Modeling and Learning Control System Design for Robots using Deep Learning Techniques

By

Raj Patel

A thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science (MSc) in Computational Sciences

The Faculty of Graduate Studies
Laurentian University
Sudbury, Ontario, Canada

© Raj Patel, 2021

THESIS DEFENCE COMMITTEE/COMITÉ DE SOUTENANCE DE THÈSE
Laurentian Université/Université Laurentienne
Office of Graduate Studies/Bureau des études supérieures

Title of Thesis Titre de la thèse	Modeling and Learning Control System Design for Robots using Deep Learning Techniques
Name of Candidate Nom du candidat	Patel, Raj
Degree Diplôme	Master of Science
Department/Program Département/Programme	Computational Sciences
Date of Defence Date de la soutenance	August 18, 2021

APPROVED/APPROUVÉ

Thesis Examiners/Examinateurs de thèse:

Dr. Kalpdrum Passi
(Co-Supervisor/Co-directeur(trice) de thèse)

Dr. Meysar Zeinali
(Co-Supervisor/Co-directeur(trice) de thèse)

Dr. Ratvinder Grewal
(Committee member/Membre du comité)

Approved for the Office of Graduate Studies
Approuvé pour la Office des études supérieures
Tammy Eger, PhD
Vice-President Research (Office of Graduate Studies)
Vice-rectrice à la recherche (Bureau des études supérieures)
Laurentian University / Université Laurentienne

Dr. Minglun Gong
(External Examiner/Examinateur externe)

ACCESSIBILITY CLAUSE AND PERMISSION TO USE

I, **Raj Patel**, hereby grant to Laurentian University and/or its agents the non-exclusive license to archive and make accessible my thesis, dissertation, or project report in whole or in part in all forms of media, now or for the duration of my copyright ownership. I retain all other ownership rights to the copyright of the thesis, dissertation or project report. I also reserve the right to use in future works (such as articles or books) all or part of this thesis, dissertation, or project report. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that this copy is being made available in this form by the authority of the copyright owner solely for the purpose of private study and research and may not be copied or reproduced except as permitted by the copyright laws without written authority from the copyright owner.

Abstract

In recent years, research and development in artificial intelligence dramatically impacted fields like health care, agriculture, and manufacturing. In the robotics field, innovation such as recognition, interaction, and manipulation made a breakthrough in "How a robot can help to improve the quality of human life?". Deep learning for real-time applications became viable as computer power and data became more readily available. In the past classical methods of modeling such as Euler-Lagrange method widley have been used to construct accurate dynamic model of the robots, but due to the uncertainty in the parameters and external disturbances, the accurate and generalized analytical model is too complex to build. On the other hand, deep learning algorithms are general non-linear modelling techniques that can learn from input-output data, making them a good choice for modelling and model-based control system design for robots. In this research, recurrent neural network techniques such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are adopted to overcome the issues of modelling. Then, by effectively combining adaptive sliding mode controller (ASMC) with the deep-learning-based (LSTM-based) inverse dynamic model of the robot a model-based control can be developed. . The deep-learning-based controller is applied for trajectory tracking on various scenarios to verify the effectiveness of this approach. The primary strategy of designing a controller with a deep recurrent neural network is to benefit from its memory, resulting in better generalization, accuracy enhancement, and better estimation of time-varying parameters and disturbances. The simulation and experimental findings show that the suggested controller operates adequately on unknown trajectories and disturbances even without tuning settings.

Keywords: Robot Learning Control, Deep Learning, Deep Recurrent Neural Network, Adaptive Sliding Mode Control, Long Short-Term Memory, Gated Recurrent Unit

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Meysar Zeinali, and Dr. Kalpdrum Passi for their continuous encouragement and fruitful discussion about concepts and theories related to my research work. Dr. Zeinali enthusiasm kept me engaged in my research work. He has been an inspiration to me as I struggle through the challenges in the deployment on the robot. His continuous support and knowledge about robotics made my journey much easier. I would like to thank Dr. Passi for providing advice related to my research and necessary support throughout my graduation journey. His knowledge about Data Science was beneficial in this research.

Furthermore, I also would like to thank Mark Thompson from the IT department for providing all the support related to the required software and hardware for computation in this research work.

Finally, I would like to thank my parents for believing in me and providing support and love throughout this journey.

Dedication

Dedicated to my parents,

for their endless love, support, and encouragement

Table of Contents

List of Tables

List of Figures

List of Symbols

$A \cdot B$	Dot product of two matrix
$A * B$	Element wise multiplication
q	Actual position of each joint in a robot manipulator
$\dot{q}$	Actual velocity of each joint in a robot manipulator
$\ddot{q}$	Actual acceleration of each joint in a robot manipulator
q_d	The vector of desired position
$\dot{q}_d$	The vector of desired velocity
$\ddot{q}_d$	The vector of desired acceleration
$\ddot{q}_r$	The vector of reference acceleration
τ	The vector of actuator torque in a robot manipulator
θ	Position in revolute joint
$\dot{\theta}$	Velocity in revolute joint
$\ddot{\theta}$	Acceleration in revolute joint
M	The inertia matrix of a robot manipulator
C	The centrifugal and Coriolis matrix of a robot manipulator
F	Generalized force due to Viscus and Coulomb friction
G	Gravitational term in a robot manipulator
d	External disturbance
F_c	Coulomb friction
F_v	Viscus friction
$\hat{M}$	Estimated inertia matrix of a manipulator
$\hat{C}$	Estimated centrifugal and Coriolis matrix of a manipulator
$\hat{F}$	Estimated friction matrix of a manipulator
$\hat{G}$	Estimated gravity matrix of a manipulator
ΔM	Unknown part of inertia matrix
ΔC	Unknown part of centrifugal and Coriolis matrix of a manipulator
ΔF	Unknown part of friction matrix in a manipulator
δ	Lumped uncertainty

$\hat{\tau}$	Nominal dynamics of a robot
ρ	Unknown bounding function of uncertainties
K	Proportional gain in a manipulator
S	The vector of a sliding function
Γ	The adaption gain matrix in a manipulator
e	Position error
$\dot{e}$	Velocity error
Λ	The positive definite sliding function design parameter
τ_m	Torque from Model based component
τ_{NN}	Torque from neural network
τ_{ASMC}	Torque from adaptive sliding mode controller
τ_{LSTM}	Torque predicted by Long Short-Term Memory
τ_{GRU}	Torque predicted by Gated Recurrent Unit
$\tau_{ASMC+NN}$	Torque predicted by a combination of adaptive sliding mode control and neural network
x^t	Input for recurrent neural network at time t
z_h^t	Weighted sum in recurrent neural network
h^t	Sigmoid of weighted sum in recurrent neural network
b_h	Bias vector of hidden layer in recurrent neural network
W	Weight matrix in recurrent neural network
$\hat{y}^t$	Predicted output at time t
f_i^t	Forget gate computation at time t
u_i^t	Update gate computation at time t
o_i^t	Output gate computation at time t
c_i^t	Candidate gate computation at time t
r_i^t	Reset gate computation at time t
L	Loss calculated using loss function
λ	L2 regularization parameter
p_{drop}	Dropout probability in dropout layer
s_t	First order moments in Adam optimization

r_t	Second order moments in Adam optimization
β_1, β_2	Exponential decay rate in Adam optimization
α	Initial learning rate
ζ	Positive definite diagonal matrix
I	Moment of inertia
$\overline{x}$	Mean value of x
σ_x	Standard deviation of x
$x^{'}$	Normalized value of x

Acronyms

IFR	International Federation of Robotics
PV	Process Variable
SMC	Sliding Mode Control
ASMC	Adaptive Sliding Mode Control
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
PID	Proportional-Integral-Derivative
RFSMC	Resilient Fuzzy Sliding-mode Control
AFSMC	Adaptive Fuzzy Sliding-mode Control
UAVs	Unmanned Aerial Vehicles
RUAVs	Rotatory Unmanned Aerial Vehicles
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
ReLU	Recursive Linear Unit
DCNN	Deep Convolutional Neural Network
FC	Fully Connected Layer
BPTT	Backpropagation through time
SGD	Stochastic Gradient Descent
MSE	Mean Square Error
RMSE	Root-mean-square error
ONNX	Open Neural Network Exchange

Chapter 1

Introduction

As the world is advancing rapidly with the benefit of the 4th industrial revolution, the robotics field is not an exception. Robotics is an innovative and developing field of modern technology that overpasses conventional engineering boundaries by benefiting from multiple domains. To understand the complexity of a robot and its working, knowledge of computer science and other engineering fields are required [1]. From ancient Greece times, the idea of designing a machine that works autonomously has evolved, but until the 20th century, the research and possible uses of robots did not grow. Advancement in artificial intelligence and machine learning made a dramatic impact on the robotics field. With the help of intelligent algorithms and computation power, robots are becoming more intelligent than ever before.

Robots are excellent at calculations, searching and processing an enormous amount of data, and doing repetitive work. Moreover, it can handle extreme environment for which humans are not capable such as radiation, temperature, pressure, toxic chemicals, and hazardous gases, which make them distinctive in terms of operation. Due to its unique characteristics, it plays a significant role not only in manufacturing but also in some other domain such as search and rescue, , scientific exploration, and public health. Example For instance, Spot from Boston Dynamics [2] is an example of an advanced robot that can automate the inspection task and give clear visibility at hazardous places in industries.

According to the application, it is essential to design a robot that works faster with more precision. Consider a robot working in medical science where humans are involved. A robot needs to be more

precise in control, safe in operation, and adaptive to its environment in such cases. With the rise of computer vision and deep learning, autonomous robots can drive the car or even dance like humans making the possibility of doing things boundless.

Advanced robots which are designed to work in human environment requires safe and friendly interaction with human. Increase human-robot-interaction necessitates a robot's perception and precision in the surroundings, which in turn requires more robust and intelligent control system.

On the other hand, Deep Learning algorithms are general non-linear algorithms that can solves this problem by learning the inverse dynamics of the robot directly from the input-output data of a robut. A robot with an intelligent controller can replace anything requiring safe, repetitive, precise, and fast work that can be carefully controlled and monitored in a dynamic environment. It could be more helpful in many areas of the industries. With the help of these intelligent controllers, the gap between human and robots at the workplace is becoming blur.

1.1 Robot Definition and Motion Control

According to the Robot Institute of America, a robot is “A reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through a variable programmed motion for the performance of a variety of tasks” [5]. Robot learning includes acquiring a novel skill or adapt to its environment using a learning algorithm for a particular task using human supervision or a trial and error approach.

A controller is responsible for the movements of the mechanical arms in the environment. The controller is the heart of the robot, which decides and controls movement in the environment.

It is more difficult to classify robots according to the application. According to M. W. Spong et al., robots can be classified based on fixed or mobile, geometry or kinematic structure (serial or

parallel), methods of actuation, and rigid or flexible [5]. A serial manipulator consists of links connected in series, which form an open-loop mechanism whereas, a parallel manipulator consists of multiple kinematic chains attached to the base that constructs a closed-loop mechanism.

As shown in Figure. 1.1 robotic system consists of a robotic arm, external power source, sensors, end effector, data acquisition device, control computer, and input device.

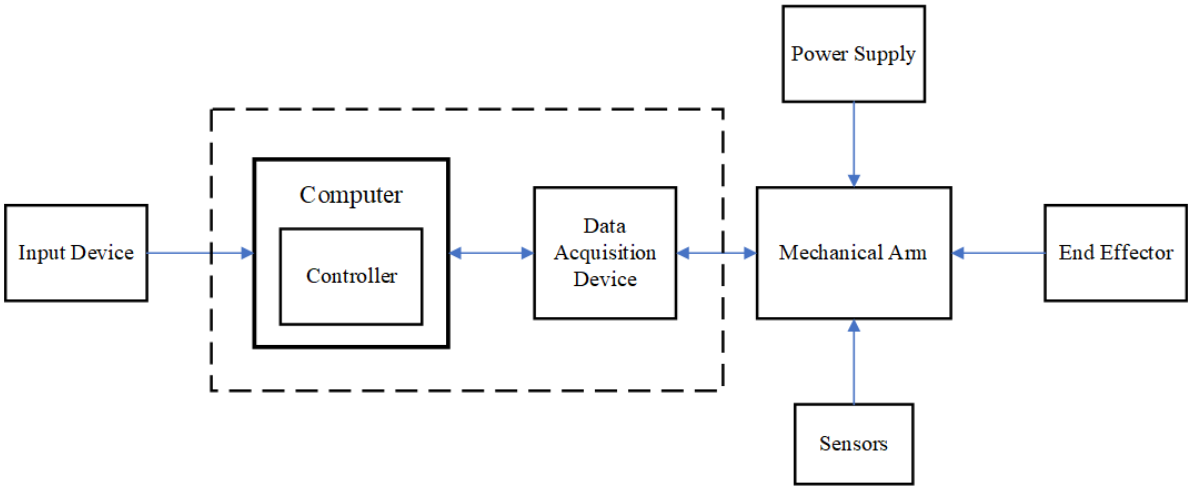

Figure 1.1: A structure of a robotic system [5].

The problem of controlling a manipulator can be formulated as that to determine the time history of joint actuator inputs (torque) required to cause the end-effector to execute a commanded task while satisfying given transient and steady-state requirements or desired behavior of all joints motion. For each joint axis, the motion controller of robots produces two sets of values: **i**) reference values for position, velocity, and acceleration. **ii**) a torque value (τ). The joint positions are calculated by the inverse kinematic model, while the dynamic model calculates the motor torques. The data acquisition device establishes real-time communication of data between the mechanical arm and controller. In some robot, the controller of the robot communicates to the robotic arms wirelessly. The input device is helpful to reprogram the robot using a computer. In

some advanced application, cameras are used to measure the exact position of the end effector in 3D.

1.2 Objective

Robot control is a centerpiece in Robotics that drives the robot to perform the desired task, such as welding in automotive industry. Conventionally, an engineer or a team of engineers work on this task to get optimized robot control. The process of optimizing the parameters of a robot controller is a tedious task, as many parameters need to be tuned to get better accuracy, stability, and robustness. Designing a robot controller that performs a task with more precision in a dynamic and noisy environment is challenging. For these reasons, deep learning algorithms have seen exponential growth in the robotics field in recent years. Rather than spending lots of time tuning the parameters and finding the robot's exact dynamic model, deep learning algorithms allow the controller to learn the system dynamic from the input-output data (i.e., inputs are joints' position, velocity and acceleration; and outputs are joints' torque). In the actual world, a generic deep learning model can handle a variety of circumstances, and the generalization of these deep learning models may be improved by training them on additional datasets, which is highly useful.

In this research, the primary objective is to construct the inverse dynamics of the manipulator using deep-learning techniques (deep recurrent neural network). The secondary objective involves a formulation of deep learning model rather than using the components.

With the help of deep learning-based controller, the final objective is to make an improved, precise, and generalized controller for a robot manipulator.

1.3 Motivation

Robot manipulator consists of links to form a kinematic chain and is connected through revolute joints or prismatic joints. Throughout this thesis, the revolute joint is considered and joint variable is denoted as θ. A configuration set in a manipulator that provides the entire definition of the position of every point on the manipulator is known as configuration space. Joint position (θ), velocity ($\dot{\theta}$), and acceleration ($\ddot{\theta}$) can all be used to specify the manipulator's state.

As discussed earlier, the robot faces extreme challenges in the real environment, making it exceptionally difficult to model the complex system accurately or process by the mathematical model. As shown in Figure 1.2, a feedback control works based on the error calculated with the help of the feedback signal. The controller generates a torque value for a given trajectory. Although the input to the system is known in the physical system, the torque value is not accurate due to uncertainty. This uncertainty arises from two sources in the physical system: ***i***) unknown or unpredictable inputs (e.g., disturbance, friction, noise) ***ii***) complicated dynamics.

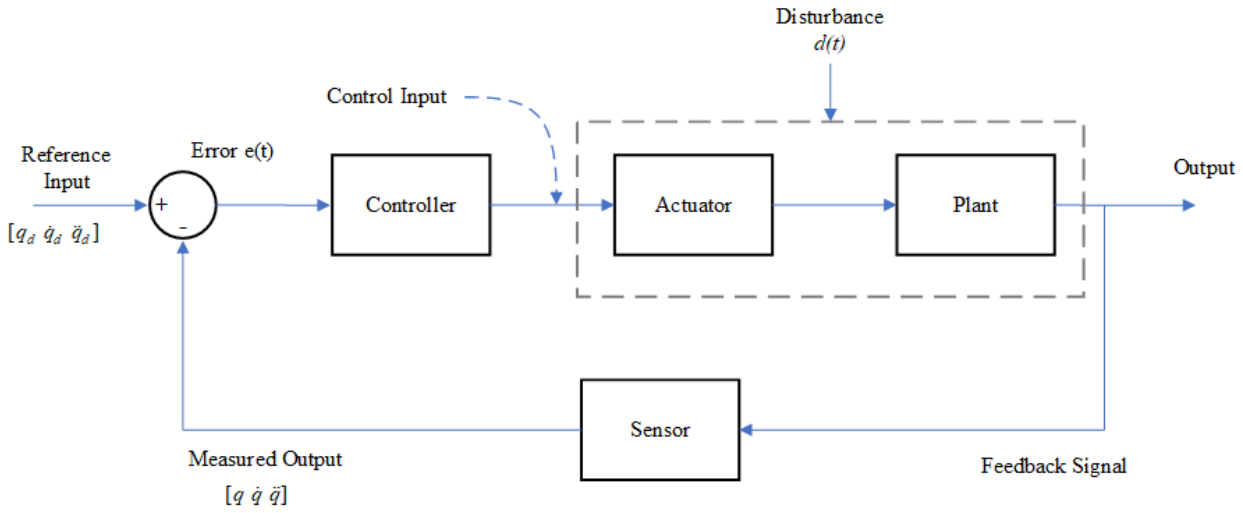

Figure 1.2: A general block diagram of feedback control [5].

The approximate values of the parameters are combined in one parameter called lumped uncertainty in a physical system. A comprehensive discussion of lumped uncertainty can be found in Chapter 3.

1.4 Overview of Purposed Method

A recurrent neural network has the qualities to perform better on sequential data. A manipulator inverse dynamics characteristics are sequential, which allows using recurrent neural network models such as LSTM and GRU with adaptive sliding mode controller. The primary strategy of designing a controller with recurrent neural networks and adaptive sliding mode control has its own benefits such as better generalization, accuracy enhancement, compensate for fast time-varying disturbances and handling error estimation. Joint position, velocity, and acceleration are the measurable and dynamic equation of the manipulator are assumed to be known.

The detailed information about deep learning model training and evaluation is available in chapter 4. Our approach for using a recurrent neural network to get the benefit of its “memory”. From the industry perspective, getting real-time data from the robot and training the model using the proposed controller improves the existing controller as it learns new data and remembers the previous data.

1.5 Outline of the Thesis

The remaining chapters are organized as follows:

Chapter-2 [Literature Review]: This chapter introduces a literature review related to the proposed method. Firstly, it discusses a brief history of robot and control theory. Secondly, it examines some existing controllers used in robot manipulators such as PID control, sliding mode control, adaptive control, adaptive sliding mode control, and intelligent control. It explores and

compares the advantages and disadvantages of various techniques. Lastly, it presents the justification of the proposed method.

Chapter-3 [Background and Theory]: This chapter illustrates a theoretical description required for understanding the proposed methodology, such as the representation of uncertainty term (lumped uncertainty). In addition to that, it presents the description of adaptive sliding mode control and deep recurrent neural network. It covers the mathematical formulas to represent the deep recurrent neural networks such as LSTM and GRU and the fundamental idea of the hyperparameters in the neural networks.

Chapter-4 [Implementation and Simulation Results]: This chapter describes the technique to generate the dataset to train the deep recurrent neural network. It covers the model training and validation process to optimize the deep recurrent neural network model for given hyperparameters and dataset. It encompasses the mathematical formulas to combine an adaptive sliding mode controller with a deep recurrent neural network. Finally, it compares the proposed controller with adaptive sliding mode control in various cases.

Chapter-5 [Experimental Results and Discussion]: This chapter clarifies the experimental setup and parameters of the robot used in this research work. It explains hardware details of the Quanser 2-DOF robot and the deep learning model deployment process on target hardware. In addition to that, it represents experimental results of the proposed method on various cases (e.g., with and without disturbance, different trajectories) and conversation about each experiment.

Chapter-6 [Conclusion, Contribution and Future Work]: The last chapter concludes this research work, discuss about the contribution, and some ideas for future work.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides an overview of the literature related to our research work. The central idea of this chapter is to give a breif knowledge of some existing robot controllers such as PID, sliding mode control, adaptive control, adaptive sliding mode control, and intelligent control. Furthermore, these controllers' merits and downsides are explored. Finally, it justifies the need for intelligent control in combination with adaptive sliding mode control to the handle problem in the control system.

2.2 A Brief History of Control Theory

Control theory is a centerpiece in robotics which deals with the control of robots. The goal of control theory is to create a model or algorithm that governs the application of system inputs to drive the system to the desired state (e.g., desired trajectory), while reducing any delay, overshoot, or steady-state error by ensuring the system's stability.

The theoretical underpinning for the operation of governors was first defined in the 19th century by James Clerk Maxwell [6]. Edward Routh in 1874, Charles Sturm in 1895, and Adolf Hurwitz in 1895 all contributed to the formation of control stability criteria which became essential in robotics. Control theory had grown in importance by World War II. In 1922, PID or three-term control law, was first developed using theoretical analysis by Nicolas Minorsky [7], which significantly impacted the robotics field.

Feedback control theory is applicable anywhere wherever feedback is required . The use of feedback controllers increased dramatically in the twentieth century. Some application includes

fire-control system, steam generation-control system, electric motor speed control, ship and aircraft stabilizers. It has applications in biological sciences, pure and applied mathematics, engineering (aerospace, chemical, computer, electrical, and mechanical), sociology, and operations research. A summary of some available control methodologies are shown in Figure 2.1.

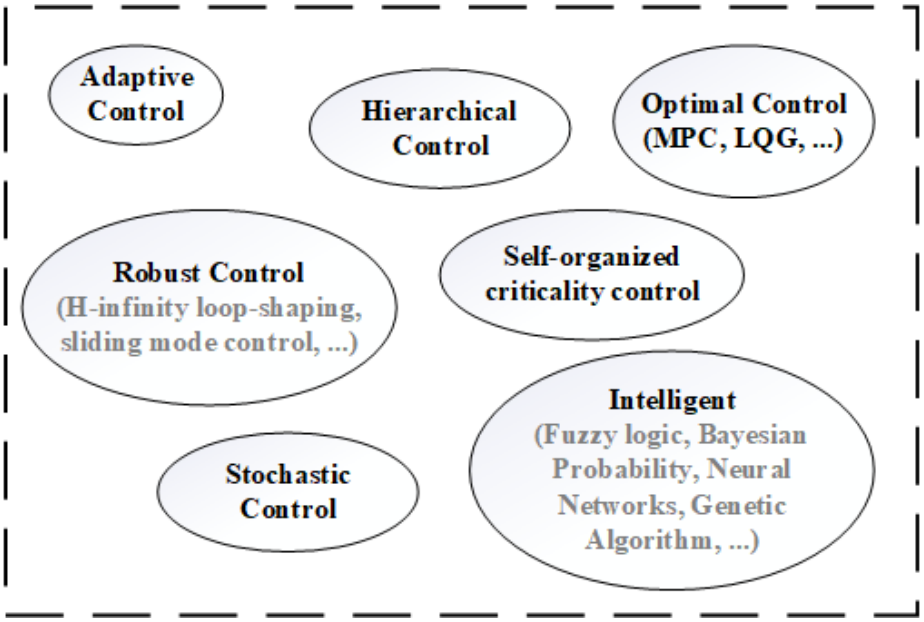

Figure 2.1: A summary of some available control methodologies

2.3 A Brief History of Robotics

As discussed earlier, the idea of controlling machines autonomously has evolved from ancient Greece time. The word “Robot” first appeared in 1921 in Karel Capek’s play Rossum’s Universal Robots [8]. Apart from the historical significance, this play expresses critical issues from a philosophical and technological perspective. The play's storyline shows how robotics can provide huge conveniences while also having enormous implications on people's lives, necessitating immediate legislation to prevent them. Thus, this play has a significant impact on the robotics community.

George Devol patented the first digitally operated and programmable robot in 1954 (accepted in 1961) [9], representing the foundation to modern industrial robots. The 2,700-pound “Unimate” prototype was initially placed on an assembly line at a General Motors diecasting factory in Trenton, New Jersey, in 1959. General Motors became the first company that uses robots in the production line [10] in 1961. The critical roadblocks to advances in control technology were the high cost of processing, a lack of appropriate sensors, and fundamental knowledge of robot dynamics. In that era, the introduction of transistors into computers significantly impacted reducing the size and increasing the performance of robots. The development and deployment of advanced, sensor-based control has been aided by the increasing speed and lowering cost of computation.

Over time, the typical heavy manipulator is expected to replace with new, lighter, faster, and more intelligent systems. In 2000, Honda showed ASIMO, a highly advanced humanoid robot that can run, walk, communicate with humans, recognize faces, environment, voices, posture, and interact with its environment. In recent years, there is a significant usage of robots in every field, and in the near future, robots could become a part of human life.

Along with the advancement in computer-aided design and manufacturing, from the 1960s, robotics opened new frontiers in every field. Artificial intelligence made a dramatic impact on robotics. With the help of computer vision, natural language understanding, and deep learning algorithms, robots are becoming more intelligent than ever before. Pick and place, screwing, drilling, and welding are just a few of the activities that integrated robotic platforms like the arm and end-effector can accomplish. Robots must follow the desired trajectory precisely in a highly constrained environment to perform all these tasks.

From the simulation in the lab to real-world, robotic industry faces extreme challenges. Growing Human-Robot Interaction necessitates the robot gaining more knowledge and precision in order to assure the protection of human life. Human behaviour interpretation is a challenging issues for which human-made solutions are difficult to come up with [11]. In such a scenario, the controller needs to be more intelligent to be collaborative with humans.

As the industry expands, robots will be required to do a variety of tasks, which is challenging for many learning algorithms to manage. In order to execute a task in the real world, a robot must follow the desired trajectory more precisely in a highly constrained environment. However, in real applications, the manipulator's inverse dynamic model may not be accurate because of the following: ***i***) nonlinear dynamic behavior of robot manipulators ***ii***) uncertainty of parameters ***iii***) time-varying effect such as tear and wear ***iv***) external disturbances ***v***) data acquisition and filtering errors due to the noise or inaccurate data coming from sensors.

A system with the ability to learn from data plays a critical role in such a scenario. Numerous scholars have studied intelligent control, variable structure control, adaptive control, and other aspects of robotics and control [12]–[14]. In this research the goal is to tackle the problem of learning from data through simulation and experimental study. The subsequent section discusses some robot control methodologies such as feedback linearization, computed torque control, sliding mode control, adaptive control, fuzzy control, genetic algorithm-based control, neural-network-based control.

2.4 Robot Control Methodologies

Since the 1920s, there are numerous controllers proposed by researchers. Some well-known robot control methodologies discussed in this section are:, PID, computed torque control (feedback

linearization), sliding mode control, adaptive control, passivity-based control, and intelligent control (i.e., fuzzy logic, genetic algorithm, neural network).

The three-term control, also known as PID, was first developed by Nicolas Minorsky using theoretical analysis [7]. PID is the most common controller in the industrial manipulator because of having a basic mathematical model, where parameters of the dynamic system and noise from the environment are accurately known [15]. Many researchers used this controller as a base controller to create a new controller [9] or combine it with other algorithms to get better performance [16]. Rafael Kelly proposed simple rules for tuning PID controller in [17]. Another way to optimize the PID control can be found in [18]. It has been claimed that the PID control method in high-speed applications cannot manage nonlinear systems with uncertainty [19]. It is critical to have a controller that can deal with the unpredictability of maintaining a trajectory.

The differential geometric approach of feedback linearization, which has been applied to various practical issues both within and outside of robotics, was inspired by prior research on inverse dynamic and computed torque control [20]. In computed torque control, the goal is to create a nonlinear feedback system that balances out the nonlinearities in a robot manipulator. The closed-loop system then becomes strictly linear or somewhat linear, depending on the model's accuracy, and thus a linear controller such as PD or PID can be used. In high-speed situations, computed torque control based on feedback linearization performs well; however, it requires the exact dynamic model of the robot, which is very difficult to obtain due to uncertainties. To deal with uncertainties about unknown parameters, computed torque control-based adaptive control and neural network techniques are applied in many applications [14], and [21].

In the mid-1980s, Slotine and Li developed a new adaptive robot control method [22] that consists of a PD feedback component and a full dynamics feedforward compensation portion. The unknown manipulator and payload parameters are calculated online in this method. The adaptive control problem necessitated the understanding of two key characteristics of Lagrangian dynamical systems: linearity in inertia parameters and the skew-symmetry feature of the robot inertia matrix [23]. Adaptive control is asymptotically stable in the Lyapunov sense in the face of system uncertainties, resulting in closed-loop asymptotic tracking with all remaining signals restricted. The main problem with adaptive control is to handle unmodelled dynamic and external disturbances.

Sliding mode control (SMC), a well-known variable structure control technique, has been presented to fulfil robust tracking. Sliding mode control is well-known due to its capability to deal with uncertainties, good transient response, and fast response [24]. The chattering problem still arises in this method, may lead to an actuator damage and inefficient control input. As discussed earlier, numerous sliding mode controllers have been proposed to satisfy robust tracking [13]. The SMC is built upon a two-phase switching control law. ***i***) For any bounded disturbances, the sliding surface must be established, and the state variable must be configured to follow the appropriate trajectory in joint coordinates. ***ii***) Modify the switching gain to be greater than the unknown bounded disturbance and dynamic parameters [19]. This control methodology is known as "variable structure control" since the structure of the control input switches from one structure to another.

Terminal sliding mode control, as opposed to SMC, is used to predict uncertainty in order to optimize the transient response, such as faster convergence to the equilibrium point and increased resilience [25]. In [26], a neural network-based terminal sliding mode is proposed to approximate

the nonlinear dynamics of the robot manipulator. Although, the chattering phenomena leads to accuracy and robustness reduction in the manipulator. P. Kachroo presented a new approach using a new function inside the boundary layer to diminish the chattering effect [27].

As previously stated, numerous sliding mode controllers have been proposed to provide robust tracking with or without combination with other controls. Online identification of process parameters or adjustment of controller gains are used in adaptive control to provide high resilience characteristics [28]–[31]. Adaptive control uses online identification of the robot dynamic parameters or modification of the controller gains, improving to obtain strong robustness properties. When these two strategies are combined, they are powerful enough to eradicate unknown disturbances. In the 1950s, adaptive controller applied in the field of aerospace.

Intelligent controller uses various innovative approaches such as fuzzy logic, genetic algorithms, artificial neural networks, and a combination of those.

In machine control systems, fuzzy logic is commonly employed. In 1965, professor from University of California, L. Zadeh invented fuzzy logic [32]. Fuzzy modelling consists of a set of rules (IF-THEN rules) and a reasoning mechanism [33]. The reasoning mechanism is a process for assuming a good result based on rules and data. Fuzzy systems can be used to solve situations for which a thorough mathematical description is unavailable or when using a precise model is extremely difficult. A fuzzy system's capacity to assimilate erroneous information makes it a good tool for control processes, system identification, decision support, and signal and image processing, among other things.

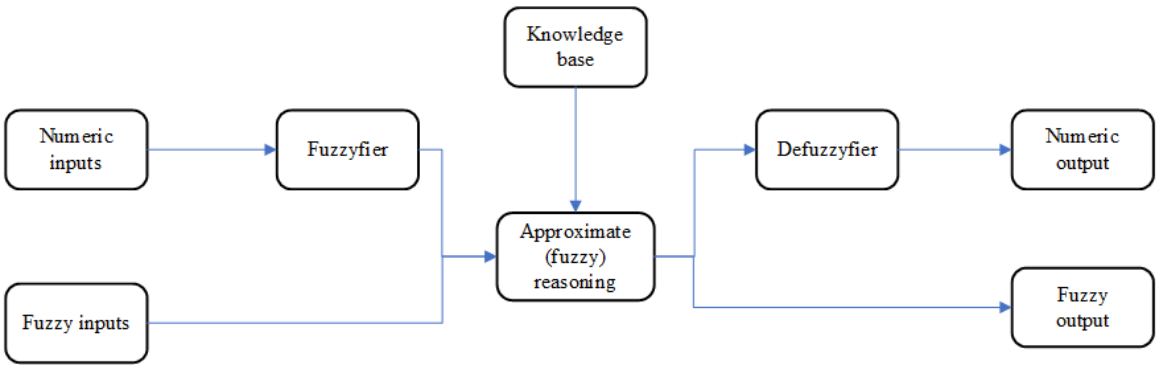

Figure 2.2: A typical structure of a fuzzy system [33].

Unlike traditional or digital logic, which works with discrete values of 1 or 0, this mathematical method analyses analogue input data using logical variables with continuous values between 0 and 1.

Fuzzy sliding mode is another way to deal with undesirable chattering and unknown bounds of uncertainty. Based on Takagi-Sugeno, Hwang presented a unique adaptive fuzzy sliding-mode controller [34]. In this method, a nonlinear dynamic system is approximated by N fuzzy-based linear state-space subsystem. Furthermore, the same fuzzy sets of the system rule are utilised to develop resilient fuzzy sliding-mode control (RFSMC) and adaptive fuzzy sliding-mode control (AFSMC). The controller alternates between RFSMC and AFSMC, making implementation problematic due to the unpredictable transient performance. M. Zeinali proposed an adaptive chattering-free sliding mode control using fuzzy model of the system [35] which is the combination of the adaptive and continuous sliding mode control method proposed in [31] and the systematic adaptive fuzzy modelling method proposed in [36], and [37].

Genetic algorithm-based robot control is another exciting area of research. A search heuristic based on Charles Darwin's natural selection concept is known as a genetic algorithm [38]. This algorithm mimics natural selection, in which the fittest individuals are chosen for reproduction to create the

following generation's children. To produce high-quality solutions to optimization and search issues, genetic algorithms use biologically inspired operators like as mutation, crossover, and selection. The fitness function in a genetic algorithm calculates a fitness score by determining how close is a given solution to an optimum solution of the problem at hand. The goal of the selection phase is to find the fittest individuals based on this fitness score and let them pass their genes along to future generations. A genetic algorithm's most crucial phase is crossover. It's a genetic operator that combines the genetic information of two parents to create new children. Mutation happens to maintain population variety and avoid premature convergence. The algorithm terminates upon successful convergence of the population.

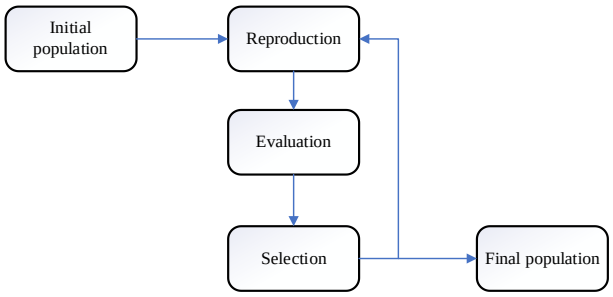

Figure 2.3: The typical structure of a genetic algorithm.

V. Maniezzo used a genetic algorithm in Boolean function learning and robot control (for weight distribution in the neural network) [39]. Vincent Roberge et al. used a genetic algorithm and the practical swarm optimization algorithm for path planning of unmanned aerial vehicles (UAVs) [40].

Artificial intelligence and machine learning approaches are used in SMC to decrease the chattering effect [41]. Because of its capacity to approximate any function, neural network-based approaches have grown in popularity over the last decade. The mixture of traditional control and deep learning-based approaches resulted in the development of new research techniques. S.P. Chan developed a

neural network-based technique to learn the inverse dynamics of the flexible manipulator in [42]. In comparison to a traditional controller, the neural network-based method requires more processing power.

The popularity of using neural network in combination with existing controllers has increased. With the goal of achieving robust tracking performance on desired trajectories as quickly as possible. B. Ozyer proposed an adaptive fast sliding controller coupled with a neural network and a global fast sliding mode controller [19]. In order to learn quadrotor dynamics for flight control, S. Bansal et al. employed a neural network-based approach [43]. S. Edhah et al. used a feedforward neural network to control the altitude, hovering mode of Rotary Unmanned Aerial Vehicles (RUAVs) by using a greedy layer-wise approach [44]. The greedy layer approach uses multiple separately trained layers into one network to get more accuracy. However, it increases training time significantly if the depth of the network is higher.

To work with humans, several studies were performed to introduce human-like behavior, which is beneficial to improve accuracy in such an environment. Huang et al. [45] founded a novel method for controlling a dual-arm exoskeleton robot that successfully transfers human-like impedance. Beretta et al. [46] used a redundant robot to achieve adaptive human-like hands-on control in surgery, allowing them to reach and accomplish the required job. De Momi et al. [47] used the feedforward artificial neural network to obtain human-like motion in teleoperated anthropomorphic manipulators for trajectory planning.

Furthermore, for geophysical exploration [48], motion tracking [49], and biomedical application [50], artificial neural network-based approaches had been applied. Generally speaking, compared to traditional nonlinear methods, the artificial neural network-based approachs can ensure high

accuracy [51], [52]. Z. Su et al. [53] used a three-layer neural network to model the inverse dynamic of two-link flexible robots. Using a combination of backstepping, neural network, and indirect method, W. Chatlatanagulchai et al. [54] proposed a model to control the motion of a two-link flexible joint robot. Mori et al. [55] proposed a forward propagation rule for a neural network to learn an inverse robot model. Jiang et al. [56] created a dynamic compensator using a three-layer neural network to improve a robot's dynamic model with identified parameters.

Deep Learning has seen tremendous success in image and video processing, object recognition, recommender systems, and natural language processing in recent years. Deep Learning is a subset of a larger category of machine learning approaches that learn from data. In deep learning, each layer extracts new features making it possible to predict the desired output of the system. Here, learning methodology includes supervised, unsupervised, and semi-supervised. The learning procedure depends on the structure of a neural network. In a case of object detection using convolutional deep neural network, top layers learn higher-level features, whereas lower-level characteristics are learned by the last layers. For the first time, Yann LeCunn et al. [57] proposed a new technique to predict hand-written digits taken from the US Mail. This technique uses backpropagation algorithm to update the weights in the neural network. In [34], Krizhevsky, Sutskever, and Hinton presented a deep convolutional neural network that improved image identification considerably. For managing redundancy optimization for robot control, H. Su et al. [58] used a deep convolutional neural network method.

Humans have biological intelligence that processes information in little chunks while maintaining an internal model of processing that is founded on past knowledge and modified when new information is acquired [59]. Using this idea, the goal behind using a recurrent neural network-based model is to keep the information in a cell (memory) so that it can be used for prediction. In

[60]–[62], a three-layer recurrent neural network is used to model the inverse dynamic of the robot. The multiplicative gradient in neural network may be exponentially decreasing/increasing with respect to the number of layers leading to vanishing gradient problem in recurrent neural networks. The main reason behind this phenomenon is the difficulty in capturing long-term relationships. Because of insufficient weight shifts in the network, a number of layer increase, long-term dependencies are challenging to understand [63], [64]. Advanced recurrent neural networks such as the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) overcome this problem . In 1997, S. Hochreiter and J. Schmidhuber proposed Long Short-Term Memory [65]. J. Chung et al. proposed a new type of recurrent neural network by focusing on more sophisticated units that implement gating mechanism [66]. These two networks are explained in detail in Chapter 3. For Smart Cities and Factories, N. Liu et al. employed an LSTM-based deep learning method to describe robot inverse dynamics [67]. Rueckert et al. [68] proposed LSTM based approach to model inverse dynamics of a robot.

The use of deep learning approaches to predict the inverse dynamics of the manipulator and create a robust controller is still relatively new and needs additional investigation. In this study, robust adaptive motion control is suggested by integrating existing adaptive sliding mode controller (ASMC) with recurrent neural networks like Long Short-Term Memory and Gated Recurrent Unit . This approach is verified by performing simulation and experiments on actual robot.

Chapter 3

Background and Theory

3.1 Introduction

As discussed earlier, due to the existence of non-linear parameter such as friction, backlash, unknown disturbance, payload fluctuation, flexibility in joint connections, and time-varying parameters such as tear and wear, and friction coefficient, it is exceedingly difficult to establish the exact dynamic model of a robot. Because of these factors, the majority of robot manipulators model is an approximation of a real manipulator. This chapter focuses on explaining background of the adaptive sliding mode controller and advanced recurrent neural networks.

This chapter is structured as follows. A problem definition and formulation are discussed in Section 3.2. This includes a detailed description about adaptive sliding mode controller. Proposed controller and detailed description of mathematical formulas are presented in Section 3.3. The deep learning methodology is described in detail in Sections 3.4 to 3.10. These sections include a general description of deep learning, overview of recurrent neural network and its usage, advanced recurrent neural networks such as Long Short-Term Memory and Gated Recurrent Unit. Following sections includes the structure and detailed explanation of fully connected layer. In the end of this chapter, regularization and optimization techniques are discussed.

3.2 Problem Definition and Formulation

The general dynamic model for n-link manipulator in joint space can be described by a system of second-order differential equation as follows [35]:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(q,\dot{q}) + G(q) + d(t) \tag{3.1}$$

In above equation, joint position, velocity, and acceleration are denoted as q, $\dot{q}$, and $\ddot{q}$, respectively.

Here, $M(q) \in R^{(n\times n)}$, $C(q,\dot{q}) \in R^{(n\times n)}$, $F(\dot{q}) \in R^{(n\times 1)}$, $G \in R^{(n\times 1)}$, $d(t) \in R^{(n\times 1)}$ and τ refers to inertia matrix, which is positive definite, Coriolis terms, viscus and Coulomb friction coefficient, gravitational term, bounded disturbances, and torque, respectively. Viscus (F_v) and Coulomb (F_c) friction term defined as the following equation:

$$F(q,\dot{q}) = F_v\,\dot{q} + F_c \tag{3.2}$$

Genrally speaking, it is very difficult to build exact dynamic model or invers dynamic models of a robot, because it dynamic models are based on parameters that are often need to be identified accuratley. Moreover, dynamic models might fail to accurately represent all the mechanical structure's physical properties because not all the effects are observed such as external disturbance, payload variation, and friction or because some of the effects are too difficult to model. As a result, the dynamic model is an approximate representation of the real robot, and the components stated in (3.1) may be recast as follows:

$$\tau = (\hat{M}(q) + \Delta M(q))\ddot{q} + (\hat{C}(q,\dot{q}) + \Delta C(q,\dot{q}))\dot{q} + (\hat{F}(q,\dot{q}) + \Delta F(q,\dot{q})) + (\hat{G}(q) + \Delta G(q)) + d(t) \tag{3.3}$$

And

$$M(q) = \hat{M}(q) + \Delta M(q);\ C(q,\dot{q}) = \hat{C}(q,\dot{q}) + \Delta C(q,\dot{q});\ F(q,\dot{q}) = \hat{F}(q,\dot{q}) + \Delta F(q,\dot{q});\ G(q) = \hat{G}(q) + \Delta G(q) \tag{3.4}$$

Here, the estimated values of inertia matrix, Coriolis terms, friction terms, and gravity terms are denoted as $\hat{M}$, $\hat{C}$, $\hat{F}$, and $\hat{G}$, respectively. Furthermore, an unknown portion of these terms are

defined as $\Delta M, \Delta C, \Delta F,$ and ΔG. In this research work, it is assumed that without the loss of gravity, the friction term defined as above $F(q, \dot{q})$ is completely unknown. Therefore, equation (3.4) can be written as follows:

$$\tau = \hat{\tau}(q, \dot{q}, \ddot{q}) + \delta(q, \dot{q}, \ddot{q}) \tag{3.5}$$

$$\hat{\tau}(q, \dot{q}, \ddot{q}) = \hat{M}(q)\ddot{q} + \hat{C}(q, \dot{q})\dot{q} + \hat{G}(q) \tag{3.6}$$

Where $\hat{\tau}(q, \dot{q}, \ddot{q})$ is a nominal dynamic of a robot and $\delta(q, \dot{q}, \ddot{q})$ is an unknown function that combines various uncertainty term into one lumped uncertainty term and can be defined as follows:

$$\delta(q, \dot{q}, \ddot{q}) = \Delta M(q)\ddot{q} + \Delta C(q, \dot{q})\dot{q} + \Delta F(q, \dot{q}) + \Delta G(q) + d(t) \tag{3.7}$$

The following assumption is made in order to build the controller.

Assumption: In Euclidian norm, the uncertainty vector $\delta(q, \dot{q}, \ddot{q})$ and its partial derivatives are limited as follows:

$$\| \delta(q, \dot{q}, \ddot{q}) \| \leq \rho(q, \dot{q}, \ddot{q}) \tag{3.8}$$

where $\| . \|$ is Euclidian norm. The unknown bounding function of the uncertainties is defined as $\rho(q, \dot{q}, \ddot{q})$. The reader is advised to refer [31] for more extensive information on the stability. An option in contrast to engineering a fixed dynamic model attempting to anticipate the reality is to allow the robot to learn the characteristics of its own dynamics over time and during operation. With the help of a deep learning algorithm, one can build a model that can learn the features of various robot and successfully replace implementation of the analytical dynamic model. The deep learning based inverse dynamic model can be evaluated by measuring the performance of the

controller constructed based on the deep learning model. . This approach could be applied in following circumstances: ***i***) An exact dynamic model of a robot is not known or is extremely complex to describe the dynamic of robot analytically. ***ii***) When building the parameterized dynamic model of robot and performing parameter identification to build adaptive control is time consuming, costly, and is not well generalized.

3.3 Proposed Controller

The deep-learning based controller can be built based on adaptive sliding mode (ASMC) controller presented in [31], and [69] as:

$$\tau_{ASMC} = \hat{M}(q)\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{G}(q) - KS - \Gamma\int S\ dt \tag{3.9}$$

In equation (3.9), K and S are design parameters and they are symmetric positive definite diagonal matrix. Here, the sliding variable S is defined as follows:

$$S = (\frac{d}{dt} + \Lambda)^2(\int e\ dt) = \dot{e} + 2\Lambda e + \Lambda^2\int e\ dt \tag{3.10}$$

And

$$e = q - q_d,\ \ \dot{e} = \dot{q} - \dot{q}_d \tag{3.11}$$

where Λ is an $n \times n$ symmetric positive definite diagonal constant matrix and $e = q - q_d,\ \ \dot{e} = \dot{q} - \dot{q}_d$ are the position error and velocity error vectors, respectively. q_d and q are the desired position and measured position of each link in the robot. To guarantee that there is no offset mistake, the integral of error is included. The control law in equation (3.9) includes two parts:

I. A model-based component, i.e., $\hat{M}(q)\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{G}(q)$. This component can be constructed with a use of available knowledge about the dynamic model of the robot manipulator.

II. A sliding function-based proportional-integral component $\tau_{PI} = -KS - \Gamma\int Sdt$. It is built on the dynamic behavior of sliding function and replaces the discontinuous term in standard sliding mode, for detail please refer to [31], [69].

The objective of this study is to replace the model-based component defined as follows:

$$\hat{\tau}_m = \hat{M}(q)\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{G}(q) \tag{3.12}$$

with the component constructed using deep learning techniques and is called τ_{NN} in this thesis. In fact, τ_{NN} is a data-driven parameterized model of the robot built using deep learning techniques and off-line training on remote server utilizing data gathered from the robot using existing ASMC controller. In this deep learning model, the model's parameters being the network's weights. The component based on deep learning is a nonlinear function of joint variables and is as follows:

$$\tau_{NN} = f(q,\dot{q},\ddot{q}) \tag{3.13}$$

Where, the input variables of the neural network are joint position, velocity, and acceleration of denoted as $(q,\dot{q},\ddot{q})$, and the output of the neural network is the predicted torques (τ_1,τ_2) of the corresponding joints. Replacing the model-based component defined in equation (3.9) with neural network-based component τ_{NN}, control law can be written as the following equation and the block digram of the controller as depicted in Figure 3.1:

$$\tau_{ASMC+NN} = \tau_{NN} - KS - \Gamma \int S \, dt \tag{3.14}$$

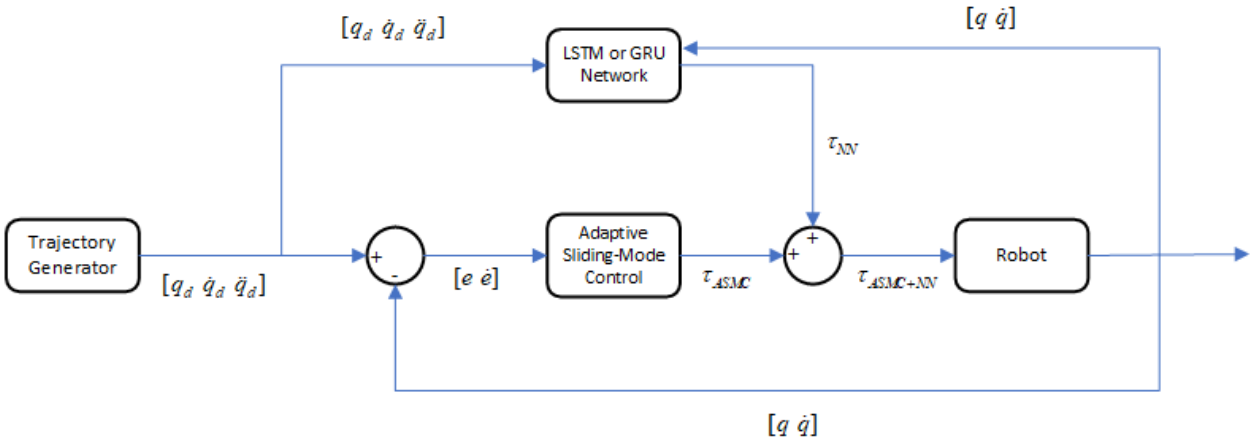

Figure 3.1: A block diagram of the proposed controller.

As illustrated in Figure 3.1, the proposed controller is a hybrid of an adaptive sliding mode controller and a deep recurrent neural network. The torque of the deep recurrent neural network (τ_{NN}) and adaptive sliding mode controller (τ_{ASMC}) combined as ($\tau_{ASMC+NN}$).

$$\tau_{ASMC+NN} = \tau_{NN} + \tau_{ASMC} \tag{3.15}$$

Here τ_{NN} is the output from a deep recurrent neural network. The output of the LSTM can be defined as τ_{LSTM} and GRU as τ_{GRU}. The algorithm for implementation is divided into three parts as shown below:

1. Reading dataset from the Quanser 2 DOF flexible link robot. After that, divide the dataset into suitable training, validation, and testing ratios.

2. Building and training a deep recurrent neural network model such as LSTM and GRU using the dataset generated from the robot.

3. Result validation and hyperparameter tuning.

3.4 Deep Learning

Deep learning is a subset of a wider class of machine learning algorithms based on representation learning and artificial neural networks. G. Hinton et al. [71] proposed deep learning as a new research area of machine learning to learn new features and find meaningful representation from data. The multi-level architecture of these concepts enables the computer to learn complex concepts by building them out of more simplistic algorithms, which is the main reason to call this approach "deep". In order to learn new things, a person's daily existence need a vast quantity of information, and the human brain processes millions of new things. When compared to human life, computers require an enormous quantity of world information, the majority of which is subjective and intuitive, making it exceedingly challenging.

A computer can reason automatically about assertions in the formal language using logical inference rules in a knowledge-based manner. The challenge that this system faces implies that artificial intelligence should be able to learn on its own by extracting patterns from raw data. This process is called as “Machine learning”. For example, the naive Bayes method, which is a simple machine learning technique, can identify essential emails from a list of spam emails. The format of the data that the machine learning algorithm is given has a significant impact on its performance.

In machine learning, it is crucial to design the right set of features to get the best output from the machine learning model, which is called feature engineering. In image-related task such as object detection, it is very difficult to know what features should be extracted whereas in a signal processing task, removing noise and get the important frequency from the signal is important. In such cases, an immense amount of time and expert knowledge are required.

Deep learning solves this problem with multi-level representation built on top of each other. The feedforward neural network or multilayer perceptron is a simple example of a deep learning model. In deep learning, there are two main ways to measure the depth of the model as following:

I. Using computational graph

II. Using probabilistic graph

Deep learning algorithms have evolved tremendously as tools and techniques have developed, to the point that they can now surpass humans at classifying photos, defeat the world's greatest GO player, and enable automobiles to drive themselves without human supervision. Deep learning is experiencing explosive growth on account of GPUs' use to accelerate their execution, big data to satisfy its data hungriness, faster network connectivity, and better infrastructure for distributed computing.

For a deep learning, there are three things that needs to take in account such as

- *Input data* – For regression task such as stock market data prediction, this input data could be continuous data point of stock with timestamp. In classification task such as object detection task, this input could be image.

- *Expected output data* – In stock market data prediction, the output data could be the next point of the stock. For object detection task, the output data could be the labels of object names such as "car", "house", and so on.

- *A way to measure the prediction of the algorithm* – In order to determine the distance between the algorithm's prediction and the original output, this measure is important. It is

utilized as a feedback signal to tune the weight of the layers which results in the change of the algorithm's output.

In deep learning, the individual level learns to transform its input data into a slightly wider abstract and composite illustration, which significantly accelerates the work and reduces time in feature engineering task. With the use of a multi-level structure and layer-wise training, a variety of data such as images, text, audio, and video etc., can be uncovered.

3.5 Recurrent Neural Networks

Recurrent neural networks, or RNNs [72], are a type of neural network that was created specifically for sequence modelling. Vanilla network accepts fixed size of input and returns fixed size of output which is a limitation. In contrast, recurrent neural network has a capability to process variable length of sequence $x^1, x^2, x^3, ..., x^T$ where each sample point x^t represent input at t time. It is critical in a recurrent neural network to exchange parameters across various portions of the model in order to generalize across them. This idea of sharing parameters found in machine learning and statistical modeling in 1980s. Each layer's output in a recurrent neural network is a function of the preceding layer's output. It is produced using the same computation carried out in the previous outputs resulting in the parameter sharing through a deep computational graph.

- **One-to-one:** Vanilla mode of processing the input to predict the output without recurrent neural network. The data in both the input and output is in a standard format. Image classification is an example of this structure where network accepts image as input and predict the label.

If either the input or output data is sequence, then it can be in the form of one of the following:

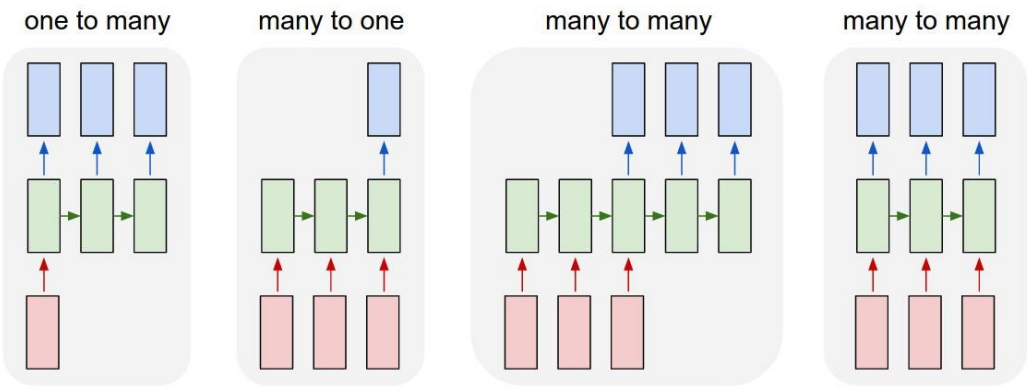

Figure 3.2: Various types of recurrent neural networks based on application [73]

- **One-to-many:** The input data is in a standard format and the output data is a sequence. For example, in video captioning, a recurrent neural network gets one frame as input and predict caption according to it.

- **Many-to-one:** In this structure, the input data is a sequence, and the output data is a label or fixed length of data point. For example, in the sentiment analysis task, the input data can be a sequence of text or audio and the output data is positive or negative sentiment label.

- **Many-to-many:** Input and output in this structure are sequence. This structure may be further split into two portions based on synchronization or delay, as illustrated in Figure 3.2. Video classification is the best example of synchronized many-to-many structure. In this structure, recurrent neural network labels every frame to its label. Machine Translations is an example of delayed many-to-many structure. In this type, recurrent neural network accepts the whole sequence in one language and generate another sequence in another language. For example, a whole sequence of English language fed into the network and the network produces another sequence in Hindi language.

In the standard neural network, the information flows from input to hidden layer to the output. On the other hand, in recurrent neural network, the information flows from input and previous hidden layer to the next hidden layer to the output as shown in Figure 3.3.

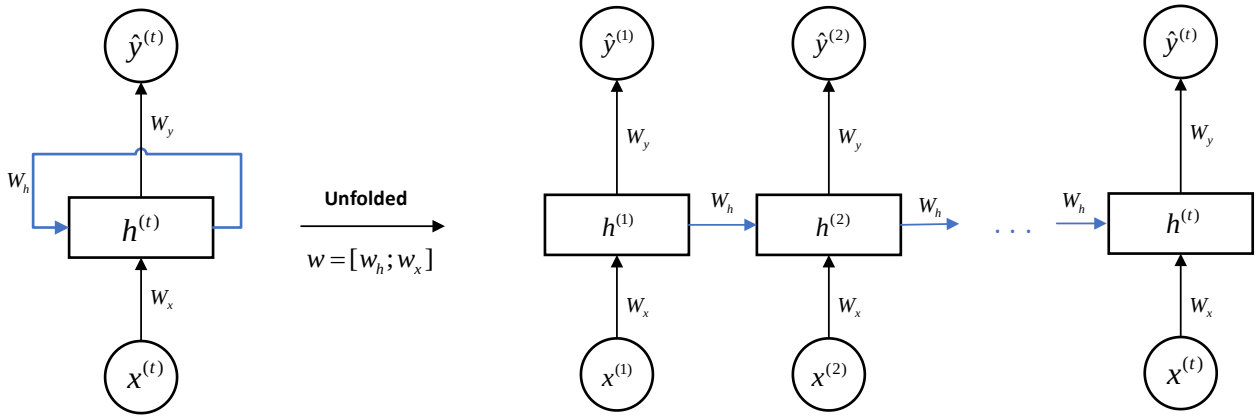

Figure 3.3: Unfolded structure of simple recurrent neural network [74]

Where W_x, W_h, W_y, x^t, and $\hat{y}^{(t)}$ are weight matrix from input to hidden layer, weight matrix from hidden layer to next hidden layer, weight matrix from hidden layer to output layer, input layer, and output layer respectively. The flow of information through hidden layer to the next hidden layer helps to persist memory.

As defined in fully connected layer, the computation in Recurrent neural network can be represented by using following formulas:

$$z_h^{(t)} = W_x x^{(t)} + W_h h^{(t-1)} + b_h \tag{3.16}$$

$$h^{(t)} = \sigma(z_h^{(t)}) \tag{3.17}$$

$$\hat{y}^{(t)} = \sigma(W_y h^{(t)} + b_y) \tag{3.18}$$

For the backpropagation, it uses Backpropagation Through Time(BPTT) defined in [75]. The overall loss function L is a sum of all the losses $L(t)$ generated in each time step t from $t=1$ to $t=T$ as shown below:

$$L = \sum_{t=1}^{t=T} L(t) \tag{3.19}$$

Computing the gradient through time using chain rule as follows

$$\frac{\partial L^{(t)}}{\partial W_h} = \frac{\partial L(t)}{\partial \hat{y}^{(t)}} \times \frac{\partial \hat{y}^{(t)}}{\partial h^{(t)}} \times \left(\sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial W_h} \right) \tag{3.20}$$

Traditional neural network-based deep learning algorithms have a problem in remembering previously learnt circumstances. In account of long-range dependencies, as shown in equation $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ has $t-k$ multiplication resulting in w^{t-k}. A small change in weight matrix w makes a significant difference in the computation of the gradient. For example, if $w=1$ and $T=50$ then the computation of the gradient becomes 117.4 making it exploding gradient, where as if $w=0.9$ then it becomes 0.00515 leads to vanishing gradient. This detailed explanation can be found in [76]. To handle this problem of "vanishing" or "exploding", there are several solutions available as described below:

- Gradient Clipping using Truncated backpropagation through time (TBPTT). This method clips the gradient above the given threshold. While updating the weight, the truncation restricts the number steps making it effective in backpropagation.

- Using some sophisticated recurrent neural network architecture (such as LSTM or GRU).

3.6 Long Short-Term Memory (LSTM)

Hochreiter and Schmidhuber [77] proposed a new type of recurrent neural network called Long Short Term Memory networks in 1997. Classic RNNs, in theory, may track arbitrary long-term relationships in input sequences. The gradients that are back-propagated can "vanish" or "explode" due to the computations needed in the creation of the backpropagation method, which uses finite-precision integers while training a standard (or "vanilla") RNN.

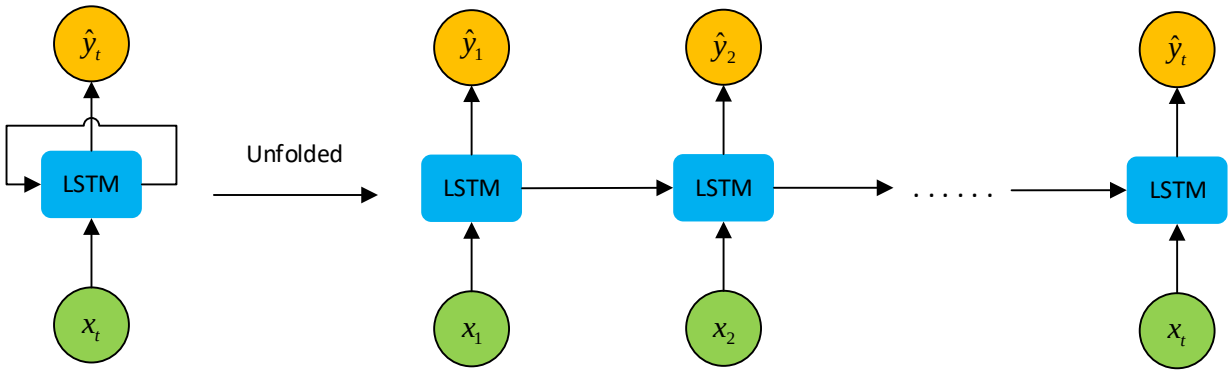

Figure 3.4: An unrolled structure of Long Short-Term Memory [78]

As depicted in Figure 3.4, The Long Short-Term Memory Network's essential components are a sequence input layer and an LSTM layer. A loop permits information to move from one stage to the next, and an LSTM looks at the input x and outputs $\hat{y}$ value.

The horizontal line extending through the top of the diagram, as illustrated in Figure 3.5, is responsible for information persistence over time. It runs straight down the entire chain with only a few linear interactions.

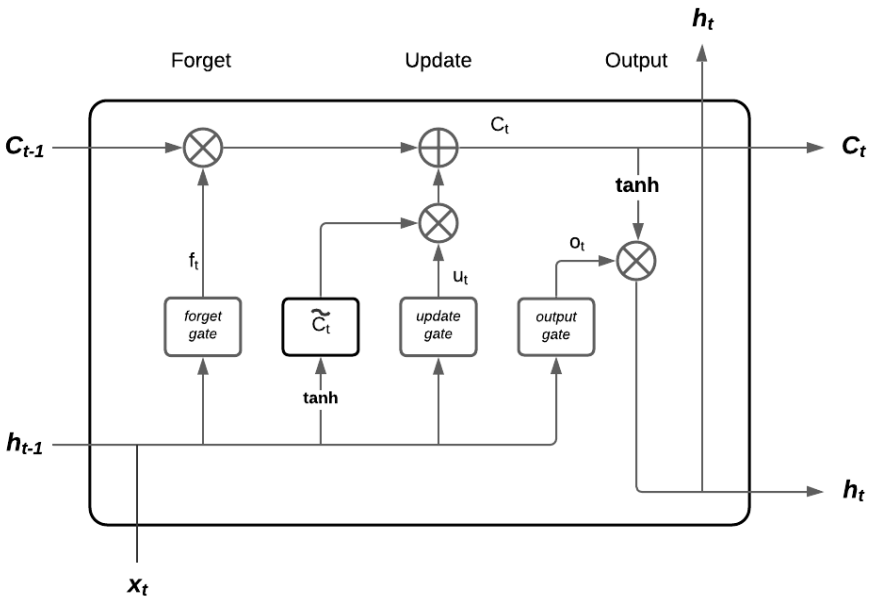

Figure 3.5: Structure of LSTM Cell [78]

The following equations can be used to express LSTM mathematically [79]:

$$f_i^{(t)} = \sigma(b_i^f + \sum_j W_{i,j}^f h_j^{t-1} + \sum_j R_{i,j}^f x_j^t) \tag{3.21}$$

$$u_i^{(t)} = \sigma(b_i^u + \sum_j W_{i,j}^u h_j^{t-1} + \sum_j R_{i,j}^u x_j^t) \tag{3.22}$$

$$o_i^{(t)} = \sigma(b_i^o + \sum_j W_{i,j}^o h_j^{t-1} + \sum_j R_{i,j}^o x_j^t) \tag{3.23}$$

$$c_i^{(t)} = f_i^{(t)} * c_i^{(t-1)} + u_i^{(t)} * \sigma(b_i^c + \sum_j W_{i,j}^c h_j^{t-1} + \sum_j R_{i,j}^c x_j^t) \tag{3.24}$$

$$\hat{y}_i^{(t)} = h_i^{(t)} = \tanh(c_i^{(t)}) * o_i^{(t)} \tag{3.25}$$

Here, forget gate, update gate, and output gate can be represented by $f_i^{(t)}, u_i^{(t)},$ and $o_i^{(t)}$, respectively. The input to the current cell is denoted as $x_j^{(t)}$ and output from the LSTM cell is

represented as h_j^{t-1}. Here, tanh stands for tangent activation function and σ signifies a sigmoid activation function. $w_{i,j}^f, w_{i,j}^\mu, w_{i,j}^c,$ and $w_{i,j}^o$ are weight matrix, and $b_j^f, b_j^u, b_j^c,$ and b_j^o refer to bias vectors. The flow of information is regulated by the multiplication of f_i^t and c_i^{t-1}. It also decides the usefulness of previous information in current prediction.

3.7 Gated Recurrent Unit (GRU)

Gated recurrent unit is another type of recurrent neural network introduced by Cho et al. [80] in 2014. A single gating unit controls both the forgetting factor and the choice to update the state unit at the same time in LSTM where as in GRU, there is no forget gate.

The following equations can be used to express GRU mathematically [79]:

$$r_i^{(t)} = \sigma(b_i^r + \sum_j R_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}) \tag{3.26}$$

$$u_i^{(t)} = \sigma(b_i^u + \sum_j R_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}) \tag{3.27}$$

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) * \sigma(b_i + \sum_j R_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}) \tag{3.28}$$

where $r_i^{(t)}$ and $u_i^{(t)}$ stands for reset and update gate. What information should be supplied to the output is determined by the update and reset gates in GRU. They are unique in that they can be trained to retain knowledge from the past without having to wash it away over time or eliminate information that is unrelated to the forecast. Figure 3.6 shows a detailed structure of a simplified GRU cell.

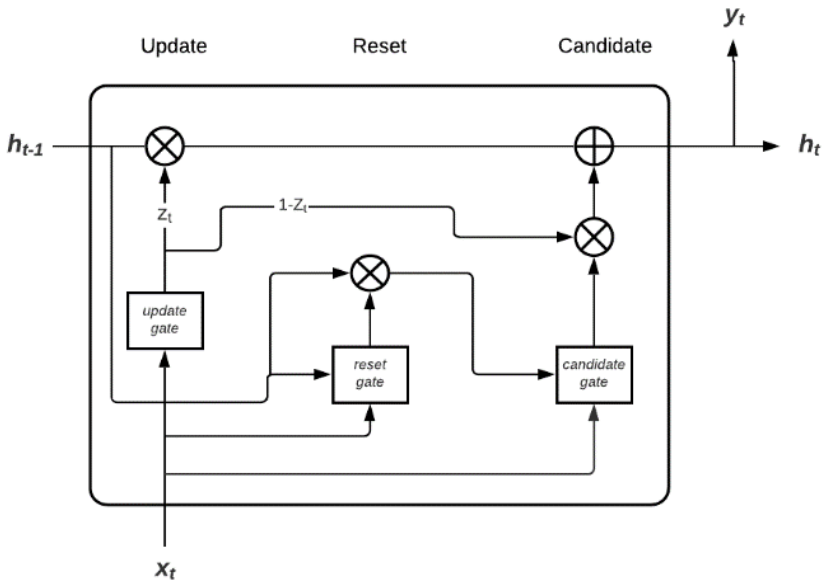

Figure 3.6: A simplified structure of GRU cell [80]

3.8 Fully Connected Layer

A fully connected layer is the basis of many important deep neural networks. The goal of the fully connected layer to approximate any function f^*. In the case of classification, it predicts the possible classes of the input whereas in regression task, it predicts the numerical value of the input. In most of the cases this layer is attached to end of deep neural network such as deep convolutional network or deep recurrent neural network.

The information moves through the function being evaluated from x, utilizing the intermediate computations that were used to define f, with a final output $\hat{y}$ known as forward propagation. In the classification task, number of neurons in the last layer depends on the number of classes whereas in the regression task, it depends on the number of outputs.

Considering input vector $x \in \mathbb{R}^m$, weight matrix $w \in \mathbb{R}^{i \times m}$, and bias vector $b \in \mathbb{R}^m$ then i^{th} input of the network becomes

$$\hat{y}_{i+1} = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + ... + w_m x_m + b_i) \tag{3.29}$$

Here σ is the activation function and $\hat{y}_i$ is the output of the i^{th} layer. In general, this equation can be written as following

$$\hat{y} = \sigma(w^T x + b) \tag{3.30}$$

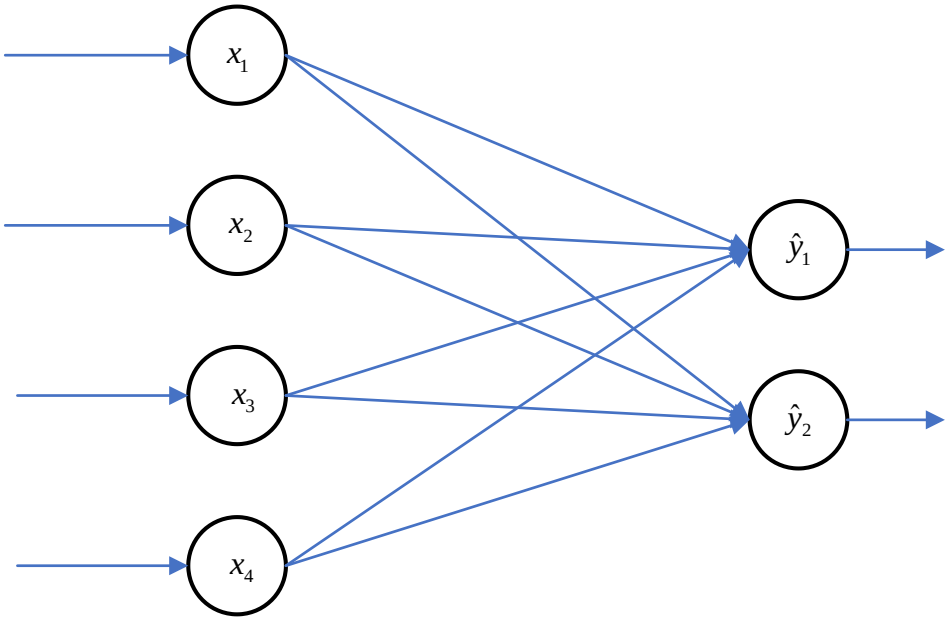

Figure 3.7: Architecture of fully connected layer

3.8.1 Cost Function

A cost function calculates the error between predicted output of the deep learning model ($\hat{y}$) with the actual output (y). The forward propagation continues in the training until the cost function L is calculated and it can be defined in several ways. In this study, the Mean Squared Error is utilized as a cost function which can be defined as follows:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.31}$$

3.8.2 Gradient Descent and Backpropagation

Gradient descent is an optimization technique that iteratively updates learnable parameters (such as weight) to minimize the loss calculated by loss function. Gradient of the loss function decides the direction of local or global minima. As shown in Figure 3.11, the sign of $\alpha \frac{\partial L}{\partial w_i}$ decides the direction of the minimal value.

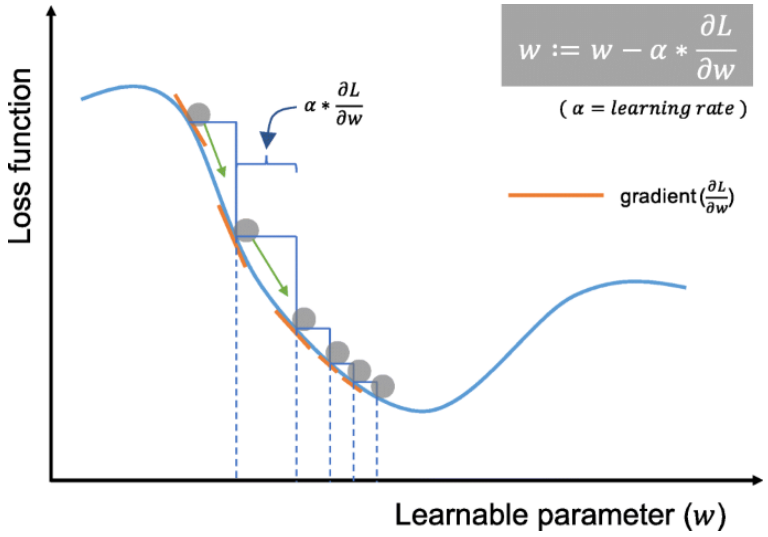

Figure 3.8: Gradient descent optimization [81]

Back-propagation is a technique for estimating gradients for weights in a neural network graph structure using an automated differentiation process. Rumelhart et al. discovered the backpropagation technique [72] and its broad application in neural networks in 1986. This approach has been independently rediscovered in a variety of fields. For a single input-output pair, backpropagation creates the gradient of the loss function with respect to the neural network

weights. In order to compute gradient, the backpropagation algorithm allows information to flow backward from the cost function across the network.

Computationally, calculating the gradient can be time-consuming. The backpropagation algorithm made it easy and inexpensive procedure. The derivative of the functions is calculated using the chain rule of calculus. These functions are created by combining various functions with known derivatives.

As shown in figure 3.7, if there are 4 inputs and 2 outputs, then using backpropagation, the weight can be updated in following way:

I. Complete forward propagation using steps defined in equation 3.31 and 3.32.

II. Calculate loss using mean square error

III. Compute the change in weight with respect to total loss using chain rule

IV. Update the weight using weight update formula as shown below

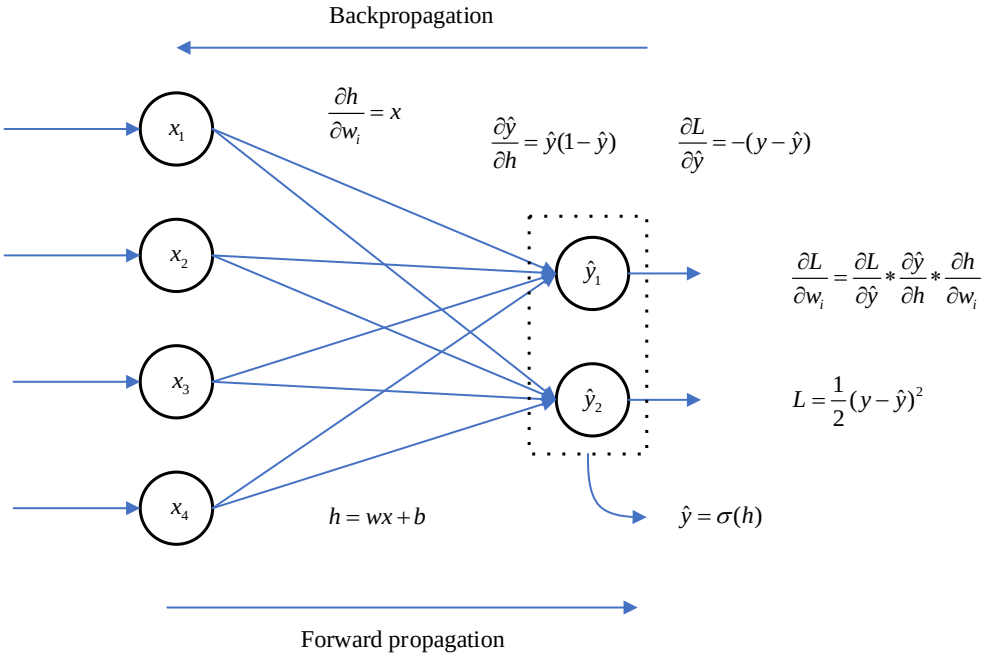

Figure 3.9: Mechanism of backpropagation in simple neural network

$$w_i^+ = w_i - \alpha \frac{\partial L}{\partial w_i} \tag{3.32}$$

where α, w_i, and $\frac{\partial L}{\partial w_i}$ are learning rate, i^{th} weight value, and rate of change in loss with respect to i^{th} weight, respectively.

3.9 Regularization

Choosing the appropriate depth for a neural network has always been a difficult task. In neural network, the weight matrix and the number of layers have higher influence in the accuracy and robustness of the network. These needs to be adjusted carefully to get optimal results. Small neural networks, with a small number of parameters, have poor performance, resulting in underfitting since learning the underlaying structure of a large dataset is extremely difficult. On the other hand, overfitting occurs when a big neural network learns more quickly and memorizes the training data, resulting in excellent performance on the training set. In some cases, this network shows poor performance on the test set making it mysterious.

There are various methods for overcoming this issue and obtaining consistent prediction on both the training and testing sets. Regularizations are a collection of strategies for reducing error and avoiding overfitting by fitting a function correctly on the provided training set. In the case of neural networks, to do well on the training set, one solution is to design a network with a large number of layers (somewhat bigger than what is really required). Multiple regularization techniques, such as Early stopping, L2 regularization, and dropout, can be used to tackle the overfitting problem at this stage.

The next sections discuss about Early stopping, L2 regularization, and dropout in detail to provide a comprehensive idea about the regularization techniques and its benefit in neural networks.

3.9.1 Early Stopping

This method is most popular in regularization due to its effectiveness and simplicity. When training a neural network, usually the validation loss steadily decreases over time but in some exceptional cases the validation loss rises again. Thus, there is a chance of getting optimized parameters in between which performs better or consistent on validation set.

This problem can be handled by storing model parameters in every epoch and comparing the performance (the error on validation set). When no parameters have improved beyond the best recorded validation set error over a pre-defined epoch, the method ends. This approach is also helpful to get an optimized hyperparameters. The implementation details are specified in Algorithm 3.3 and visual description is shown in Figure 3.9.

Algorithm 3.1: *Early Stopping Implementation in Long Short-Term Memory*

Input: *Initial parameters* $\boldsymbol{P_0}$, ***loss_overall***, ***loss_epoch***
Output: *Best parameters* ***P_best***
Let n be the number of steps between evaluation, and p is the patience number (till what number to check loss before giving up)
Let P_0 *be the initial hyperparameters (num_of_epoch, batch_size, learning_rate), P_best is the best hyperparameters evaluated in the whole process, loss_overall is the overall best loss, and loss_epoch is the loss calculated in every epoch.*
LSTM is the function that accepts parameters and returns loss for single epoch.

```
for i = 1 to n
    if P_best == null
        loss_epoch ⟵ LSTM(P_0)
    else
        loss_epoch ⟵ LSTM(P_best)
    end
    save parameters of this epoch as p_i
    if i > p
        break
    else
        While loss_epoch < loss_overall do
            loss_overall ⟵ loss_epoch
```

```
13                    P_best ⟵ p_i
14                end
15            end
16        end
```

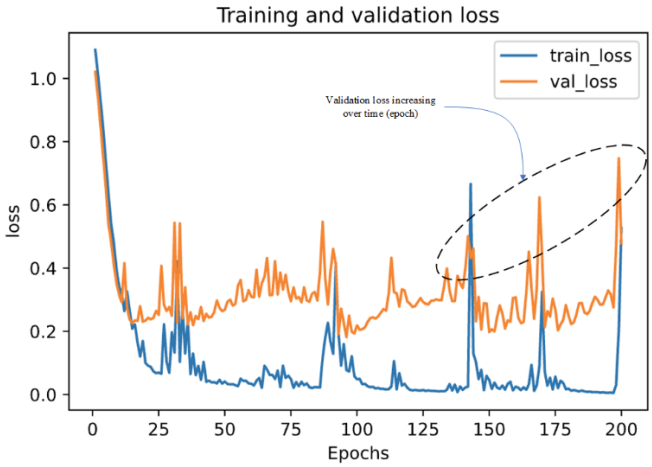

Figure 3.10: Inconsistency in validation loss [82]

3.9.2 L2 Parameter regularization

L2 is the common parameter norm penalty which is also known as weight decay, rigid regression, and Tikhonov regularization. By including a regularization component L2 in the goal function, this regularization method brings the weights closer to the origin.

$$L2 : \| w \|_2^2 = \lambda \sum_{j=1}^{m} w_j^2 \tag{3.33}$$

It is crucial how λ is selected in this case. If λ is excessively big, it will contribute too much weight to the equation, resulting in under-fitting. This method is highly effective in avoiding the problem of over-fitting.

3.9.3 Dropout

Dropout, a popular regularization approach that performs remarkably well for deep neural networks, has emerged in recent years. During the training phase of a neural network, dropout

occurs when a percentage of the hidden units is randomly discarded with probability p_{drop} at each iteration. The common choice of dropout rate is 0.5. The weight associated with the remaining neurons is rescaled to account for the missing neurons when a specific proportion of input neurons is removed.

As a result of this unpredictable dropout, the network is forced to learn complex representation of the data. In dropout, the hidden unit may be turned off (with a use of p_{drop}) anytime during the training which makes it unpredictable at any point of training. This nature allows the network to learn more complex and robust patterns from the data. Overfitting may be easily avoided with this random dropout.

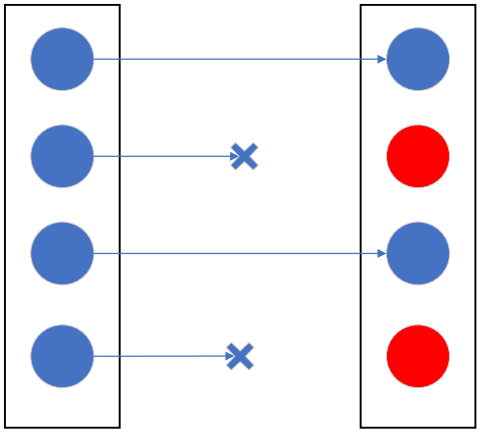

Figure 3.11: Dropout with probability p_{drop} 0.5

As shown in Figure 3.10, half of the input randomly deactivated at the training time. In the evaluation, all the inputs must be active.

3.10 Adam Optimization

Optimization techniques are useful to get optimal hyperparameters in the neural network. Stochastic Gradient Descent (SGD), SGD with momentum, AdaGrad, RMSProp, and Adam are

some of the optimization algorithms available to get optimized hyperparameters. Gradient Descent which is the simplest algorithm for optimization is already discussed in section 3.8.2. This section discusses about adaptive learning rate-based Adam optimization algorithm. Adam optimization is the most common optimization algorithm used in the neural networks. This adaptive learning rate optimization algorithm was first introduced by Kingma and Ba [83] in 2014.

In Adam, fist order moments (s_t) and second order moments (r_t) can be defined as follows:

$$s_t = \beta_1 s_{t-1} + (1-\beta_1) g \tag{3.34}$$

$$r_t = \beta_2 r_{t-1} + (1-\beta_2) g \square\ g \tag{3.35}$$

$$\hat{s}_t = \frac{s_t}{1-\beta_1} \tag{3.36}$$

$$\hat{r}_t = \frac{r_t}{1-\beta_2} \tag{3.37}$$

$$\Delta w_t = -\alpha \frac{\hat{s}_t}{\sqrt{\hat{r}_t} + \delta} \tag{3.38}$$

$$w_{t+1} = w_t + \Delta w_t \tag{3.39}$$

Here, β_1 and β_2; δ; α; and g are exponential decay rate, constant for stabilization, initial learning rate; and gradient at time *t*, respectively.

Momentum is immediately included as an estimate of the gradient's first-order moment in Adam. It accounts for its initialization at origin by applying bias adjustments to the estimations of both

first order and second-order moments. With a few key differences from RMSProp and momentum, this method performs well on to get optimized learning rate.

The next chapter discusses about the implementation method used in this research and simulation results.

Chapter 4

Implementation and Simulation Results

4.1 Introduction

In this chapter, the detailed structure and development of the deep learning-based adaptive sliding mode controller, the deep learning workflow, and simulation results are presented. This chapter is organized as follows. Section 4.2 presents implementation of the proposed controller using deep learning. In this method, the adaptive sliding mode controller is used for dataset generation. Section 4.3 explains the workflow of deep learning which includes dataset generation; dataset cleaning and normalization; model training and evaluation; and deployment. An algorithm for training the network with multiple hyperparameters is presented and an end-to-end deep learning pipeline is discussed. Section 4.3 explains the simulation results constructed on a two degree of freedom (DOF) serial robot manipulator to investigate the different aspects of the proposed controller. An optimized deep learning model is used in the simulation.

4.2 Implementation of Proposed Controller

The exact dynamic model of the manipulator is not known because of uncertainty and external disturbance. In this context, the aproximatley known dynamic model of a 2-DOF robot shown in Figure 4.1 can be written as follows [22]:

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}=\begin{bmatrix}\hat{M}_{11} & \hat{M}_{12}\\ \hat{M}_{21} & \hat{M}_{22}\end{bmatrix}\begin{bmatrix}\ddot{q}_1\\ \ddot{q}_2\end{bmatrix}+\begin{bmatrix}\hat{C}_{11} & \hat{C}_{12}\\ \hat{C}_{21} & \hat{C}_{22}\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix}+\begin{bmatrix}\hat{g}_1\\ \hat{g}_2\end{bmatrix} \tag{29}$$

$$\hat{M}_{11}=a_1+2a_3\cos(q_1)+2a_4\sin(q_2),\hat{M}_{12}=a_2+a_3\cos(q_2)+a_4\sin(q_2),\ \hat{M}_{21}=\hat{M}_{12},\hat{M}_{22}=a_2,$$

$$\hat{C}_{11}=-h\dot{q}_2+v1,\ \hat{C}_{12}=-h(\dot{q}_1+\dot{q}_2),\ \hat{C}_{21}=h\dot{q}_1,\ \hat{C}_{22}=v2,$$

$$h=a_3\sin(q_2)-a_4\cos(q_2),\ \hat{g}_1=a_5\cos(q_1)+a_6\cos(q_1+q_2),\ \hat{g}_2=a_6*\cos(q_1+q_2)$$

$$a_1 = I_1 + m_1 l_{c1}^2 + I_e + m_e l_{ce}^2 + m_e l_1^2,\ a_2 = I_e + m_e l_{ce}^2,\ a_3 = I_e + m_e l_1 l_{ce}\cos(30),\ a_4 = m_e l_1 l_{ce}\sin(30)$$

$$a_5 = (m_1 l_{c1} + m_e l_1)g,\ a_6 = \frac{m_e l_{ce} g}{\cos(\frac{\pi}{6})}$$

$$K = \begin{bmatrix} 400 & 0 \\ 0 & 200 \end{bmatrix},\ \zeta = \begin{bmatrix} 25 & 0 \\ 0 & 15 \end{bmatrix},\ \Gamma = \begin{bmatrix} 20000 & 0 \\ 0 & 10000 \end{bmatrix},\ \Lambda = 50,\ d(t) = \begin{bmatrix} 250 & 0 \\ 0 & 100 \end{bmatrix}$$

The parameters of the two-link robot manipulator are mentioned in the Table 4.1.

Table 4.1: 2-DOF robot parameters (As mentioned in Figure 4.1)

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Link 1 mass m_1	1 kg	Initial position q_1	0.1 radians	l_{c1}	0.5 m
m_e	2 kg	Initial position q_2	0.05 radians	g	9.81 m/s^2
Link 1 length l_1	0.51 m	Moment of inertia I_1	0.12 $kg\,m^2$	v1	0
l_{ce}	0.6 m	I_e	0.25 $kg\,m^2$	v2	2.7

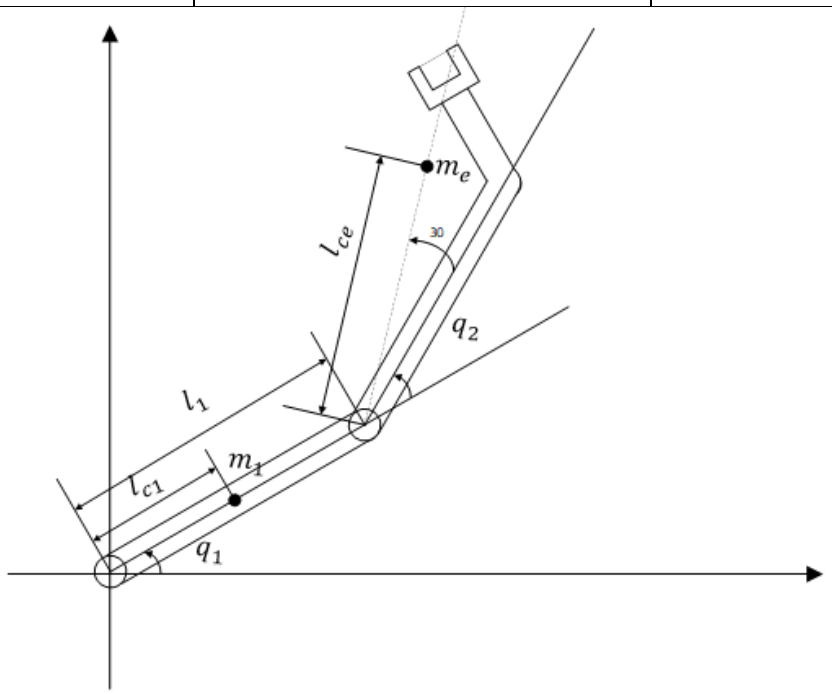

Figure 4.1: 2-DOF serial robot manipulator [31].

As discussed in chapter 3, the adaptive sliding mode controller can be defined using following equation [70]:

$$\tau = \hat{M}\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{g}(q) - KS - \Gamma\int S\,dt \tag{4.1}$$

Where K and S are symmetric positive definite diagonal matrix and sliding variable, respectively. For the detailed information, please refer Section 3.3.

4.3 Deep Learning Workflow

An end-to-end deep learning pipeline implemented in this research. The final goal of this implementation is to get an optimized deep learning model. Finding optimized hyperparameters is a challenging task which can be automated. In this research work, a common function handles data cleaning and normalization on training and validation dataset. A separate algorithm is accepting dataset and hyperparameters and train Long Short-Term Memory and Gated Recurrent Unit according to it.

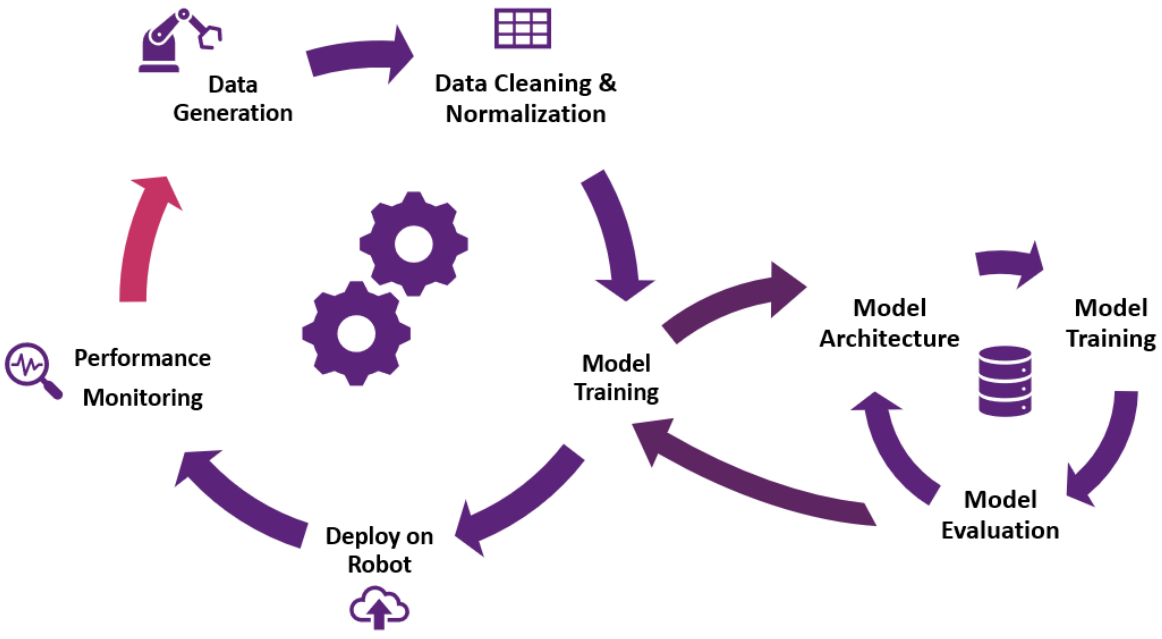

Figure 4.2: An end-to-end deep learning pipeline implemented in this research

Following sections explains the step-by-step implementation of this deep learning workflow.

4.3.1 Dataset Generation

The adaptive sliding mode controller is used to generate dataset which includes position, velocity, and acceleration as input and torque value as output. For the input, actual values of position, velocity, and acceleration are considered. The calculation of acceleration is done by using following formula:

$$\ddot{q}_r = \ddot{q}_d - 2\zeta\dot{e} - \zeta^2 e \tag{4.2}$$

Where $\ddot{q}_r$, $\ddot{q}_d$, $\dot{e}$, e, and ζ are reference acceleration, desired acceleration, velocity error, position error, and diagonal positive definite matrix, respectively. The sampling time (TS) is set at 0.003 to generate training. The initial values (robot parameters) for dataset generation and formulas to calculate inverse kinematic of the robot are defined in section 4.2. The sample of generated dataset is presented in Figure 4.3.

	q1	q2	qv1	qv2	qar1	qar2	t1	t2
251	-3.163969	-2.133623	-0.350336	0.805638	245.493018	162.563782	135.567418	62.329503
252	-3.163373	-2.130105	0.015608	1.050269	247.797888	160.085312	137.131315	61.976564
253	-3.161669	-2.125870	0.384040	1.291189	249.956532	157.704114	138.940803	61.684965
254	-3.158848	-2.120928	0.754758	1.528550	251.961174	155.425192	140.989324	61.455186
255	-3.154906	-2.115290	1.127549	1.762511	253.804787	153.252279	143.271033	61.287304
...	...	...	...	...	...	...	...	...
1661	-0.520205	-0.657658	-3.600555	-4.805690	34.093527	44.447110	56.113078	15.793840
1662	-0.530769	-0.671784	-3.547699	-4.741172	34.552154	45.992024	56.575062	16.146414
1663	-0.541171	-0.685708	-3.494005	-4.674644	35.006452	47.507357	56.985709	16.494196
1664	-0.551407	-0.699424	-3.439489	-4.606125	35.457207	48.990515	57.346255	16.836928
1665	-0.561477	-0.712925	-3.384165	-4.535643	35.905156	50.439036	57.658076	17.174351

42450 rows × 8 columns

Figure 4.3: Dataset generated using adaptive sliding mode controller

Here first six columns are input and last two are output for the neural network.

4.3.2 Data Cleaning and Normalization

Data cleaning is an essential part in deep learning. Although deep learning has capability to learn meaningful representation from given data, usage of data cleaning significantly improves the performance of a deep learning model in some cases. There are numerous ways to clean the data depending on the data type, size, and structure. In this research, all the columns of the dataset include only numerical values which can be cleaned as follows:

- Consistency in data: The degree to which data is consistent, either within the same data set (same type of trajectory data) or across various data sets (various types of data such as square wave, sine wave etc.), is referred to as consistency. It is important to note constancy in dataset while merging multiple trajectory data. This can be handled via good data distribution among multiple datasets. Data
- Duplicate values: When combining data from different sources, duplicates in data is a common problem. According to the use case, this should be handled. In the case of combining same trajectory data, duplicates may have a negative effect on the prediction. It should be removed if such a case.

Normalization is very important in neural networks. The goal of normalizing data is to give all qualities the same weight. Normalization can be accomplished in a variety of ways. The z-score normalization approach is utilized to standardize the data in this study. The z-score normalization can be expressed by following equations:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.3}$$

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N} | x_i - \bar{x} |^2}{N-1}} \tag{4.4}$$

$$x_i^{'} = \frac{x_i - \bar{x}}{\sigma_x} \tag{4.5}$$

Where x_i, $\bar{x}$, σ_x, and N are sample at i^{th} position, mean value of x, standard deviation of x, and number of samples, respectively. When the real minimum and maximum of attribute x are unknown, this approach of normalizing is beneficial.

4.3.3 Deep Learning Model Training

This section discusses about the deep learning model training and selection process. The deep learning model can be trained on various parameters called as "hyperparameters". The selection of best hyperparameters in the training could be time consuming and require extensive computation power. In this research, the model selection process is based on the error rate calculated with root-mean-square error (RMSE).

Following section shows step by step evaluation of Long Short-Term Memory and Gated Recurrent Unit on different condition such as change in learning rate, number of layers, normalization type, batch size, and epoch. The architecture of deep learning model is shown in Figure 4.4. The LSTM and GRU have variable of layers and other layers are fixed. Dropout layer is used to eliminate overfitting and 2 neurons in last fully connected layer is responsible for joint 1 and joint 2 torque. The mean squared error (MSE) is used as a loss function in this research. Each layer of LSTM or GRU has a fixed 200 neurons.

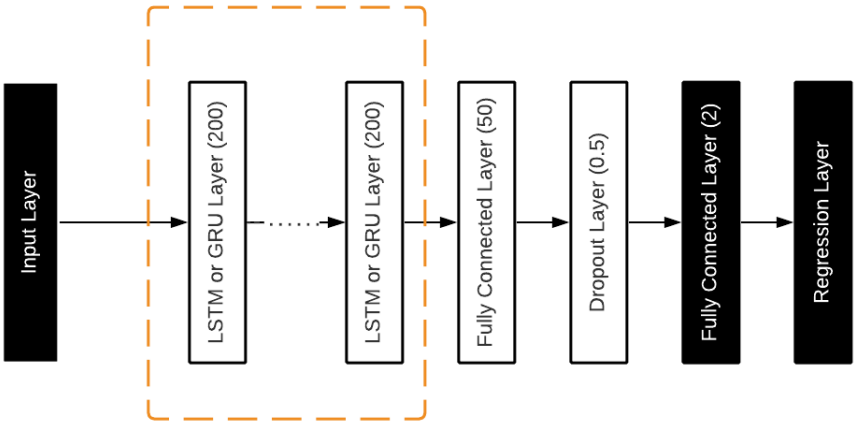

Figure 4.4: A deep learning architecture of LSTM or GRU

A. Effect of number of layers

Table 4.2: LSTM model training on variable number of layers

LSTM									
No	LSTM Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.003	128	100	0.0001	15.95	9.16	12.55
2	2	128,128	0.003	128	100	0.0001	15.38	10.38	12.88
3	3	128,128,128	0.003	128	100	0.0001	17.81	9.39	13.6
4	4	128,128,128,128	0.003	128	100	0.0001	14.82	5.63	10.22

Table 4.3: GRU model training on variable number of layers

GRU									
No	GRU Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.003	128	100	0.0001	23.33	13.54	18.43

2	2	128, 128	0.003	128	100	0.0001	15.60	12.10	13.85
3	3	128,128,128	0.003	128	100	0.0001	16.45	11.93	14.19
4	4	128,128,128,128	0.003	128	100	0.0001	20.16	12.58	16.37

B. Effect of different learning rate

Table 4.4: LSTM model training on different learning rate

LSTM									
No	LSTM Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.001	128	100	0.0001	15.32	9.39	12.35
2			0.003	128	100	0.0001	15.95	9.16	12.55
3			0.005	128	100	0.0001	13.53	6.39	9.96
4			0.007	128	100	0.0001	17.19	10.14	13.66
5			0.0001	128	100	0.0001	22.37	11.33	16.85

Table 4.5: GRU model training on different learning rate

GRU									
No	GRU Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.001	128	100	0.0001	19.53	12.53	16.03
2			0.003	128	100	0.0001	23.33	13.54	17.93
3			0.005	128	100	0.0001	16.16	10.77	13.46
4			0.007	128	100	0.0001	17.41	13.89	15.65
5			0.0001	128	100	0.0001	21.76	14.05	17.90

C. Effect of different batch size

Table 4.6: LSTM model training on different batch size

LSTM									
No	LSTM Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.003	72	100	0.0001	15.95	9.16	12.55
2			0.003	128	100	0.0001	15.95	9.16	12.55
3			0.003	256	100	0.0001	15.95	9.16	12.55
4			0.003	512	100	0.0001	15.95	9.16	12.55

Table 4.7: GRU model training on different batch size

GRU									
No	GRU Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.003	72	100	0.0001	23.33	13.54	18.43
2			0.003	128	100	0.0001	23.33	13.54	18.43
3			0.003	256	100	0.0001	23.33	13.54	18.43
4			0.003	512	100	0.0001	23.33	13.54	18.43

D. Effect of Normalization

Table 4.8: LSTM model training on different normalization

LSTM					
No	LSTM Layers	Neurons in each layer	Normalization	Torque RMSE		Average RMSE
				Joint 1	Joint 2	

1	1	128	z-score	15.95	9.16	12.55
2			min-max	57.69	37.32	47.50

Table 4.9: GRU model training on different normalization

GRU						
No	GRU Layers	Neurons in each layer	Normalization	Torque RMSE		Average RMSE
				Joint 1	Joint 2	
1	1	128	z-score	23.33	13.54	18.43
2			min-max	48.12	29.25	38.68

E. Effect of different epoch

Table 4.10: LSTM model training on different epoch

LSTM									
No	LSTM Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.003	128	100	0.0001	15.95	9.16	12.55
2			0.003	128	200	0.0001	10.39	6.12	8.25
3			0.003	128	300	0.0001	7.53	4.68	6.10
4			0.003	128	400	0.0001	7.83	3.73	5.78

Table 4.11: GRU model training on different epoch

GRU									
No	GRU Layers	Neurons in each layer	Hyperparameters				Torque RMSE		Average RMSE
			Learning Rate	Batch Size	Epoch	L2 Regularization	Joint 1	Joint 2	
1	1	128	0.003	128	100	0.0001	23.33	13.54	18.43

2			0.003	128	200	0.0001	17.51	9.78	13.64
3			0.003	128	300	0.0001	12.39	9.36	10.87
4			0.003	128	400	0.0001	15.21	9.56	12.38

As shown above from Table 4.2 to Table 4.10, training the deep learning model on various hyperparameters gives different results. The optimized results can be calculated with root-mean-square error which can be defined as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}} \tag{4.6}$$

Where N , x_i , and $\hat{x}_i$ are number of non-missing data points, actual observation, and predicted value, respectively. The best model is used to predict the torque value. Analyzing root-mean-square error, the best combination of hyperparameters for LSTM is with 400 epoch, 0.005 initial learning rate, 128 batch size, and 4 layers. In the same way, the finest hyperparameters combination for GRU is 300 epoch, 0.005 initial learning rate, 128 batch size, and 2 layers. Furthermore, increasing epoch for certain point has positive effect on LSTM and GRU. Algorithm 4.1 describes a detail description of a common function to get best hyperparameters.

Algorithm 4.1: *A common function to get best combination of hyperparameters*

Input: dataset**, **model, parameter_combinations
Output: best_combination
Let n be the number of hyperparameters combinations
Let Pi be the hyperparameters (num_of_epoch, batch_size, initial_learning_rate, normalization, number_of_layers) at i^{th} position, param_best is the best hyperparameters evaluated in the whole process. Parallel for is the parallel for loop to speed up the whole process using multiple gpu.
LSTM or GRU is the function that accepts parameters and returns root-mean-square error.

1 ***Parallel for*** *i = 1* ***to*** *n*
2 *current_param ⟵ parameter_combinations[i]*
3 ***if*** *model == LSTM*

```
        rmse,state ⟵ LSTM(dataset, current_param )
    else
        rmse,state ⟵ GRU(dataset, current_param )
    end
    Generate model name with unique id
    save rmse and state of LSTM or GRU at i^th position in Datatable
end
for i=1 to length of Datatable
    if Datatable[i].model_name == model
        Filter all the rows using model name
        Filter data and get lowest rmse by epoch wise, initial learning rate wise,
        batch_size wise, number_of_layers wise
        Compare it with current best_combination and overwrite it if rmse is low
    end
end
```

As discussed earlier, in this research, τ_{LSTM} is used for the predicted torque value from LSTM and τ_{GRU} for predicted torque value from GRU. For more details, please refer to section 3.3. The combination of LSTM with ASMC is defined as "LSTM + ASMC" and the combination of GRU with ASMC is denoted as "GRU + ASMC" in the simulation figures.

4.4 Simulation Results

This section discusses about the simulation results using Long Short-Term Memory and Gated Recurrent Unit on various aspects. In the beginning, ASMC results are presented to give understanding of error and required torque. The main goal of this research is to replace a model base component from ASMC with the model built based on deep recurrent neural networks. The simulation results included in this section are with 0.001 sampling time for 10 seconds. The deep learning models are trained on 27 trajectories and tested on 3 trajectories. In the last, tracking error is presented using RMSE in Table 4.12.

4.4.1 ASMC Only

In this section, the results are generated with highly optimized adaptive sliding mode controller. All the robot parameters used in the simulation are already discussed in section 4.2.

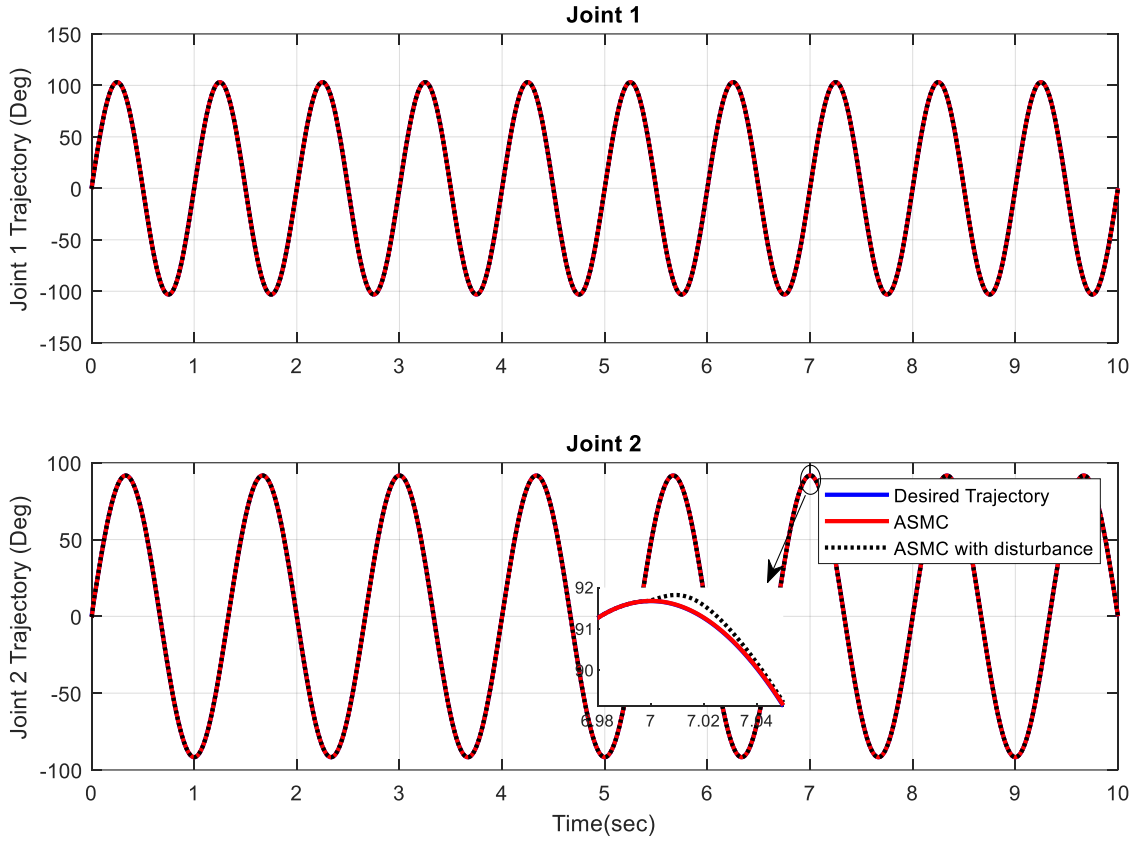

Figure 4.5: ASMC trajectory tracking with and without disturbance.

As shown in Figure 4.6, the adaptive sliding mode controller generates higher torque value to get lower error rate. The peak value of torque is much higher and should be handled with corrector before deploying in the actual robot. The disturbance is applied at $t = 7$ leading to a small jump in the torque value and error. Over time the ASMC controller handle the disturbance and follows the desired trajectory again. The error rate in ASMC is around +0.35 degree to -0.25 degree as shown in Figure 4.7.

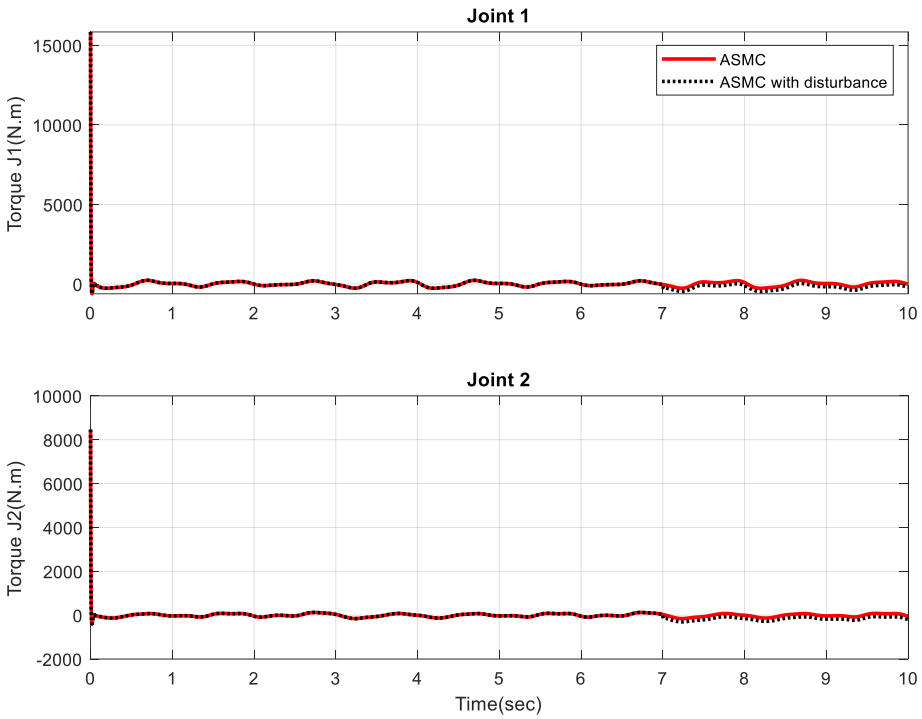

Figure 4.6: ASMC torque with and without disturbance.

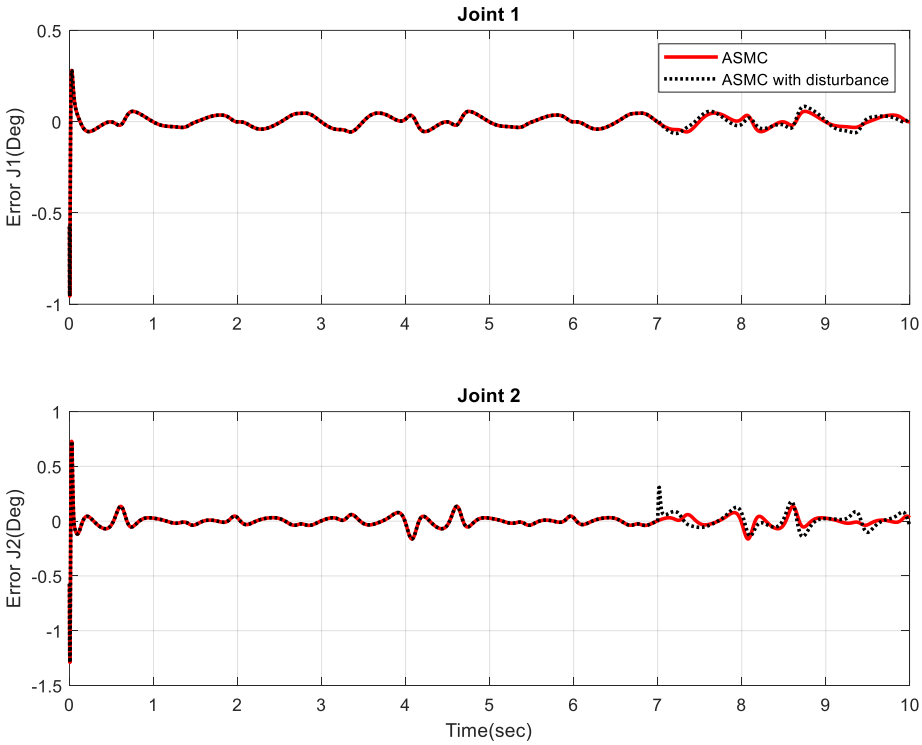

Figure 4.7: ASMC error comparison with and without disturbance.

The next sections present the results from LSTM only, GRU only, LSTM with ASMC, GRU with ASMC, and comparison of all of them.

4.4.2 LSTM Only

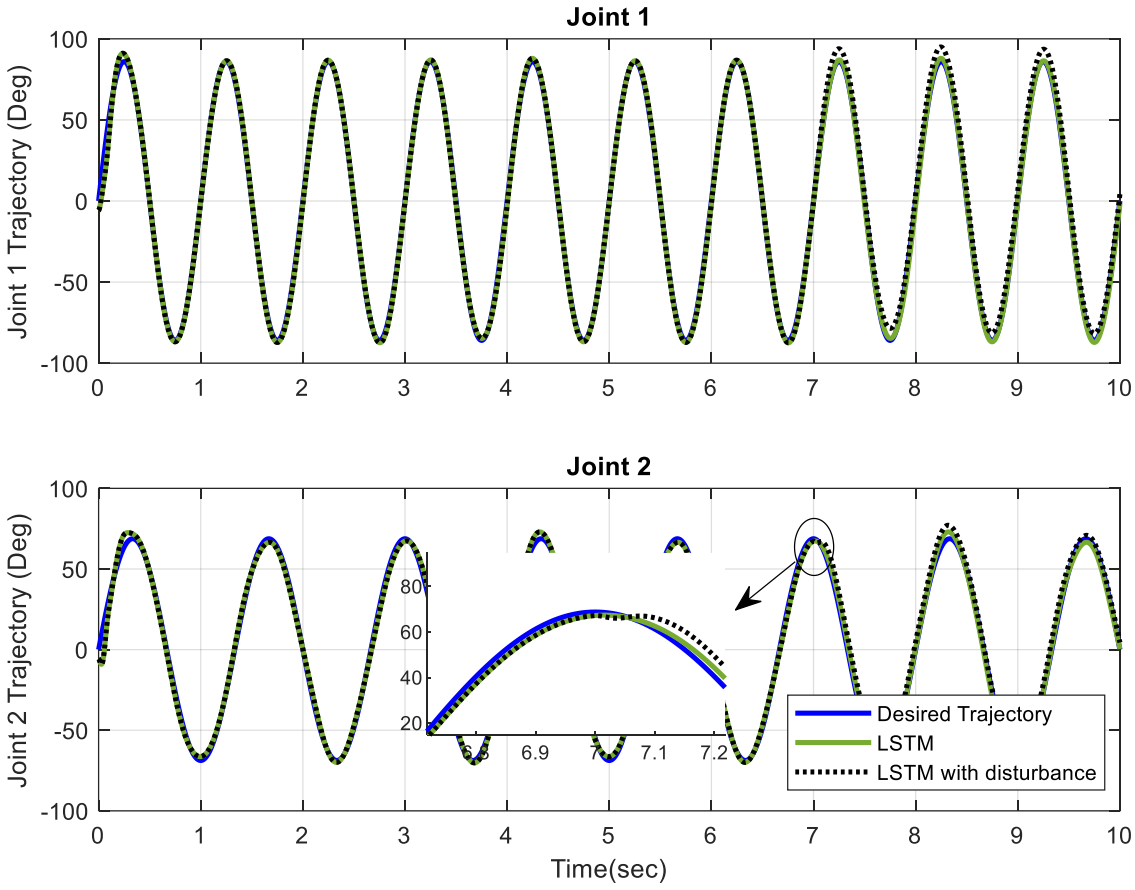

Figure 4.8: LSTM trajectory tracking results with and without disturbance.

As shown in Figure 4.8, the LSTM model follows trajectory accurately with its knowledge about the trajectory data. In the same way, as shown in Figure 4.11, GRU model also follows trajectory precisely. Applying disturbance at $t = 7$ significantly affect the tracking resulting a small jump in error. After $t = 7$, LSTM and GRU take some time to follow the trajectory again. This can be avoided with the help of adaptive sliding mode controller or online training.

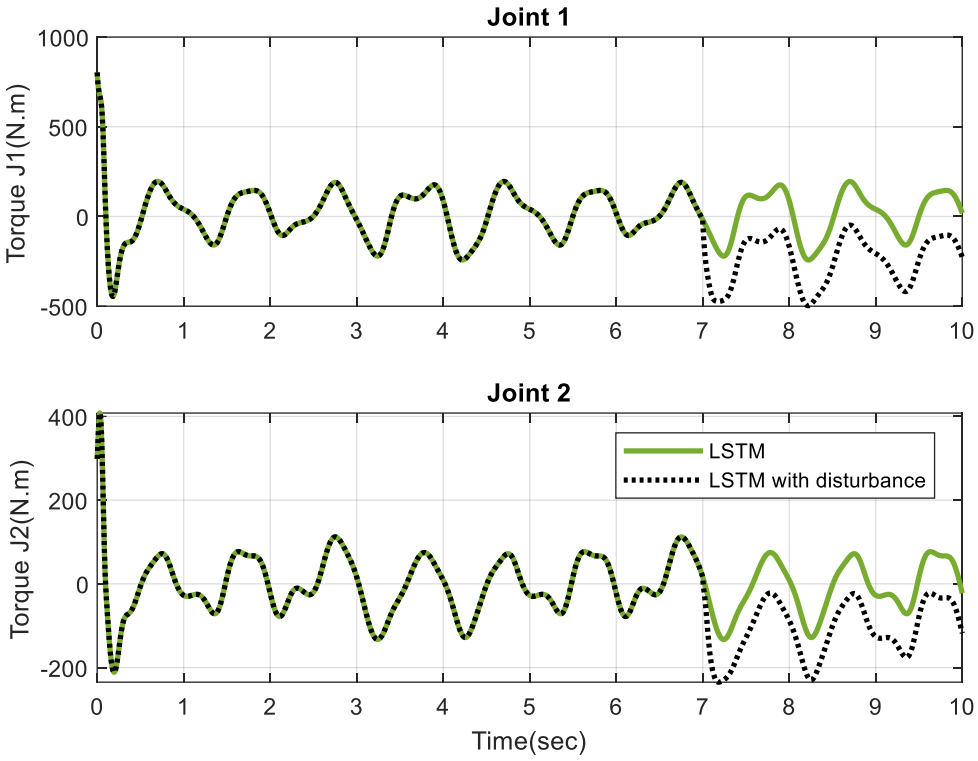

Figure 4.9: LSTM torque results with and without disturbance

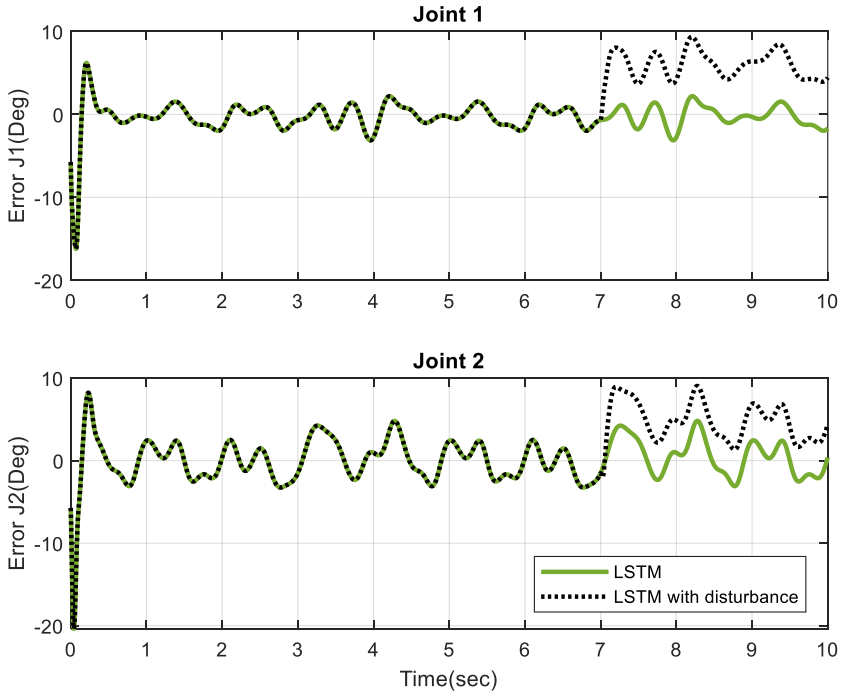

Figure 4.10: LSTM error results with and without disturbance

4.4.3 GRU Only

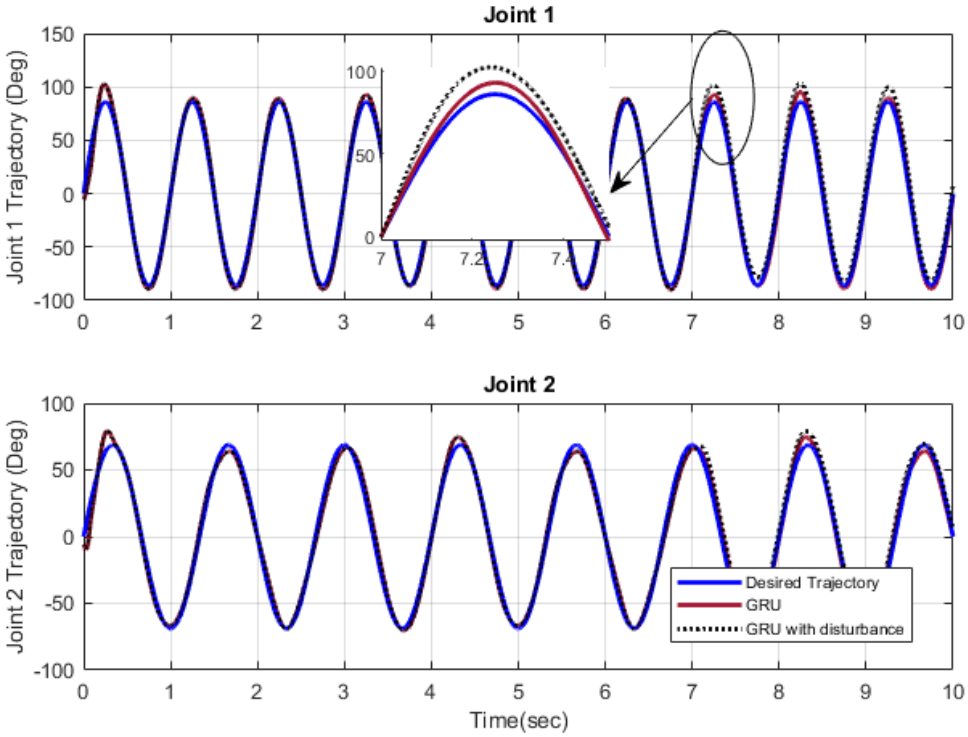

Figure 4.11: GRU trajectory tracking results with and without disturbance

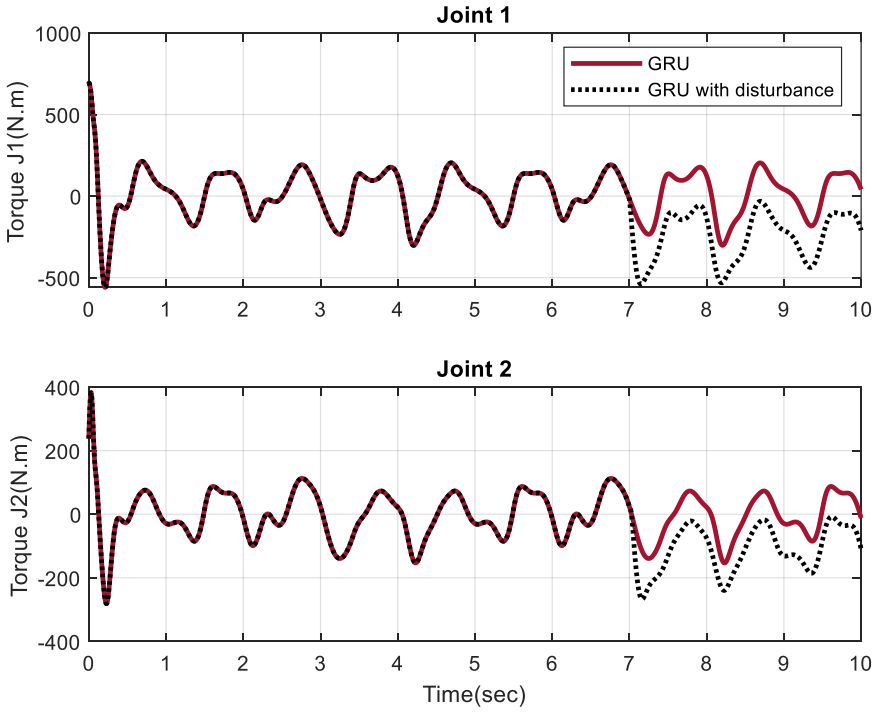

Figure 4.12: GRU torque results with and without disturbance

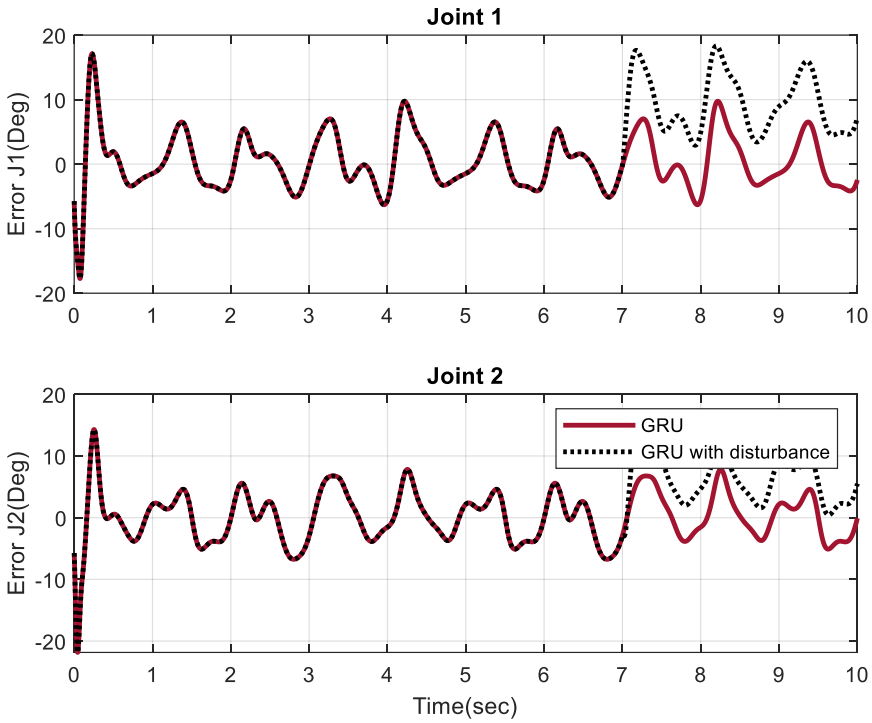

Figure 4.13: GRU error results with and without disturbance

4.4.4 LSTM + ASMC with and without disturbance

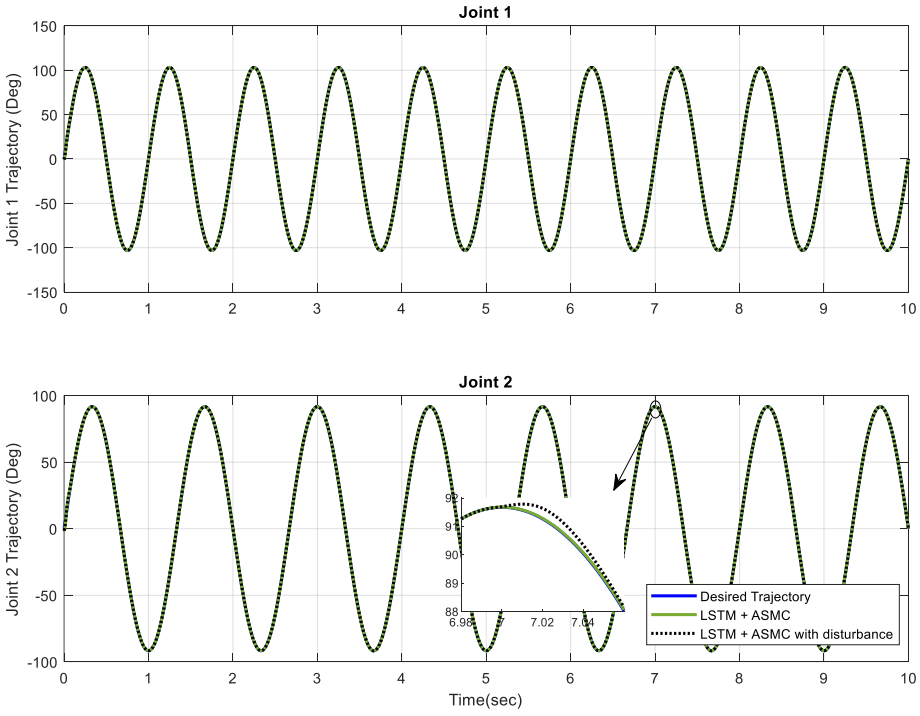

Figure 4.14: LSTM with ASMC trajectory tracking with and without disturbance

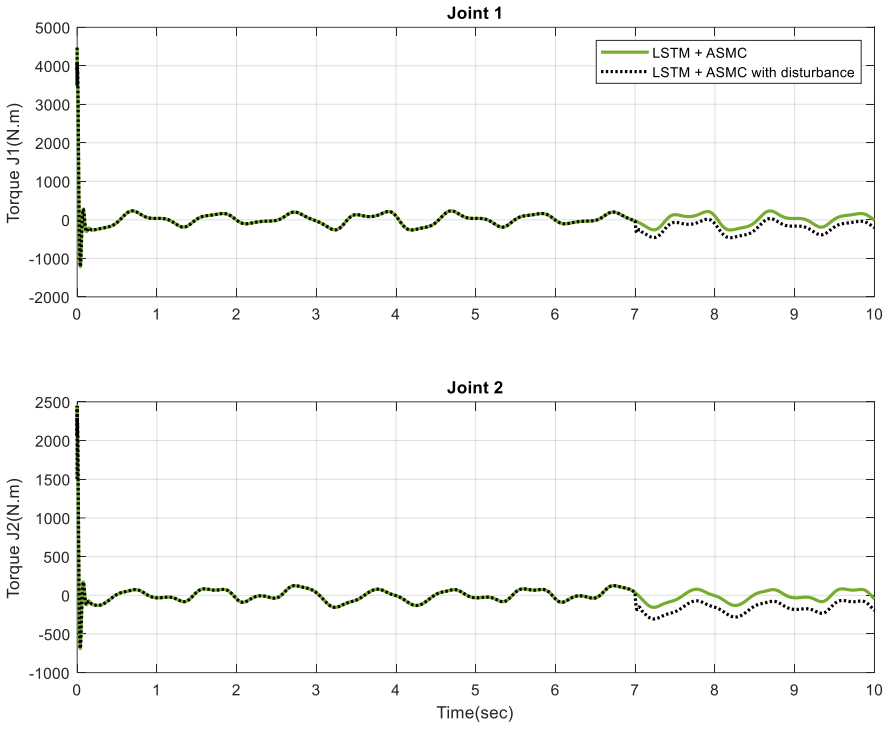

Figure 4.15: LSTM with ASMC torque with and without disturbance

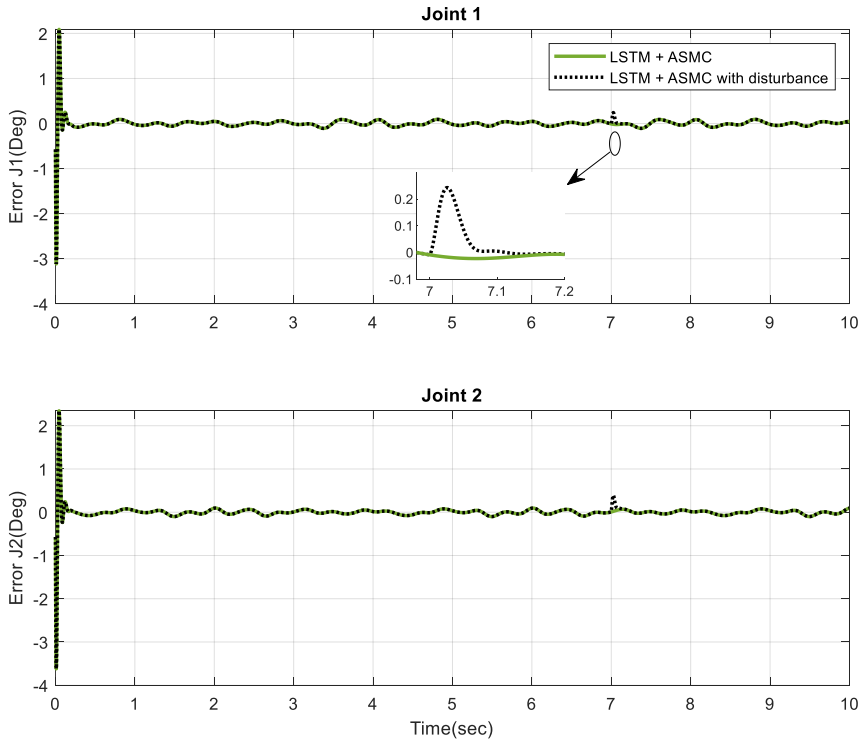

Figure 4.16: LSTM with ASMC error with and without disturbance

4.4.5 GRU + ASMC with and without disturbance

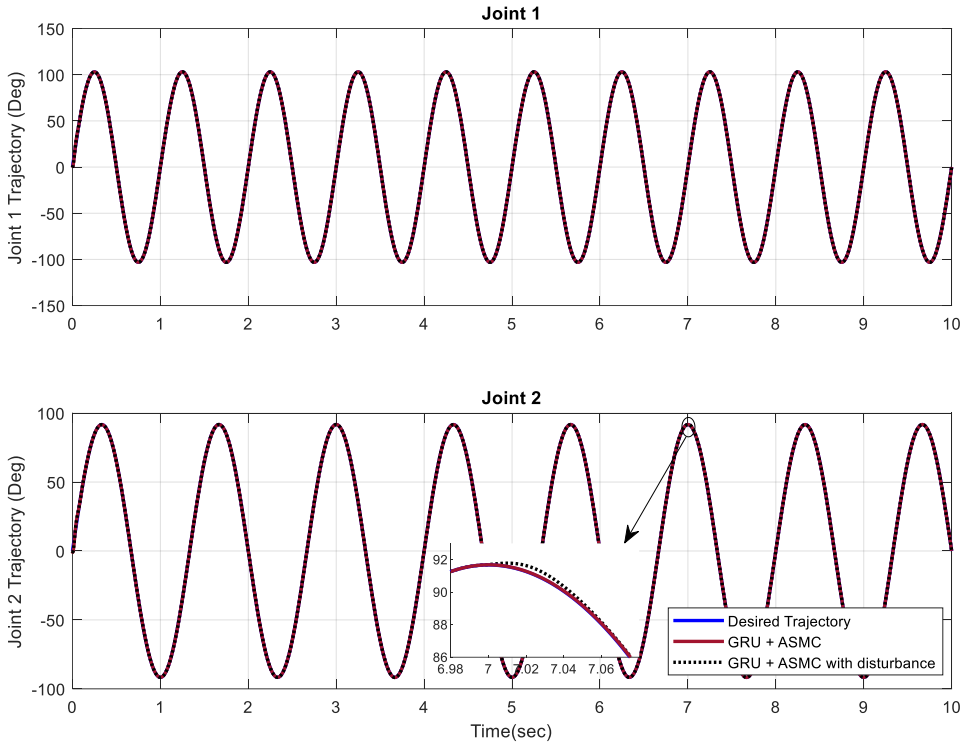

Figure 4.17: GRU with ASMC trajectory tracking with and without disturbance

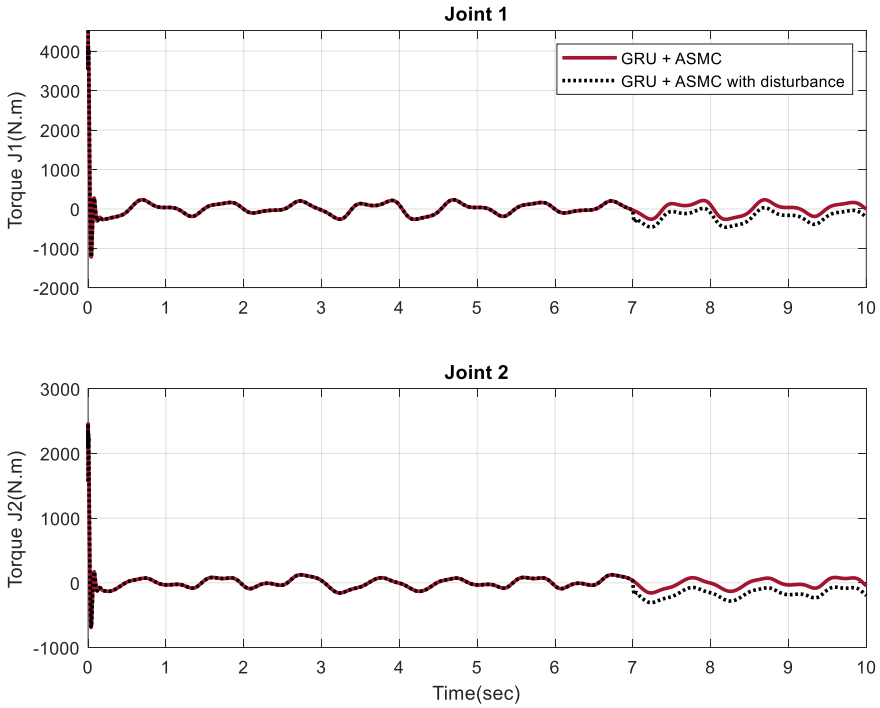

Figure 4.18: GRU with ASMC torque with and without disturbance

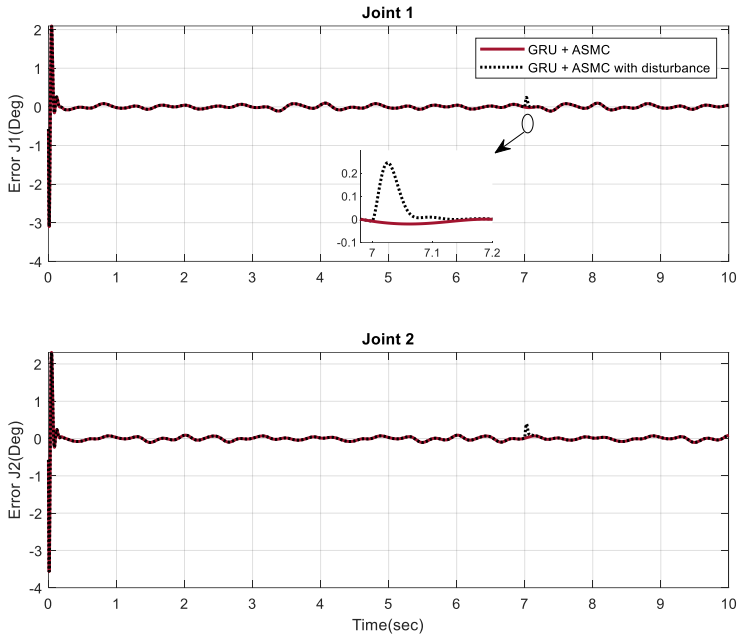

Figure 4.19: GRU with ASMC error with and without disturbance

4.4.6 LSTM, GRU, and ASMC without disturbance

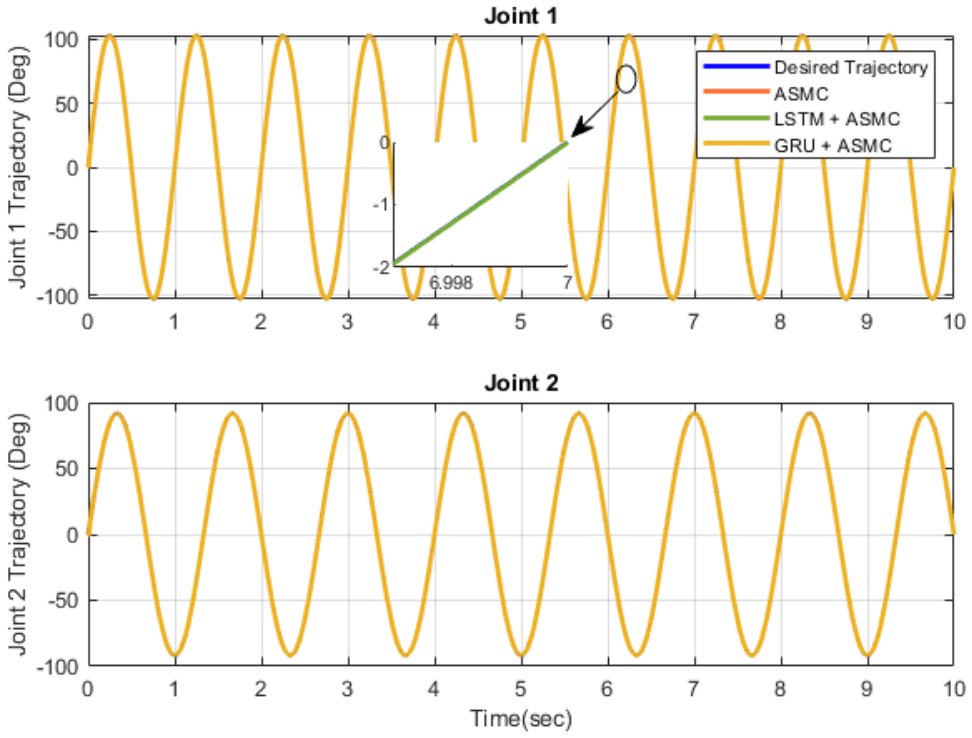

Figure 4.20: A comparison of trajectory tracking in ASMC, LSTM with ASMC, and GRU with ASMC

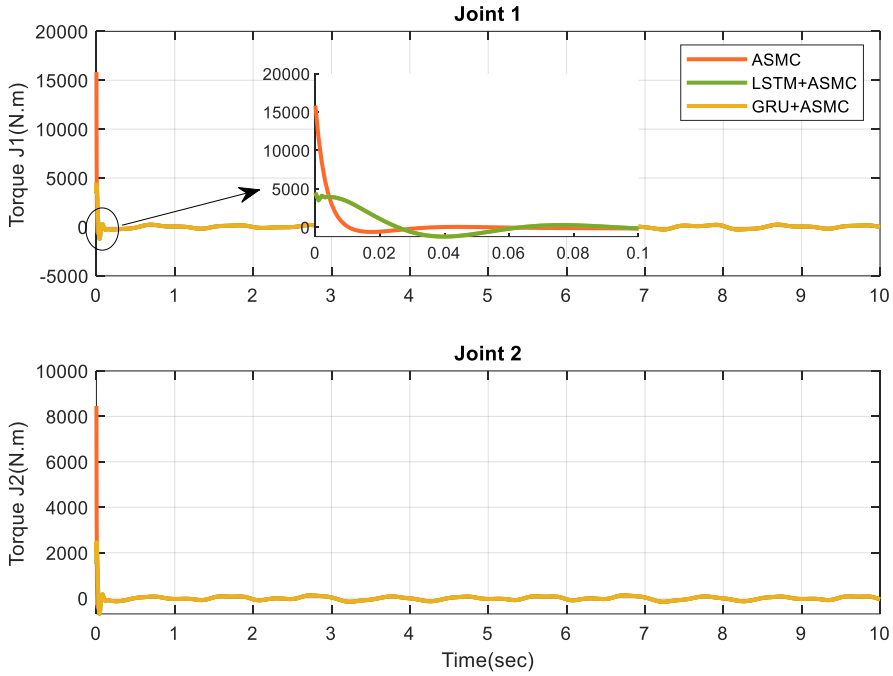

Figure 4.21: A comparison of torque in ASMC, LSTM with ASMC, and GRU with ASMC

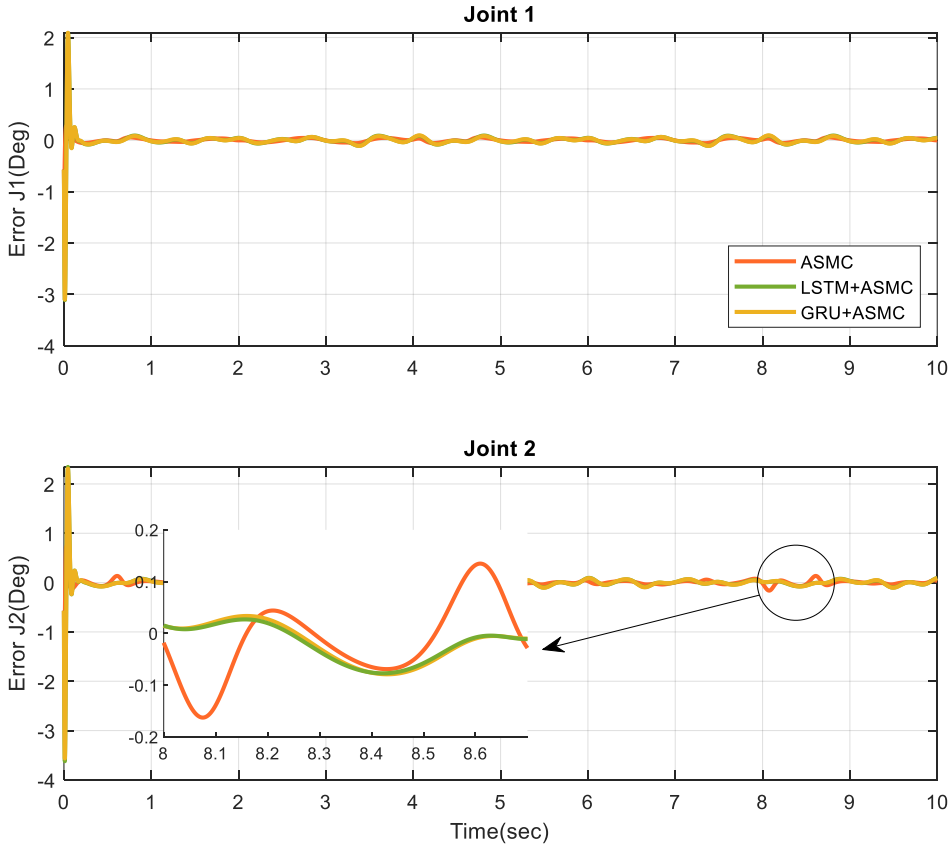

Figure 4.22: A comparison of error in ASMC, LSTM with ASMC, and GRU with ASMC

4.4.7 LSTM, GRU, and ASMC with disturbance

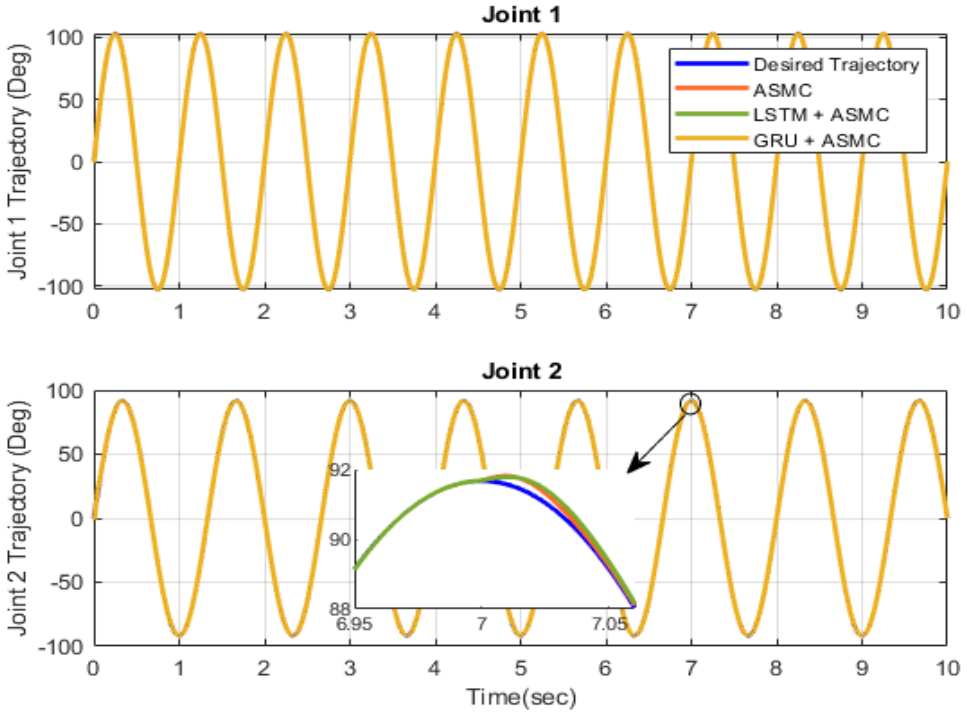

Figure 4.23: A comparison of trajectory tracking in ASMC, LSTM with ASMC, and GRU with ASMC (with disturbance)

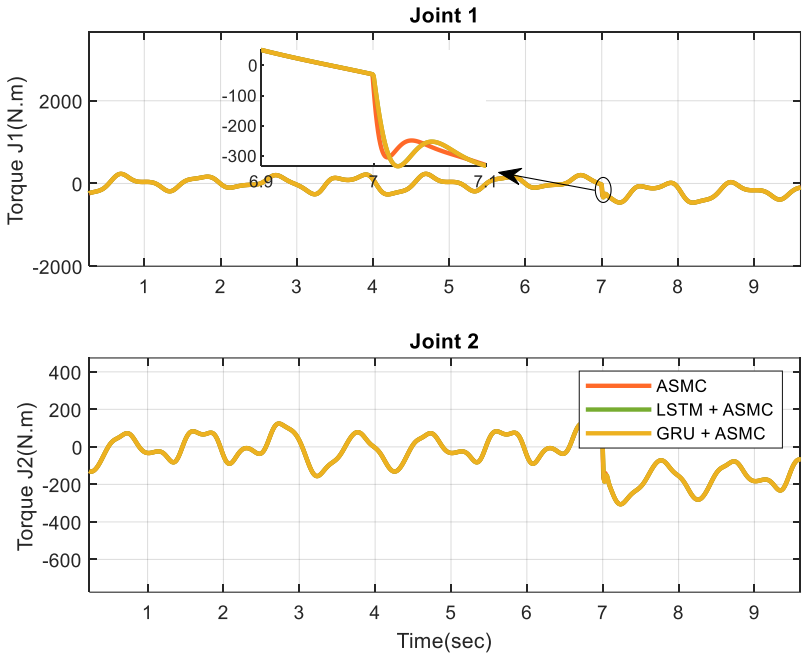

Figure 4.24: A comparison of torque in ASMC, LSTM with ASMC, and GRU with ASMC (with disturbance)

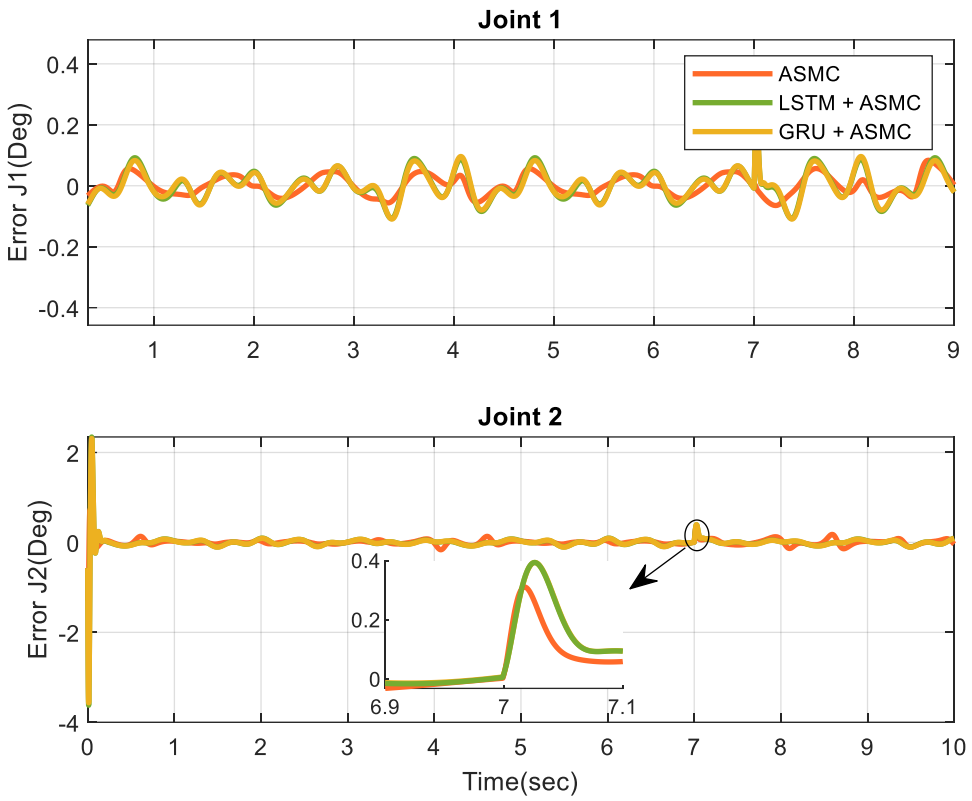

Figure 4.25: A comparison of error in ASMC, LSTM with ASMC, and GRU with ASMC (with disturbance)

Table 4.12: Deep Recurrent Neural Network Model performance with and without external disturbance (in degree)

Model	**Trajectory tracking RMSE**		**Average RMSE**	**Disturbance**
	RMSE for Joint 1	RMSE for Joint 2		
ASMC	0.0413	0.0616	0.0514	No
LSTM	1.9057	2.6633	2.2845	No
GRU	4.3634	4.1702	4.2668	No
LSTM + ASMC	0.1354	0.1637	0.1495	No
GRU + ASMC	0.1362	0.1652	0.1507	No
ASMC	0.0429	0.0685	0.0557	t = 7 sec
LSTM	3.8296	3.7571	3.7933	t = 7 sec
GRU	6.8267	5.3469	6.0868	t = 7 sec
LSTM + ASMC	0.1453	0.1604	0.1528	t = 7 sec
GRU + ASMC	0.1490	0.1686	0.1588	t = 7 sec

As shown in Table 4.12, LSTM and GRU follows the desired trajectory with a good tracking error. After combining with low gain ASMC significantly increases the accuracy and reduces the RMSE. The tracking error follows the same pattern in the simulation results with disturbance at t = 7 sec. It confirms that without using model-based component discussed in section 3.3, the proposed method generates acceptable results without tuning the parameters of the robot.

In the simulation, the LSTM with ASMC controller shown a good tracking performance with and without disturbance. To verify this approach, this controller is implemented on an actual robot which is examined in the next chapter.

Chapter 5

Experimental Results and Discussion

5.1 Introduction

This chapter covers experimental setup, Simulink model, experimental results, and discussion of the proposed method based on three essential aspects: performance, precision, and robustness. This chapter is structured as follows. The first section of this chapter contains a brief description of software tools and hardware used for implementation. The deep learning model conversion process is clarified, and some common problems in the deployment are mentioned here. In addition to that, the detailed Simulink model and configuration parameters of the robot are explained. Furthermore, the proposed method is deployed on a 2-DOF robot and tested on several trajectories to verify this approach. In the end, the discussion about the proposed method is presented.

5.2 Experimental Setup

This section explains software tools and hardware used in this research. As discussed in chapter 4, all the deep learning models use the same configuration. For the experiment, it is divided into two steps:

1. Create a deep learning model using TensorFlow and Keras and optimize hyperparameters. The selection of the best deep learning model is made by using root-mean-square error (RMSE).
2. As the robot only supports MATLAB code, this model is converted into a global deep learning model (.h5 file format) and imported into the MATLAB workspace. It is tough to use a deep learning model in real-time (with 0.001 sampling time) using a client-server architecture with low system configuration, so the whole model is converted to C++ files

to use as an S-Function in Simulink directly. The whole complexity in deployment on actual robots is defined in section 5.2.2.

5.2.1 Software Tools

There are several frameworks and libraries available to implement the deep learning model. In recent years, parallel computing and multi-GPU made the deep learning model training process much more manageable. While working with real-time systems, there are several things to consider: performance of deep learning model, memory usage, and accuracy of deep learning model.

a) **TensorFlow:** TensorFlow is a free and open-source machine learning and deep learning library developed by Google Brain Team. It was first used by Google for internal projects before being released in 2015 under the Apache License 2.0. It provides support to multiple platforms such as embedded device, web application, and mobile application. The benefit of using TensorFlow is to leverage multiple CPUs and GPUs to train deep learning models.

b) **Keras:** It is an open-source software library for artificial neural networks that include a Python interface It was created as part of the Open-ended Neuro-Electronic Intelligent Robot Operating System project's research endeavour. Keras includes several implementations of basic and advanced neural network building components like layers, activation functions, optimizers, and other tools. Keras now supports TensorFlow for constructing machine learning and deep learning models as of version 2.4. The best way to use all these libraries and frameworks in python is to create a virtual environment and install all the dependencies. This allows the reuse and interchange of necessary libraries and frameworks.

c) **NumPy:** In deep learning, matrix multiplication of a multi-dimensional array is a big challenge when complexity increases. NumPy is a free, open-source Python library that supports huge, multi-dimensional arrays and matrices, as well as a wide number of high-level mathematical functions for manipulating them. This was originally written by Travis Oliphant.

d) **Pandas:** The accuracy of deep learning models can be increased by data cleaning and manipulation. Pandas is a free and open-source Python data manipulation and analysis toolkit. It has inbuilt procedures and data structures for manipulating numerical quantities and time series. In Pandas, the table containing the dataset is called DataFrame. Pandas allow various data cleaning methods, data wrangling features, and data manipulation methods such as merging, reshaping and selecting. Reading and writing on multiple CSV or Excel become much easier by using Pandas.

e) **CUDA:** GPU usage (Graphical Processing Unit) significantly reduces the training time required by deep learning models. Nvidia established the CUDA (Compute Unified Device Architecture) parallel computing platform and application programming interface (API) paradigm. This platform allows software developers and engineers to use a CUDA-enabled graphics processing unit (GPU) for general-purpose processing. This is highly useful in a production environment where the size of the dataset is extensive.

f) **Open Neural Network Exchange (ONNX):** ONNX is a machine learning and deep learning model representation format that is open source. It helps to create a general machine and deep learning model which uses common set of operators and building blocks, making it extremely useful for engineers and researchers. This is exceptionally helpful while working with multiple programming languages.

Table 5.1 represents the version of all the programming languages, libraries and frameworks used in this research.

Table 5.1: Software tools and their version

Programming Language / Framework / Library	**Version**
Python	3.7
Microsoft Visual C++	2017
Java	1.8
TensorFlow	2.2.3
Keras	2.3.0
NumPy	1.18.5
Pandas	1.1.5
Matlab	R2021a (9.10)
Matlab Coder	5.2
Matlab Deep Learning Toolbox	14.2
Quanser Quarc Complete Edition	2021
oneDNN (MKL-DNN)	1.0
CMake	2.8.11

5.2.2 Hardware Configuration

In this research, a 2-DOF serial flexible link robot from Quanser is used to implement and evaluate the performance of the developed algorithm.

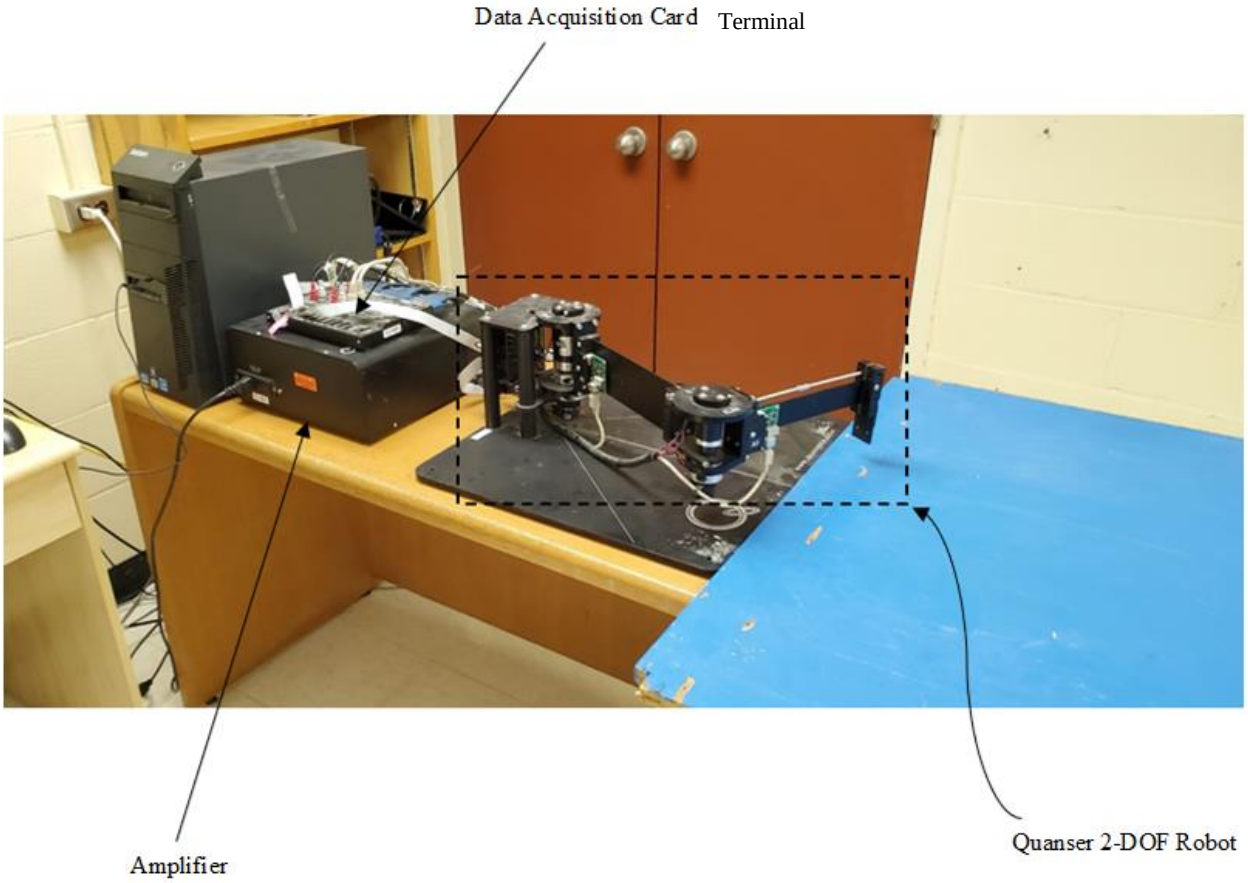

Figure 5.1: 2-DOF serial flexible link robot from Quanser

The training configuration and testing configuration is different, as defined in Table 5.2. The remote server is used to train the deep learning model and the local computer to perform experiments.

Table 5.2: Remote Server and Local Computer Configuration

Configuration	Remote Server	Local Computer
Processor	Intel Silver 4216 Cascade Lake @ 2.1 GHz (32 Core / 64 Threads)	Intel Core i7 2600 @ 3.4 GHz (4 Core / 8 Threads)
GPU	NVIDIA V100 Volta (32GB HBM2 memory)	Intel® HD Graphics 2000
Memory	64 GB	16 GB

5.3 Data Generation using Simulink Model

As shown in Figure 5.1, the Simulink model consist of Stage 1 setpoint (Initialization of desired position, velocity, and acceleration), HIL Initializer, deep learning block, and existing LQR and ASMC controller. The deep learning model uses reference position, velocity, and acceleration as input and generate required torque. The reference acceleration is calculated with the following formula:

$$\ddot{q}_r = \ddot{q}_d + 2\zeta\dot{e} + \zeta^2 e \tag{5.1}$$

where

$$\dot{e} = \dot{q}_d - \dot{q};\ e = q_d - q \tag{5.2}$$

Where q, q_d, $\dot{q}$, $\dot{q}_d$, e, and $\dot{e}$ are actual position, desired position, actual velocity, desired velocity, position error, and velocity error. ς is a positive definite diagonal matrix, $\ddot{q}_d$ is a desired acceleration, and $\ddot{q}_r$ is a calculated acceleration used as an input of the deep learning model.

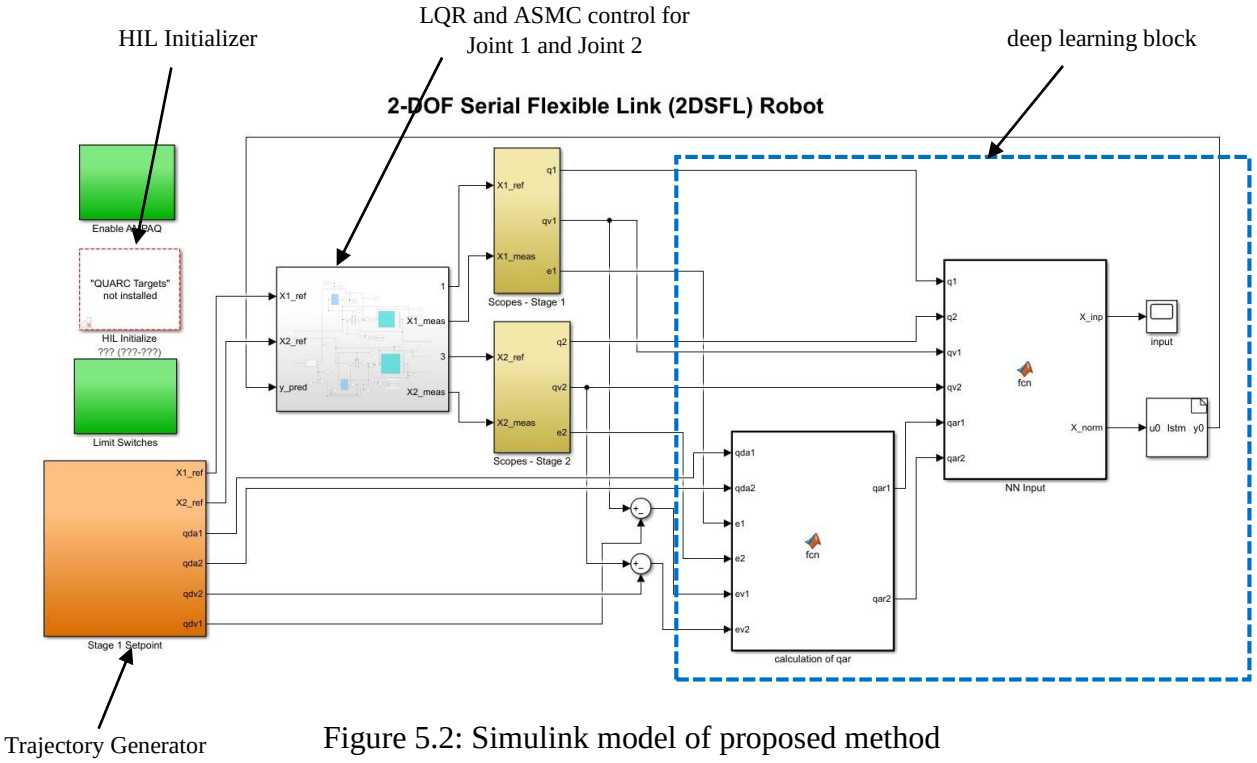

Figure 5.2: Simulink model of proposed method

The Simulink model consisting of stable ASMC controller is used to generate the dataset with a sampling time of 0.001 second. The Simulink target is set to quarc_win64.tlc (with all default configuration) for this robot. The generated dataset includes variety of trajectories such as sine wave, square wave, and step function with different amplitude and frequency . With the help of Simulink scope, data for position, velocity, acceleration, and torque is stored into workspace. As shown in the Figure 5.1, the deep learning block contains normalization function and trained deep learning model. The algorithm for normalization is provided in chapter 3.

For safety reasons, the saturation block is included to limit the current sent to the corresponding DC motor. The upper and lower limit of this current are 0.94 A and -0.94 A. While optimizing the ASMC controller, it is assumed that payload is not attached to the end-effector. The configuration of robot is defined in the reference manual provided by Quanser [84]. Initially, the setup file is executed to load all the parameters of the robot in the workspace.

5.4 Experimental Results

The experimental results include some important experiments done on variety of trajectory on different sampling rate. The robot communicates to the computer using QPIDe data acquisition card. This card generally open TCP/IP port for the communication. The Simulink model is also connected to Quarc TCP/IP block to get real-time data from the robot for data collection. The trained deep learning model is deployed using S-Function block to make faster computationally.

The following section presents trajectory tracking, error, and torque values on different conditions. Initially the sampling time was 0.004 and 0.005 due to heavy computation performed by deep learning block. Increasing the computation power in local computer was helpful to reduce the sampling time to 0.002 and 0.003. In the following sections ASMC with different percentage (%) means, lower % of gains are applied, to show that the contribution of deep learning-based model.

5.4.1 Experiment on sine wave using only ASMC with Lower Gain (10%)

In Figures 5.3 -5.8 sampling Time (TS) = 0.002, and unit of joints trajectories is radian and unit of joints torque is Newton-metere (N-m).

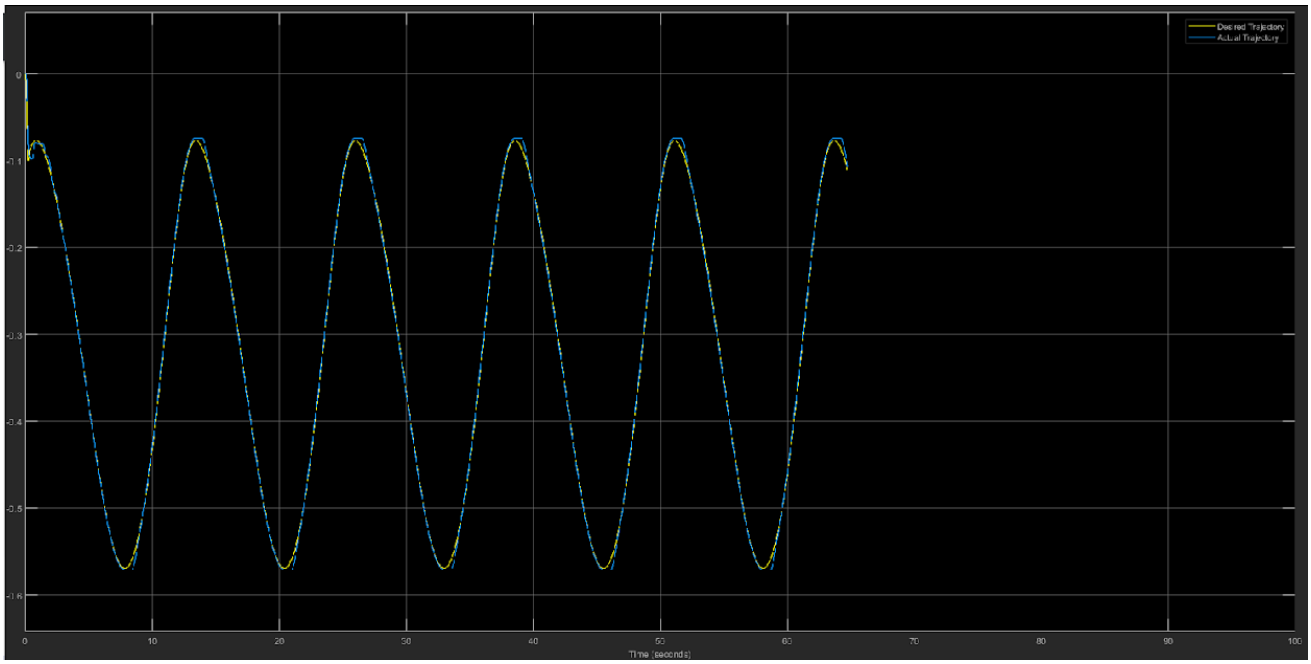

Figure 5.4: Trajectory Tracking for Joint 1 with only ASMC (10%)

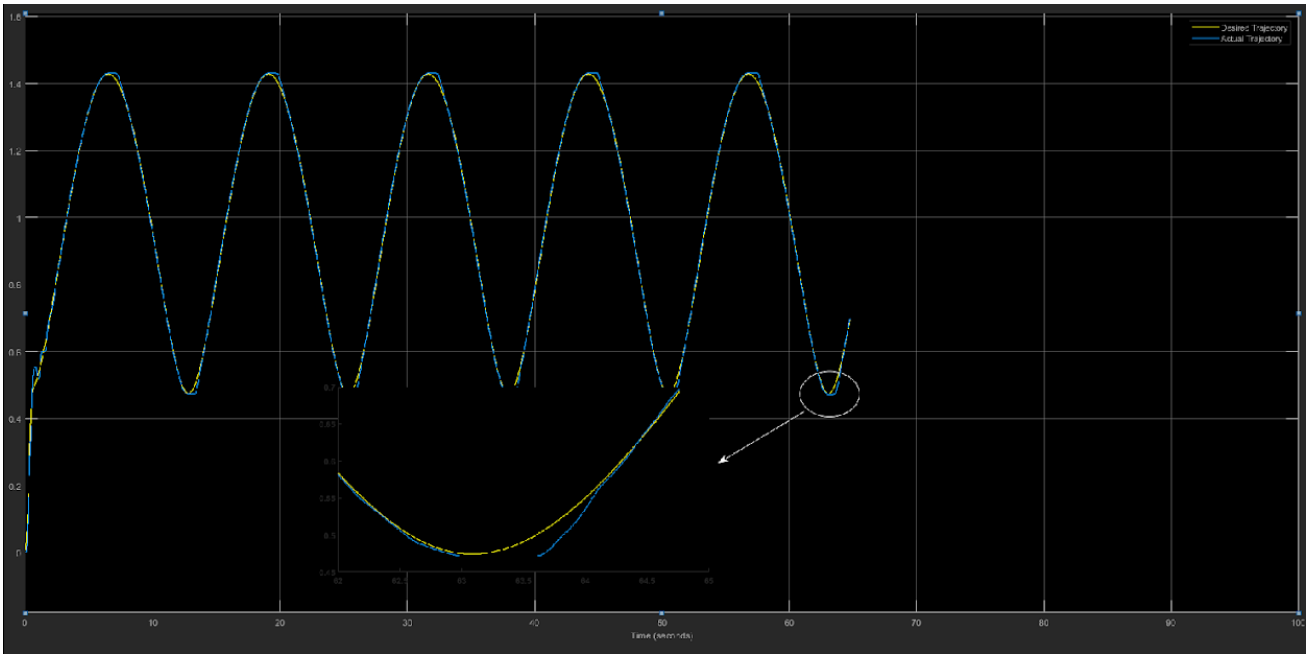

Figure 5.5: Trajectory Tracking for Joint 2 with only ASMC (10%)

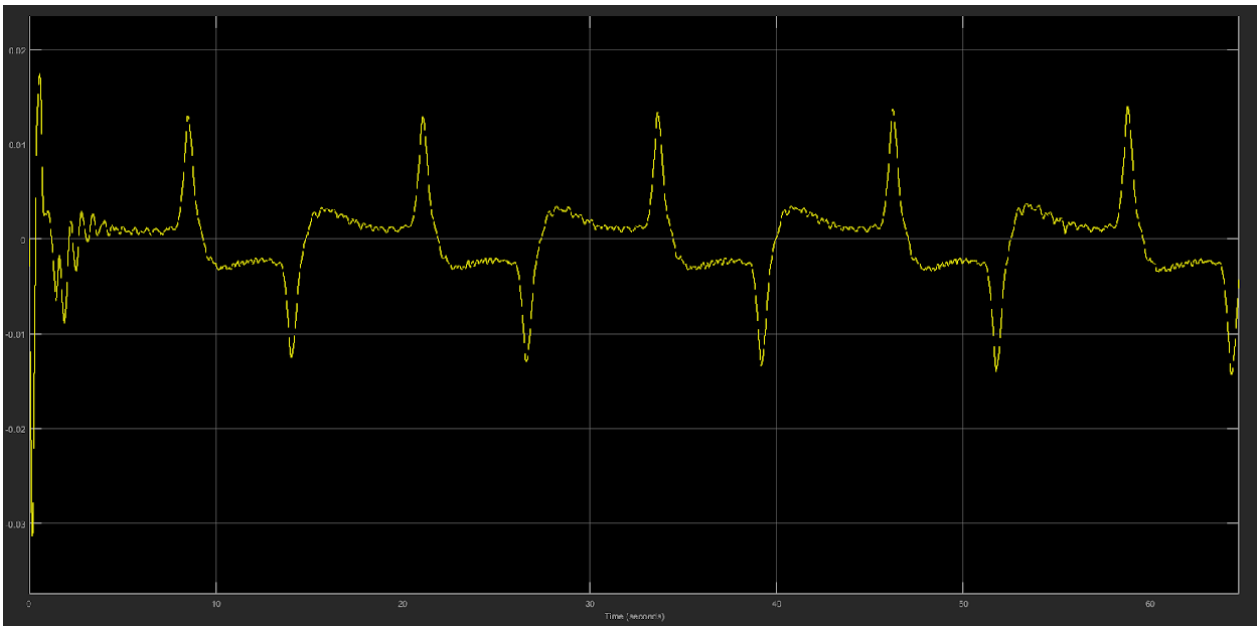

Figure 5.6: Error for Joint 1 with only ASMC (10%)

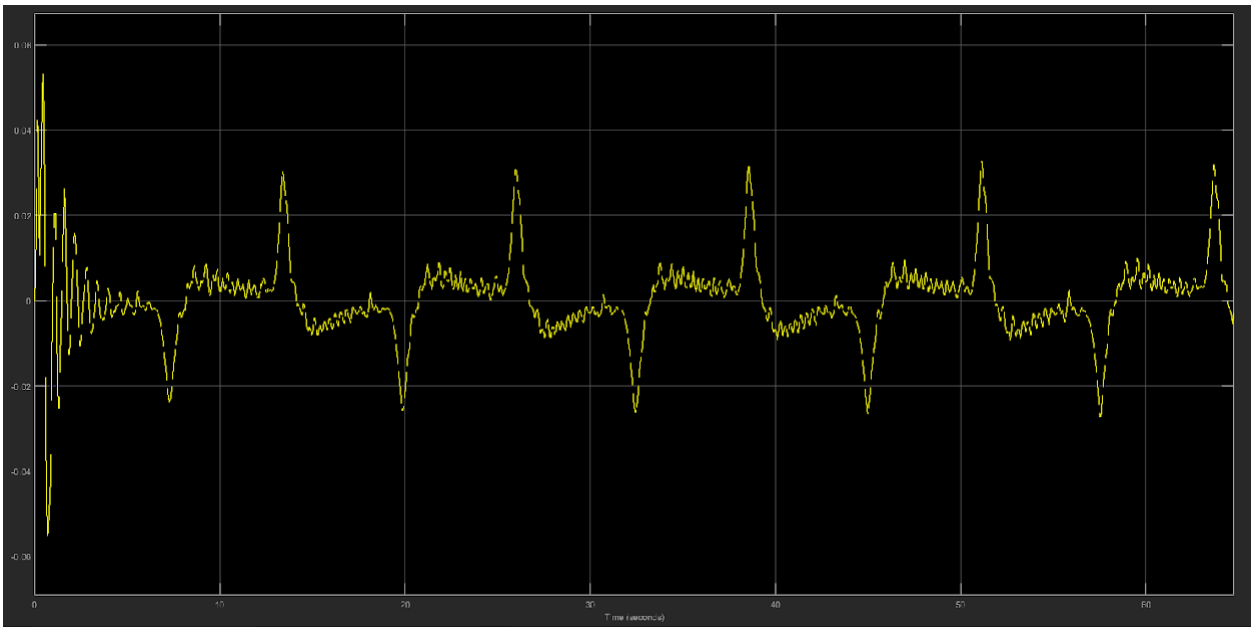

Figure 5.7: Error for Joint 2 with only ASMC (10%)

As shown in Figure 5.3 and 5.4, the ASMC controller follows the desired trajectory precisely with an error rate of ±0.0025. The error in ASMC is slightly increasing over time as shown in figure 5.6.

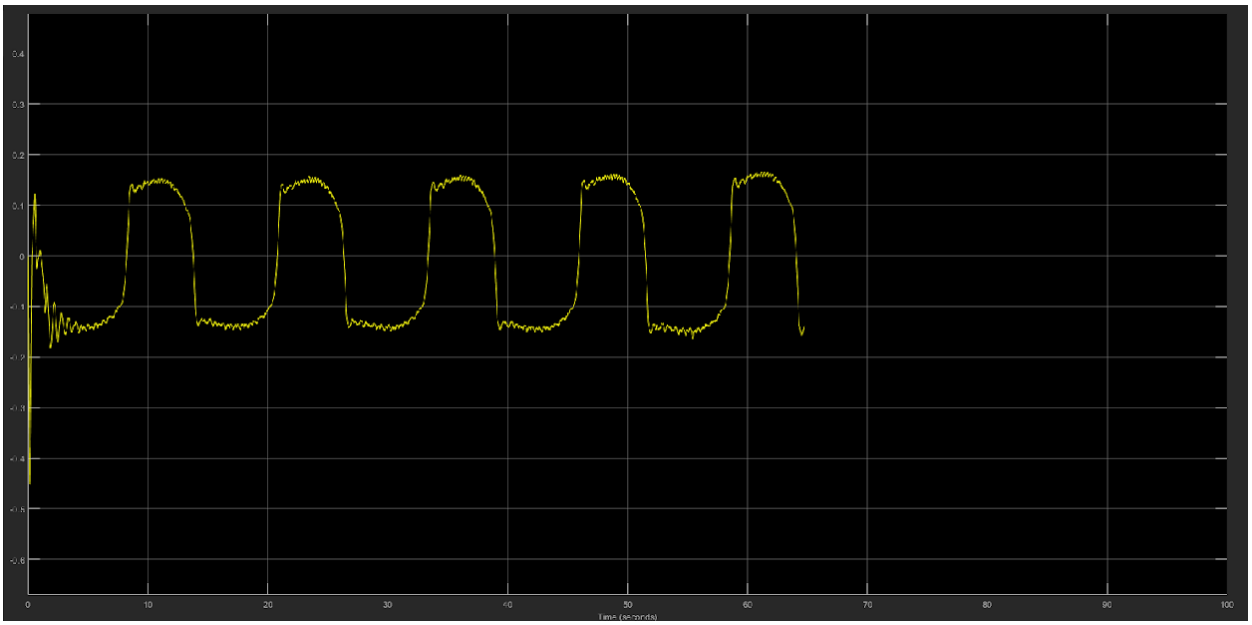

Figure 5.8: Torque for Joint 1 with only ASMC (10%)

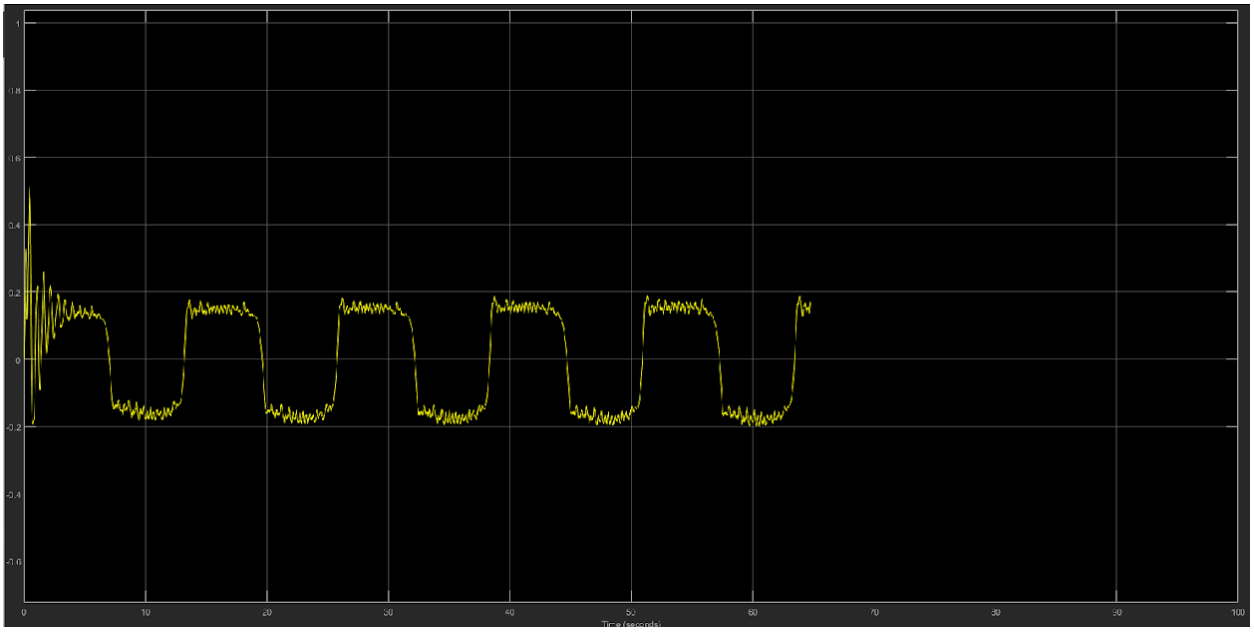

Figure 5.9: Torque for Joint 2 with only ASMC (10%)

5.4.2 Experiment on square wave using only ASMC (10%)

Sampling Time (TS)= 0.002

To check robustness, the ASMC controller performance is compared with step function and square wave. The highlighted portion in Figure 5.9 depicts that the ASMC controller follows the square wave more precisely.

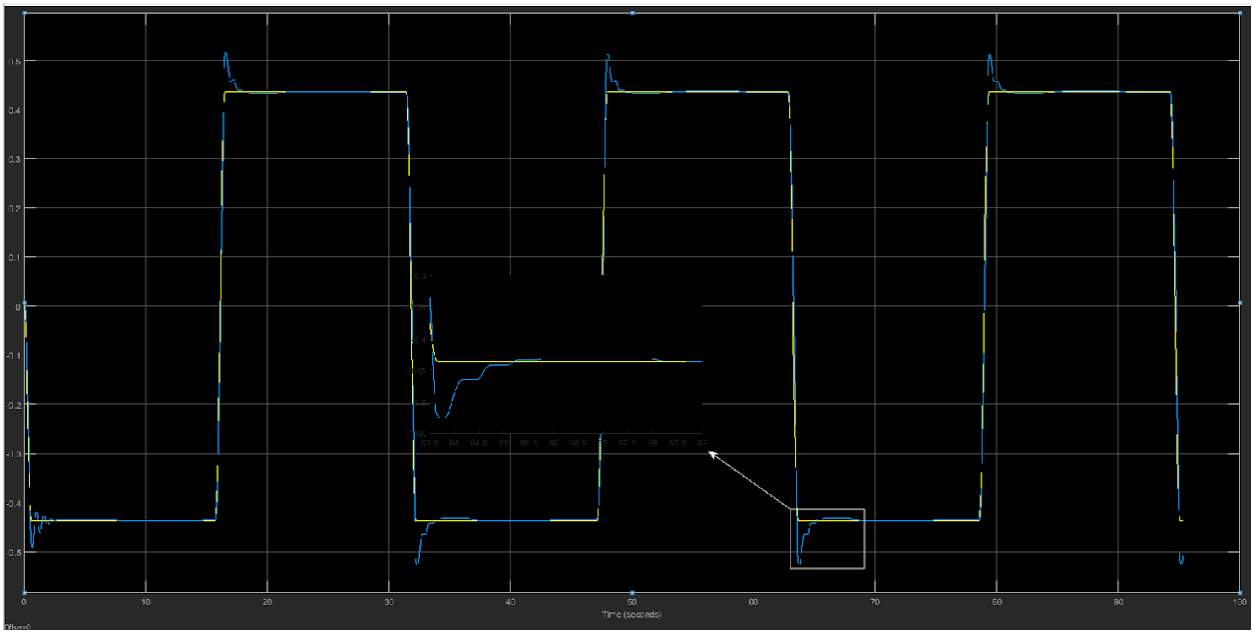

Figure 5.10: Trajectory Tracking for Joint 1 with only ASMC (10%)

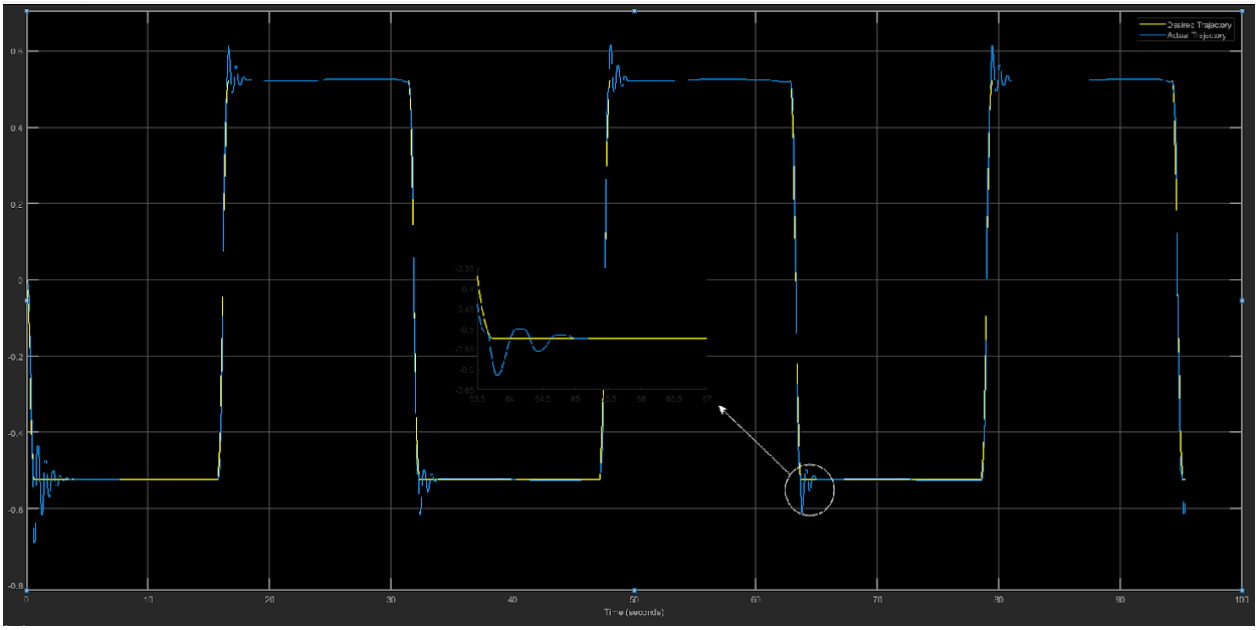

Figure 5.11: Trajectory Tracking for Joint 2 with only ASMC (10%)

The over-shoot problem occurs at the peak point where the robot moving to another direction and steady for some time. This phenomenon is clearly shown in the error for Joint 1 and 2 (Figure 5.11 and 5.12).

Figure 5.12: Error for Joint 1 with only ASMC (10%)

Figure 5.13: Error for Joint 2 with only ASMC (10%)

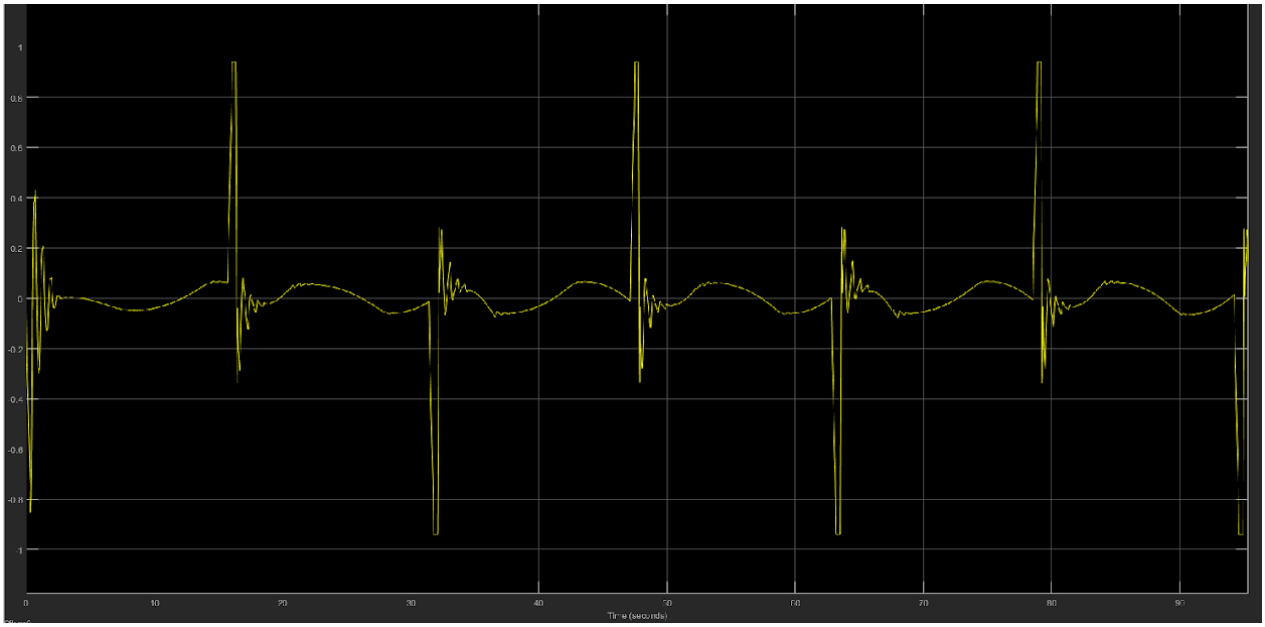

Figure 5.14: Torque for Joint 1 with only ASMC (10%)

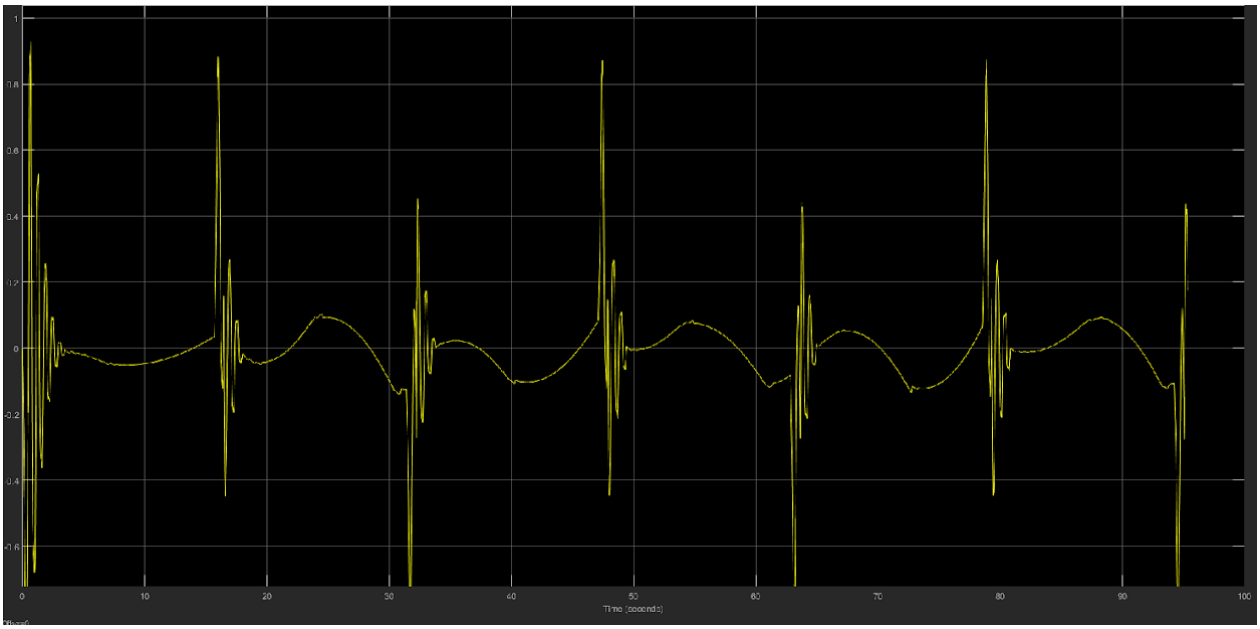

Figure 5.15: Torque for Joint 2 with only ASMC (10%)

As shown in Figure 5.14, the torque value sudden jumps to reduce the error in tracking the trajectory. Furthermore, analyzing this figure, it is safe to say that to decrease the error in trajectory tracking, ASMC generates higher torque.

5.4.3 Experiment on square wave using only ASMC (50%)

Sampling Time (TS)= 0.002

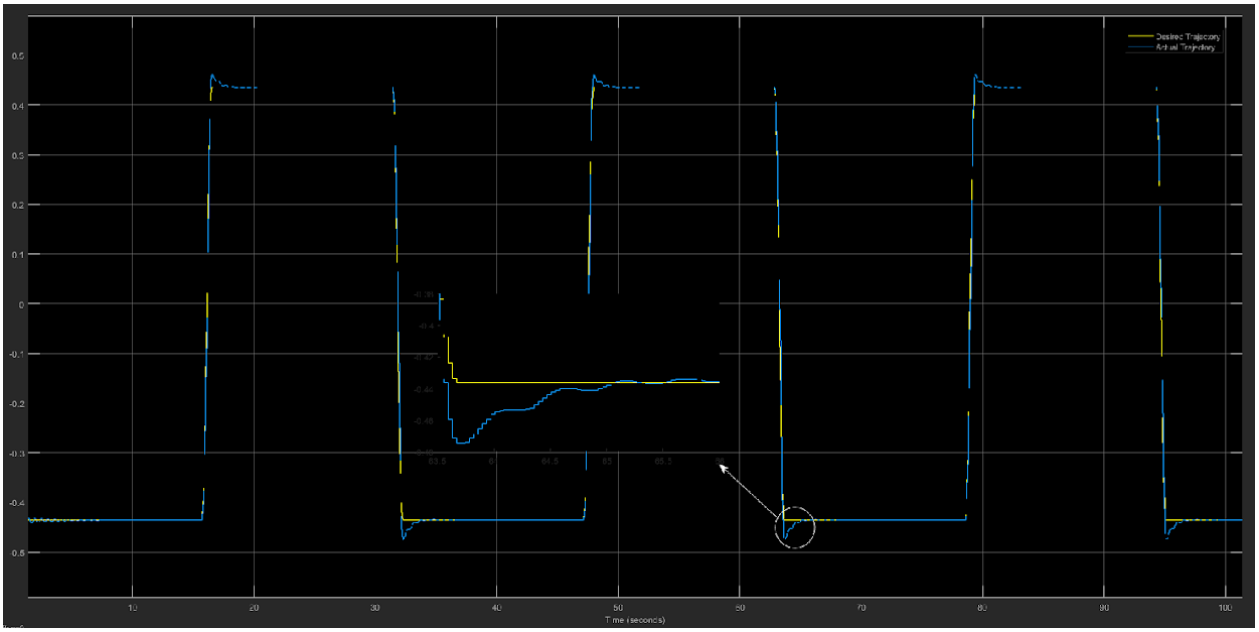

Figure 5.16: Trajectory Tracking for Joint 1 with only ASMC (50%)

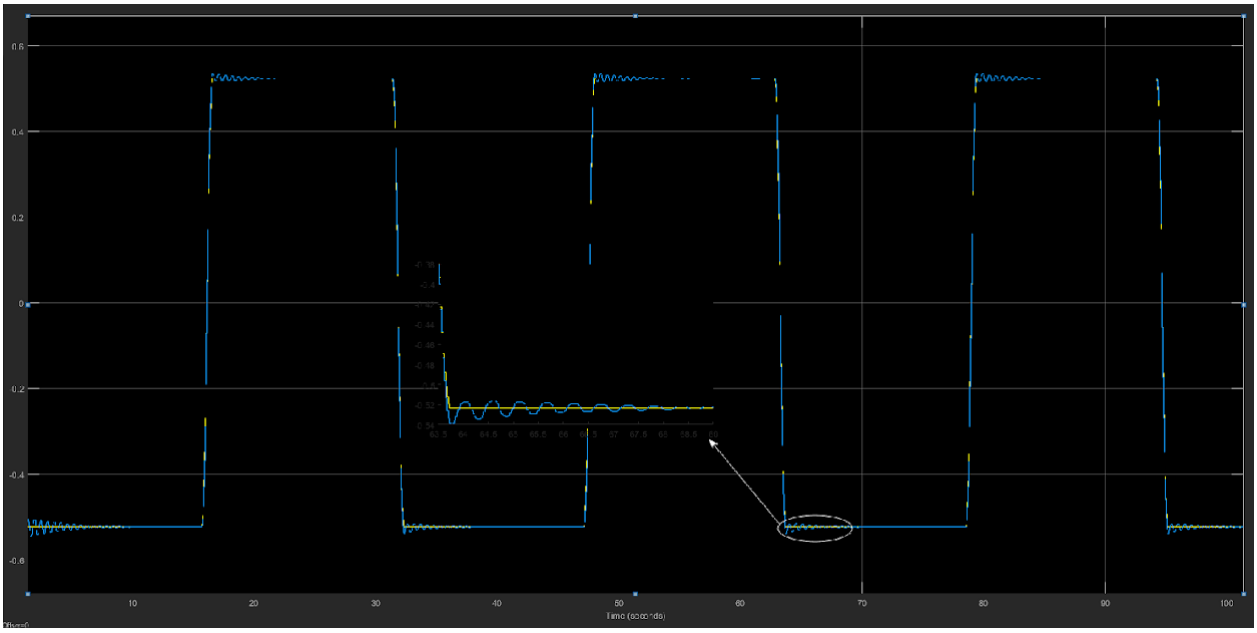

Figure 5.17: Trajectory Tracking for Joint 2 with only ASMC (50%)

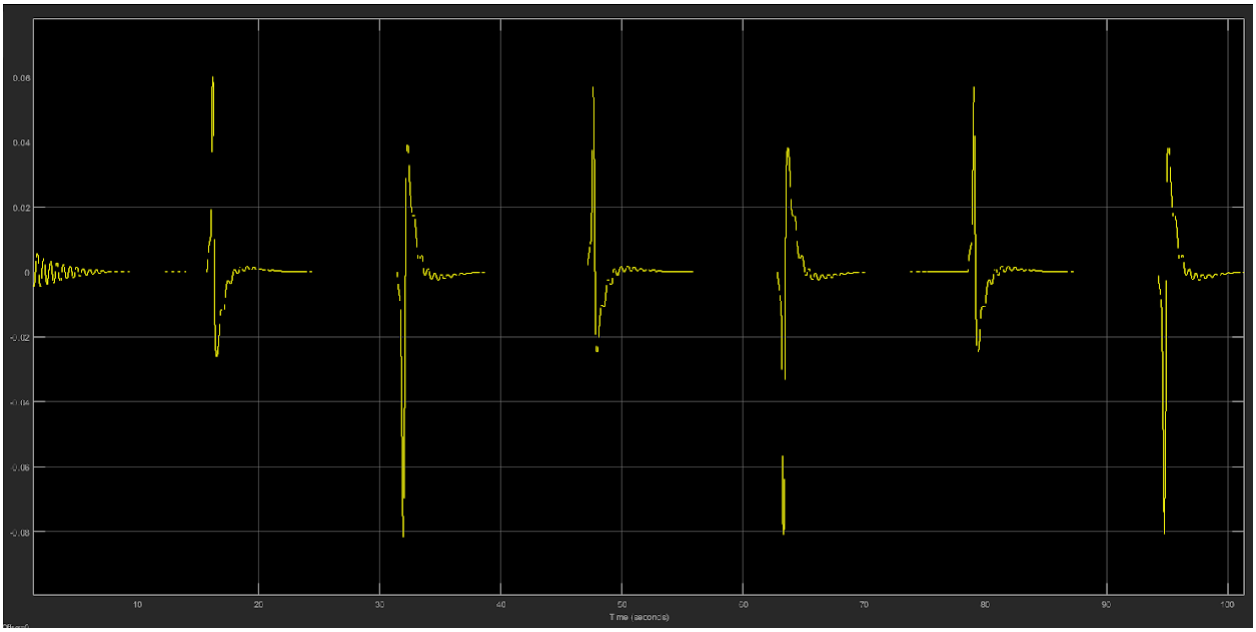

Figure 5.18: Error for Joint 1 with only ASMC (50%)

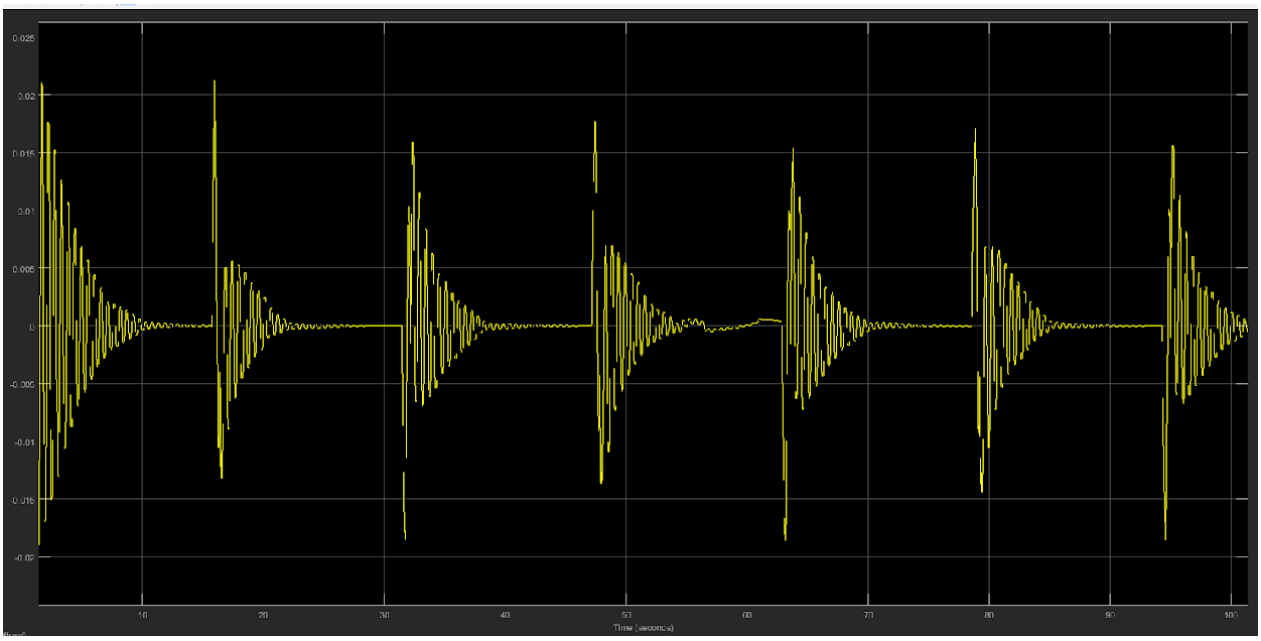

Figure 5.19: Error for Joint 2 with only ASMC (50%)

Compared to previous experiment, the ASMC controller has higher error initially on Joint 2. As the robot is a flexible link robot, the fluctuation is higher when changing the direction.

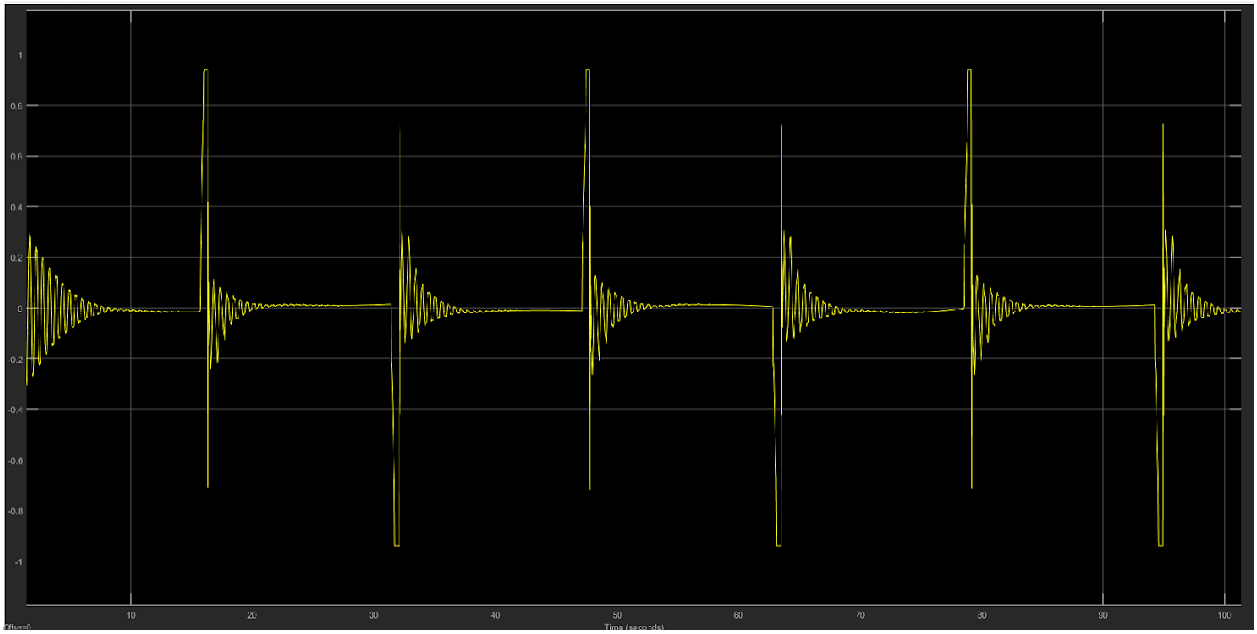

Figure 5.20: Torque for Joint 1 with only ASMC (50%)

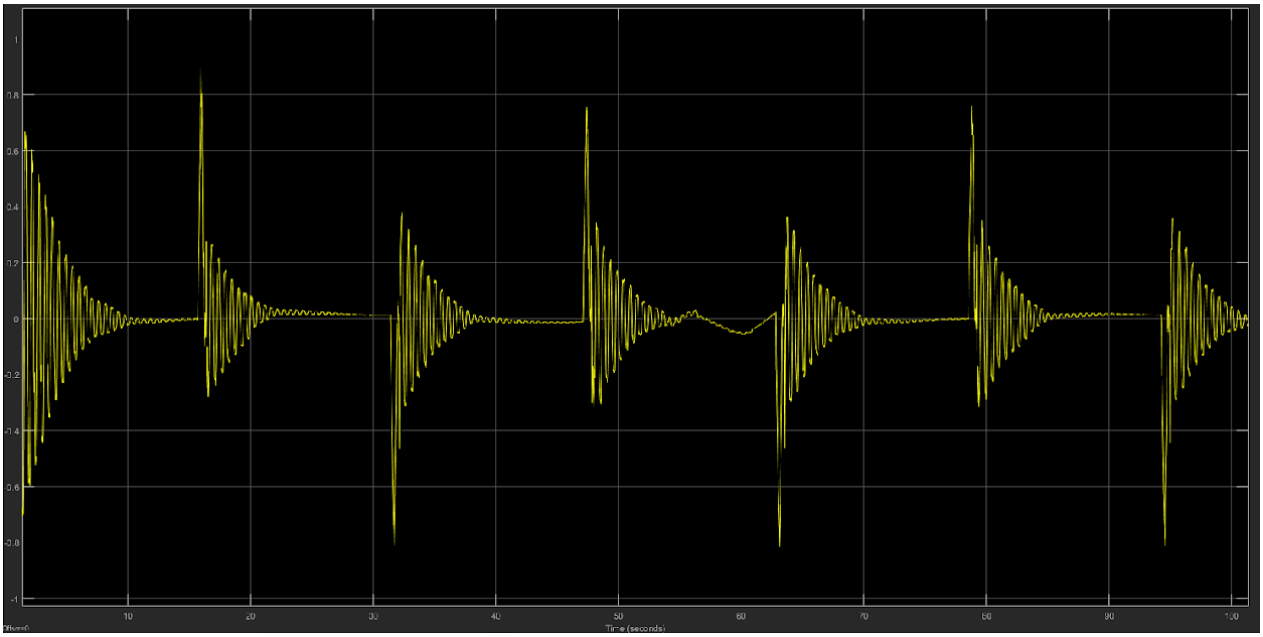

Figure 5.21: Torque for Joint 1 with only ASMC (50%)

Increasing ASMC gain from 10% to 50%, the error rate is improved for Joint 1. For joint 2, the error is higher and require more settling time on peak values as shown in Figure 5.16. The ASMC controller generates higher torque leading to more fluctuation in Joint 1 and Joint 2.

5.4.4 Experiment on sine wave using only LSTM

Sampling Time (TS)= 0.004

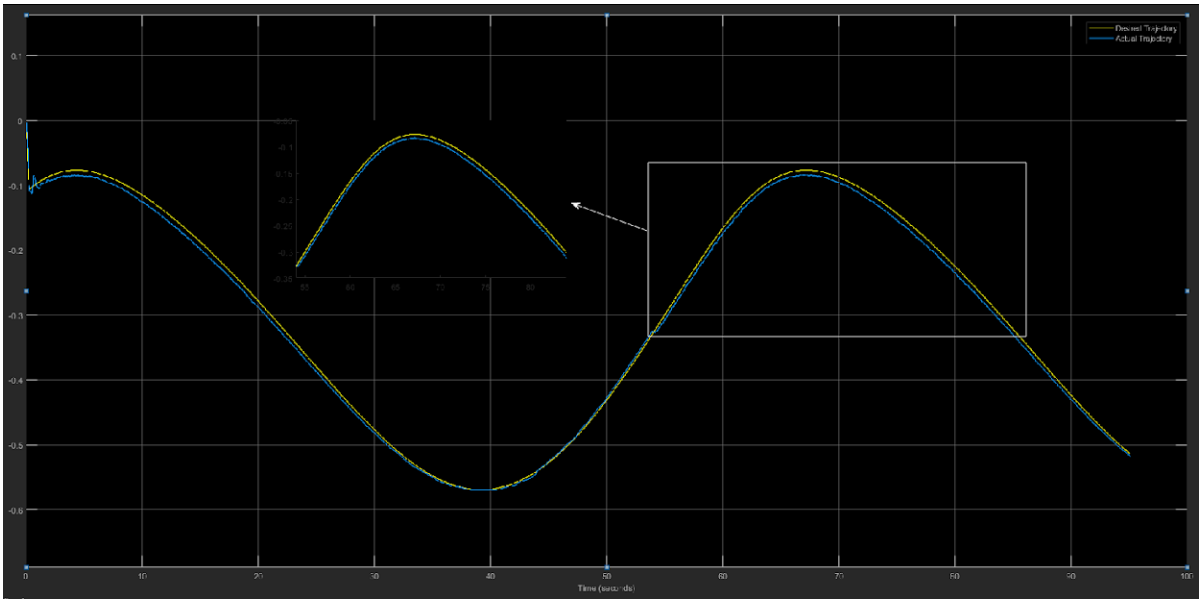

Figure 5.22: Trajectory Tracking for Joint 1 with only LSTM

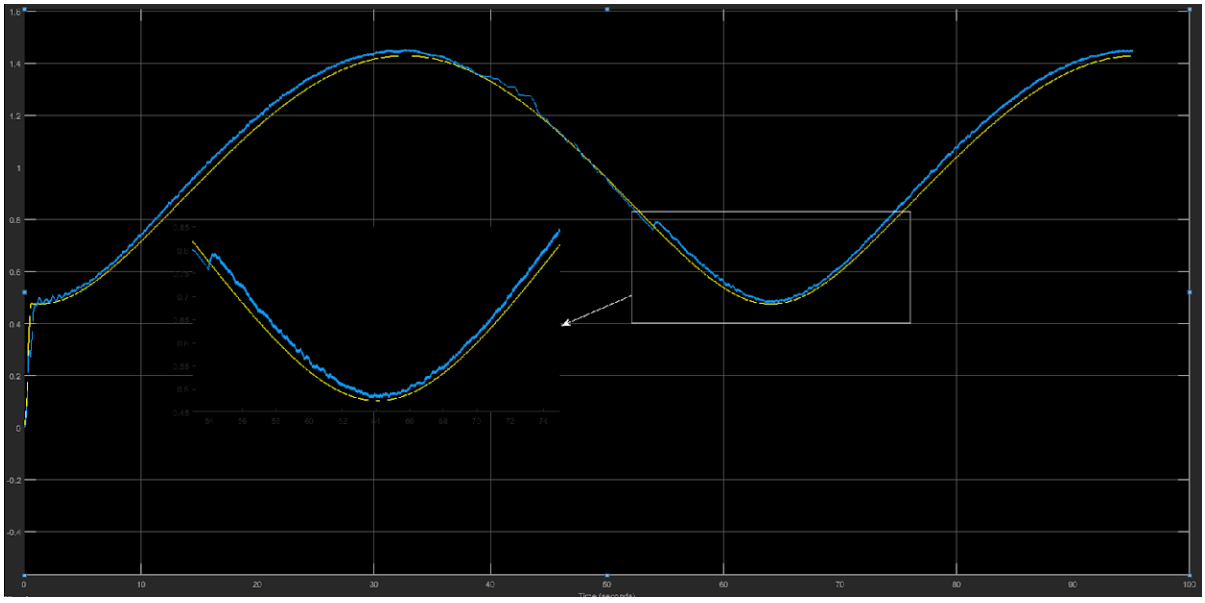

Figure 5.23: Trajectory Tracking for Joint 2 with only LSTM

In this experiment, the LSTM is trained on input-output data coming from the robot. The deep learning model is optimized using the same approach discussed in chapter 4. The LSTM block requires more computation compared to ASMC learning to higher sampling rate. To get accurate sampling rate, the sampling time and computation time blocks used in Simulink.

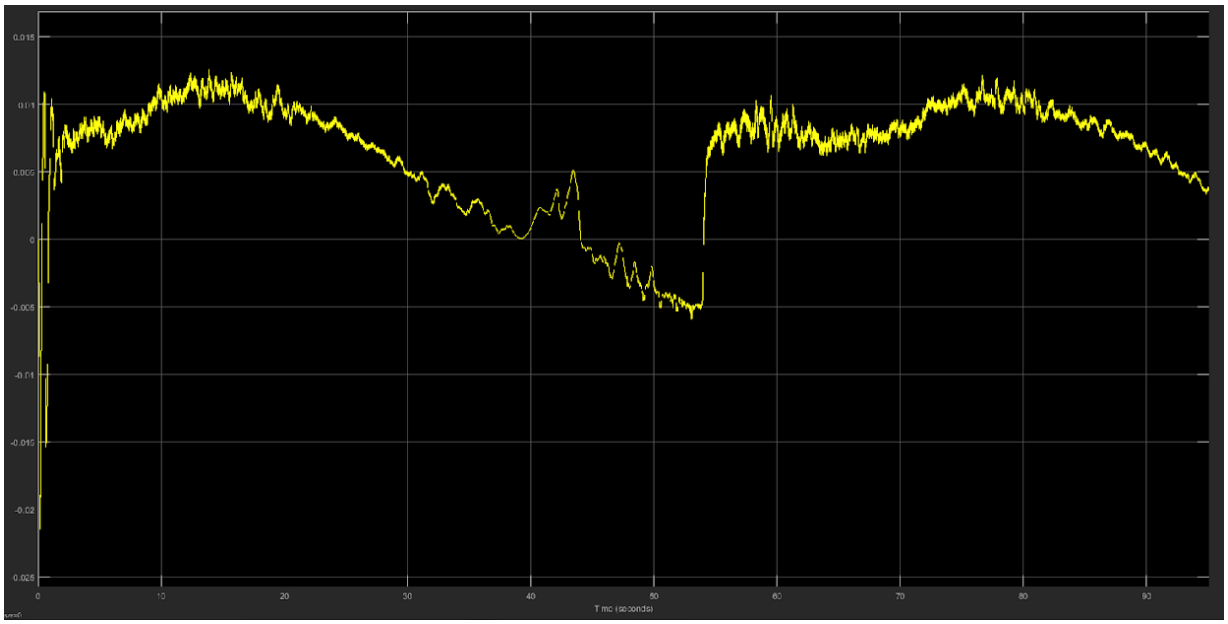

Figure 5.24: Error for Joint 1 with only LSTM

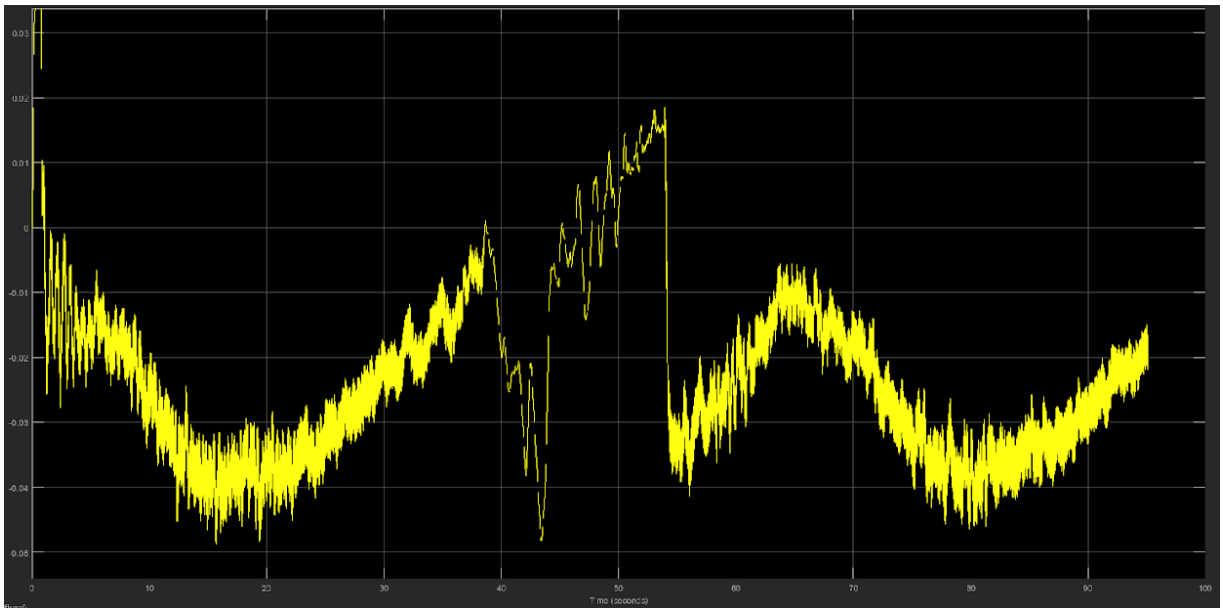

Figure 5.25: Error for Joint 2 with only LSTM

As shown in Figure 5.23, the error rate is much lower and on one sided which clearly states that there is a phase shift due to higher sampling rate.

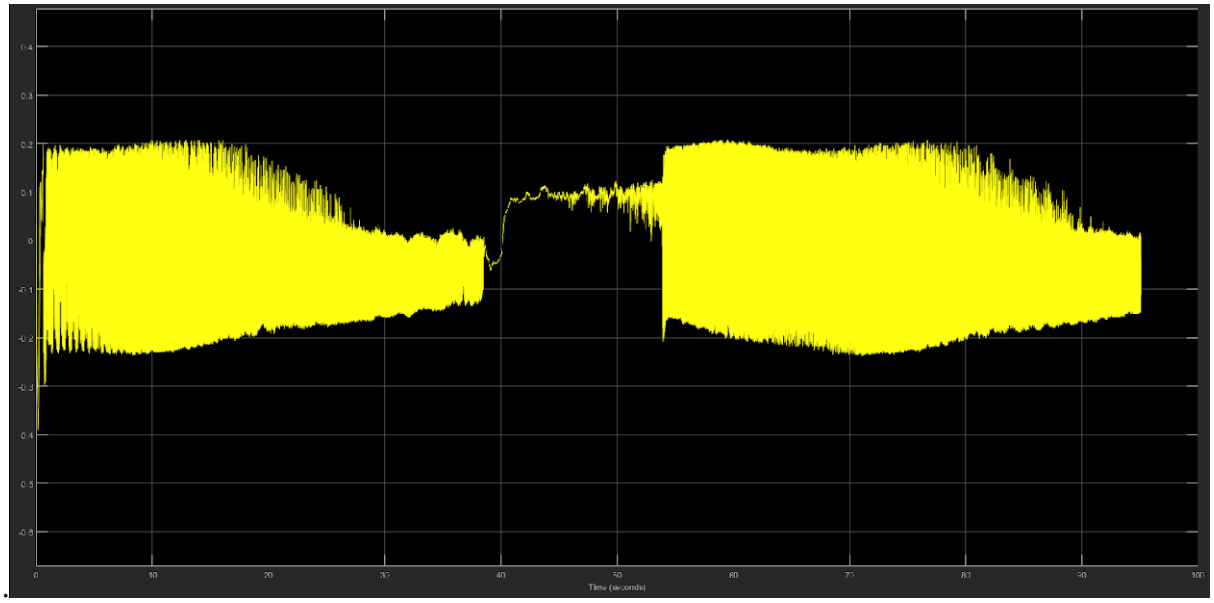

Figure 5.26: Torque for Joint 1 with only LSTM

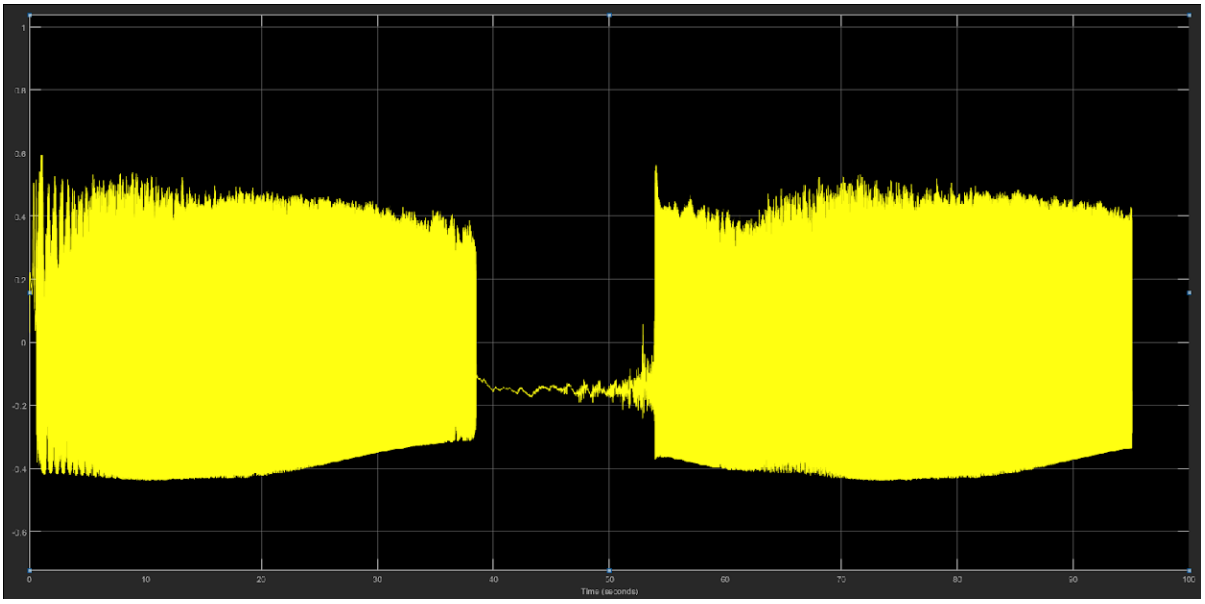

Figure 5.27: Torque for Joint 2 with only LSTM

Figure 5.25 and 5.26 clearly points out the chattering while tracking the trajectory. The ASMC controller shows almost same performance while using with higher sampling rate. This could be

eliminated by increasing the computation power in local computer. The following experiments are performed after upgrading the local computer which increases the computation power.

5.4.5 Experiment on sine wave using a combination of ASMC (10%) and LSTM

Sampling Time (TS)= 0.002

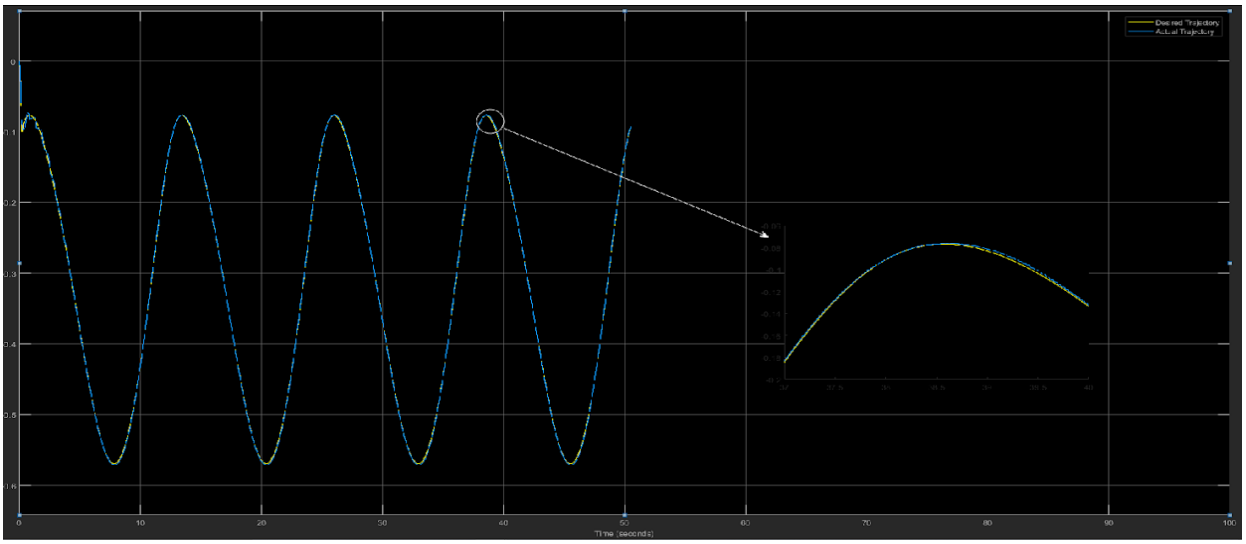

Figure 5.28: Trajectory Tracking for Joint 1 using a combination of ASMC (10%) and LSTM

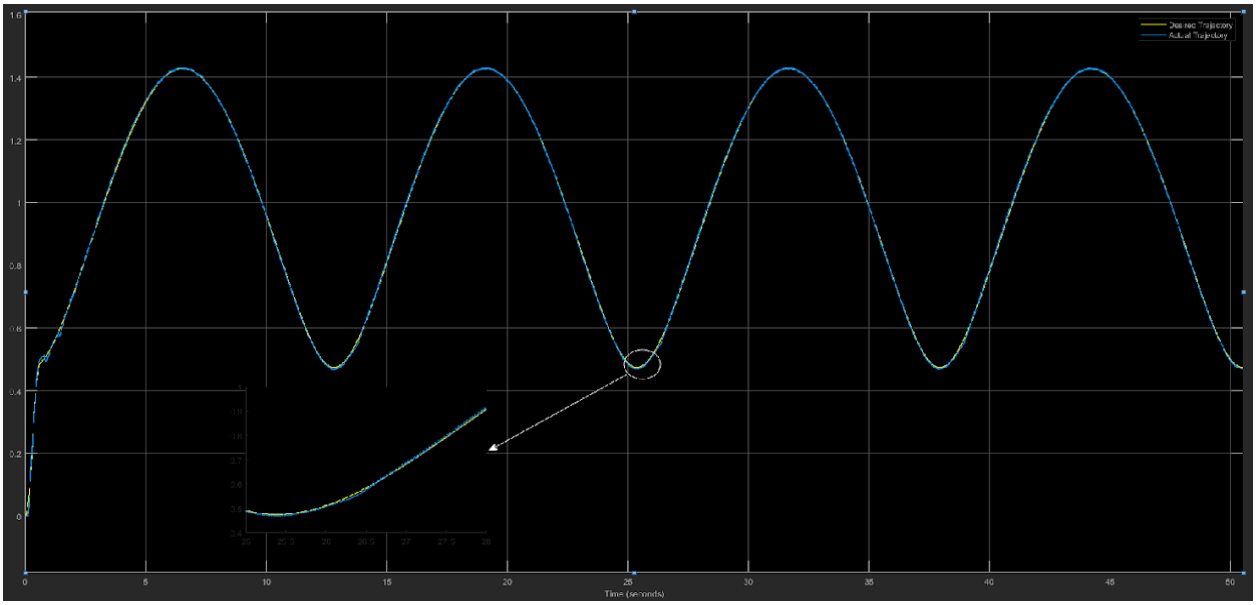

Figure 5.29: Trajectory Tracking for Joint 2 using a combination of ASMC (10%) and LSTM

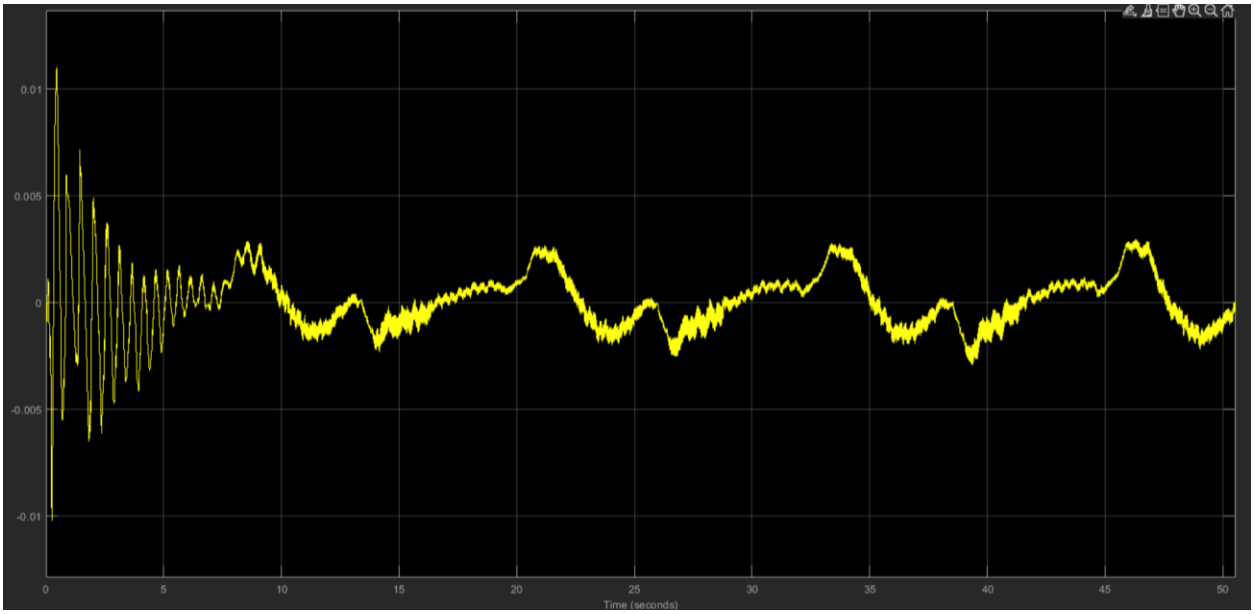

Figure 5.30: Error for Joint 1 using a combination of ASMC (10%) and LSTM

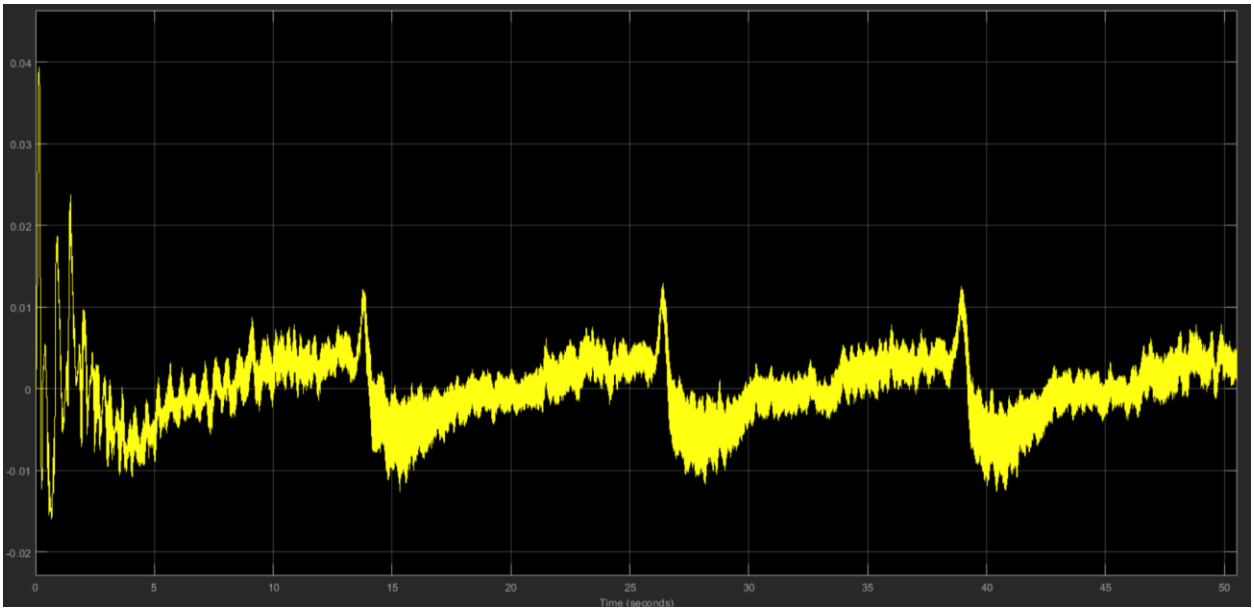

Figure 5.31: Error for Joint 2 using a combination of ASMC (10%) and LSTM

Increase in the computation power positively influence the performance of the deep learning model. The combination of deep learning model and ASMC controller is possible due to performance enhancement. The proposed controller precisely follows the trajectory as shown in Figure 5.27 and 5.28.

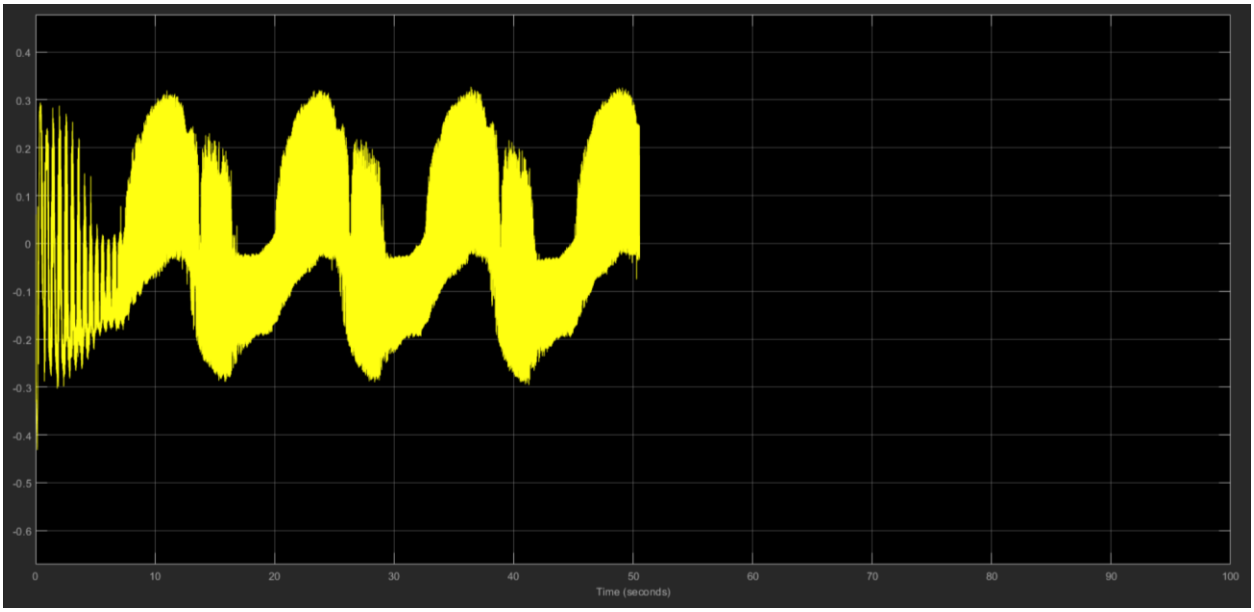

Figure 5.32: Torque for Joint 1 using a combination of ASMC (10%) and LSTM

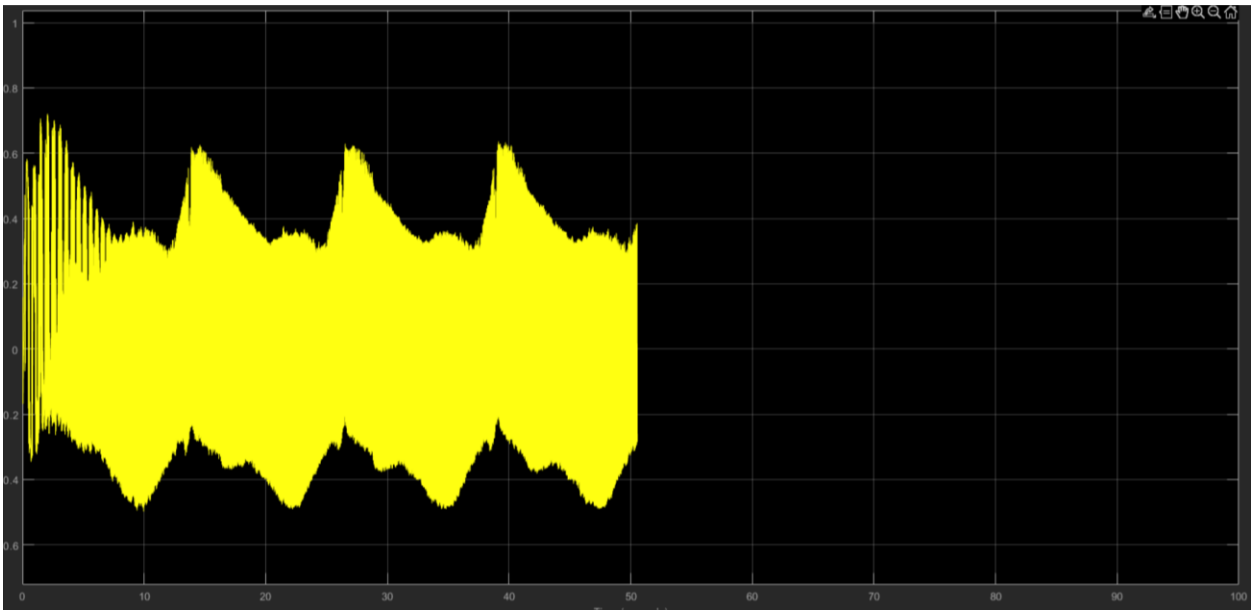

Figure 5.33: Torque for Joint 2 using a combination of ASMC (10%) and LSTM

With a use of memory based neural network such as LSTM, the proposed method follows trajectory more accurately over time. The combination of LSTM with ASMC reduces chattering in the robot.

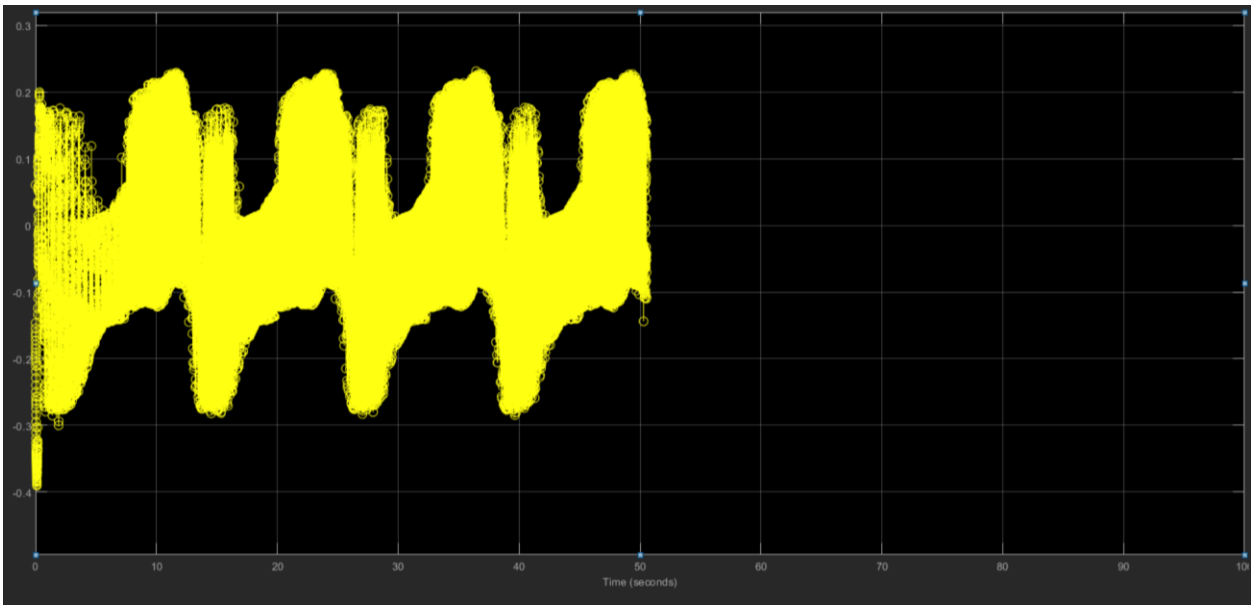

Figure 5.34: Torque generated by LSTM for Joint 1

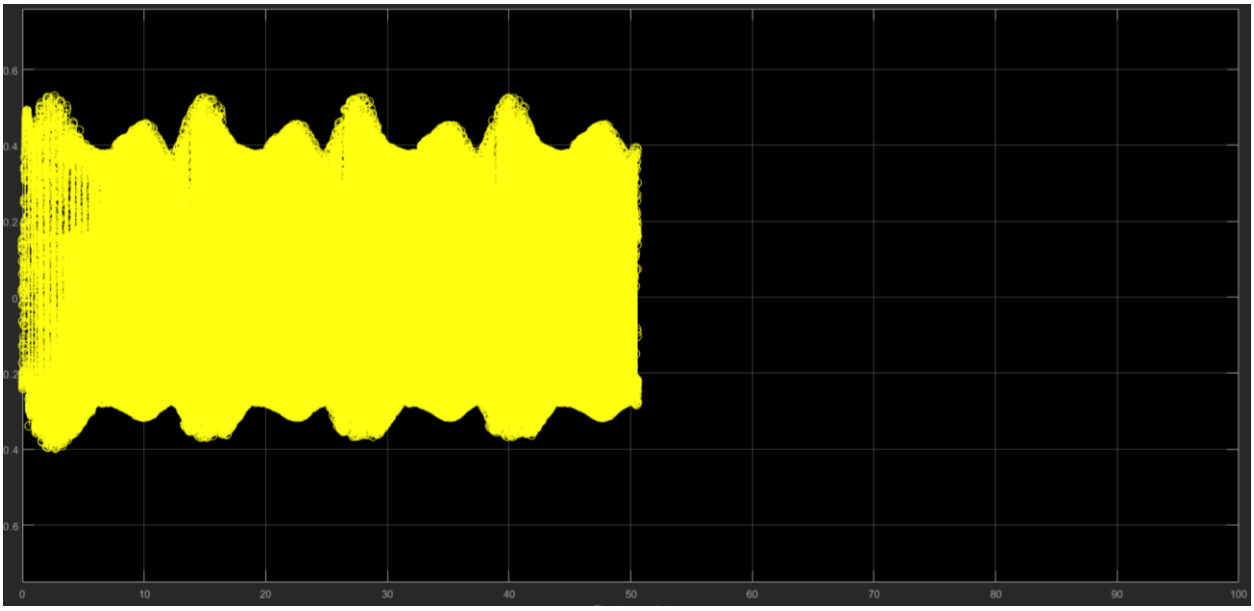

Figure 5.35: Torque generated by LSTM for Joint 2

As shown in Figure 5.33 and 5.34, the LSTM model generates required torque more precisely. In this combination, LSTM performs major role by generating more than 90% of required torque. These results are achieved with the help of proposed controller without tuning the parameters.

5.4.6 Experiment on square wave using a combination of ASMC (10%) and LSTM

Sampling time TS = 0.002

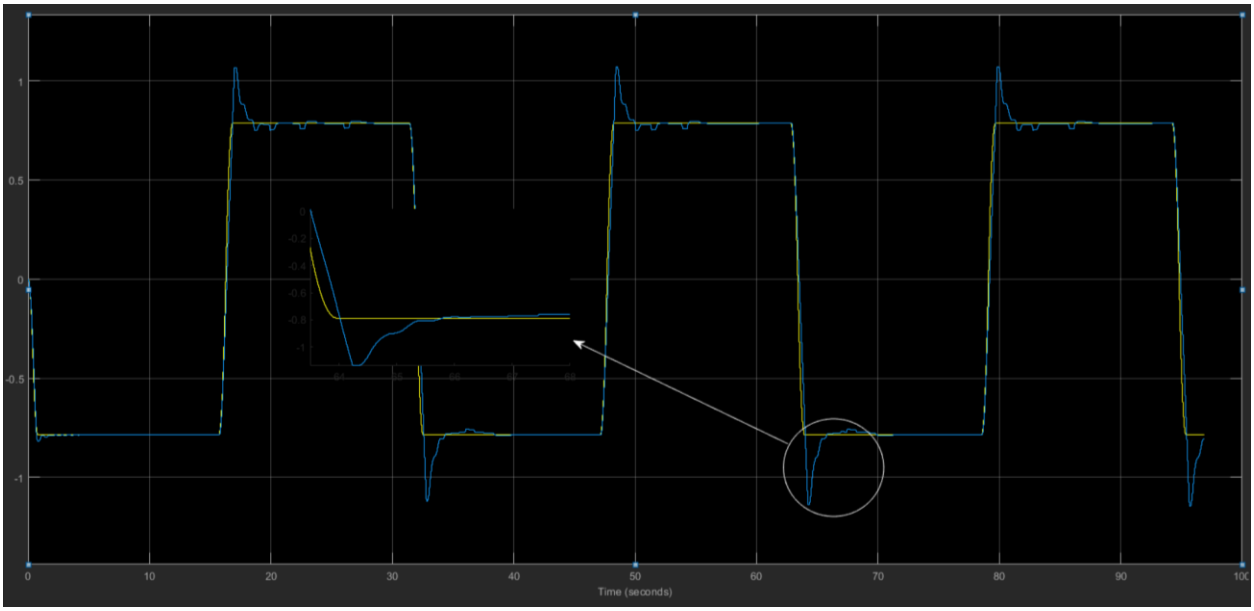

Figure 5.36: Trajectory Tracking on square wave for Joint 1 using a combination of ASMC (10%) and LSTM

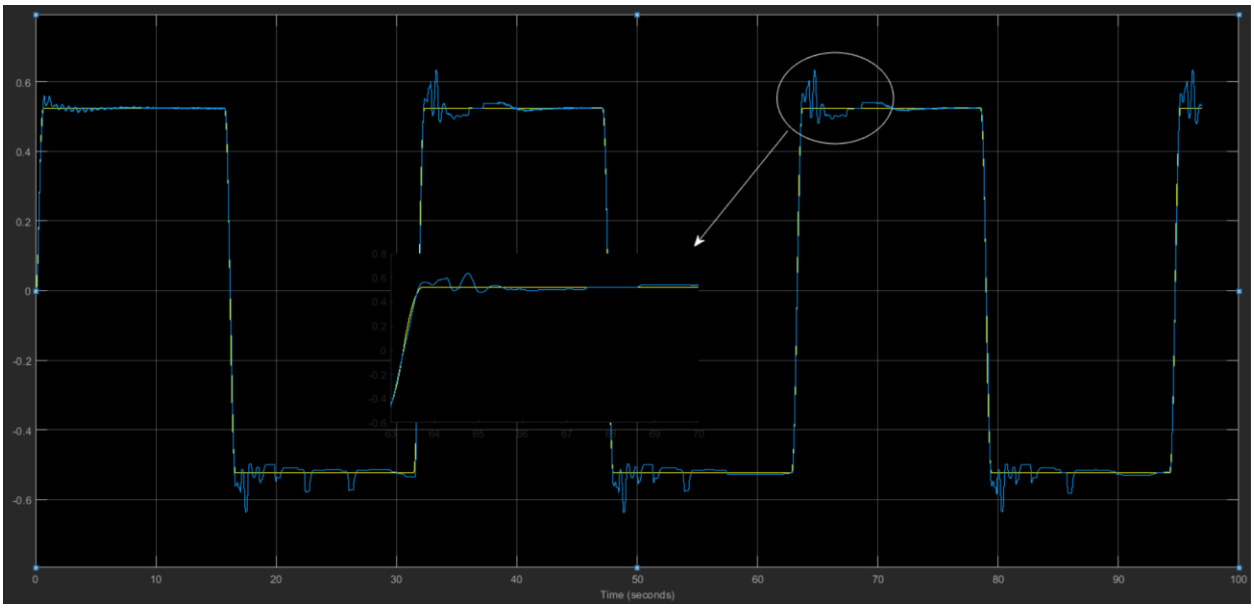

Figure 5.37: Trajectory Tracking on square wave for Joint 2 using a combination of ASMC (10%) and LSTM

To check the robustness, the proposed controller is used for tracking square wave and step function with different frequency. As shown in Figure 5.35 and 5.36, the combination of LSTM and ASMC (10%) has a good performance in tracking the desired trajectory.

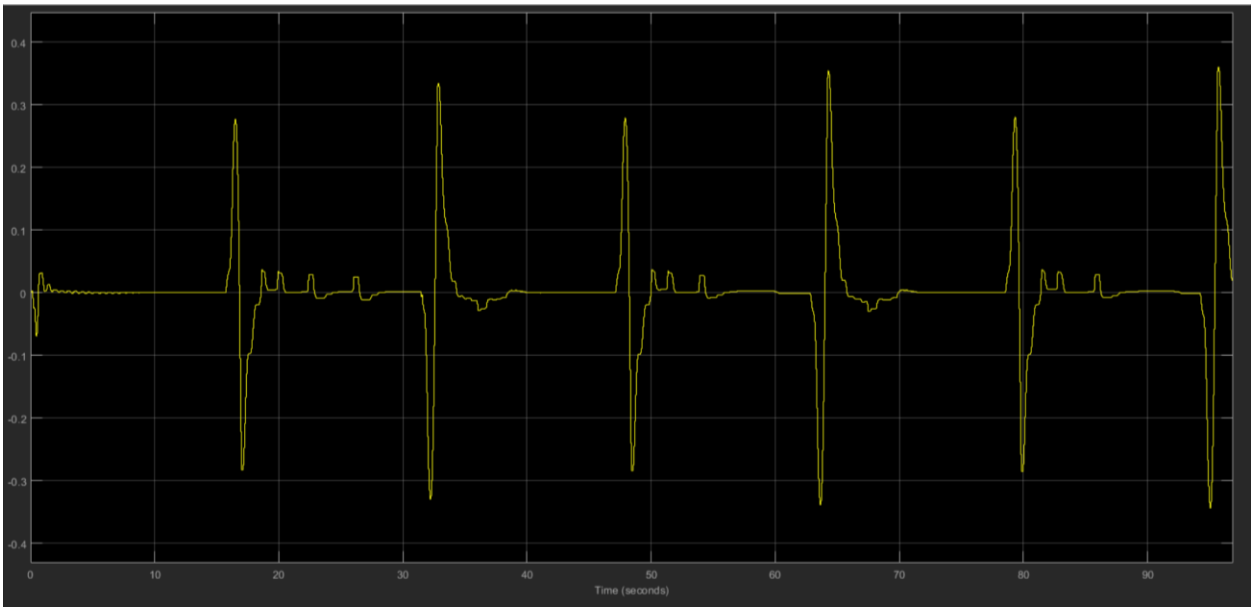

Figure 5.38: Error on square wave for Joint 1 using a combination of ASMC (10%) and LSTM

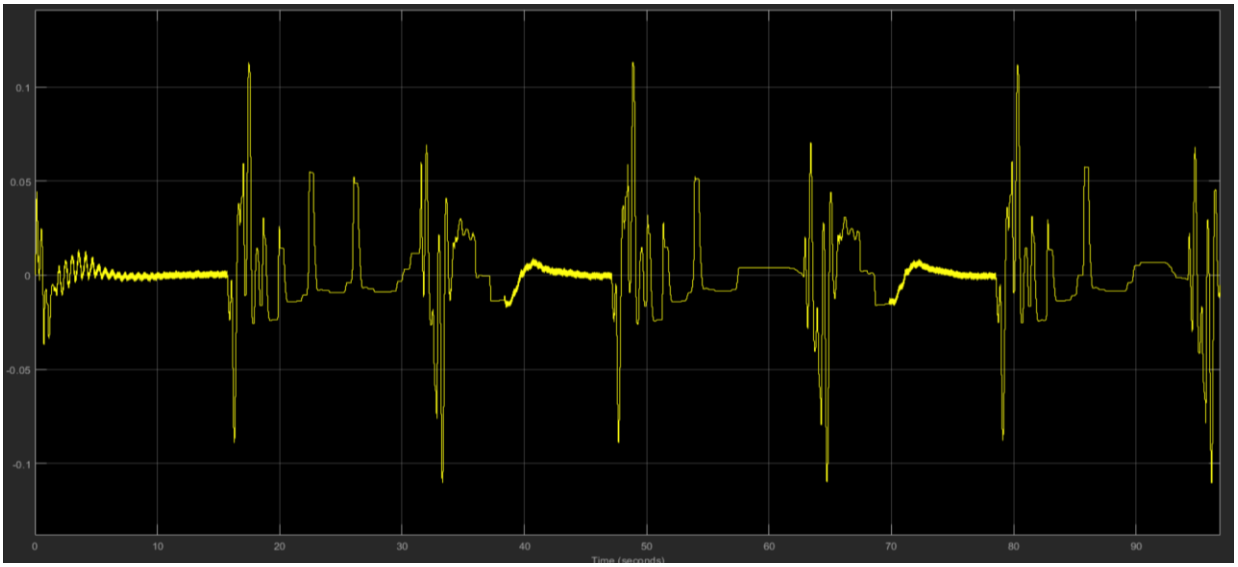

Figure 5.39: Error on square wave for Joint 2 using a combination of ASMC (10%) and LSTM

5.4.7 Experiment on square wave using a combination of ASMC and LSTM

Sampling time TS = 0.002

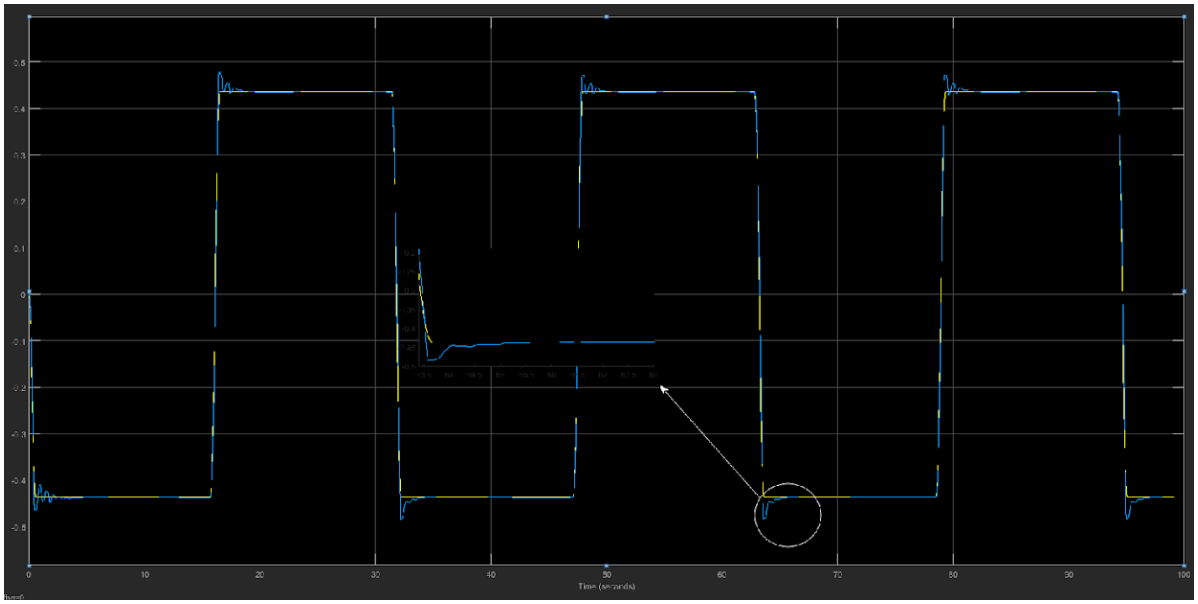

Figure 5.40: Trajectory Tracking on square wave for Joint 1 using a combination of ASMC and LSTM

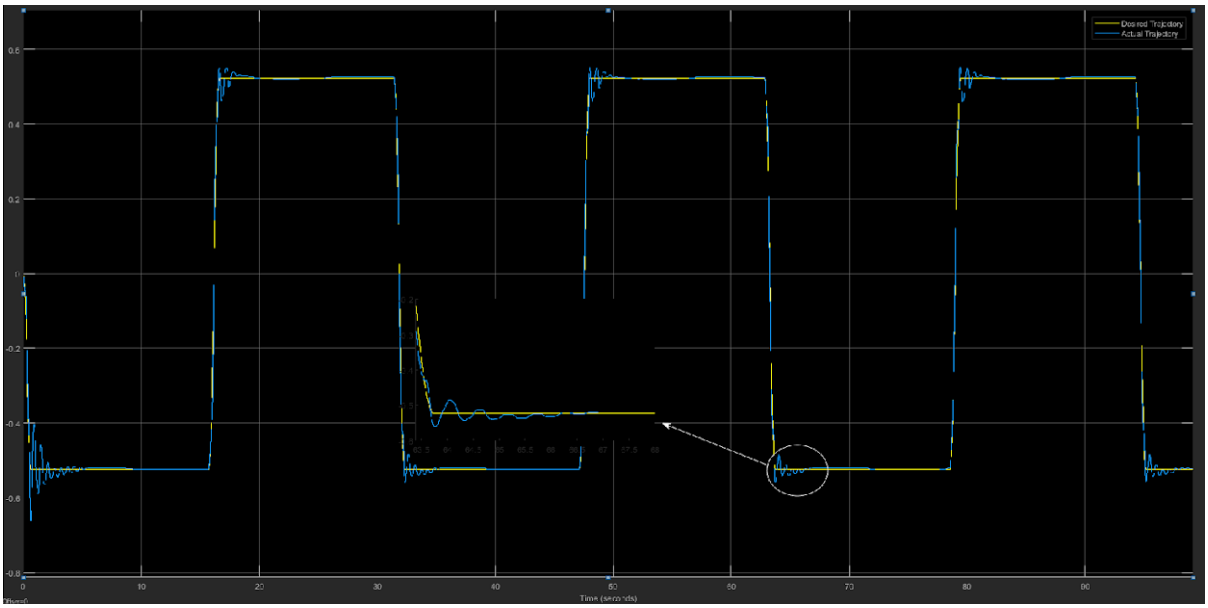

Figure 5.41: Trajectory Tracking on square wave for Joint 2 using a combination of ASMC and LSTM

As shown in 5.39 and 5.40, increasing the ASMC portion in the combination positively influences trajectory tracking. The accuracy of the proposed controller dramatically increased at peak values where the robot changes the direction and stable at certain point (highlighted portion). With the help of knowledge-based component, the error rate decreases for Joint 2.

Figure 5.42: Error on square wave for Joint 1 using a combination of ASMC and LSTM

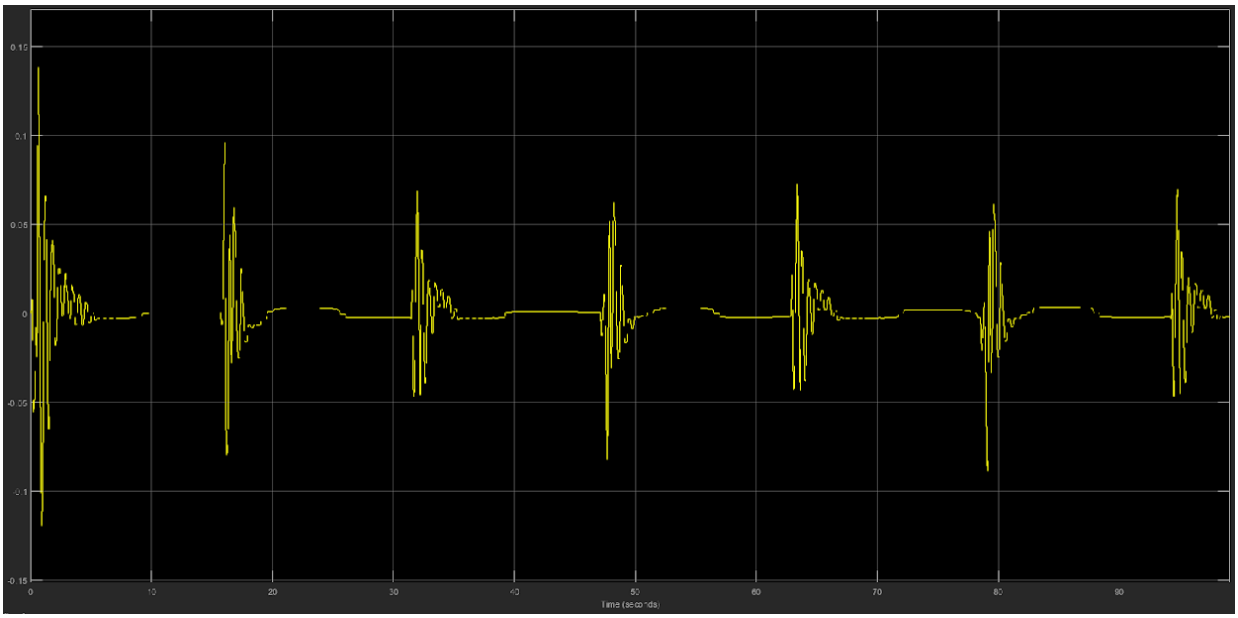

Figure 5.43: Error on square wave for Joint 2 using a combination of ASMC and LSTM

Compared with ASMC only (50%), the proposed controller achieves good trajectory tracking with less fluctuation in the torque. The error rate in this experiment is much lower (±0.08 for Joint 1 and ± 0.09 for Joint 2) compared to the previous experiment (±0.35 for Joint 1 and ± 0.12 for Joint 2).

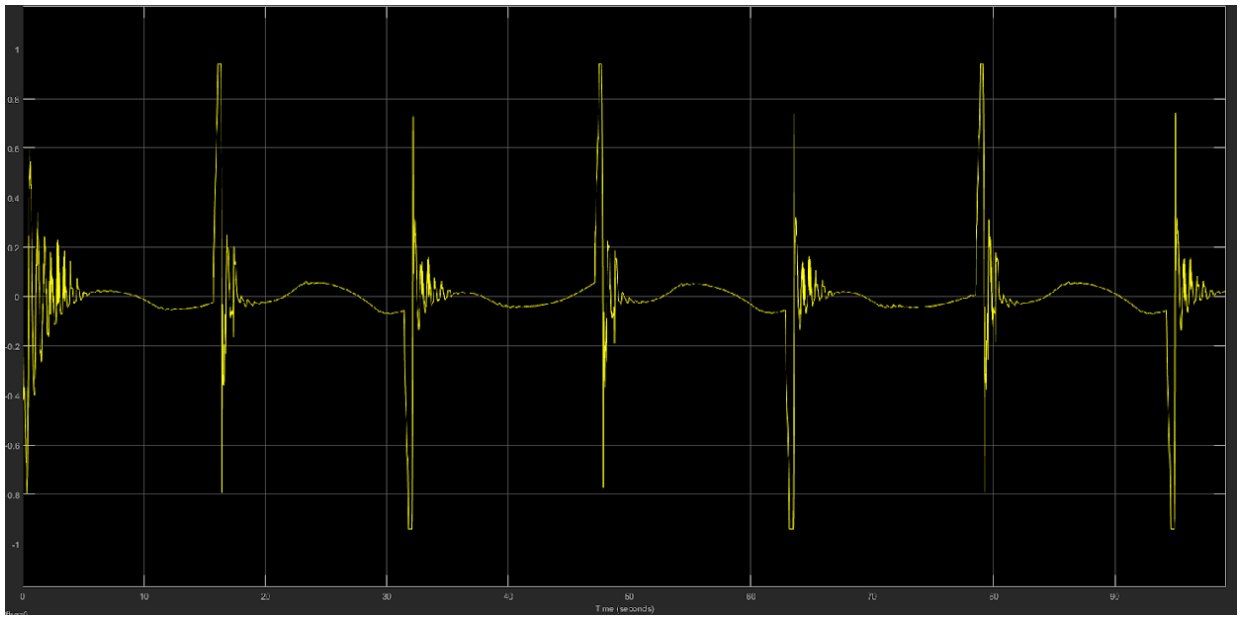

Figure 5.44: Torque on square wave for Joint 1 using a combination of ASMC and LSTM

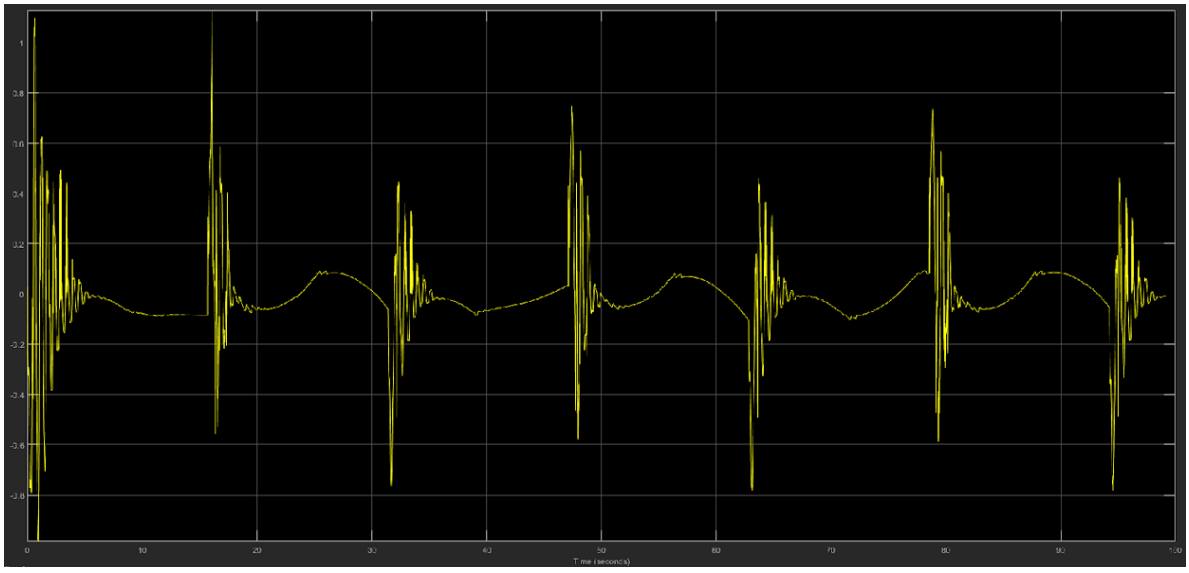

Figure 5.45: Torque on square wave for Joint 2 using a combination of ASMC and LSTM

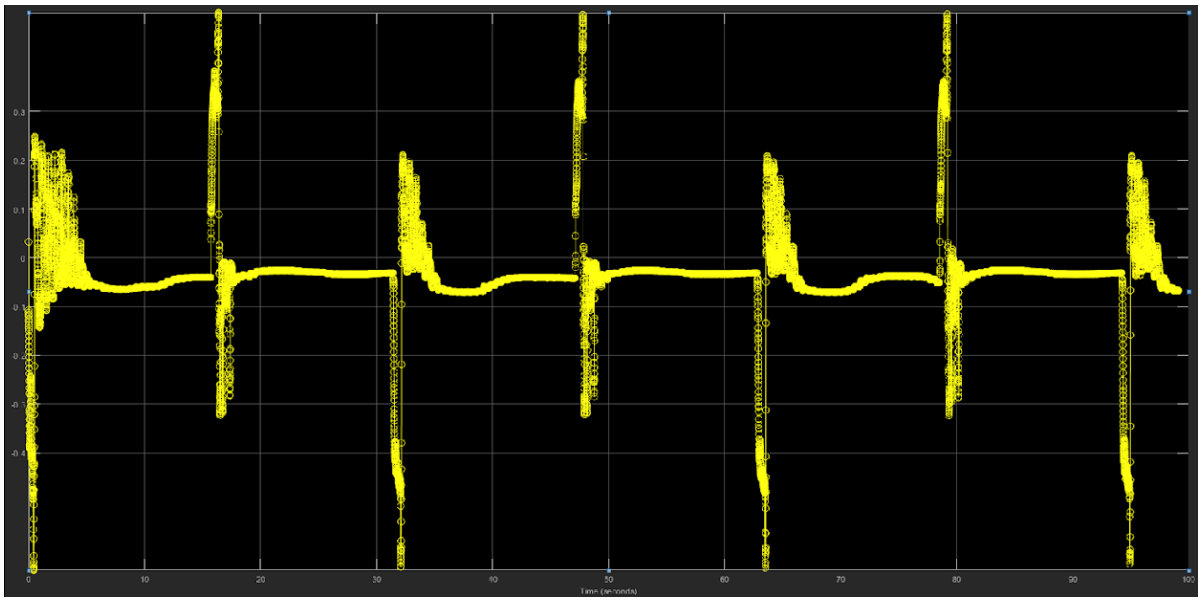

Figure 5.46: Torque generated form LSTM for Joint 1

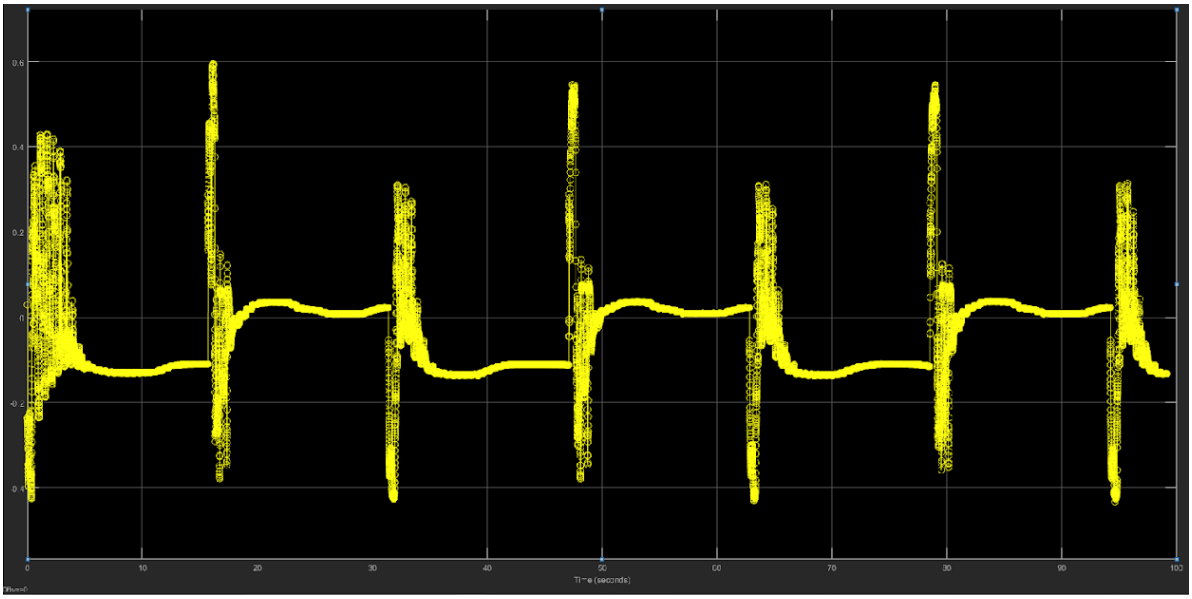

Figure 5.47: Torque generated form LSTM for Joint 2

As shown in Figure 5.45 and 5.46, the torque generated by LSTM (i.e., knowledge-based) is more accurate and reduces the fluctuation, compare to ASMC which calculates the torque based on tracking error. As shown in Figure 5.39 and 5.40, the settling time significantly reduced compared to the previous experiments (LSTM with ASMC (10%) and ASMC only (50%)).

To conclude this chapter, the proposed method has promising trajectory tracking performance and robustness. In the case of ASMC, to get higher accuracy, it is necessary to change the controller gains and parameters, which requires expert knowledge about the control systems. Moreover, changing one parameter requires tuning the controller again to get better accuracy. On the other hand, the proposed LSTM-based controller achieves good performance without tuning the parameters. As this approach uses a neural network for prediction, it only requires a higher computation power. An unseen situation such as external disturbance can be easily handled with the help of ASMC and a knowledge-based component.

With the help of these experiments, it is safe to say that ASMC with LSTM has its own benefits, such as enhancement in tracking, reduction in chattering, and adaptation to the new payload. As discussed in chapter 4, only the knowledge-based component has good accuracy with a sampling time of 0.001 sec but, in the real world, to handle the extreme situation and unknown parameters, the combination of LSTM and ASMC performs better. Comparing sections 5.4.4 and 5.4.5, it turns out that there is a huge benefit of increasing the computation power in this method. The best experiment is performed at 0.002 sampling time which clearly states that there is some room to increase the accuracy of the deep learning model.

Chapter 6

Conclusion, Contribution, and Future Work

6.1 Conclusion

In this research work, a deep recurrent neural network-based adaptive sliding mode control method is proposed to develop intelligent, high-performance, and robust controllers for robots. The proposed controller is implemented and deployed on 2-DOF serial flexible link robot. The simulation and experimental results depict that the proposed method allows predicting the torque value more precisely with and without disturbance. This is accomplished without increasing the complexity of tuning the parameters in the control system. Analyzing the simulation results, it is safe to say that initially ASMC based model lowers the error using higher torque value whereas proposed controller handles it with predicting the reasonable torque value.

6.2 Contribution

The main contribution to the research is to develop recurrent neural network architectures such as LSTM and GRU and use them in a combination with adaptive sliding mode controller. Increasing the depth of an artificial neural network have a problem of vanishing and exploding gradient. As a result, a significant variation in the input of the activation function causes a slight change in the output, making the neural network challenging to train. Recurrent neural networks solve this problem by consisting of loops in them, allowing information to persist. In theory, RNNs can handle “long-term dependencies”, but it does not seem to learn them in the real world. As the network grows linearly, the situation known by the network far before becomes challenging to remember. This problem was explained in depth by Bengio et al. [64]. Advanced architectures of recurrent neural network such as LSTMs [77] and GRUs [80] handles that situation.

The standard TCP/IP client-server architecture is built to get real-time data from the robot. The deep learning model is optimized and trained on data generated from Quanser 2-DOF serial flexible link robot. The extremely challenging part of this investigation was to train and deploy the deep learning model so that the proposed method delivers the expected result on unknown disturbance and uncertain parameters.

Furthermore, several algorithms to train the Long Short-Term Memory and Gated Recurrent Unit are developed and deployed on Cedar server from Compute Canada [85]. A common normalization function is developed to normalize the dataset which is used for training, testing, and deployment.

6.3 Future Work

An ever increase in computation power made deep learning possible to use in real-time control systems. Some of the future works are outlined as follows:

- Because this technique uses a knowledge-based component to forecast the torque value more precisely, it may be readily changed to suit the application. This is especially useful in a production setting when parameters of a robot manipulator change often.
- As shown in chapter 5, the experimental results are conducted using 0.002 sampling time. Comparing these results with simulation results, it is safe to say that there is some room for the improvement in experimental results with the help of more computation power.
- Using recurrent neural network based deep learning model has a major benefit to remember the older situations. Furthermore, the future goal of this research is to add one more layer which changes in real-time to make neural network more precise and adaptive to the environment.

- With the help of GPUs on local machine, some new and advanced deep learning models will be developed and implemented on 2-DOF robot.

References

[1] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson/Prentice Hall, 2005.

[2] “Spot from Boston Dynamics.” https://www.bostondynamics.com/solutions/inspection (accessed Jun. 01, 2021).

[3] “World Robotics Report 2020,” Sep. 2020. Accessed: Jun. 01, 2021. [Online]. Available: https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe

[4] “Next Generation Skills,” International Federation of Robotics, Nov. 2020. Accessed: Jun. 01, 2021. [Online]. Available: https://ifr.org/papers

[5] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. 2005. Accessed: Apr. 04, 2021. [Online]. Available: https://www.wiley.com/en-gb/Robot+Modeling+and+Control-p-9780471649908

[6] J. C. Maxwell, “ON GOVERNORS,” *Proceedings of the Royal Society of London*, vol. 16, pp. 270–283, Jan. 1868, doi: 10.1098/rspl.1867.0055.

[7] N. Minorsky, “Directional Stability of Automatically Steered Bodies,” *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922, doi: 10.1111/j.1559-3584.1922.tb04958.x.

[8] E. G. Christoforou and A. Müller, “R.U.R. Revisited: Perspectives and Reflections on Modern Robotics,” *Int J of Soc Robotics*, vol. 8, no. 2, pp. 237–246, Apr. 2016, doi: 10.1007/s12369-015-0327-6.

[9] J. G. C. Devol, “Programmed article transfer,” US2988237A, Jun. 13, 1961 Accessed: Jun. 11, 2021. [Online]. Available: https://patents.google.com/patent/US2988237A/en

[10] “Unimate - The First Industrial Robot,” *Automate*. https://www.automate.org/blogs/joseph-engelberger-unimate (accessed Jun. 11, 2021).

[11] A. I. Károly, P. Galambos, J. Kuti, and I. J. Rudas, “Deep Learning in Robotics: Survey on Model Structures and Training Strategies,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 266–279, Jan. 2021, doi: 10.1109/TSMC.2020.3018325.

[12] O. Cerman and P. Hušek, “Adaptive fuzzy sliding mode control for electro-hydraulic servo mechanism,” *Expert Systems with Applications*, vol. 39, no. 11, pp. 10269–10277, Sep. 2012, doi: 10.1016/j.eswa.2012.02.172.

[13] D. Zhao, S. Li, and F. Gao, “A New Terminal Sliding Mode Control for Robotic Manipulators,” *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 9888–9893, Jan. 2008, doi: 10.3182/20080706-5-KR-1001.01673.

[14] F. T. MRAD and S. AHMAD, “Adaptive control of flexible joint robots using position and velocity feedback,” *International Journal of Control*, vol. 55, no. 5, pp. 1255–1277, May 1992, doi: 10.1080/00207179208934282.

[15] I. Cervantes and J. Alvarez-Ramirez, “On the PID tracking control of robot manipulators,” *Systems & Control Letters*, vol. 42, no. 1, pp. 37–46, Jan. 2001, doi: 10.1016/S0167-6911(00)00077-3.

[16] Y. Qin, W. Zhang, J. Shi, and J. Liu, “Improve PID controller through reinforcement learning,” in *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, Aug. 2018, pp. 1–6. doi: 10.1109/GNCC42960.2018.9019095.

[17] R. Kelly, “A tuning procedure for stable PID control of robot manipulators,” *Robotica*, vol. 13, no. 2, pp. 141–148, Mar. 1995, doi: 10.1017/S0263574700017641.

[18] K. H. Ang, G. Chong, and Y. Li, “PID control system analysis, design, and technology,” *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, Jul. 2005, doi: 10.1109/TCST.2005.847331.

[19] B. Özyer, “Adaptive fast sliding neural control for robot manipulator,” *Turk J Elec Eng & Comp Sci*, vol. 28, pp. 3154–3167, 2020.

[20] B. R. Markiewicz, “Analysis of the Computed Torque Drive Method and Comparison With Conventional Position Servo for a Computer-Controlled Manipulator,” *NASA-JPL Technical Memo*, Mar. 1973.

[21] E. Rastogi and L. B. Prasad, “Comparative performance analysis of PD/PID computed torque control, filtered error approximation based control and NN control for a robot manipulator,” in *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)*, Dec. 2015, pp. 1–6. doi: 10.1109/UPCON.2015.7456706.

[22] J.-J. E. Slotine and W. Li, “On the Adaptive Control of Robot Manipulators,” *The International Journal of Robotics Research*, vol. 6, no. 3, pp. 49–59, Sep. 1987, doi: 10.1177/027836498700600303.

[23] D. Koditschek, “Natural motion for robot arms,” in *The 23rd IEEE Conference on Decision and Control*, Dec. 1984, pp. 733–735. doi: 10.1109/CDC.1984.272106.

[24] V. Utkin, “Variable structure systems with sliding modes,” *IEEE Transactions on Automatic Control*, vol. 22, no. 2, pp. 212–222, Apr. 1977, doi: 10.1109/TAC.1977.1101446.

[25] M. Zhihong, A. P. Paplinski, and H. R. Wu, “A robust MIMO terminal sliding mode control scheme for rigid robotic manipulators,” *IEEE Transactions on Automatic Control*, vol. 39, no. 12, pp. 2464–2469, Dec. 1994, doi: 10.1109/9.362847.

[26] A.-M. Zou, K. D. Kumar, Z.-G. Hou, and X. Liu, “Finite-Time Attitude Tracking Control for Spacecraft Using Terminal Sliding Mode and Chebyshev Neural Network,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 4, pp. 950–963, Aug. 2011, doi: 10.1109/TSMCB.2010.2101592.

[27] P. Kachroo and M. Tomizuka, “Chattering reduction and error convergence in the sliding-mode control of a class of nonlinear systems,” *IEEE Transactions on Automatic Control*, vol. 41, no. 7, pp. 1063–1068, Jul. 1996, doi: 10.1109/9.508917.

[28] X. Zhang and X. Zhu, “Adaptive SMC Design for Nonlinear Systems Based on Wavelet Observers,” in *2008 Fourth International Conference on Natural Computation*, Oct. 2008, vol. 2, pp. 281–285. doi: 10.1109/ICNC.2008.799.

[29] J. Baek, M. Jin, and S. Han, “A New Adaptive Sliding-Mode Control Scheme for Application to Robot Manipulators,” *IEEE Transactions on Industrial Electronics*, vol. 63, no. 6, pp. 3628–3637, Jun. 2016, doi: 10.1109/TIE.2016.2522386.

[30] H. Wang, “Adaptive Control of Robot Manipulators With Uncertain Kinematics and Dynamics,” *IEEE Transactions on Automatic Control*, vol. 62, no. 2, pp. 948–954, Feb. 2017, doi: 10.1109/TAC.2016.2575827.

[31] M. Zeinali and L. Notash, “Adaptive sliding mode control with uncertainty estimator for robot manipulators,” *Mechanism and Machine Theory*, vol. 45, no. 1, pp. 80–90, Jan. 2010, doi: 10.1016/j.mechmachtheory.2009.08.003.

[32] L. A. Zadeh, “Fuzzy sets,” *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965, doi: 10.1016/S0019-9958(65)90241-X.

[33] R. Czabanski, M. Jezewski, and J. Leski, “Introduction to Fuzzy Systems,” in *Theory and Applications of Ordered Fuzzy Numbers: A Tribute to Professor Witold Kosiński*, P. Prokopowicz, J. Czerniak, D. Mikołajewski, Ł. Apiecionek, and D. Ślęzak, Eds. Cham: Springer International Publishing, 2017, pp. 23–43. doi: 10.1007/978-3-319-59614-3_2.

[34] C.-L. Hwang, “A novel Takagi-Sugeno-based robust adaptive fuzzy sliding-mode controller,” *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 5, pp. 676–687, Oct. 2004, doi: 10.1109/TFUZZ.2004.834811.

[35] M. Zeinali, “Adaptive chattering-free sliding mode control design using fuzzy model of the system and estimated uncertainties and its application to robot manipulators,” in *2015 International Workshop on Recent Advances in Sliding Modes (RASM)*, Apr. 2015, pp. 1–6. doi: 10.1109/RASM.2015.7154652.

[36] M. Zeinali and L. Notash, “Systematic adaptive fuzzy logic modelling of complex systems from input-output data,” *Transactions of the Canadian Society for Mechanical Engineering*, vol. 29, pp. 569–580, Dec. 2005, doi: 10.1139/tcsme-2005-0036.

[37] M. Zeinali and A. Khajepour, “Development of an adaptive fuzzy logic-based inverse dynamic model for laser cladding process,” *Engineering Applications of Artificial Intelligence*, vol. 23, pp. 1408–1419, Dec. 2010, doi: 10.1016/j.engappai.2009.11.006.

[38] K. F. Man, K. S. Tang, and S. Kwong, “Genetic algorithms: concepts and applications [in engineering design],” *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, Oct. 1996, doi: 10.1109/41.538609.

[39] V. Maniezzo, “Genetic evolution of the topology and weight distribution of neural networks,” *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 39–53, Jan. 1994, doi: 10.1109/72.265959.

[40] V. Roberge, M. Tarbouchi, and G. Labonte, “Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 132–141, Feb. 2013, doi: 10.1109/TII.2012.2198665.

[41] L. Wang, T. Chai, and L. Zhai, "Neural-Network-Based Terminal Sliding-Mode Control of Robotic Manipulators Including Actuator Dynamics," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 9, pp. 3296–3304, Sep. 2009, doi: 10.1109/TIE.2008.2011350.

[42] S. P. Chan, "Real time control of robot manipulator using a neural network based learning controller," in *Proceedings of IECON '93 - 19th Annual Conference of IEEE Industrial Electronics*, Nov. 1993, pp. 1825–1830 vol.3. doi: 10.1109/IECON.1993.339351.

[43] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 4653–4660. doi: 10.1109/CDC.2016.7798978.

[44] S. Edhah, S. Mohamed, A. Rehan, M. AlDhaheri, A. AlKhaja, and Y. Zweiri, "Deep Learning Based Neural Network Controller for Quad Copter: Application to Hovering Mode," in *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Nov. 2019, pp. 1–5. doi: 10.1109/ICECTA48151.2019.8959776.

[45] B. Huang, Z. Li, X. Wu, A. Ajoudani, A. Bicchi, and J. Liu, "Coordination Control of a Dual-Arm Exoskeleton Robot Using Human Impedance Transfer Skills," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 5, pp. 954–963, May 2019, doi: 10.1109/TSMC.2017.2706694.

[46] "Adaptive Hands-On Control for Reaching and Targeting Tasks in Surgery - Elisa Beretta, Elena De Momi, Ferdinando Rodriguez y Baena, Giancarlo Ferrigno, 2015." https://journals.sagepub.com/doi/10.5772/60130 (accessed Apr. 05, 2021).

[47] H. Su, N. Enayati, L. Vantadori, A. Spinoglio, G. Ferrigno, and E. De Momi, "Online human-like redundancy optimization for tele-operated anthropomorphic manipulators," *International Journal of Advanced Robotic Systems*, vol. 15, no. 6, p. 1729881418814695, Nov. 2018, doi: 10.1177/1729881418814695.

[48] T. Rymarczyk and G. Kłosowski, "Application of neural reconstruction of tomographic images in the problem of reliability of flood protection facilities," *Eksploatacja i Niezawodnosc - Maintenance and Reliability*, vol. 20, pp. 425–434, Jun. 2018, doi: 10.17531/ein.2018.3.11.

[49] S. Zernetsch, S. Kohnen, M. Goldhammer, K. Doll, and B. Sick, "Trajectory prediction of cyclists using a physical model and an artificial neural network," *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, doi: 10.1109/IVS.2016.7535484.

[50] S. Martin and C. T. M. Choi, "A Post-Processing Method for Three-Dimensional Electrical Impedance Tomography," *Scientific Reports*, vol. 7, no. 1, Art. no. 1, Aug. 2017, doi: 10.1038/s41598-017-07727-2.

[51] B. Kalantar, B. Pradhan, S. A. Naghibi, A. Motevalli, and S. Mansor, "Assessment of the effects of training data selection on the landslide susceptibility mapping: a comparison between support vector machine (SVM), logistic regression (LR) and artificial neural networks (ANN)," *Geomatics, Natural Hazards and Risk*, vol. 9, no. 1, pp. 49–69, Jan. 2018, doi: 10.1080/19475705.2017.1407368.

[52] A. Aditian, T. Kubota, and Y. Shinohara, "Comparison of GIS-based landslide susceptibility models using frequency ratio, logistic regression, and artificial neural network in a tertiary region of Ambon, Indonesia," *Geomorphology*, vol. 318, pp. 101–111, Oct. 2018, doi: 10.1016/j.geomorph.2018.06.006.

[53] Z. Su, “A neural network-based controller for a single-link flexible manipulator using the inverse dynamics approach,” masters, Concordia University, 2000. Accessed: Apr. 05, 2021. [Online]. Available: https://spectrum.library.concordia.ca/1204/

[54] W. Chatlatanagulchai and P. H. Meckl, “Motion control of two-link flexible-joint robot with actuator nonlinearities, using neural networks and direct method,” in *Proceedings of 2005 IEEE Conference on Control Applications, 2005. CCA 2005.*, Aug. 2005, pp. 1552–1557. doi: 10.1109/CCA.2005.1507353.

[55] H. Mori, Y. Ohama, N. Fukumura, and Y. Uno, “Learning of real robot’s inverse dynamics by a forward-propagation learning rule,” *Electrical Engineering in Japan - ELEC ENG JPN*, vol. 161, pp. 38–48, Dec. 2007, doi: 10.1002/eej.20456.

[56] Z. Jiang, T. Ishida, and M. Sunawada, “Neural Network Aided Dynamic Parameter Identification of Robot Manipulators,” in *2006 IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2006, vol. 4, pp. 3298–3303. doi: 10.1109/ICSMC.2006.384627.

[57] Y. LeCun *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[58] H. Su, W. Qi, C. Yang, A. Aliverti, G. Ferrigno, and E. D. Momi, “Deep Neural Network Approach in Human-Like Redundancy Optimization for Anthropomorphic Manipulators,” *IEEE Access*, vol. 7, pp. 124207–124216, 2019, doi: 10.1109/ACCESS.2019.2937380.

[59] F. Chollet, *Deep Learning with Python*. 2017. Accessed: Apr. 04, 2021. [Online]. Available: https://www.manning.com/books/deep-learning-with-python

[60] A. F. R. Araujo and H. D’Arbo, “A partially recurrent neural network to perform trajectory planning, inverse kinematics, and inverse dynamics,” in *SMC’98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, Oct. 1998, vol. 2, pp. 1784–1789 vol.2. doi: 10.1109/ICSMC.1998.728153.

[61] Ken Onozato and Yutaka Maeda, “Learning of inverse-dynamics and inverse-kinematics for two-link SCARA robot using neural networks,” in *SICE Annual Conference 2007*, Sep. 2007, pp. 1031–1034. doi: 10.1109/SICE.2007.4421135.

[62] N. Ishibashi and Y. Maeda, *Learning of inverse-dynamics for SCARA robot*. 2011, p. 1303.

[63] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, Apr. 1998, doi: 10.1142/S0218488598000094.

[64] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994, doi: 10.1109/72.279181.

[65] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[66] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *arXiv:1412.3555 [cs]*, Dec. 2014, Accessed: Jun. 14, 2021. [Online]. Available: http://arxiv.org/abs/1412.3555

[67] N. Liu *et al.*, “Modeling and Simulation of Robot Inverse Dynamics Using LSTM-Based Deep Learning Algorithm for Smart Cities and Factories,” *IEEE Access*, vol. 7, pp. 173989–173998, 2019, doi: 10.1109/ACCESS.2019.2957019.

[68] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, “Learning inverse dynamics models in O(n) time with LSTM networks,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov. 2017, pp. 811–816. doi: 10.1109/HUMANOIDS.2017.8246965.

[69] M. Zeinali, “Adaptive chattering-free sliding mode control design using fuzzy model of the system and estimated uncertainties and its application to robot manipulators,” in *2015 International Workshop on Recent Advances in Sliding Modes (RASM)*, Apr. 2015, pp. 1–6. doi: 10.1109/RASM.2015.7154652.

[70] M. Zeinali and H. Wang, “New Methodology to Design Learning Control for Robots Using Adaptive Sliding Mode Control and Multi-Model Neural Networks,” presented at the International Conference of Control, Dynamic Systems, and Robotics, Jun. 2018. doi: 10.11159/cdsr18.140.

[71] G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science (New York, N.Y.)*, vol. 313, pp. 504–7, Aug. 2006, doi: 10.1126/science.1127647.

[72] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, Art. no. 6088, Oct. 1986, doi: 10.1038/323533a0.

[73] A. Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks.” http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (accessed Apr. 06, 2021).

[74] S. Raschka, “Python Machine Learning - Third Edition,” *Packt*. https://www.packtpub.com/product/python-machine-learning-third-edition/9781789955750 (accessed Apr. 20, 2021).

[75] P. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Nov. 1990, doi: 10.1109/5.58337.

[76] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training Recurrent Neural Networks,” *arXiv:1211.5063 [cs]*, Feb. 2013, Accessed: Apr. 20, 2021. [Online]. Available: http://arxiv.org/abs/1211.5063

[77] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[78] “Understanding LSTM Networks -- colah’s blog.” https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Jun. 15, 2021).

[79] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016. Accessed: Jun. 22, 2021. [Online]. Available: https://mitpress.mit.edu/books/deep-learning

[80] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” *arXiv:1406.1078 [cs, stat]*, Sep. 2014, Accessed: Apr. 04, 2021. [Online]. Available: http://arxiv.org/abs/1406.1078

[81] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, Art. no. 4, Aug. 2018, doi: 10.1007/s13244-018-0639-9.

[82] B. Chen, “A Practical Introduction to Early Stopping in Machine Learning,” *Medium*, Aug. 03, 2020. https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd (accessed Jun. 21, 2021).

[83] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Dec. 2014, Accessed: Jun. 22, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980

[84] “2 DOF Serial Flexible Joint,” *Quanser*. http://www.quanser.com/products/2-dof-serial-flexible-joint/ (accessed Jul. 06, 2021).

[86] “Compute Canada - Calcul Canada,” *Compute Canada - Calcul Canada*. https://www.computecanada.ca (accessed Jul. 07, 2021).

Appendix A

Technical Specification of Data Acquisition Board

Appendix A.1: QPIDe data acquisition board from Quanser

As shown in figure, QPIDe is used to communicate to the robot. This card is connected to the local computer using PCIe Express slot on the motherboard. The major benefit of this card is a large buffer and multiple programming language support. The technical specification about this board is extracted from Quanser data sheet available on Quanser website (www.quanser.com) and is as follows:

Analog inputs	• Number of channels: 8 • Resolution: 16-bit • Input range: ± 10 V
Digital IO	• Number of channels: 56

Encoder inputs	• Number of channels: 8 • Count frequency in quadrature decoding: 40 MHx
PWM outputs	• Output low: 0.4 V • Output high: 2.4 V • Minimum frequency: 9.6 Hz • Maximum frequency: 20 MHz
SPI	• Max data rate: 10 MHz • Bit width range: 1-32 bit
General purpose timers	• Number of Channels (16-bit): 2 • Resolution (16-bit): 800 ns • Number of channels (32-bit): 2 • Resolution (32-bit): 25 ns