1:length(trainquadEMG1);
                   Rtrainquad(end+1)=trainquadEMG1(element);
               end
               for element =1:length(trainquadEMG2);
                   Rtrainquad(end+1)=trainquadEMG2(element);
               end
               for element =1:length(trainhamEMG1);
                   Rtrainham(end+1)=trainhamEMG1(element);
               end
               for element =1:length(trainhamEMG2);
                   Rtrainham(end+1)=trainhamEMG2(element);
               end
            end

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        L=v1{2};
        L=struct2cell(L);
        L_emg=L{3};
        L_stridetime=L{5};

        L_emg_trials=L_emg(:,1);
        Ltrials=[];
        for lT = 1:length(L_emg_trials); % right emg data indicies
            lT;
            trialnum=L_emg_trials(lT);
            trialnum=(trialnum{1,1}{1,1});
            k = strsplit(trialnum,':');
```

```
        %k=cell2mat(k);
        k=(k(2));
        k=cell2mat(k);
        k=strtrim(k);
        k=str2num(k);
        Ltrials=[Ltrials, k] ;% right emg data indicies array
    end

    L_theta=L{1};
    L_torque=L{8};
    for ltrial = Ltrials;

        stridetime1=L_stridetime(ltrial,2);
        stridetime1=table2array(stridetime1);

        removeentries1=round((delay/stridetime1));

        theta=L_theta(ltrial,2);
        theta=table2array(theta{1});

        theta1=theta(:,4);
        theta1=theta1';

        traintheta1=theta1(1:end-removeentries1);
        testtheta1=theta1(removeentries1+1:end);
        if ltrial== Ltrials(end)

            for element =1:length(traintheta1);
                Lvaltheta(end+1)=traintheta1(element);
            end
            for element =1:length(testtheta1);
                Ltargtheta(end+1)=testtheta1(element);
            end
        else
            for element =1:length(traintheta1);
                Ltraintheta(end+1)=traintheta1(element);
            end
            for element =1:length(testtheta1);
                Ltesttheta(end+1)=testtheta1(element);
            end
        end

        torque=L_torque(ltrial,2);
        torque=table2array(torque{1});

        torque1=torque(:,4);
        torque1=torque1';

        traintorque1=torque1(1:end-removeentries1);
        testtorque1=torque1(removeentries1+1:end);

        if ltrial== Ltrials(end)
            for element =1:length(traintorque1);
                Lvaltorque(end+1)=traintorque1(element);
            end
```

```
            for element =1:length(testtorque1);
                Ltargtorque(end+1)=testtorque1(element);
            end
        else
            for element =1:length(traintorque1);
                Ltraintorque(end+1)=traintorque1(element);
            end
            for element =1:length(testtorque1);
                Ltesttorque(end+1)=testtorque1(element);
            end
        end

    end
    for Ltrial = 1:length(L_emg); %collect emg data for current trial
        stridetime1=L_stridetime(Ltrials(Ltrial),2);
        stridetime1=table2array(stridetime1);

        removeentries1=round((delay/stridetime1));

        EMG=L_emg(Ltrial,2);
        EMG=EMG{1};

        quadEMG1=EMG(:,1);
        quadEMG1=table2array(quadEMG1)';

        hamEMG1=EMG(:,2);
        hamEMG1=table2array(hamEMG1)';


        trainquadEMG1=quadEMG1(1:end-removeentries1);


        trainhamEMG1=hamEMG1(1:end-removeentries1);

        testquadEMG1=quadEMG1(removeentries1+1:end);


        testhamEMG1=hamEMG1(removeentries1+1:end);



        if Ltrial== length(L_emg)
            for element =1:length(trainquadEMG1);
                Lvalquad(end+1)=trainquadEMG1(element);
            end
            for element =1:length(trainhamEMG1);
                Lvalham(end+1)=trainhamEMG1(element);
            end
            for element =1:length(testquadEMG1);
                Ltargquad(end+1)=testquadEMG1(element);
            end
            for element =1:length(testhamEMG1);
                Ltargham(end+1)=testhamEMG1(element);
            end
        else
```

```
                for element =1:length(trainquadEMG1);
                    Ltrainquad(end+1)=trainquadEMG1(element);
                end
                for element =1:length(trainhamEMG1);
                    Ltrainham(end+1)=trainhamEMG1(element);
                end
                for element =1:length(testquadEMG1);
                    Ltestquad(end+1)=testquadEMG1(element);
                end
                for element =1:length(testhamEMG1);
                    Ltestham(end+1)=testhamEMG1(element);
                end
            end


        end
    end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LEG=leg
if LEG== "L"
    Input=[Ltrainquad;Ltrainham;Ltraintheta;Ltraintorque];
    Target=[Ltesttheta;Ltesttorque];
    validation=[Lvalquad;Lvalham;Lvaltheta;Lvaltorque];
    validationTarget=[Ltargtheta;Ltargtorque];
    time=[1:length(validation)];
else
    Input=[Rtrainquad;Rtrainham;Rtraintheta;Rtraintorque];
    Target=[Rtesttheta;Rtesttorque];
    validation=[Rvalquad;Rvalham;Rvaltheta;Rvaltorque];
    validationTarget=[Rtargtheta;Rtargtorque];
    time=[1:length(validation)];
end
```

Appendix 1.3: Knee Data Extraction Function:

```
function[Input,Target,validation,validationTarget,time]=kneeExtract(delay, leg,sub)
FileName   = 'Processed_Data.mat';
for  i = sub;
    delay=delay
    Ltraintheta=[];
    Ltraintorque=[];
    Rtraintheta=[];
    Rtraintorque=[];
    Ltesttheta=[];
    Ltesttorque=[];
    Rtesttheta=[];
    Rtesttorque=[];
```

```
    Rvaltheta=[];
    Rvaltorque=[];

    Lvaltheta=[];
    Lvaltorque=[];

    Rtargtheta=[];
    Ltargtheta=[];

    Rtargtorque=[];
    Ltargtorque=[];



    FolderName = sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\knee angle emg\\Participant%i\\',i);
    File = fullfile(FolderName, FileName);
    participantData = load(File);
    participantData = struct2cell(participantData);
    velocities = participantData{1};
    velocities=struct2cell(velocities);
    for v = 1:length(velocities) %all walking velocities
        v1=velocities{v}; %current walking velocity
        v1=struct2cell(v1);
        R=v1{1};
        R=struct2cell(R);
        R_emg=R{2};
        R_stridetime=R{5};

        R_emg_trials=R_emg(:,1);
        Rtrials=[];
        for rT = 1:length(R_emg_trials); % right emg data indicies
            rT;
            trialnum=R_emg_trials(rT);
            trialnum=(trialnum{1,1}{1,1});
            k = strsplit(trialnum,':');
            %k=cell2mat(k);;
            k=(k(2));
            k=cell2mat(k);
            k=strtrim(k);
            k=str2num(k);
            Rtrials=[Rtrials, k]; % right emg data indicies array
        end
        Rtrials;

        R_theta=R{1};
        R_torque=R{8};
        for rtrial = Rtrials;

            stridetime1=R_stridetime(rtrial,2);
            stridetime1=table2array(stridetime1);

            stridetime2=R_stridetime(rtrial,3);
            stridetime2=table2array(stridetime2);
```

```
removeentries1=round((delay/stridetime1));
removeentries2=round(delay/stridetime2);

theta=R_theta(rtrial,2);
theta=table2array(theta{1});

theta1=theta(:,7);
theta1=theta1';

traintheta1=theta1(1:end-removeentries1);
testtheta1=theta1(removeentries1+1:end);

theta2=theta(:,10);
theta2=theta2';

traintheta2=theta2(1:end-removeentries2);
testtheta2=theta2(removeentries2+1:end);
if rtrial == Rtrials(end)
    for element =1:length(traintheta1);
        Rvaltheta(end+1)=traintheta1(element);
    end
    for element =1:length(traintheta2);
        Rvaltheta(end+1)=traintheta2(element);
    end
    for element =1:length(testtheta1);
        Rtargtheta(end+1)=testtheta1(element);
    end
    for element =1:length(testtheta2);
        Rtargtheta(end+1)=testtheta2(element);
    end
else
    for element =1:length(traintheta1);
        Rtraintheta(end+1)=traintheta1(element);
    end
    for element =1:length(traintheta2);
        Rtraintheta(end+1)=traintheta2(element);
    end
    for element =1:length(testtheta1);
        Rtesttheta(end+1)=testtheta1(element);
    end
    for element =1:length(testtheta2);
        Rtesttheta(end+1)=testtheta2(element);
    end
end


torque=R_torque(rtrial,2);
torque=table2array(torque{1});

torque1=torque(:,7);
torque1=torque1';

traintorque1=torque1(1:end-removeentries1);
testtorque1=torque1(removeentries1+1:end);
```

```
            torque2=torque(:,10);
            torque2=torque2';

            traintorque2=torque2(1:end-removeentries2);
            testtorque2=torque2(removeentries2+1:end);

            if rtrial == Rtrials(end)
                for element =1:length(traintorque1);
                    Rvaltorque(end+1)=traintorque1(element);
                end
                for element =1:length(traintorque2);
                    Rvaltorque(end+1)=traintorque2(element);
                end
                for element =1:length(testtorque1);
                    Rtargtorque(end+1)=testtorque1(element);
                end
                for element =1:length(testtorque2);
                    Rtargtorque(end+1)=testtorque2(element);
                end
            else
                for element =1:length(traintorque1);
                    Rtraintorque(end+1)=traintorque1(element);
                end
                for element =1:length(traintorque2);
                    Rtraintorque(end+1)=traintorque2(element);
                end
                for element =1:length(testtorque1);
                    Rtesttorque(end+1)=testtorque1(element);
                end
                for element =1:length(testtorque2);
                    Rtesttorque(end+1)=testtorque2(element);
                end
            end

        end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        L=v1{2};
        L=struct2cell(L);
        L_emg=L{2};
        L_stridetime=L{5};

        L_emg_trials=L_emg(:,1);
        Ltrials=[];
        for lT = 1:length(L_emg_trials); % right emg data indicies
            lT;
            trialnum=L_emg_trials(lT);
            trialnum=(trialnum{1,1}{1,1});
            k = strsplit(trialnum,':');
```

```
    %k=cell2mat(k);
    k=(k(2));
    k=cell2mat(k);
    k=strtrim(k);
    k=str2num(k);
    Ltrials=[Ltrials, k] ;% right emg data indicies array
end

L_theta=L{1};
L_torque=L{8};
for ltrial = Ltrials;

    stridetime1=L_stridetime(ltrial,2);
    stridetime1=table2array(stridetime1);

    removeentries1=round((delay/stridetime1));

    theta=L_theta(ltrial,2);
    theta=table2array(theta{1});

    theta1=theta(:,1);
    theta1=theta1';

    traintheta1=theta1(1:end-removeentries1);
    testtheta1=theta1(removeentries1+1:end);
    if ltrial== Ltrials(end)

        for element =1:length(traintheta1);
            Lvaltheta(end+1)=traintheta1(element);
        end
        for element =1:length(testtheta1);
            Ltargtheta(end+1)=testtheta1(element);
        end
    else
        for element =1:length(traintheta1);
            Ltraintheta(end+1)=traintheta1(element);
        end
        for element =1:length(testtheta1);
            Ltesttheta(end+1)=testtheta1(element);
        end
    end

    torque=L_torque(ltrial,2);
    torque=table2array(torque{1});

    torque1=torque(:,1);
    torque1=torque1';

    traintorque1=torque1(1:end-removeentries1);
    testtorque1=torque1(removeentries1+1:end);

    if ltrial== Ltrials(end)
        for element =1:length(traintorque1);
            Lvaltorque(end+1)=traintorque1(element);
        end
```

```
                for element =1:length(testtorque1);
                    Ltargtorque(end+1)=testtorque1(element);
                end
            else
                for element =1:length(traintorque1);
                    Ltraintorque(end+1)=traintorque1(element);
                end
                for element =1:length(testtorque1);
                    Ltesttorque(end+1)=testtorque1(element);
                end
            end

        end
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LEG=leg
if LEG== "L"
    Input=[Ltraintheta;Ltraintorque];
    Target=[Ltesttheta;Ltesttorque];
    validation=[Lvaltheta;Lvaltorque];
    validationTarget=[Ltargtheta;Ltargtorque];
    time=[1:length(validation)];
else
    Input=[Rtraintheta;Rtraintorque];
    Target=[Rtesttheta;Rtesttorque];
    validation=[Rvaltheta;Rvaltorque];
    validationTarget=[Rtargtheta;Rtargtorque];
    time=[1:length(validation)];
end
```

Appendix 2: Batch Network Training/ testing Code

Appendix 2.1: Knee Network Training/Testing Code:

```
clc
%clear all
%%%%%%ntstool
delay=[50,100,150];
LEG=["R","L"];
sub=[1,2,3,8,9,10,11,12,13,14];
row=1;
for del = delay
    for leg = LEG
        row=1;
        for subject = sub
            [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(del,
leg,subject);
            Input=KInput;
            Target=KTarget;
```

```
            done =trainnetwork(Input,Target,del, leg,subject);
            A=Kvalidation;
            B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2)];
            network=sprintf('P%d_%c_K_Pred_%d', subject, leg, del);
            network=str2func(network);
            [output,xfin]=network(A,B);
            subject
            leg
            del
            KThetaRMSE= sqrt(mse(KvalidationTarget(1,:),output(1,:)))
            KTorqueRMSE=sqrt(mse(KvalidationTarget(2,:),output(2,:)))
            row=row+1;
        end
    end
end
```

Appendix 2.2: Knee-Ankle Network Training/Testing Code:

```
clc
%clear all
%%%%%%ntstool
delay=[50,100,150];
LEG=["R","L"];
sub=[10,11,12,13,14];
row=1;
for del = delay
    for leg = LEG
        row=1;
        for subject = sub
            [AInput,ATarget,Avalidation,AvalidationTarget,Atime]=ankleExtract(del,
leg,subject);
            [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(del,
leg,subject);
            Input=[KInput;AInput];
            Target=[KTarget;ATarget];
            %done =trainnetwork_KA(Input,Target,del, leg,subject);
            A=[Kvalidation;Avalidation];

B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2);A(5,1:2);A(6,1:2);A(7,1:2);A(8,1:2)];
            network=sprintf('P%d_%c_KA_Pred_%d', subject, leg, del);
            network=str2func(network);
            [output,xfin]=network(A,B);
            subject
            leg
            del
            KThetaRMSE= sqrt(mse(KvalidationTarget(1,:),output(1,:)))
            KTorqueRMSE=sqrt(mse(KvalidationTarget(2,:),output(2,:)))
            AThetaRMSE= sqrt(mse(AvalidationTarget(1,:),output(3,:)))
            ATorqueRMSE=sqrt(mse(AvalidationTarget(2,:),output(4,:)))
            row=row+1;
        end
    end
end
```

Appendix 2.3: Knee-Ankle-Hip Network Training/Testing Code:

```
clc
%clear all
%%%%%%ntstool
delay=[50,100,150];
LEG=["R","L"];
sub=[10,11,12,13,14];
row=1;
for del = delay
    for leg = LEG
        row=1;
        for subject = sub
            [AInput,ATarget,Avalidation,AvalidationTarget,Atime]=ankleExtract(del,
leg,subject);
            [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(del,
leg,subject);
            [HInput,HTarget,Hvalidation,HvalidationTarget,Htime]=hipExtract(del,
leg,subject);
            Input=[KInput;AInput;HInput];
            Target=[KTarget;ATarget;HTarget];

            %done =trainnetwork_KAH_noemg(Input,Target,del, leg,subject);
            A=[Kvalidation;Avalidation;Hvalidation];

B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2);A(5,1:2);A(6,1:2);A(7,1:2);A(8,1:2);A(9,1:2);A
(10,1:2)];
            network=sprintf('P%d_%c_KAH_Pred_%d', subject, leg, del);
            network=str2func(network);
            [output,xfin]=network(A,B);
            subject
            leg
            del
            KThetaRMSE= sqrt(mse(KvalidationTarget(1,:),output(1,:)))
            KTorqueRMSE=sqrt(mse(KvalidationTarget(2,:),output(2,:)))
            AThetaRMSE= sqrt(mse(AvalidationTarget(1,:),output(3,:)))
            ATorqueRMSE=sqrt(mse(AvalidationTarget(2,:),output(4,:)))
            HThetaRMSE= sqrt(mse(HvalidationTarget(1,:),output(5,:)))
            HTorqueRMSE=sqrt(mse(HvalidationTarget(2,:),output(6,:)))
            row=row+1;
        end
    end
end
```

Appendix 2.4: Knee-Ankle-Hip- No EMG Network Training/Testing Code:

```
clc
%clear all
%%%%%%ntstool
delay=[50,100,150];
LEG=["R","L"];
sub=[1,2,3,8,9,10,11,12,13,14];
row=1;
```

```
for del = delay
    for leg = LEG
        row=1;
        for subject = sub
            [AInput,ATarget,Avalidation,AvalidationTarget,Atime]=ankleExtract(del,
leg,subject);
            [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(del,
leg,subject);
            [HInput,HTarget,Hvalidation,HvalidationTarget,Htime]=hipExtract(del,
leg,subject);
            Input=[KInput(3:4,:);AInput(3:4,:);HInput];
            Target=[KTarget;ATarget;HTarget];

            %done =trainnetwork_KAH_noemg(Input,Target,del, leg,subject);
            A=[Kvalidation(3:4,:);Avalidation(3:4,:);Hvalidation];
            B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2);A(5,1:2);A(6,1:2)];
            network=sprintf('P%d_%c_KAH_Pred_%d_noemg', subject, leg, del);
            network=str2func(network);
            [output,xfin]=network(A,B);
            subject
            leg
            del
            KThetaRMSE= sqrt(mse(KvalidationTarget(1,:),output(1,:)))
            KTorqueRMSE=sqrt(mse(KvalidationTarget(2,:),output(2,:)))
            AThetaRMSE= sqrt(mse(AvalidationTarget(1,:),output(3,:)))
            ATorqueRMSE=sqrt(mse(AvalidationTarget(2,:),output(4,:)))
            HThetaRMSE= sqrt(mse(HvalidationTarget(1,:),output(5,:)))
            HTorqueRMSE=sqrt(mse(HvalidationTarget(2,:),output(6,:)))
            row=row+1;
        end
    end
end
```

Appendix 4: Single Network Training Function

Appendix 4.1: Knee Network Training Function:

```
function[done]=trainnetwork(Input,Target,delay, LEG,sub)
% Solve an Input-Output Time-Series Problem with a Time Delay Neural Network
% Script generated by Neural Time Series app.
% Created 17-Jun-2022 03:57:37
%
% This script assumes these variables are defined:
%
%   Input - input time series.
%   Target - target time series.

X = tonndata(Input,true,false);
T = tonndata(Target,true,false);

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
```

```
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  %bayesian regularization.

% Create a Time Delay Network
inputDelays = 1:2;
hiddenLayerSize = 10;
net = timedelaynet(inputDelays,hiddenLayerSize,trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular network,
% shifting time by the minimum amount to fill input states and layer
% states. Using PREPARETS allows you to keep your original time series data
% unchanged, while easily customizing it for networks with differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,X,T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);
pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\kneeModelsv2\\P%d_%c_K_Pred_%d',sub, LEG,delay);
genFunction(net,pathname,'MatrixOnly','yes')
% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep early.
% The original network returns predicted y(t+1) at the same time it is
% given x(t+1). For some applications such as decision making, it would
% help to have predicted y(t+1) once x(t) is available, but before the
% actual y(t+1) occurs. The network can be made to return its output a
% timestep early by removing one delay so that its minimal tap delay is now
% 0 instead of 1. The new network returns the same outputs as the original
% network, but outputs are shifted left one timestep.
nets = removedelay(net);
%netname=sprintf('P%d_%c_Pred_%d',sub, LEG,delay)
```

```
%pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\kneeModelsv2\\P%d_%c_K_Pred_%d',sub, LEG,delay)
%genFunction(nets,pathname,'MatrixOnly','yes')
nets.name = [net.name ' - Predict One Step Ahead'];
%view(nets)
[xs,xis,ais,ts] = preparets(nets,X,T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys);
done =1;
```

Appendix 4.2: Knee-Ankle Network Training Function:

```
function[done]=trainnetwork_KA(Input,Target,delay, LEG,sub)
% Solve an Input-Output Time-Series Problem with a Time Delay Neural Network
% Script generated by Neural Time Series app.
% Created 17-Jun-2022 03:57:37
%
% This script assumes these variables are defined:
%
%   Input - input time series.
%   Target - target time series.

X = tonndata(Input,true,false);
T = tonndata(Target,true,false);

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  %bayesian regularization.

% Create a Time Delay Network
inputDelays = 1:2;
hiddenLayerSize = 10;
net = timedelaynet(inputDelays,hiddenLayerSize,trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular network,
% shifting time by the minimum amount to fill input states and layer
% states. Using PREPARETS allows you to keep your original time series data
% unchanged, while easily customizing it for networks with differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,X,T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);
```

```
pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--RESEARCH\\knee-
ankle-models\\P%d_%c_KA_Pred_%d',sub, LEG,delay);
genFunction(net,pathname,'MatrixOnly','yes')
% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep early.
% The original network returns predicted y(t+1) at the same time it is
% given x(t+1). For some applications such as decision making, it would
% help to have predicted y(t+1) once x(t) is available, but before the
% actual y(t+1) occurs. The network can be made to return its output a
% timestep early by removing one delay so that its minimal tap delay is now
% 0 instead of 1. The new network returns the same outputs as the original
% network, but outputs are shifted left one timestep.
nets = removedelay(net);
%netname=sprintf('P%d_%c_Pred_%d',sub, LEG,delay)
%pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\kneeModelsv2\\P%d_%c_K_Pred_%d',sub, LEG,delay)
%genFunction(nets,pathname,'MatrixOnly','yes')
nets.name = [net.name ' - Predict One Step Ahead'];
%view(nets)
[xs,xis,ais,ts] = preparets(nets,X,T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys);
done =1;
```

Appendix 4.3: Knee-Ankle-Hip Network Training Function:

```
function[done]=trainnetwork_KA(Input,Target,delay, LEG,sub)
% Solve an Input-Output Time-Series Problem with a Time Delay Neural Network
% Script generated by Neural Time Series app.
% Created 17-Jun-2022 03:57:37
%
% This script assumes these variables are defined:
%
%   Input - input time series.
%   Target - target time series.
```

```
X = tonndata(Input,true,false);
T = tonndata(Target,true,false);

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  %bayesian regularization.

% Create a Time Delay Network
inputDelays = 1:2;
hiddenLayerSize = 10;
net = timedelaynet(inputDelays,hiddenLayerSize,trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular network,
% shifting time by the minimum amount to fill input states and layer
% states. Using PREPARETS allows you to keep your original time series data
% unchanged, while easily customizing it for networks with differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,X,T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);
pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--RESEARCH\\knee-
ankle-hip-models\\P%d_%c_KAH_Pred_%d',sub, LEG,delay);
genFunction(net,pathname,'MatrixOnly','yes')
% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep early.
% The original network returns predicted y(t+1) at the same time it is
% given x(t+1). For some applications such as decision making, it would
```

```
% help to have predicted y(t+1) once x(t) is available, but before the
% actual y(t+1) occurs. The network can be made to return its output a
% timestep early by removing one delay so that its minimal tap delay is now
% 0 instead of 1. The new network returns the same outputs as the original
% network, but outputs are shifted left one timestep.
nets = removedelay(net);
%netname=sprintf('P%d_%c_Pred_%d',sub, LEG,delay)
%pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\kneeModelsv2\\P%d_%c_K_Pred_%d',sub, LEG,delay)
%genFunction(nets,pathname,'MatrixOnly','yes')
nets.name = [net.name ' - Predict One Step Ahead'];
%view(nets)
[xs,xis,ais,ts] = preparets(nets,X,T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys);
done =1;
```

Appendix 4.4: Knee-Ankle-Hip- No EMG Network Training Function:

```
function[done]=trainnetwork_KA(Input,Target,delay, LEG,sub)
% Solve an Input-Output Time-Series Problem with a Time Delay Neural Network
% Script generated by Neural Time Series app.
% Created 17-Jun-2022 03:57:37
%
% This script assumes these variables are defined:
%
%   Input - input time series.
%   Target - target time series.

X = tonndata(Input,true,false);
T = tonndata(Target,true,false);

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  %bayesian regularization.

% Create a Time Delay Network
inputDelays = 1:2;
hiddenLayerSize = 10;
net = timedelaynet(inputDelays,hiddenLayerSize,trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular network,
% shifting time by the minimum amount to fill input states and layer
% states. Using PREPARETS allows you to keep your original time series data
% unchanged, while easily customizing it for networks with differing
% numbers of delays, with open loop or closed loop feedback modes.
[x,xi,ai,t] = preparets(net,X,T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
```

```
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);
pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--RESEARCH\\knee-
ankle-hip-models-noemg\\P%d_%c_KAH_Pred_%d_noemg',sub, LEG,delay);
genFunction(net,pathname,'MatrixOnly','yes')
% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep early.
% The original network returns predicted y(t+1) at the same time it is
% given x(t+1). For some applications such as decision making, it would
% help to have predicted y(t+1) once x(t) is available, but before the
% actual y(t+1) occurs. The network can be made to return its output a
% timestep early by removing one delay so that its minimal tap delay is now
% 0 instead of 1. The new network returns the same outputs as the original
% network, but outputs are shifted left one timestep.
nets = removedelay(net);
%netname=sprintf('P%d_%c_Pred_%d',sub, LEG,delay)
%pathname=sprintf('E:\\documents\\matlab scripts\\knee engle emg--
RESEARCH\\kneeModelsv2\\P%d_%c_K_Pred_%d',sub, LEG,delay)
%genFunction(nets,pathname,'MatrixOnly','yes')
nets.name = [net.name ' - Predict One Step Ahead'];
%view(nets)
[xs,xis,ais,ts] = preparets(nets,X,T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys);
done =1;
```

Appendix 5: Principal Component Analysis Code

Appendix 5.1: Principal Component Analysis Code for Knee-Ankle-Hip Networks

```
clc
%clear all
%%%%%%ntstool
delay=[0];
LEG=["R"];
sub=[1];
row=1;
for del = delay
    for leg = LEG
        row=1;
        for subject = sub
            [AInput,ATarget,Avalidation,AvalidationTarget,Atime]=ankleExtract(del,
leg,subject);
            [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(del,
leg,subject);
            [HInput,HTarget,Hvalidation,HvalidationTarget,Htime]=hipExtract(del,
leg,subject);
            Input=[KInput;AInput;HInput]';
            [coeff,score,latent,tsquared,explained] =pca(Input);
            coeff
            explained

        end
    end
end
```

Appendix 6: Network Grand Average Error Code

Appendix 6.1: Knee Network Grand Average Error Code:

```
clc
%clear all
%%%%%%ntstool
delay=150
LEG="L"
sub=[1,2,3,8,9,10,11,12,13,14]
futureangle=[]
futuretorque=[]
predictedangle=[]
predictedtorque=[]
row=1
for subject = sub
    futureangle=[]
    futuretorque=[]
    predictedangle=[]
    predictedtorque=[]
    subject
    [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(delay,
LEG,subject);
```

```
    Input=KInput;
    Target=KTarget;
    A=Kvalidation;
    B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2)];
    angle=KvalidationTarget(1,:);
    torque=KvalidationTarget(2,:);
    futureangle(row,:)=angle;
    futuretorque(row,:)=torque;
    network=sprintf('P%d_%c_K_Pred_%d', subject, LEG, delay)
    network=str2func(network)
    [output,xfin]=network(A,B);
    predictedangle(row,:)=output(1,:);
    predictedtorque(row,:)=output(2,:);
    row=row+1;
end
grndavg_angle=mean(futureangle);
grndavg_torque=mean(futuretorque);
grndavg_predictedangle=mean(predictedangle);
grndavg_predictedtorque=mean(predictedtorque);
anglediff=abs(grndavg_angle-grndavg_predictedangle);
torquediff=(grndavg_torque-grndavg_predictedtorque);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1)
subplot(2,1,1)
plot(Ktime,grndavg_predictedangle);
hold on;
plot(Ktime,grndavg_angle);
legend('predicted theta','actual theta')
ylabel('Joint Angle (degrees)')
subplot(2,1,2)

plot(Ktime,grndavg_predictedtorque);
hold on;
plot(Ktime,grndavg_torque);
hold off;
legend('predicted T','actual T')
xlabel('Normalized Stride Time (1001 points/stride)')
ylabel('Torque (N-m)')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)
subplot(2,1,1)
plot(Ktime,grndavg_angle,'color','r');
hold on;
plot(Ktime,grndavg_angle+anglediff);
patch([Ktime fliplr(Ktime)], [grndavg_angle fliplr(grndavg_angle+anglediff)], 'r')
ylabel('Joint Angle (degrees)')
hold off

subplot(2,1,2)
plot(Ktime,grndavg_torque,'color','r');
hold on;
plot(Ktime,grndavg_torque+torquediff);
```

```
patch([Ktime fliplr(Ktime)], [grndavg_torque fliplr(grndavg_torque+torquediff)], 'r')
xlabel('Normalized Stride Time (1001 points/stride)')
ylabel('Torque (N-m)')
hold off;
```

Appendix 6.2: Knee-Ankle Network Grand Average Error Code:

```
clc
clear all
%%%%%%ntstool
delay=150
LEG="L"
sub=[1,2,3,8,9,10,11,12,13,14]
row=1
for subject = sub
    future_ankle_angle=[]
    future_ankle_torque=[]
    predicted_ankle_angle=[]
    predicted_ankle_torque=[]
    future_knee_angle=[]
    future_knee_torque=[]
    predicted_knee_angle=[]
    predicted_knee_torque=[]
    subject
    [AInput,ATarget,Avalidation,AvalidationTarget,Atime]=ankleExtract(delay,
LEG,subject);
    [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(delay,
LEG,subject);
    Input=[KInput;AInput];
    Target=[KTarget;ATarget];

    %done =trainnetwork_KAH_noemg(Input,Target,del, leg,subject);
    A=[Kvalidation;Avalidation];
    B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2);A(5,1:2);A(6,1:2);A(7,1:2);A(8,1:2)];
    %angle=KvalidationTarget(1,:);
    %torque=KvalidationTarget(2,:);
    future_ankle_angle(row,:)=AvalidationTarget(1,:);
    future_ankle_torque(row,:)=AvalidationTarget(2,:);
    future_knee_angle(row,:)=KvalidationTarget(1,:);
    future_knee_torque(row,:)=KvalidationTarget(2,:);
    network=sprintf('P%d_%c_KA_Pred_%d', subject, LEG, delay)
    network=str2func(network)
    [output,xfin]=network(A,B);
    predicted_knee_angle(row,:)=output(1,:);
    predicted_knee_torque(row,:)=output(2,:);
    predicted_ankle_angle(row,:)= output(3,:);
    predicted_ankle_torque(row,:)=output(4,:);
    row=row+1;
end
grndavg_knee_angle=mean(future_knee_angle);
grndavg_knee_torque=mean(future_knee_torque);
grndavg_ankle_angle=mean(future_ankle_angle);
grndavg_ankle_torque=mean(future_ankle_torque);
```

```
grndavg_predicted_knee_angle=mean(predicted_knee_angle);
grndavg_predicted_knee_torque=mean(predicted_knee_torque);
grndavg_predicted_ankle_angle=mean(predicted_ankle_angle);
grndavg_predicted_ankle_torque=mean(predicted_ankle_torque);

knee_anglediff=abs(grndavg_knee_angle-grndavg_predicted_knee_angle);
ankle_anglediff=abs(grndavg_ankle_angle-grndavg_predicted_ankle_angle);


knee_torquediff=abs(grndavg_knee_torque-grndavg_predicted_knee_torque);
ankle_torquediff=abs(grndavg_ankle_torque-grndavg_predicted_ankle_torque);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1)
subplot(4,1,1)
plot(Ktime,grndavg_knee_angle,'color','r');
hold on;
plot(Ktime,grndavg_knee_angle+knee_anglediff);
patch([Ktime fliplr(Ktime)], [grndavg_knee_angle
fliplr(grndavg_knee_angle+knee_anglediff)], 'r')
ylabel('Joint Angle (degrees)')
legend('Knee Angle Error')
hold off

subplot(4,1,2)
plot(Ktime,grndavg_knee_torque,'color','r');
hold on;
plot(Ktime,grndavg_knee_torque+knee_torquediff);
patch([Ktime fliplr(Ktime)], [grndavg_knee_torque
fliplr(grndavg_knee_torque+knee_torquediff)], 'r')
ylabel('Torque (N-m)')
legend('Knee Torque Error')
%xlabel('Normalized Stride Time (1001 points/stride)')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%figure(2)
subplot(4,1,3)
plot(Atime,grndavg_ankle_angle,'color','r');
hold on;
plot(Atime,grndavg_ankle_angle+ankle_anglediff);
patch([Atime fliplr(Atime)], [grndavg_ankle_angle
fliplr(grndavg_ankle_angle+ankle_anglediff)], 'r')
ylabel('Joint Angle (degrees)')
legend('Ankle Angle Error')
hold off

subplot(4,1,4)
plot(Atime,grndavg_ankle_torque,'color','r');
hold on;
plot(Atime,grndavg_ankle_torque+ankle_torquediff);
patch([Atime fliplr(Atime)], [grndavg_ankle_torque
fliplr(grndavg_ankle_torque+ankle_torquediff)], 'r')
```

```
ylabel('Torque (N-m)')
legend('Ankle Torque Error')
xlabel('Normalized Stride Time (1001 points/stride)')
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix 6.3: Knee-Ankle-Hip Network Grand Average Error Code:

```
clc
clear all
%%%%%%ntstool
delay=150
LEG="L"
sub=[1,2,3,8,9,10,11,12,13,14]
row=1
for subject = sub
    future_ankle_angle=[]
    future_ankle_torque=[]
    predicted_ankle_angle=[]
    predicted_ankle_torque=[]
    future_knee_angle=[]
    future_knee_torque=[]
    predicted_knee_angle=[]
    predicted_knee_torque=[]
    future_hip_angle=[]
    future_hip_torque=[]
    predicted_hip_angle=[]
    predicted_hip_torque=[]
    subject
    [AInput,ATarget,Avalidation,AvalidationTarget,Atime]=ankleExtract(delay,
LEG,subject);
    [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(delay,
LEG,subject);
    [HInput,HTarget,Hvalidation,HvalidationTarget,Htime]=hipExtract(delay,
LEG,subject);
    Input=[KInput;AInput;HInput];
    Target=[KTarget;ATarget;HTarget];

    %done =trainnetwork_KAH_noemg(Input,Target,del, leg,subject);
    A=[Kvalidation;Avalidation;Hvalidation];

B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2);A(5,1:2);A(6,1:2);A(7,1:2);A(8,1:2);A(9,1:2);A
(10,1:2)];
    %angle=KvalidationTarget(1,:);
    %torque=KvalidationTarget(2,:);
    future_ankle_angle(row,:)=AvalidationTarget(1,:);
    future_ankle_torque(row,:)=AvalidationTarget(2,:);
    future_knee_angle(row,:)=KvalidationTarget(1,:);
    future_knee_torque(row,:)=KvalidationTarget(2,:);
    future_hip_angle(row,:)=HvalidationTarget(1,:);
    future_hip_torque(row,:)=HvalidationTarget(2,:);
    network=sprintf('P%d_%c_KAH_Pred_%d', subject, LEG, delay)
    network=str2func(network)
```

```
    [output,xfin]=network(A,B);
    predicted_knee_angle(row,:)=output(1,:);
    predicted_knee_torque(row,:)=output(2,:);
    predicted_ankle_angle(row,:)= output(3,:);
    predicted_ankle_torque(row,:)=output(4,:);
    predicted_hip_angle(row,:)= output(5,:);
    predicted_hip_torque(row,:)=output(6,:);
    row=row+1;
end
grndavg_knee_angle=mean(future_knee_angle);
grndavg_knee_torque=mean(future_knee_torque);
grndavg_ankle_angle=mean(future_ankle_angle);
grndavg_ankle_torque=mean(future_ankle_torque);
grndavg_hip_angle=mean(future_hip_angle);
grndavg_hip_torque=mean(future_hip_torque);

grndavg_predicted_knee_angle=mean(predicted_knee_angle);
grndavg_predicted_knee_torque=mean(predicted_knee_torque);
grndavg_predicted_ankle_angle=mean(predicted_ankle_angle);
grndavg_predicted_ankle_torque=mean(predicted_ankle_torque);
grndavg_predicted_hip_angle=mean(predicted_hip_angle);
grndavg_predicted_hip_torque=mean(predicted_hip_torque);

knee_anglediff=abs(grndavg_knee_angle-grndavg_predicted_knee_angle);
ankle_anglediff=abs(grndavg_ankle_angle-grndavg_predicted_ankle_angle);
hip_anglediff=abs(grndavg_hip_angle-grndavg_predicted_hip_angle);

knee_torquediff=abs(grndavg_knee_torque-grndavg_predicted_knee_torque);
ankle_torquediff=abs(grndavg_ankle_torque-grndavg_predicted_ankle_torque);
hip_toprquediff=abs(grndavg_hip_torque-grndavg_predicted_hip_torque);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1)
subplot(6,1,1)
plot(Ktime,grndavg_knee_angle,'color','r');
hold on;
plot(Ktime,grndavg_knee_angle+knee_anglediff);
patch([Ktime fliplr(Ktime)], [grndavg_knee_angle
fliplr(grndavg_knee_angle+knee_anglediff)], 'r')
ylabel('Joint Angle (degrees)')
legend('Knee Angle Error')
hold off

subplot(6,1,2)
plot(Ktime,grndavg_knee_torque,'color','r');
hold on;
plot(Ktime,grndavg_knee_torque+knee_torquediff);
patch([Ktime fliplr(Ktime)], [grndavg_knee_torque
fliplr(grndavg_knee_torque+knee_torquediff)], 'r')
ylabel('Torque (N-m)')
xlabel('Normalized Stride Time (1001 points/stride)')
legend('Knee Torque Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%figure(1)
subplot(6,1,3)
plot(Atime,grndavg_ankle_angle,'color','r');
hold on;
plot(Atime,grndavg_ankle_angle+ankle_anglediff);
patch([Atime fliplr(Atime)], [grndavg_ankle_angle
fliplr(grndavg_ankle_angle+ankle_anglediff)], 'r')
ylabel('Joint Angle (degrees)')
legend('Ankle Angle Error')
hold off

subplot(6,1,4)
plot(Atime,grndavg_ankle_torque,'color','r');
hold on;
plot(Atime,grndavg_ankle_torque+ankle_torquediff);
patch([Atime fliplr(Atime)], [grndavg_ankle_torque
fliplr(grndavg_ankle_torque+ankle_torquediff)], 'r')
ylabel('Torque (N-m)')
xlabel('Normalized Stride Time (1001 points/stride)')
legend('Ankle Torque Error')
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%figure(1)
subplot(6,1,5)
plot(Htime,grndavg_hip_angle,'color','r');
hold on;
plot(Htime,grndavg_hip_angle+hip_anglediff);
patch([Htime fliplr(Htime)], [grndavg_hip_angle
fliplr(grndavg_hip_angle+hip_anglediff)], 'r')
ylabel('Joint Angle (degrees)')
legend('Hip Angle Error')
hold off

subplot(6,1,6)
plot(Htime,grndavg_hip_torque,'color','r');
hold on;
plot(Htime,grndavg_hip_torque+hip_toprquediff);
patch([Htime fliplr(Htime)], [grndavg_hip_torque
fliplr(grndavg_hip_torque+hip_toprquediff)], 'r')
ylabel('Torque (N-m)')
xlabel('Normalized Stride Time (1001 points/stride)')
legend('Hip Torque Error')
hold off;
```

Appendix 6.4: Knee-Ankle-Hip- No EMG Grand Average Error Code:

```
clc
%clear all
%%%%%%ntstool
delay=100
LEG="R"
```

```
sub=[1,2,3,8,9,10,11,12,13,14]
futureangle=[]
futuretorque=[]
predictedangle=[]
predictedtorque=[]
row=1
for subject = sub
    subject
    [KInput,KTarget,Kvalidation,KvalidationTarget,Ktime]=kneeExtract(delay, LEG,sub);
    A=Kvalidation;
    B=[A(1,1:2);A(2,1:2);A(3,1:2);A(4,1:2)];
    angle=KvalidationTarget(1,:);
    torque=KvalidationTarget(2,:);
    futureangle(row,:)=angle;
    futuretorque(row,:)=torque;
    network=sprintf('P%d_%c_Pred_%d', subject, LEG, delay)
    network=str2func(network)
    [output,xfin]=network(A,B);
    predictedangle(row,:)=output(1,:);
    predictedtorque(row,:)=output(2,:);
    row=row+1;
end
grndavg_angle=mean(futureangle);
grndavg_torque=mean(futuretorque);
grndavg_predictedangle=mean(predictedangle);
grndavg_predictedtorque=mean(predictedtorque);
anglediff=abs(grndavg_angle-grndavg_predictedangle);
torquediff=(grndavg_torque-grndavg_predictedtorque);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1)
subplot(2,1,1)
plot(Ktime,grndavg_predictedangle);
hold on;
plot(Ktime,grndavg_angle);
legend('predicted theta','actual theta')
ylabel('Joint Angle (degrees)')
subplot(2,1,2)

plot(Ktime,grndavg_predictedtorque);
hold on;
plot(Ktime,grndavg_torque);
hold off;
legend('predicted T','actual T')
xlabel('Normalized Stride Time (1001 points/stride)')
ylabel('Torque (N-m)')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)
subplot(2,1,1)
plot(Ktime,grndavg_angle,'color','r');
hold on;
plot(Ktime,grndavg_angle+anglediff);
patch([Ktime fliplr(Ktime)], [grndavg_angle fliplr(grndavg_angle+anglediff)], 'r')
```

```
hold off

subplot(2,1,2)
plot(Ktime,grndavg_torque,'color','r');
hold on;
plot(Ktime,grndavg_torque+torquediff);
patch([Ktime fliplr(Ktime)], [grndavg_torque fliplr(grndavg_torque+torquediff)], 'r')
hold off;
```

LIST OF REFRENCES

[1]Center For Disease Control, https://www.cdc.gov/media/releases/2018/p0816-disability.html

[2] Coker, Jordan, Howard Chen, Mark C. Schall Jr., Sean Gallagher, and Michael Zabala. 2021. "EMG and Joint Angle-Based Machine Learning to Predict Future Joint Angles at the Knee" Sensors 21, no. 11: 3622. https://doi.org/10.3390/s21113622

[3] Moreira, L., Figueiredo, J., Fonseca, P. et al. Lower limb kinematic, kinetic, and EMG data from young healthy humans during walking at controlled speeds. Sci Data 8, 103 (2021). https://doi.org/10.1038/s41597-021-00881-3

[4]Majidi Fard Vatan, H., Nefti-Meziani, S., Davis, S. *et al.* A review: A Comprehensive Review of Soft and Rigid Wearable Rehabilitation and Assistive Devices with a Focus on the Shoulder Joint. *J Intell Robot Syst* **102,** 9 (2021). https://doi.org/10.1007/s10846-021-01353-x

[5] Intisar, Muhatasim, Mohammad Monirujjaman Khan, Mehedi Masud, and Mohammad Shorfuzzaman. "Development of A Low-Cost Exoskeleton for Rehabilitation and Mobility." In *Intelligent Automation & Soft Computing*, vol. 31, no. 1, pp. 101-115. Tech Science Press, 2021.

[6]Franco Molteni, Giulio Gasperini, Giovanni Cannaviello, Eleonora Guanziroli, Exoskeleton and End-Effector Robots for Upper and Lower Limbs Rehabilitation: Narrative Review, PM&R, Volume 10, Issue 9, Supplement 2, 2018, Pages S174-S188, ISSN 1934-1482, https://doi.org/10.1016/j.pmrj.2018.06.005.

[7] Setiawan, Joga D., Mochammad Ariyanto, Sri Nugroho, M. Munadi, and Rifky Ismail. "A soft exoskeleton glove incorporating motor-tendon actuator for hand movements assistance." *Int. Rev. Autom. Control* 13 (2020): 1-11.

[8] Farfán, F. D., Politti, J. C. & Felice, C. J. Evaluation of EMG processing techniques using Information Theory. *Biomed. Eng. Online* **9** (2010).

[9] Lopes, M. C. S., Costa, M. C. A., & Ebecken, N. F. F. (1998). A comparison of methods for customer classification. *Transactions on Information and Communications Technologies*, *19*. https://doi.org/10.2495/DATA980251

[10] Ma, X.; Liu, Y.; Song, Q.; Wang, C. Continuous Estimation of Knee Joint Angle Based on Surface Electromyography Using a Long Short-Term Memory Neural Network and Time-Advanced Feature. Sensors **2020**, 20, 4966.

[11] Marzieh Mostafavizadeh Ardestani, Xuan Zhang, Ling Wang, Qin Lian, Yaxiong Liu, Jiankang He, Dichen Li, Zhongmin Jin, Human lower extremity joint moment prediction: A wavelet neural network approach, Expert Systems with Applications, Volume 41, Issue 9, 2014, Pages 4422-4433, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2013.11.003. (https://www.sciencedirect.com/science/article/pii/S0957417413009032)

[12] H. Al-Fahaam, S. Davis and S. Nefti-Meziani, "Wrist rehabilitation exoskeleton robot based on pneumatic soft actuators," *2016 International Conference for Students on Applied Engineering (ICSAE)*, 2016, pp. 491-496, doi: 10.1109/ICSAE.2016.7810241.

[13] Xinyi Zhang, Haoping Wang, Yang Tian, Laurent Peyrodie, Xikun Wang, Model-free based neural network control with time-delay estimation for lower extremity exoskeleton, Neurocomputing, Volume 272, 2018, Pages 178-188, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2017.06.055. (https://www.sciencedirect.com/science/article/pii/S0925231217311839)