## ABSTRACT

# NEURAL NETWORK-BASED PID COMPENSATION FOR NONLINEAR SYSTEMS: BALL-ON-PLATE EXAMPLE

A. Mohammadi, M.S.
Department of Mechanical Engineering
Northern Illinois University, 2018
Dr. Ji-Chul Ryu, Director

Controller design of a nonlinear system is in general very difficult. One way to avoid such complexity is using a simplified model so that certain nonlinear control techniques can be easily applied. Using a linearized model could make the controller design even simpler. However, some control error is inevitable with a simplified model. Therefore, in this thesis, a neural network-based approach is proposed in order to compensate for the errors caused by using a simplified dynamic model.

The base controller which is designed by using the simplified dynamic model will be compensated by a PID controller with adjustable gains. A neural network is used to update the PID gains during control process. Finally, the outputs of the NN-based PID compensator and the base controller are added together to control the actual nonlinear system. This way, the NN-based PID compensator tries to compensate for the effects of the ignored nonlinear terms of the dynamic model.

The performance of the proposed control method is verified on the ball-on-plate system that is built for this study. Approximate feedback linearization is applied as the base controller on a simplified decoupled dynamic model. A NN-based PID compensator is added to each decoupled ball-on-beam system. Experimental results that show better stabilization and trajectory tracking performance are provided and discussed in the thesis.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

AUGUST 2018

# NEURAL NETWORK-BASED PID COMPENSATION FOR NONLINEAR SYSTEMS: BALL-ON-PLATE EXAMPLE

BY

A. MOHAMMADI

A THESIS SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING

Thesis Director:
Dr. Ji-Chul Ryu

# ACKNOWLEDGEMENTS

I would like to appreciate Dr. Ji-Chul Ryu for his support in the process of completing the thesis. I received both technical and financial support from my advisor.

Internship experience at Apellix, summer 2017, helped me a lot to write the programming part of the thesis. I used almost everything that I learned there.

Research group members also gave me suggestions that led to new ideas, so I would like to appreciate Revanth Raghuram Konda, Sandeep Reddy Erumalla, and Tyler Thompson. They also helped me in choosing the right motor for the experimental setup.

Last but not least, I would like to appreciate the Mechanical Engineering Department for their partial funding support to purchase experimental setup materials.

# DEDICATION

There is nothing more important in life than faith in God. He helped me a lot to feel stronger in each step of my education. I also need to appreciate my family, especialy my parents, who helped me and were supportive. I would like to dedicate the present research to my parents. I hope I can make them happier than before.

# TABLE OF CONTENTS

Page

LIST OF TABLES . . . . . . . . . . vi

LIST OF FIGURES. . . . . . . . . . vii

LIST OF APPENDICES . . . . . . . . . . xii

Chapter

1 INTRODUCTION . . . . . . . . . . 1

1.1 Motivation . . . . . . . . . . 3

1.1.1 Neural Networks . . . . . . . . . . 4

1.2 Literature Review . . . . . . . . . . 6

1.3 Objective . . . . . . . . . . 8

2 DYNAMIC MODEL: BALL-ON-PLATE EXAMPLE . . . . . . . . . . 10

2.1 Ball-on-Plate Dynamics . . . . . . . . . . 10

3 CONTROLLER DESIGN . . . . . . . . . . 17

3.1 Overal Structure of the Controller . . . . . . . . . . 17

3.2 Feedback Linearization . . . . . . . . . . 19

3.2.1 Feedback Linearization Conditions and Procedure . . . . . . . . . . 20

3.2.2 Simplified Ball-on-Beam System. . . . . . . . . . . 23

3.3 Neural Networks . . . . . . . . . . 25

3.3.1 Neural Network Structure . . . . . . . . . . 25

3.3.2 NN-Based PID Controller . . . . . . . . . . 28

Chapter Page

4 EXPERIMENTAL SETUP AND RESULTS . . . . . . . . . . 34

4.1 Experimental Setup . . . . . . . . . . 34

4.1.1 Mechanical Design . . . . . . . . . . 35

4.1.2 Vision System . . . . . . . . . . 38

4.1.3 Controller Board . . . . . . . . . . 40

4.2 Simulation and Experimental Results . . . . . . . . . . 42

4.2.1 Stabilization . . . . . . . . . . 43

4.2.1.1 Simulation Results . . . . . . . . . . 44

4.2.1.2 Experimental Results . . . . . . . . . . 46

4.2.2 Trajectory Tracking . . . . . . . . . . 51

4.2.2.1 Simulation Results . . . . . . . . . . 52

4.2.2.2 Experimental Results . . . . . . . . . . 56

5 CONCLUSION . . . . . . . . . . 60

REFERENCES . . . . . . . . . . 62

APPENDICES . . . . . . . . . . 65

# LIST OF TABLES

Table | Page

2.1 Ball-on-plate system generalized coordinates . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.2 Ball-on-plate system parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

4.1 Parameters and values for the ball-on-plate system. . . . . . . . . . . . . . . . . . . . 38

C.1 Mechanical setup items of the ball-on-plate system. . . . . . . . . . . . . . . . . . . . 77

# LIST OF FIGURES

Figure Page

1.1 Feedback linearization approach, ball-on-wheel system [1] . . . . . . . . . . . . . . 2

1.2 Feedback linearization approach, disk-on-disk system [2] . . . . . . . . . . . . . . . 2

1.3 Approximate feedback linearization approach, ball-on-sphere system [5] . . . . 3

1.4 Neural network structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.5 PID-like neural network controller, double inverted pendulum system [10] . . 6

1.6 Reference compensation technique (RCT), two-dimensional inverted pendulum system [13] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

1.7 Neural network controller, PAM manipulator [7] . . . . . . . . . . . . . . . . . . . . . 7

2.1 Ball-on-plate system . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.2 Local $oxyz$ and global $OXYZ$ coordinate systems . . . . . . . . . . . . . . . . . . 12

3.1 Decoupled controller design for the ball-on-plate system . . . . . . . . . . . . . . . 17

3.2 Overal structure of the controller applied on the ball-on-plate system . . . . . 18

3.3 Inside of the controller block diagram ($x$ axis) . . . . . . . . . . . . . . . . . . . . . . 18

3.4 Neural network structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.5 A neuron in a neural network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.6 Neural network structure (along $x$ axis) . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.7 Tangent sigmoid activation function . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.8 Linear activation function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.9 Sign function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

Figure Page

4.1 Experimental setup . . . . . 34

4.2 Ball-on-plate system designed in SolidWorks . . . . . 35

4.3 Dynamixel motor[2] . . . . . 35

4.4 Linkage mechanism for the plate rotation . . . . . 36

4.5 Camera used in the experiment . . . . . 38

4.6 Dynamixel2USB adapter . . . . . 40

4.7 Torque-current graph of Dynamixel motors[4] . . . . . 41

4.8 Simulation on using AFL in order to stabilize the ball at $(0, 0)$ (center point simulation) . . . . . 45

4.9 Simulation on using AFL in order to stabilize the ball at $(0.1, -0.1)$ (offset point simulation). . . . . . 45

4.10 Experiment on using AFL in order to stabilize the ball at $(0, 0)$ . . . . . 46

4.11 Experiment on using AFL in order to stabilize the ball at $(0.1, -0.1)$. . . . . . 47

4.12 Experiment on using AFLNNPID in order to stabilize the ball at $(0.1, -0.1)$ 48

4.13 Experiment on PID gains in $x$ axis in order to balance the ball at $(0.1, -0.1)$ 49

4.14 Experiment on AFL control input of AFLNNPID in $x$ axis in order to balance the ball at $(0.1, -0.1)$ . . . . . 49

4.15 Experiment on NNPID control input of AFLNNPID in $x$ axis in order to balance the ball at $(0.1, -0.1)$. . . . . . 50

4.16 Experiment on total control input (AFLNNPID) in $x$ axis in order to balance the ball at $(0.1, -0.1)$ . . . . . 50

4.17 Simulation on using AFL for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m) . . . . . 52

4.18 Simulation on using AFL for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m) . . . . . 52

4.19 Simulation on using AFLNNPID for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m), simulation . . . . . 53

Figure Page

4.20 Simulation on PID gains in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m) . . . . . 54

4.21 Simulation on AFL control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m) . . . . . 54

4.22 Simulation on NNPID control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m) . . . . . 55

4.23 Simulation on total control input (AFLNNPID) in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m) . . . . . 55

4.24 Experiment on using AFL for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m), experiment . . . . . 56

4.25 Experiment on using AFLNNPID for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m), experiment . . . . . 56

4.26 Experiment on PID gains in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m) . . . . . 57

4.27 Experiment on AFL control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m) . . . . . 58

4.28 Experiment on NNPID control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m) . . . . . 58

4.29 Experiment on total control input (AFLNNPID) in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m) . . . . . 59

B.1 Inside of controller block diagram ($y$ axis) . . . . . 70

B.2 Neural network structure ($y$ axis) . . . . . 72

C.1 Base plate designed in SolidWorks . . . . . 78

C.2 Connecting rod of the linkage mechanism designed in SolidWorks . . . . . 78

C.3 Center rod designed in SolidWorks . . . . . 79

C.4 Rod end of the linkage mechanism designed in SolidWorks . . . . . 79

C.5 Top plate (off-the-shelf item) . . . . . 80

Figure Page

C.6 Motor bottom bracket (off-the-shelf item) . . . . . 80

C.7 Motor back frame (off-the-shelf item) . . . . . 81

C.8 Dynamixel motor (off-the-shelf item) . . . . . 81

C.9 Rod end attached to motor side bracket (off-the-shelf item) . . . . . 81

C.10 Motor spacer (off-the-shelf item). . . . . . 82

C.11 Universal joint (off-the-shelf item) . . . . . 82

D.1 Experiment on PID gains in $y$ axis in order to balance the ball at $(0.1, -0.1)$ 84

D.2 Experiment on AFL control input of AFLNNPID in $y$ axis in order to balance the ball at $(0.1, -0.1)$ . . . . . 84

D.3 Experiment on NNPID control input of AFLNNPID in $y$ axis in order to balance the ball at $(0.1, -0.1)$. . . . . . 85

D.4 Experiment on total control input (AFLNNPID) in $y$ axis in order to balance the ball at $(0.1, -0.1)$ . . . . . 85

D.5 Simulation on PID gains in $y$ axis for trajectory tracking ($\omega = 6.0$rad/s and R $= 0.06$m) . . . . . 86

D.6 Simulation on AFL control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega = 6.0$rad/s and R $= 0.06$m) . . . . . 86

D.7 Simulation on NNPID control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega = 6.0$rad/s and R $= 0.06$m) . . . . . 87

D.8 Simulation on total control input (AFLNNPID) in $y$ axis for trajectory tracking ($\omega = 6.0$rad/s and R $= 0.06$m) . . . . . 87

D.9 Experiment on PID gains in $y$ axis for trajectory tracking ($\omega = 2.0$rad/s and R $= 0.08$m) . . . . . 88

D.10 Experiment on AFL control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega = 2.0$rad/s and R $= 0.08$m) . . . . . 88

D.11 Experiment on NNPID control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega = 2.0$rad/s and R $= 0.08$m) . . . . . 89

D.12 Experiment on total control input (AFLNNPID) in $y$ axis for trajectory tracking ($\omega = 2.0$rad/s and R = 0.08m) . . . . . . . . . . . . . . . . . . . . . . . . . . 89

# LIST OF APPENDICES

Appendix Page

A EQUATIONS OF MOTION MATRICES . . . . . . . . . . 65

A.1 Equations of Motion Matrices. . . . . . . . . . 66

B Y AXIS CONTROLLER. . . . . . . . . . 69

B.1 Y Axis Controller . . . . . . . . . . 70

B.1.1 Approximate Feedback Linearization . . . . . . . . . . 70

B.1.2 NN-Based PID Controller . . . . . . . . . . 72

C EXPERIMENTAL SETUP . . . . . . . . . . 76

C.1 Experimental Setup. . . . . . . . . . 77

D Y AXIS RESULTS . . . . . . . . . . 83

D.1 Experiment on Offset Point Stabilization. . . . . . . . . . 84

D.2 Simulation on Trajectory Tracking . . . . . . . . . . 85

D.3 Experiment on Trajectory Tracking . . . . . . . . . . 87

# CHAPTER 1
# INTRODUCTION

Degrees of freedom and number of actuators in a dynamic system are very important. If the number of actuators is less than the degrees of freedom of the dynamic system, it is called underactuated. Balancing an object on another object is an example of an underactuated system in which the object motion can be controlled using a specific control strategy. However, some other underactuated systems are not controllable, which are not in the scope of this thesis.

Control laws are classified into two main categories: *linear* and *nonlinear*. The basic idea behind linear controllers is designing a controller that works within a specific region around an equilibrium point. Therefore, linear controllers are easy to design and only work successfully in a confined state space. One of the most widely used linear controllers is PID controller. A PID is easy to design and robust to some extent due to its feedback nature. These two features have made PID common in industry. However, it lacks the optimality consideration of the control input. LQR is the linear controller that tries to optimize the controller. It considers both state convergence and the amount of control input. On the other hand, nonlinear controllers are designed based on the original system's nonlinear dynamic model. As a result, they have a better control performance and/or a larger workspace (in many times the entire state space of the system), but they are more complex and difficult to design in general.

There are different methods for nonlinear control design, and in this thesis, input-output feedback linearization method is used. Feedback linearization transforms system dynamics to a new space in which the system is represented by a linear form. This way one can take advantage of the linear control theory in the new space and use the mapping relations of the system to go from the transformed space to the original one. Among numerous related studies, Ho et al. in [1] used the method to balance a ball on a wheel as shown in Fig. 1.1. A DC motor attached to the wheel acts as the input and the ball angle is the output.

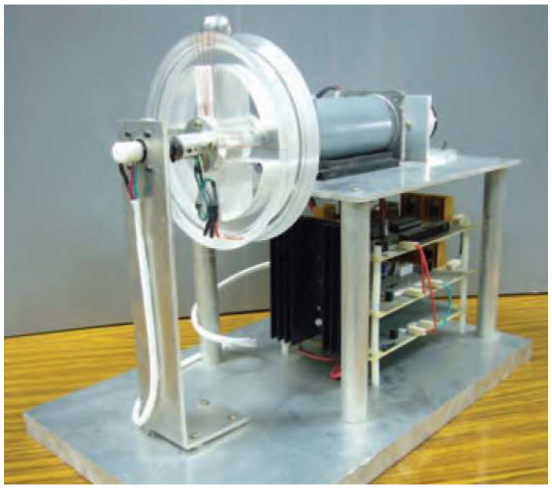

Figure 1.1: Feedback linearization approach, ball-on-wheel system [1]

Ryu et al. in [2] used feedback linearization to balance a disk on a disk, Fig. 1.2. The lower disk is attached to a DC motor that works as input, and the upper disk which, is free to roll, is supposed to be balanced.

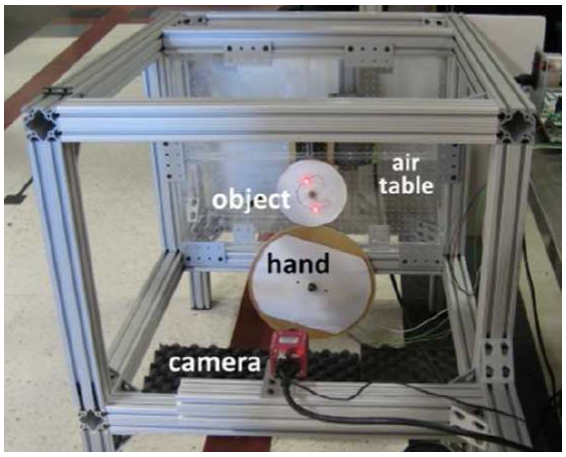

Figure 1.2: Feedback linearization approach, disk-on-disk system [2]

## 1.1 Motivation

Even though feedback linearization is a powerful tool in control design of nonlinear systems, only a set of systems that satisfy the necessary conditions are feedback linearizable. For those of systems that do not satisfy the conditions, a possible solution would be dynamic model approximation. Hauser et al. in [3] showed that the ball-on-beam system is not feedback linearizable, so the simplified dynamic model of the system is used to apply feedback linearization. They demonstrated the performance of the approximated controller through simulations. Due to the approximation, the control law has a limited operation domain because the controller is susceptible to the initial offset from the target states. Guo et al. in [4] used a fuzzy dynamic model of the system and derived two separate linearization laws within those two subspaces. The resulting controller shows better performance in simulation compared to [3]. Liu et al. in [5] also simplified the dynamic model of the ball-on-sphere system to two independent ball-on-wheel systems and then applied feedback linearization on each of them. Since ball-on-wheel system, Fig. 1.1, is feedback linearizable, this nonlinear method stabilized the ball-on-sphere system, shown in Fig. 1.3.

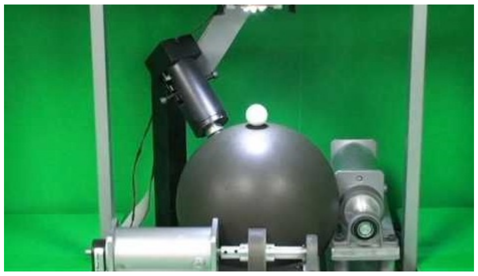

Figure 1.3: Approximate feedback linearization approach, ball-on-sphere system [5]

Ho et al. in [6] applied an approximate feedback linearization controller on the ball-on-plate system. Since ball-on-plate system is not feedback linearizable, they ignored coupling terms to derive two independent ball-on-beam systems. Finally, they applied approximate feedback linearization on each simplified nonlinear system and provided experimental results.

Due to the approximations that have been made in this type of controller design, it seems necessary to find a way to compensate for the effects of the ignored nonlinear terms in the dynamic model. This compensation strategy should be adaptive in order to adjust itself to different system conditions, and it should be robust enough to stabilize the system. For example, the effect of friction is usually ignored in the control design, so adaptivity of the compensator may play a critical role here.

Since neural networks have shown adaptivity, they can be a good candidate to be used as a compensator. Moreover, PID controllers have shown reliable performance. Therefore, this error-based control method of the PID can be used for neural network training. For these collective reasons, the design of a NN-based PID controller is proposed in this work. The next part discusses the general structure of neural networks.

### 1.1.1 Neural Networks

Neural networks are sets of nodes or neurons, connections, math operations, inputs, and outputs. From the moment a neural network receives inputs, math operations are applied until outputs are generated. Fig. 1.4 shows the overall structure of a general neural network.

While more complex structures of neural network are available, this thesis uses a feed-forward neural network. Neural networks have the ability to learn by changing network attributes. This way the neural network can generate desired outputs. The forementioned characteristics of adaptivity or learning feature make neural networks favorable in the control problems.

As shown in Fig. 1.4, a neural network is divided into a few layers. Each layer consists of some nodes or neurons, and all nodes of each layer is connected to all nodes of the next. The connection between the previous and the current layers is established by weights. Moreover, there are biases that are added to the neuron input. These weights along with biases define the input of the neuron. Next, the input of the neuron goes into a transfer (or activation) function that generates neuron output. This process continues until neural network outputs are generated. Among different learning processes available in neural networks, the supervised learning process is used in this thesis. Supervised learning changes network weights and biases in a way that neural network outputs converge to the desired values. There are several network training algorithms such as backpropagation and Levenberge-Marquardt. These algorithms update the network in each iteration based on a criterion (or function). This function is called an error function, and it determines how close the network outputs are to the desired values. Math functions, training algorithm, and details of the neural networks will be discussed in Chapter 3.

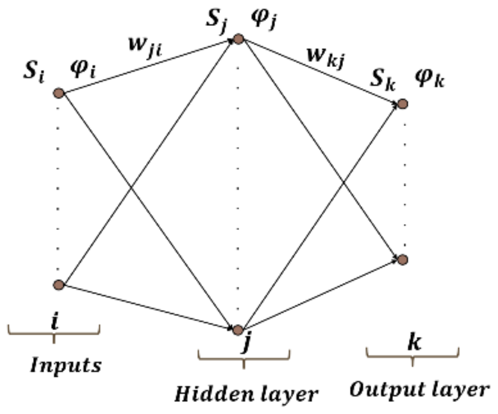

Figure 1.4: Neural network structure

## 1.2 Literature Review

There has been growing attention to the ability of neural networks in system control. Adaptivity of neural networks is a great strength in controller design. Neural network control design has been used in a wide range of applications from the robotic manipulators ([7] and [8]) to the car industry ([9]).

Cong and Liang in [10] proposed a PID-like neural network controller that has better performance compared to LQR. They applied their controller on single and double inverted pendulum, shown in Fig. 1.5, and compared the results with LQR. One disadvantage of this control approach is that the controller performance depends on the initial conditions. If the initial conditions are far from the target states, the controller may not be able to stabilize the system or show satisfactory performance. They provided stability analysis of the controller. The difference between this PID-like controller and the controller proposed in this work is the neural network. The latter has different neural network structure and it generates PID gains, whereas the former generates torque. The NN-based PID controller is designed to be used as a compensator working with other controllers in order to reduce the error.

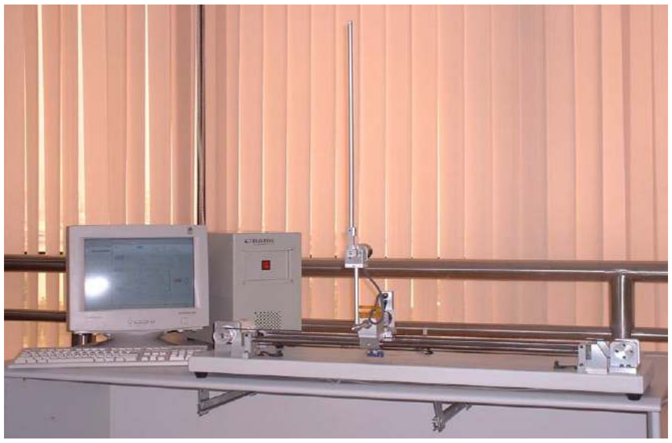

Figure 1.5: PID-like neural network controller, double inverted pendulum system [10]

Jung et al. in [11]-[13] designed a neural network controller based on reference compensation technique (RCT) that can stabilize a two-dimensional inverted pendulum, shown in Fig. 1.6. The neural network in RCT represents the inverse dynamics of the system, which is not the case in NN-based PID controller.

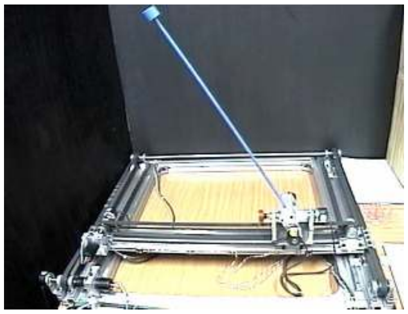

Figure 1.6: Reference compensation technique (RCT), two-dimensional inverted pendulum system [13]

A neural controller is investigated by Thanh and Ahn in [7] and [8]. They designed a neural network controller to control PAM manipulator, shown in Fig. 1.7. The superiority of the proposed controller over conventional PID is verified by experimental results. A backpropagation algorithm updates neural network online.

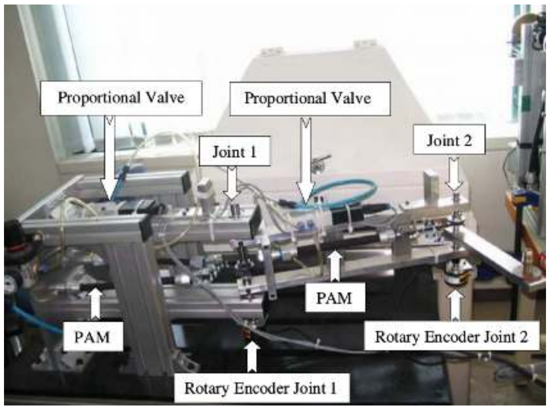

Figure 1.7: Neural network controller, PAM manipulator [7]

A novel robust neural network controller is proposed by Chen and Sheu in [14] for speed control of an induction motor. They designed a neural network estimator for the dynamic model of the system along with a neural network PI controller. Projection algorithm was used as the training algorithm. The control performance is verified by experimental and simulation results. This control scheme shows better performance in presence of the disturbance. Adaptivity of the neural network helps it to correct itself online. It's worth mentioning that although using a neural network to represent system dynamics is useful in neural network controller design, noisy data from state derivatives make it difficult to train the neural network for this purpose.

Ge et al. in [15] used an observer with high gains to estimate state derivatives. They used a state feedback neural network controller and discussed the control stability by Lyapunov function. This control method doesn't need to be trained offline, but neural network gains should have a good initial estimation of the system.

## 1.3 Objective

Nonlinear controller design is difficult in general. Applying dynamic approximations can be a solution, but it introduces errors in the control performance. In this work, the objective is to reduce this error using a NN-based PID compensator. The controller structure of the system consists of two parts using this method: *base controller* and *NN-based PID*. The base controller can be any classic linear or nonlinear controller that can perform stabilization and/or trajectory tracking. The error of the base controller, which may be induced by dynamic model simplifications, is compensated by using NN-based PID. In this work, approximate feedback linearization is used as the base controller, which has some steady state error as will be shown in Chapter 4. Next, NN-based PID compensator is added to the controller to reduce the error. The NN-based PID controller has two main parts: *neural network* and *PID*. The neural network part receives the state errors and generates PID gains as the outputs. Neural network is also updated online using backpropagation learning algorithm. Next, the PID controller receives the updated gains from the neural network along with the state errors and generates the compensatory control input. Finally, the output of the PID is added to the output of the base controller. In this structure, the NN-based PID controller constantly modifies control input of the base controller to overcome the error. This control approach is implemented on the ball-on-plate system and verified to show improvements in both stabilization and trajectory tracking. It's worth noting that the proposed method can be applied on any nonlinear system.

Neural network in this control approach needs offline training. Another limitation is that NN-based PID controller is activated only when approximate feedback linearization settling time is met. This means when there is steady state error from the base controller, neural network comes into the picture and reduces the error. In this work, NN-based PID controller activated one second after the base controller in both stabilization and trajectory tracking.

The rest of the thesis is organized as follows. Dynamic model of the ball-on-plate system is derived in Chapter 2, followed by Chapter 3, which describes the new control law. Chapter 4 presents experimental setup and results, and Chapter 5 ends the thesis with concluding remarks.

# CHAPTER 2

# DYNAMIC MODEL: BALL-ON-PLATE EXAMPLE

This chapter provides the ball-on-plate dynamic model, then it will be simplified to two independent ball-on-beam systems such that rotational motion of the ball-on-plate system will be controlled by the ball-on-beam system.

## 2.1 Ball-on-Plate Dynamics

Ball-on-plate system is a nonlinear, coupled system that will be modelled in this section. Fig.2.1 shows the 3D modeling of the system.

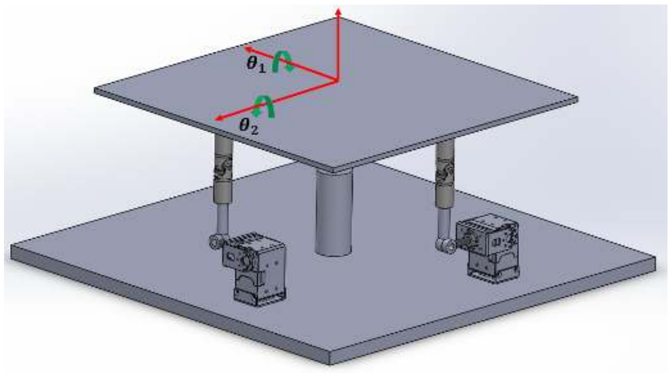

Figure 2.1: Ball-on-plate system

Lagrange equation is used to derive the equations of motion:

$$L = T - V \tag{2.1}$$

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}}) - \frac{\partial L}{\partial q} = Q \tag{2.2}$$

where $T$ and $V$ are kinetic and potential energies of the system, respectively. $L$ is called system Lagrangian; $t$, $q$, $\dot{q}$, and $Q$ are time, system generalized coordinates, time derivative of generalized coordinates, and generalized forces, respectively. Eq. (2.2) derives equations of motion in matrix form. Torques applied on the ball-on-plate system are given below:

$$Q = \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}^T \tag{2.3}$$

where $\tau_1$ and $\tau_2$ are torques applied about $x$ and $y$ axes, respectively. The ball-on-plate system has four generalized coordinates, explained in Table 2.1, that are given below:

$$q = \begin{bmatrix} \theta_1 & \theta_2 & x & y \end{bmatrix}^T \tag{2.4}$$

Table 2.1: Ball-on-plate system generalized coordinates

| | |
|---|---|
| $\theta_1$ | plate angle about $x$ axis (see Fig. 2.1 for the direction) |
| $\theta_2$ | plate angle about $y$ axis (see Fig. 2.1 for the direction) |
| $x$ | ball position along $x$ axis(local coordinate system) |
| $y$ | ball position along $y$ axis(local coordinate system) |

There is one local $oxyz$ coordinate frame which is attached to the plate, and one global $OXYZ$ frame as shown in Fig. 2.2. The system parameters are explained in Table 2.2. Plate moment of inertia with respect to the local $oxyz$ frame is

$$I_p^{oxyz} = \begin{bmatrix} I_{px} & 0 & 0 \\ 0 & I_{py} & 0 \\ 0 & 0 & I_{pz} \end{bmatrix} \tag{2.5}$$

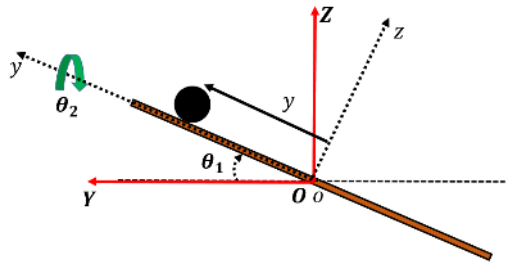

Figure 2.2: Local $oxyz$ and global $OXYZ$ coordinate systems

Table 2.2: Ball-on-plate system parameters

| | |
|---|---|
| $I_{px}$ | moment of inertia of the plate about $x$ axis |
| $I_{py}$ | moment of inertia of the plate about $y$ axis |
| $I_{pz}$ | moment of inertia of the plate about $z$ axis |
| $I_b$ | ball moment of inertia (all three axes have the same moment of inertia) |
| $m$ | mass of the ball |
| $r$ | radius of the ball |
| $g$ | gravity |

A ball is a symmetric object, so the ball moments of inertia in all three axes are the same. It will be shown that the ball moment of inertia with respect to the global $OXYZ$ frame is also the same as the moment of inertia with respect to the local $oxyz$ frame.

$$I_b^{oxyz} = \begin{bmatrix} I_b & 0 & 0 \\ 0 & I_b & 0 \\ 0 & 0 & I_b \end{bmatrix} \tag{2.6}$$

The plate is rotated by $\theta_1$ about $x$ axis, then it will be rotated by $\theta_2$ about $y$ axis with respect to the local $oxyz$ frame attached to the plate. The rotation matrix between the local $oxyz$ frame to the global $OXYZ$ frame is given below:

$$R = R_{oxyz}^{OXYZ} = R_{x,\theta_1} R_{y,\theta_2} = \begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) \\ \sin(\theta_1)\sin(\theta_2) & \cos(\theta_1) & -\cos(\theta_2)\sin(\theta_1) \\ -\cos(\theta_1)\sin(\theta_2) & \sin(\theta_1) & \cos(\theta_1)\cos(\theta_2) \end{bmatrix} \tag{2.7}$$

Given the ball position in the local $oxyz$ coordinate frame,

$$P_b^{oxyz} = \begin{bmatrix} x & y & r \end{bmatrix}^T \tag{2.8}$$

Using the rotation matrix in Eq. (2.7), the ball position with respect to the global $OXYZ$ coordinate frame is given by

$$P_b^{OXYZ} = R \times P_b^{oxyz} = \begin{bmatrix} \cos(\theta_2)x + \sin(\theta_2)r \\ \sin(\theta_1)\sin(\theta_2)x + \cos(\theta_1)y - \cos(\theta_2)\sin(\theta_1)r \\ -\cos(\theta_1)\sin(\theta_2)x + \sin(\theta_1)y + \cos(\theta_1)\cos(\theta_2)r \end{bmatrix}, \tag{2.9}$$

By taking time derivative from position in Eq. (2.9), the ball velocity can be obtained with respect to the global $OXYZ$ coordinate frame.

$$V_b^{OXYZ} = \begin{bmatrix} V_{bx}^{OXYZ} & V_{by}^{OXYZ} & V_{bz}^{OXYZ} \end{bmatrix}^T \tag{2.10}$$

$$V_{bx}^{OXYZ} = \frac{dP_{bx}^{OXYZ}}{dt} = -\sin(\theta_2)\dot{\theta}_2 x + \cos(\theta_2)\dot{x} + \cos(\theta_2)\dot{\theta}_2 r \tag{2.11}$$

$$V_{by}^{OXYZ} = \frac{dP_{by}^{OXYZ}}{dt} = \cos(\theta_1)\dot{\theta}_1 \sin(\theta_2)x + \sin(\theta_1)\cos(\theta_2)\dot{\theta}_2 x + \sin(\theta_1)\sin(\theta_2)\dot{x} - \sin(\theta_1)\dot{\theta}_1 y + \cos(\theta_1)\dot{y} + \sin(\theta_2)\dot{\theta}_2 \sin(\theta_1) r - \cos(\theta_2)\cos(\theta_1)\dot{\theta}_1 r \tag{2.12}$$

$$V_{bz}^{OXYZ} = \frac{dP_{bz}^{OXYZ}}{dt} = \sin(\theta_1)\dot{\theta}_1 \sin(\theta_2)x - \cos(\theta_1)\cos(\theta_2)\dot{\theta}_2 x - \cos(\theta_1)\sin(\theta_2)\dot{x} + \cos(\theta_1)\dot{\theta}_1 y + \sin(\theta_1)\dot{y} - \sin(\theta_1)\dot{\theta}_1 \cos(\theta_2) r - \cos(\theta_1)\sin(\theta_2)\dot{\theta}_2 r \tag{2.13}$$

Equation $S(\omega_p) = \dot{R}R^{-1}$ gives angular velocity of the plate in the form of a skew symmetric matrix. Hence:

$$S(\omega_p^{OXYZ}) = \begin{bmatrix} 0 & -\dot{\theta}_2 \sin(\theta_1) & \dot{\theta}_2 \cos(\theta_1) \\ \dot{\theta}_2 \sin(\theta_1) & 0 & -\dot{\theta}_1 \\ -\dot{\theta}_2 \cos(\theta_1) & \dot{\theta}_1 & 0 \end{bmatrix} \tag{2.14}$$

Now, angular velocity of the plate with respect to the global $OXYZ$ coordinate frame is

$$\omega_p^{OXYZ} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 \cos(\theta_1) & \dot{\theta}_2 \sin(\theta_1) \end{bmatrix}^T \tag{2.15}$$

Assuming no slip condition between the ball and the plate, the angular velocity of the ball with respect to the local $oxyz$ coordinate frame is given by

$$\omega_b^{oxyz} = \begin{bmatrix} -\frac{\dot{y}}{r} & \frac{\dot{x}}{r} & 0 \end{bmatrix}^T \tag{2.16}$$

The ball is rolling on the plate, and the local frame is attached to the plate. Consequently the angular velocity of the ball with respect to the global $OXYZ$ coordinate frame is given by

$$\omega_b^{OXYZ} = R\omega_b^{oxyz} + \omega_p^{OXYZ} = \begin{bmatrix} -\frac{\cos(\theta_2)\dot{y}}{r} + \dot{\theta}_1 \\ -\frac{\sin(\theta_1)\sin(\theta_2)\dot{y}}{r} + \frac{\cos\theta_1 \dot{x}}{r} + \dot{\theta}_2\cos(\theta_1) \\ \frac{\cos(\theta_1)\sin(\theta_2)\dot{y}}{r} + \frac{\sin(\theta_1)\dot{x}}{r}\dot{\theta}_2\sin(\theta_1) \end{bmatrix} \tag{2.17}$$

The superscript $OXYZ$ will be dropped in the rest of the thesis because it is assumed that all positions and velocities are defined with respect to the global $OXYZ$ coordinate frame. If a variable is defined with respect to the local frame, it will be specified. Now, kinetic and potential energy of the ball and plate can be derived as below:

$$T_p = \frac{1}{2}\omega_p^T R I_p^{oxyz} R^T \omega_p \tag{2.18}$$

$$V_p = 0 \tag{2.19}$$

$$T_b = \frac{1}{2}\omega_b^T R I_b^{oxyz} R^T \omega_b + \frac{1}{2}V_b^T m_b V_b \tag{2.20}$$

$$V_b = m_b g P_{bz}^2 \tag{2.21}$$

$$T_{sys} = T_p + T_b \tag{2.22}$$

$$V_{sys} = V_p + V_b \tag{2.23}$$

where subscript $p$, $b$, and $sys$ refer to the plate, the ball, and the whole system, respectively. It is important to note that the local $oxyz$ frame is rotating with the plate, so the plate moment of inertia must be measured with respect to the global $OXYZ$ coordinate frame. This can be done by using the relation $I^{OXYZ} = RI^{oxyz}R^T$ as shown in Eqs. (2.18) and (2.20). This relation does not have effect on the ball because the ball is a symmetric object, so applying the relation will result to the $I_b^{oxyz}$. Substituting Eqs. (2.22) and (2.23) into Eqs. (2.1)-(2.2) gives equations of motion of the system:

$$M\ddot{q} + C\dot{q} + G = Q \tag{2.24}$$

where $M$, $C$, $G$, and $Q$ are the matrices of inertia, centrifugal and coriolis forces, potential, and external forces, respectively. These matrices are given in Appendix A.

Now, ignoring coupling effects in the equations of motion results into two independent ball-on-beam systems (Ho et al. [6]):

$$\ddot{\theta}_1 = \frac{1}{I_b + I_p + my^2}\left[\tau_1 + mgr\sin(\theta_1) - mgy\cos(\theta_1) - 2my\dot{y}\dot{\theta}_1\right] \tag{2.25}$$

$$\ddot{\theta}_2 = \frac{1}{I_b + I_p + mx^2}\left[\tau_2 + mgr\sin(\theta_2) + mgx\cos(\theta_2) - 2mx\dot{x}\dot{\theta}_2\right] \tag{2.26}$$

$$\ddot{x} = \frac{1}{m + \frac{I_b}{r^2}}\left[mx\dot{\theta}_2^2 + mg\sin(\theta_2)\right] \tag{2.27}$$

$$\ddot{y} = \frac{1}{m + \frac{I_b}{r^2}}\left[my\dot{\theta}_1^2 - mg\sin(\theta_1)\right] \tag{2.28}$$

The simlified equations will be used in the next chapter to design an approximate feedback linearization with NN-based PID controller.

# CHAPTER 3

# CONTROLLER DESIGN

This chapter discusses the controller design. A new compensator is proposed for improvements in both stabilization and trajectory tracking. Although this controller is applied on the ball-on-plate system, it can be applied on any nonlinear system. Another advantage of this controller is its simplicity. Since the decoupled ball-on-plate dynamics is used in this work, representing the controller design procedure for one axis is sufficient. This chapter considers controller design about $x$ axis only; $y$ axis controller is given in Appendix B. Fig. 3.1 shows the decoupled controller design for the ball-on-plate system. The next section describes the overal structure of the controller in more detail.

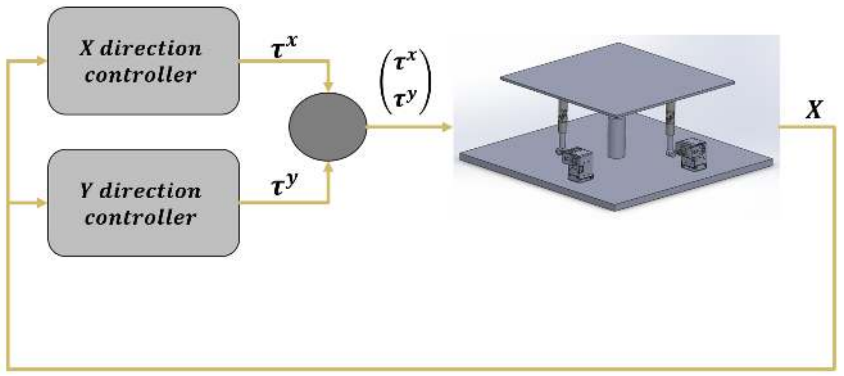

Figure 3.1: Decoupled controller design for the ball-on-plate system

## 3.1 Overal Structure of the Controller

The new controller consists of two main parts: *base controller* and *NN-based PID*. Fig. 3.2 shows the overal structure of the controller applied on the ball-on-plate system. The base controller can be any linear or nonlinear controller. It is better to choose the simplest one because NN-based PID compensator can reduce the control errors. In this work, an approximate nonlinear controller has been used as the base controller to see the effect of NN-based PID controller. This approximate nonlinear controller is approximate input-output feedback linearization. Since the ball-on-plate system dynamic model is simplified, it is called approximate. Detailed information on the base controller is given in the next section.

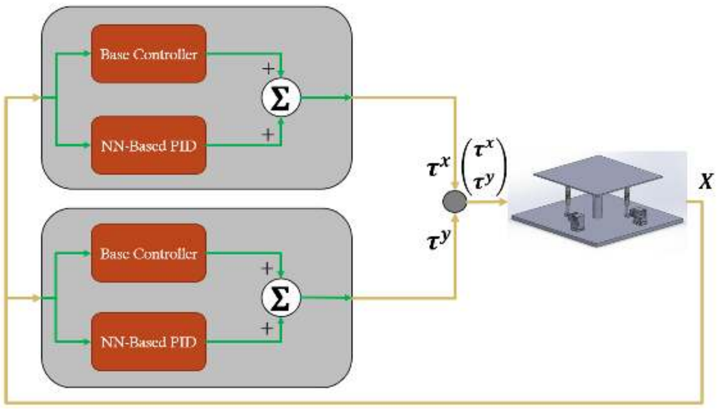

Figure 3.2: Overal structure of the controller applied on the ball-on-plate system

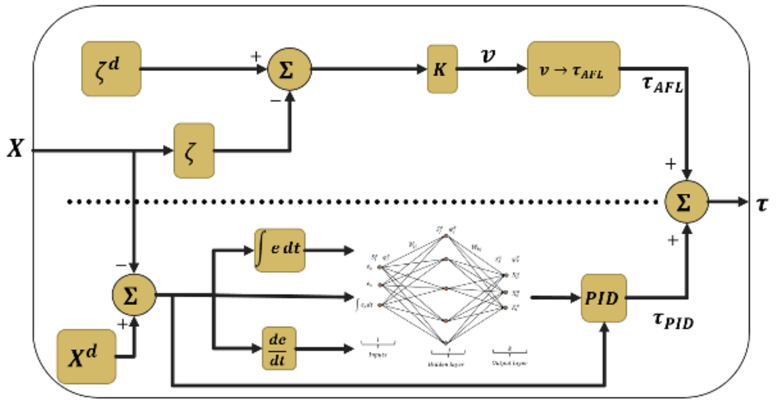

Figure 3.3: Inside of the controller block diagram ($x$ axis)

The second part of the controller is called NN-based PID controller. This controller contains two major parts: a neural network and a PID controller. The neural network part gets state error as inputs and generates PID gains. Next step, PID gets the updated gains along with the state errors to generate compensatory control input. Finally, the output of the NN-based PID controller is added to the output of the base controller. Fig 3.3 shows the details of the controller.

The rest of the chapter consists of two sections. First one describes feedback linearization and its necessary conditions, which are derived from [1], [2], and [6] mostly, and the second part is dedicated to the NN-based PID.

## 3.2 Feedback Linearization

Feedback linearization is a control approach that transforms the original space of the system to a new one. This method uses a systematic approach to find a specific change of coordinates for the transformation. As a result, a linear system which is equivalent to the original system is obtained in the new state space. States of the new system are functions of the original system states. This approach enables us to apply linear controllers on the transformed system.

There are two points that need to be made. The process of transforming a nonlinear system to a linear form is a systematic approach through a specific method. The second point is that not all systems can be transformed to a linear form. Only a class of systems that satisfy feedback linearization conditions can be transformed. If system dynamic model satisfies the conditions, the system is called feedback linearizable. However, since the decoupled ball-on-plate system does not satisfy the conditions, the ball-on-beam system dynamic model undergoes further simplification to make it feedback linearizable. The controller designed for this nonlinear simplified model is called approximate feedback linearization.

This section discusses feedback linearization and sufficient conditions on how to apply it on the simplified ball-on-beam system.

### 3.2.1 Feedback Linearization Conditions and Procedure

A nonlinear system with states of $X$ and input $u$:

$$\dot{X} = f(X) + g(X)u \tag{3.1}$$

is feedback linearizable if it satisfies the following conditions:

- The matrix has rank of $n$.

$$G(X) = \left[g(X), ad_f g(X), ad_f^2 g(X), ..., ad_f^{n-1} g(X)\right] \tag{3.2}$$

- The distribution is involutive.

$$\Delta = span \left[g(X), ad_f g(X), ad_f^2 g(X), ..., ad_f^{n-2} g(X)\right] \tag{3.3}$$

where $n$ is the number of states of the original system, and Lie bracket is defined as below:

$$ad_f g(X) = [f, g](X) = \frac{\partial g(X)}{\partial X} f(X) - \frac{\partial f(X)}{\partial X} g(X) \tag{3.4}$$

and in a generalized form:

$$ad_f^k g(X) = [f, ad_f^{k-1} g(X)](X) \tag{3.5}$$

If a system satisfies the conditions (3.2) and (3.3), it is said to be feedback linearizable. Next step is transforming the system to a linear form. Given the system output $h(X)$, the derivatives of the output give the transformed states. Derivation continues until input of the original system appears.

$$\zeta_1 = h(X) \tag{3.6}$$

$$\zeta_2 = \frac{d\zeta_1}{dt} \tag{3.7}$$

$$\cdot$$

$$\cdot$$

$$\cdot$$

$$\zeta_n = \frac{d\zeta_{n-1}}{dt} \tag{3.8}$$

where $[\zeta_1, \zeta_2, ..., \zeta_n]^T$ are the transformed states. The input of the transformed system is as follows:

$$v = \dot{\zeta}_n \tag{3.9}$$

$$v = \alpha + \beta u \tag{3.10}$$

where $\alpha$ and $\beta$ are defined as below:

$$\alpha = L_f^n h(x) \tag{3.11}$$

$$\beta = L_g L_f^{n-1} h(x) \tag{3.12}$$

Eq. (3.10) is important since it relates transformed system input, $v$, to the original system input, $u$. Finally, the system representation in the transformed space is written as

$$\dot{\zeta} = A\zeta + Bv \tag{3.13}$$

where

$$\zeta = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ . \\ . \\ . \\ \zeta_n \end{bmatrix}, A = \begin{bmatrix} 0, 1, 0, ..., 0 \\ 0, 0, 1, ..., 0 \\ . \\ . \\ . \\ 0, 0, 0, ..., 1 \\ 0, 0, 0, ..., 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ . \\ . \\ . \\ 1 \end{bmatrix} \tag{3.14}$$

Since the transformed system in Eq. (3.13) is now linear, one can take advantage of the linear control theory. Applying the pole placement method determines the controller gains:

$$P = \begin{bmatrix} p_1, p_2, ..., p_n \end{bmatrix} \tag{3.15}$$

$$K = Pole\_Placement(P) \tag{3.16}$$

$$v = -K\zeta \tag{3.17}$$

$$\dot{\zeta} = A\zeta - BK\zeta \tag{3.18}$$

where $P$, $K$, and $\zeta$ are the locations of the controller poles, the controller gains, and the transformed states, respectively. In the next part, the dynamic model of the ball-on-beam system is considered, and the application of feedback linearization is investigated.

### 3.2.2 Simplified Ball-on-Beam System

In Chapter 2, it was shown that the equations of motion of the ball-on-plate system can be simplified into two independent ball-on-beam systems. Hauser et al. in [3] showed that the ball-on-beam system does not satisfy the feedback linearization conditions, so they simplified the system dynamics to make it feedback linearizable. Ho et al. in [6] applied approximate feedback linearization on the ball-on-beam system. The same model simplification as in [6] is utilized here.

The ball-on-beam system has four states as below:

$$X = \begin{bmatrix} \theta_2 & \dot{\theta}_2 & x & \dot{x} \end{bmatrix}^T \tag{3.19}$$

From Fig. 2.2, it is clear that angular position of the plate, $\theta_2$, changes ball position along $x$ axis and vice versa. The ball-on-beam dynamics in Eqs. (2.26) and (2.27) can be rewritten as an input-affine form as in Eq. (3.1).

$$\dot{X} = \begin{bmatrix} \dot{\theta}_2 & \ddot{\theta}_2 & \dot{x} & \ddot{x} \end{bmatrix}^T \tag{3.20}$$

$$f(X) = \begin{bmatrix} \dot{\theta}_2 \\ \dfrac{1}{I_b + I_p + mx^2}\left(mgr\sin(\theta_2) + mgx\cos(\theta_2) - 2mx\dot{x}\dot{\theta}_2\right) \\ \dot{x} \\ \dfrac{1}{m + \dfrac{I_b}{r^2}}\left(mx\dot{\theta}_2^2 + mg\sin(\theta_2)\right) \end{bmatrix} \tag{3.21}$$

$$g(x) = \begin{bmatrix} 0 & \dfrac{1}{I_b + I_p + mx^2} & 0 & 0 \end{bmatrix}^T \tag{3.22}$$

$$u = \tau_2 \tag{3.23}$$

By using the position of the ball as the output:

$$h(X) = x \tag{3.24}$$

The system's transformed states can be derived as below:

$$\zeta_1 = h(X) = x \tag{3.25}$$
$$\zeta_2 = \frac{d\zeta_1}{dt} = \dot{x} \tag{3.26}$$
$$\zeta_3 = \frac{d\zeta_2}{dt} = \ddot{x} = \frac{1}{m + \frac{I_b}{r^2}}\left(mx\dot{\theta}_2^2 + mg\sin(\theta_2)\right) \tag{3.27}$$
$$\zeta_4 = \frac{d\zeta_3}{dt} = \frac{1}{m + \frac{I_b}{r^2}}\left(m\dot{x}\dot{\theta}_2^2 + mg\dot{\theta}_2\cos(\theta_2)\right) \tag{3.28}$$
$$\zeta = \begin{bmatrix}\zeta_1 & \zeta_2 & \zeta_3 & \zeta_4\end{bmatrix}^T \tag{3.29}$$

where the term $2mx\dot{\theta}_2\ddot{\theta}_2$ is ignored from the $\zeta_4$ in order to make system feedback linearizable. By using Eqs. (3.9) and (3.10), the transformed control input is

$$\dot{\zeta}_4 = \alpha + \beta\tau_2 \tag{3.30}$$

where

$$\alpha = A_x^2 m\dot{\theta}_2^2\left(mx\dot{\theta}_2^2 + mg\sin(\theta_2)\right) - A_x mg\dot{\theta}_2^2\sin(\theta_2) + \tag{3.31}$$
$$A_x B_x\left(2m\dot{x}\dot{\theta}_2 + mg\cos(\theta_2)\right)\left(-2mx\dot{x}\dot{\theta}_2 + mgx\cos(\theta_2) + mgr\sin(\theta_2)\right) \tag{3.32}$$
$$\beta = A_x B_x\left(2m\dot{x}\dot{\theta}_2 + mg\cos(\theta_2)\right) \tag{3.33}$$
$$A_x = \frac{1}{m + \frac{I_b}{r^2}} \tag{3.34}$$

$$B_x = \frac{1}{mx^2 + I_p + I_b} \tag{3.35}$$

Since the transformed system is now linear, the pole placement method can be used to determine the controller gains. The same procedure is applied for the $y$ axis. The details are given in Appendix B.

## 3.3 Neural Networks

Neural networks have been capturing more attention in recent years. Adaptivity and often simplicity in structure are the main reasons for their popularity in many control problems.

In this study a PID compensator with adjustable gains is designed using neural network. It is designed to compensate for the errors caused by the approximate feedback linearization. This controller is applied on the ball-on-plate system, but it can be used on any nonlinear system. Another positive point of this controller is that it takes advantage of having a simple design. It is applied to the ball-on-beam system which has simpler dynamics than the ball-on-plate system.

This section is dedicated to the neural network part of the controller. First part describes the generals of the neural networks on how they work, and the second part goes through making PID controller adjustable.

### 3.3.1 Neural Network Structure

Neural networks have inputs and outputs. Mathematical operations are carried out on the inputs, and then outputs are generated in the last stage.

A neural network with two layers is shown in Fig. 3.4. There is one hidden layer and one output layer. Throughout this work, inputs are not considered as a separate layer, and the input, hidden, and output neurons are defined by $i$, $j$, and $k$ subscripts, respectively. Each layer is connected to the next using network weights. In addition, there are biases that are added to the neuron input. Finally, the neuron input goes through a function called transfer function (or activation function) and produces neuron output. Fig. 3.5 shows a neuron.

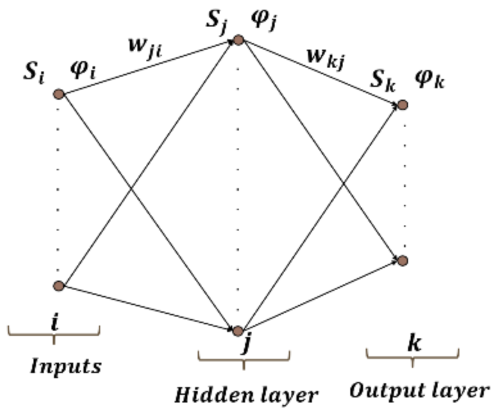

Figure 3.4: Neural network structure

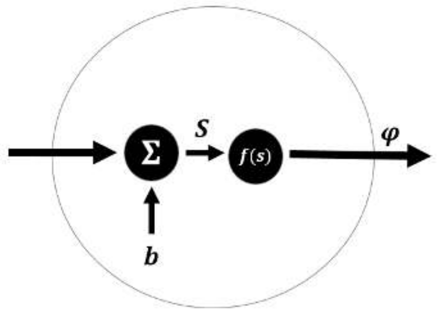

Figure 3.5: A neuron in a neural network

If a neural network input matrix is defined by $S_i$, neural outputs, $\phi_k$, can be obtained through the following equations:

$$\phi_i = S_i \tag{3.36}$$

$$S_j = w_{ji}\phi_i + b_j \tag{3.37}$$

$$\phi_j = f_h(S_j) \tag{3.38}$$

$$S_k = w_{kj}\phi_j + b_k \tag{3.39}$$

$$\phi_k = f_o(S_k) \tag{3.40}$$

where $f_h(\cdot)$ and $f_o(\cdot)$ define transfer functions of the hidden and output layers, respectively. Variables denoted by $\phi$ represent neuron output, and $w_{ji}$ and $w_{kj}$ are the weights connecting the inputs to the hidden layer and the hidden layer to the output layer, respectively; $b_j$ and $b_k$ are biases of the hidden and output layers, respectively.

Neural network adaptivity helps generate desired outputs. There are numerous methods to train a neural network. When desired outputs are known, one can use supervised learning method. This learning is based on a reference function or error function, $E$, that shows how far network outputs are from the desired values. One of the most common neural network training methods is gradient descent. This method is described by the following equations:

$$dw_{kj}^n = \frac{\partial E}{\partial w_{kj}^n} \tag{3.41}$$

$$w_{kj}^{n+1} = w_{kj}^n - \eta dw_{kj}^n + \alpha dw_{kj}^{n-1} \tag{3.42}$$

$$db_k^n = \frac{\partial E}{\partial b_k^n} \tag{3.43}$$

$$b_k^{n+1} = b_k^n - \eta db_k^n + \alpha db_k^{n-1} \tag{3.44}$$

$$dw_{ji}^n = \frac{\partial E}{\partial w_{ji}^n} \tag{3.45}$$

$$w_{ji}^{n+1} = w_{ji}^{n} - \eta dw_{ji}^{n} + \alpha dw_{ji}^{n-1} \tag{3.46}$$

$$db_j^n = \frac{\partial E}{\partial b_j^n} \tag{3.47}$$

$$b_j^{n+1} = b_j^n - \eta db_j^n + \alpha db_j^{n-1} \tag{3.48}$$

where the superscript $n$ denotes the iteration number. The learning rate, $\eta$, and the momentum rate, $\alpha$, are typically determined through experiments. Learning rate specifies the amount of correction jump in each iteration, i.e., speed of convergence, and the momentum rate is used to avoid local minima.

### 3.3.2 NN-Based PID Controller

The combination of PID and the neural network is used as the compensator. It's worth mentioning that PID controller is designed before adding the neural network, and it is used for the neural network offline training. Once PID gains are determined through experiment, those gains can be used as neural network desired outputs. Finally, this neural network is included in the controller to update the PID gains; see Fig. 3.3:

$$\tau_{NN} = K_p(x^d - x) + K_d(\dot{x}^d - \dot{x}) + K_i\left[\int_0^t (x^d - x)dt\right] \tag{3.49}$$

Neural network online training enables the PID controller to update itself. In other words, the PID adapts itself to the different situations of system states. The structure of the neural network used in the controller is shown in Fig. 3.6. Position error, velocity error, and integral of the position error are the inputs to the neural network, and PID gains are the outputs. Neural network consists of two layers in which all neurons of one layer are connected to the all neurons of the next. A tangent sigmoid and linear transfer functions are used for the hidden and the output layers, respectively. Figs. 3.7 and 3.8 show the functions.

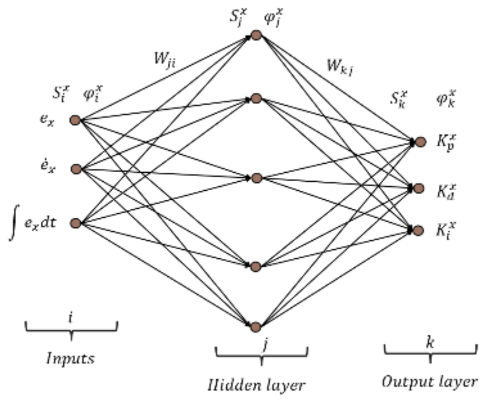

Figure 3.6: Neural network structure (along $x$ axis)

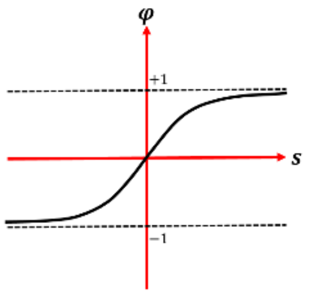

Figure 3.7: Tangent sigmoid activation function

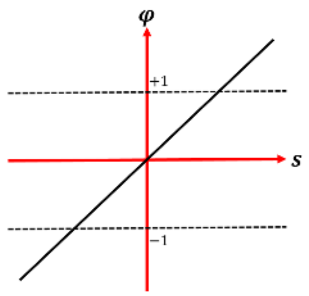

Figure 3.8: Linear activation function

The neural network has three inputs, five neurons in the hidden layer, and three neurons in the output layer. Inputs of the neural network are

$$S_1 = x^d - x \tag{3.50}$$

$$S_2 = \dot{x}^d - \dot{x} \tag{3.51}$$

$$S_3 = \int_0^t (x^d - x)dt \tag{3.52}$$

The transfer functions shown in Figs. 3.7 and 3.8 are defined by

$$f_{tansig}(s) = \frac{2}{1 + e^{-2s}} - 1 \tag{3.53}$$

$$f_{lin}(s) = s \tag{3.54}$$

The outputs of the neural network are

$$\phi_1 = K_p \tag{3.55}$$

$$\phi_2 = K_d \tag{3.56}$$

$$\phi_3 = K_i \tag{3.57}$$

As shown in Fig. 3.3,

$$\tau = \tau_{AFL} + \tau_{NN} \tag{3.58}$$

The error function is defined by

$$E = \frac{1}{2}(x^d - x)^2 \tag{3.59}$$

Finally, the gradient descent method [16] is used to update neural network outputs.

$$\frac{\partial E}{\partial w_{kj}} = (\frac{\partial E}{\partial x})(\frac{\partial x}{\partial \tau})(\frac{\partial \tau}{\partial \tau_{NN}})(\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial w_{kj}}) \tag{3.60}$$

$$\frac{\partial E}{\partial b_k} = (\frac{\partial E}{\partial x})(\frac{\partial x}{\partial \tau})(\frac{\partial \tau}{\partial \tau_{NN}})(\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial b_k}) \tag{3.61}$$

$$\frac{\partial E}{\partial w_{ji}} = (\frac{\partial E}{\partial x})(\frac{\partial x}{\partial \tau})(\frac{\partial \tau}{\partial \tau_{NN}}) \left[ \sum_{k=1}^{3} (\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial \phi_j}) \right] (\frac{\partial \phi_j}{\partial S_j})(\frac{\partial S_j}{\partial w_{ji}}) \tag{3.62}$$

$$\frac{\partial E}{\partial b_j} = (\frac{\partial E}{\partial x})(\frac{\partial x}{\partial \tau})(\frac{\partial \tau}{\partial \tau_{NN}}) \left[ \sum_{k=1}^{3} (\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial \phi_j}) \right] (\frac{\partial \phi_j}{\partial S_j})(\frac{\partial S_j}{\partial b_j}) \tag{3.63}$$

From Eq. (3.59),

$$\frac{\partial E}{\partial x} = -(x^d - x) \tag{3.64}$$

Due to the nature of the gradient descent method, only the direction of the term $\frac{\partial x}{\partial \tau}$ is important. Using finite difference method, one can simplify this to

$$\frac{\partial x}{\partial \tau} = sign(\frac{\Delta x}{\Delta \tau}) = sign(\frac{x - x_{prev}}{\tau - \tau_{prev}}) \tag{3.65}$$

where $x_{prev}$ and $\tau_{prev}$ are the state and torque values in the previous iteration; $sign(.)$ is the sign function as shown in Fig. 3.9. Using the sign function of the term $\frac{\partial x}{\partial \tau}$ is important because the exact value of the finite difference fraction is noisy. Consequently, it may create large updates at some iterations, which is undesirable.

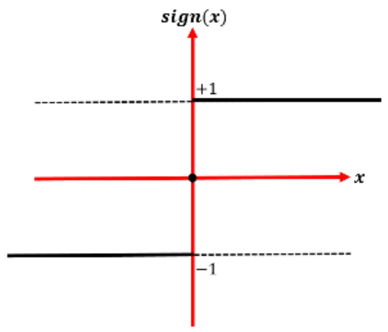

Figure 3.9: Sign function

From Eq. (3.58),

$$\frac{\partial \tau}{\partial \tau_{NN}} = 1 \tag{3.66}$$

Since neural network outputs are PID controller gains,

$$\frac{\partial \tau_{NN}}{\partial \phi_1} = x^d - x \tag{3.67}$$

$$\frac{\partial \tau_{NN}}{\partial \phi_2} = \dot{x}^d - \dot{x} \tag{3.68}$$

$$\frac{\partial \tau_{NN}}{\partial \phi_3} = \int_0^t (x^d - x)dt \tag{3.69}$$

From the neural network structure in Eqs. (3.37) and (3.39),

$$\frac{\partial S_k}{\partial w_{kj}} = \phi_j \tag{3.70}$$

$$\frac{\partial S_k}{\partial b_k} = 1 \tag{3.71}$$

$$\frac{\partial S_k}{\partial \phi_j} = w_{kj} \tag{3.72}$$

$$\frac{\partial S_j}{\partial w_{ji}} = \phi_i \tag{3.73}$$

$$\frac{\partial S_j}{\partial b_j} = 1 \tag{3.74}$$

Also, from transfer functions in Eqs. (3.53) and (3.54),

$$\frac{\partial \phi_k}{\partial S_k} = 1 \tag{3.75}$$

$$\frac{\partial \phi_j}{\partial S_j} = 1 - \phi_j^2 \tag{3.76}$$

Finally, following Eqs. (3.41)-(3.48) will update the weight and biases of the neural network. The learning and the momentum rates are determind empirically in the next chapter. The same procedure is applied to the $y$ axis ball-on-beam system.

# CHAPTER 4
# EXPERIMENTAL SETUP AND RESULTS

This chapter presents the experimental setup and results. It first describes the mechanical setup and the design, followed by the simulation and experimental results.

## 4.1 Experimental Setup

The experimental setup consists of three main components: *mechanical system*, *vision system*, and *controller system*. Fig. 4.1 shows the experimental setup. Appendix C gives detailed information on the part list and mechanical drawings.

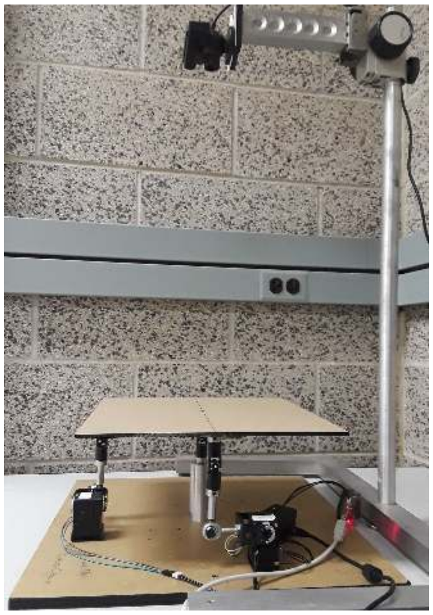

Figure 4.1: Experimental setup

### 4.1.1 Mechanical Design

As shown in Fig. 4.2, a setup is designed such that a ball is free to roll on the plate and the plate is controlled by two motors through two linkage mechanisms. Each mechanism includes a universal joint that enables the plate to rotate in both directions at the same time. There is also a third universal joint which is attached to the center rod and the plate.

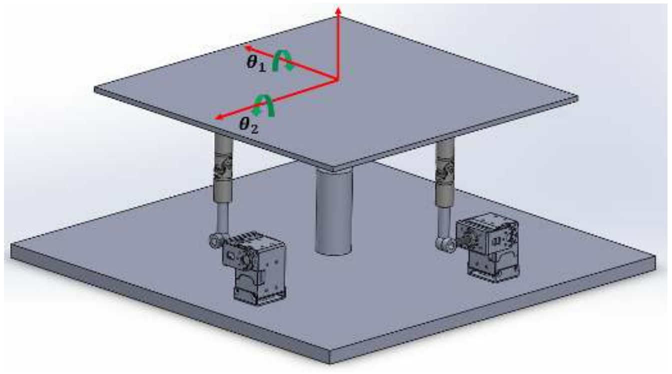

Figure 4.2: Ball-on-plate system designed in SolidWorks

Dynamixel servo motors (Fig. 4.3), are used as actuators and attached to the end of each linkage. The bottom side of the motors are mounted on the base plate using bolts and nuts.

Figure 4.3: Dynamixel motor[1]

[1]Source: `www.trossenrobotics.com/dynamixel-xm430-w210-t.aspx`

Fig. 4.4 shows the linkage mechanism.

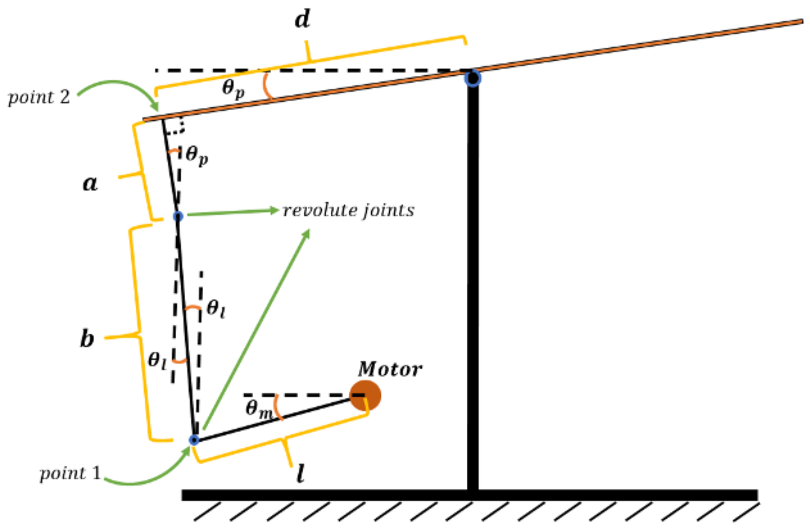

Figure 4.4: Linkage mechanism for the plate rotation

Initially, when the motor is not rotated, the vertical distance between motor and point 2 is

$$h_2^1 = a + b \tag{4.1}$$

where subscript 2 denotes point 2, and superscript 1 denotes the initial position of this point. Next, the motor is rotated by the amount of $\theta_m$. The vertical distance between the point 2 to the motor is given by

$$h_2^2 = a\cos(\theta_p) + b\cos(\theta_l) - l\sin(\theta_m) \tag{4.2}$$

Now, vertical displacement of the point 2 from the initial position to the final configuration is calculated by

$$\Delta h_2 = h_2^1 - h_2^2 = a + b - a\cos(\theta_p) - b\cos(\theta_l) + l\sin(\theta_m) \tag{4.3}$$

Assuming the angles $\theta_p$ and $\theta_l$ are small,

$$\cos(\theta_l) \simeq 1 \tag{4.4}$$

$$\cos(\theta_p) \simeq 1 \tag{4.5}$$

Now, Eq. (4.3) can be simplified to

$$\Delta h_2 = l\sin(\theta_m) \tag{4.6}$$

Eq. (4.6) shows that the amount of the vertical displacement of point 1 in Fig. 4.4 is equal to the amount of the vertical displacement of point 2. From Fig. 4.4,

$$\Delta h_2 = d\sin(\theta_p) \tag{4.7}$$

Using Eqs. (4.6) and (4.7),

$$l\sin\theta_m = d\sin\theta_p \tag{4.8}$$

If plate angle is small enough, then $\sin\theta_p \simeq \theta_p$; therefore,

$$\theta_p = \frac{l}{d}\sin\theta_m \tag{4.9}$$

Table 4.1 shows the system parameters and their values for the actual system.

Table 4.1: Parameters and values for the ball-on-plate system

| Parameters | Description | Values |
|---|---|---|
| $I_b$ | moment of inertia of the ball | $4.077\times10^{-3}$(kg.m$^2$) |
| $I_{px}$ | moment of inertia of the plate about $x$ axis | $4.315\times10^{-6}$(kg.m$^2$) |
| $I_{py}$ | moment of inertia of the plate about $y$ axis | $4.315\times10^{-6}$(kg.m$^2$) |
| $I_{pz}$ | moment of inertia of the plate about $z$ axis | $8.630\times10^{-6}$(kg.m$^2$) |
| $r$ | radius of the ball | 0.0127(m) |
| $m$ | mass of the ball | 0.067(kg) |
| $g$ | gravity | 9.81(m/s$^2$) |
| $l$ | length of the motor link (4.4) | 0.056(m) |
| $a$ | plate width (square plate) | 0.140(m) |

### 4.1.2 Vision System

A Sony SLEH-00448 USB camera is used to track the ball on the plate, Fig. 4.5.

Figure 4.5: Camera used in the experiment

This camera runs at the frame rate of 120 Hz at 320 × 240 pixels resolution. OpenCV library is used for the image processing. The camera is mounted at a height of 18.875in from the plate. It is important to note that the camera detects the ball position with respect to the fixed global coordinate frame, but the ball position with respect to the local coordinate frame, which is attached to the plate, is needed for the controller. It means that the rotational matrix in Eq. (2.7) is needed to transform the global ball coordinates to the local coordinates.

$$P_b^{oxyz} = R^{-1} P_b^{OXYZ} \tag{4.10}$$

Next, the ball velocity with respect to the local coordinate frame can be obtained by using numerical differentiation and a low pass filter:

$$V_{bx}^{oxyz} = \frac{\Delta P_{bx}^{oxyz}}{\Delta t} \tag{4.11}$$

$$V_{bx}^{oxyz} = \lambda V_{bx}^{oxyz} + (1-\lambda) V_{bx,prev}^{oxyz} \tag{4.12}$$

where subscript *prev* represents the previous value of the velocity. The same procedure is applied on the $y$ and $z$ axes to obtain $V_{by}^{oxyz}$ and $V_{bz}^{oxyz}$, respectively. The low pass filter value used in this work is $\lambda = 0.5$, and it is determined empirically.

A second candidate for the ball position detection is using a touchpad. Touchpads are easy to use and detect the ball position based on the amount of the voltage. However, regular touchpads are noisy and are not as accurate as the vision system. The high-quality touchpads are expensive, so it would not be economical to use them in the ball-on-plate system.

### 4.1.3 Controller Board

A 2.1 GHz laptop with Linux 16.04 LTS is used as the controller board in this thesis. Dynamixel motors can be run at 330 Hz using low latency timer in Linux. However, the vision system is running at 120 Hz, so it slows down the entire closed loop frequency of the system. Moreover, the controller math operations are performing while running the program, and some of the controller operations take more time, like neural network (NN hereafter) training. As a result, the closed loop frequency of the system including all system components goes down to 100 Hz.

Dynamixel motors in this thesis are designed to use TTL communication. Unlike serial communication, TTL communication uses only one wire for sending and receiving data. Dynamixel motors are connected together in series, and the first motor is connected to the Dynamixel2USB adapter. The Dynamixel2USB adapter is an interface that provides the connection between TTL communication and USB. This adapter is shown in Fig. 4.6. Finally, Dynamixel2USB adapter is connected to laptop USB port for sending and receiving commands to and from the motors.

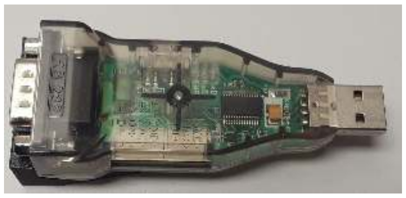

Figure 4.6: Dynamixel2USB adapter

Dynamixel servo motors are controlled through the Dynamixel library, Dynamixel SDK 3.5.4, which is provided by the manufacturer. The library is downloadable online and has a useful forum for troubleshooting. Since both the Dynamixel SDK and OpenCV library support C$^{++}$ language, the entire program is written in this language and run in a laptop.

The X series of Dynamixel motors have current control mode that can be used for torque input by using a graph that relates torque to the current. The torque-current relation in Fig. 4.7 can be fitted to the linear equation given below:

$$\tau = 1.3i - 0.32 \tag{4.13}$$

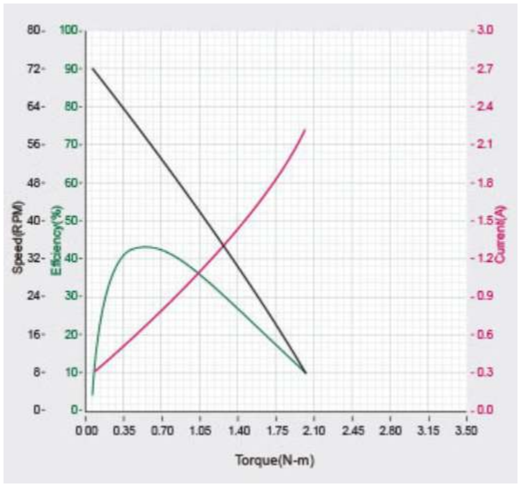

Figure 4.7: Torque-current graph of Dynamixel motors[2]

There are some other options for using a small size controller board to make the system portable. Arduino board is easy to program, but it is hard to find a camera with 120 Hz frequency compatible with Arduino. Raspberry PI is also easy to use and can handle both OpenCV library and Dynamixel SDK, but it is not convenient to install OpenCV library on it. It seems that fast processing boards like Minnowboard and UP board would also be good options. Minnowboard and UP board are good candidates to be used in the system, and they don't have the drawbacks mentioned for Arduino and Raspberry PI. One drawback of these fast processing boards may be that they are a bit costly. For example, UP board is almost \$170, which is more expensive than a Raspberry PI board (\$35). A full version of Linux

[2]Source: `http://emanual.robotis.com/docs/en/dxl/x/xm430-w210/`

16.04 along with Open CV library can easily be installed on UP board and Minnowboard, and data processing is fast enough.

## 4.2 Simulation and Experimental Results

This section presents simulation and experimental results on both stabilization and trajectory tracking.

The desired pole locations need to be determined for the approximate feedback linearization (AFL hereafter) designed in Chapter 3. The pole locations are chosen through experiments such that

$$P = \begin{bmatrix} -7 & -7 & -7 & -7 \end{bmatrix} \tag{4.14}$$

Next, using the linear transformed model of the ball-on-beam system in Eq. (3.13) with the pole locations above, the controller gain matrix can be calculated using the pole placement method. The same gain matrix is used for both $x$ and $y$ axes, and the controller gain matrix is the same throughout the experiments and simulations.

$$K = \begin{bmatrix} K_1 & K_2 & K_3 & K_4 \end{bmatrix} = \begin{bmatrix} 2401 & 1372 & 294 & 28 \end{bmatrix} \tag{4.15}$$

As mentioned in Chapter 3, a conventional PID controller is needed to be designed along with AFL before adding NN. Next, the designed PID gains are used for the NN offline training. These PID gains are given below:

$$K_p = 1.0 \tag{4.16}$$

$$K_d = 0.1 \tag{4.17}$$

$$K_i = 1.0 \tag{4.18}$$

### 4.2.1 Stabilization

In order to stabilize the ball at point $(x, y) = (x^d, y^d)$, the control input in Eq. (3.17) can be designed such that

$$v_x(t) = (x^d)^{(4)}(t) + K_4\Big(\dddot{x}^d(t) - \dddot{x}(t)\Big) + K_3\Big(\ddot{x}^d(t) - \ddot{x}(t)\Big) + K_2\Big(\dot{x}^d(t) - x(t)\Big) + K_1\Big(x^d(t) - x(t)\Big) \tag{4.19}$$

where

$$x = \zeta_{1x} \tag{4.20}$$

$$\dot{x} = \zeta_{2x} \tag{4.21}$$

$$\ddot{x} = \zeta_{3x} \tag{4.22}$$

$$\dddot{x} = \zeta_{4x} \tag{4.23}$$

and $K_i$s are the elements of the controller gain matrix of the AFL in Eq. (4.15). The superscript $d$ represents the desired value. The same procedure is applied for the $y$ axis.

$$v_y(t) = (y^d)^{(4)}(t) + K_4\Big(\dddot{y}^d(t) - \dddot{y}(t)\Big) + K_3\Big(\ddot{y}^d(t) - \ddot{y}(t)\Big) + K_2\Big(\dot{y}^d(t) - y(t)\Big) + K_1\Big(y^d(t) - y(t)\Big) \tag{4.24}$$

where

$$y = \zeta_{1y} \tag{4.25}$$

$$\dot{y} = \zeta_{2y} \tag{4.26}$$

$$\ddot{y} = \zeta_{3y} \tag{4.27}$$

$$\dddot{y} = \zeta_{4y} \tag{4.28}$$

When the ball is designed to be balanced at a fixed point on the plate, all the time derivatives of the desired terms are zero:

$$\dot{x}^d = \ddot{x}^d = \dddot{x}^d = (x^d)^{(4)} = 0 \tag{4.29}$$

All the time derivatives of the desired states in $y$ axis are also zero:

$$\dot{y}^d = \ddot{y}^d = \dddot{y}^d = (y^d)^{(4)} = 0 \tag{4.30}$$

It is worth mentioning that the desired values, $x^d$ and $\dot{x}^d$ in the NN-based PID controller (NNPID hereafter) in Eq. (3.49) also need to be changed accordingly.

#### 4.2.1.1 Simulation Results

As mentioned in Chapter 3, each axis of the ball-on-plate system is controlled independently. This section provides the simulated results of using such a controller on the ball-on-plate system. It is worth mentioning that the original nonlinear dynamic model is used in the simulation study. Fig. 4.8 shows the simulation result for stabilizing the ball at the center of the plate using AFL only. Fig. 4.9 shows the same simulation result, but the ball is supposed to be balanced at the point $(0.1, -0.1)$.

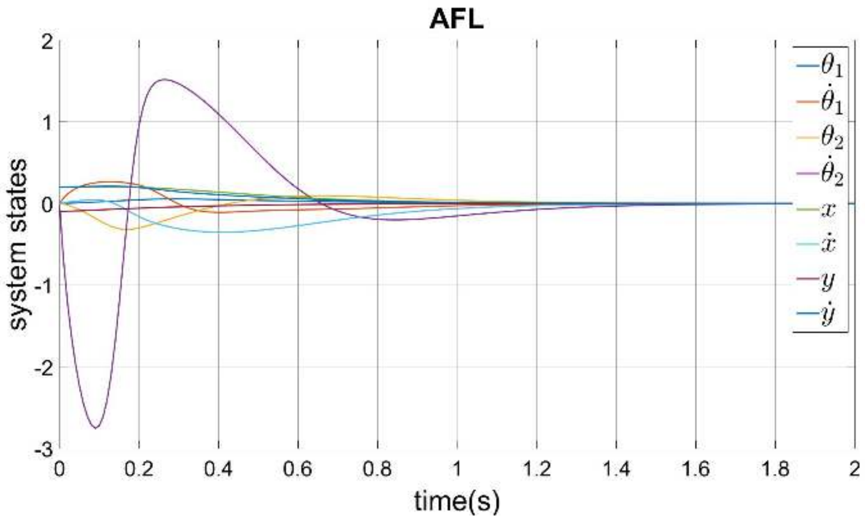

Figure 4.8: Simulation on using AFL in order to stabilize the ball at $(0, 0)$ (center point simulation)

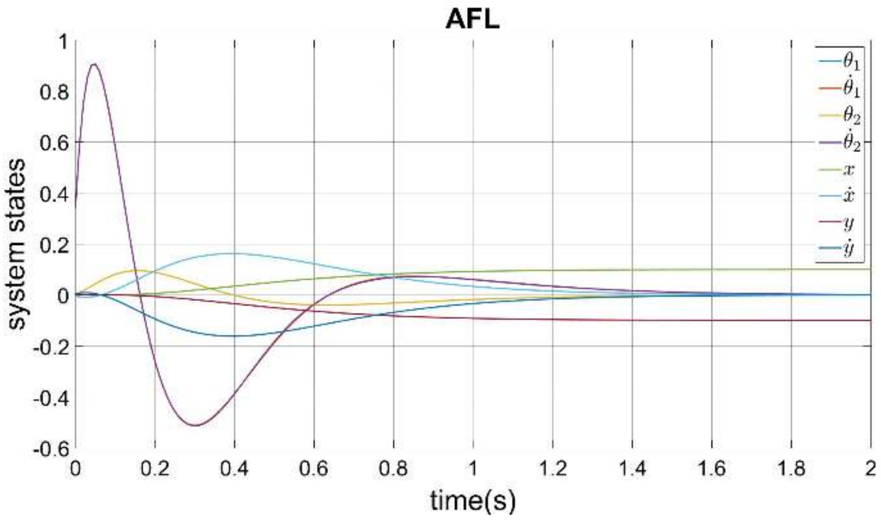

Figure 4.9: Simulation on using AFL in order to stabilize the ball at $(0.1, -0.1)$ (offset point simulation)

The initial conditions used for the center point simulation are given below. All the states started from zero for the offset point simulation.

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.2 & 0 & -0.1 & 0.2 \end{bmatrix}^T \tag{4.31}$$

As it is clear from the simulated results, the amount of errors are very small. The ball is balanced at the point $(1.79 \times 10^{-4}, -4.49 \times 10^{-5})$ and $(0.099, -0.099)$ for the center point and the offset point simulation, respectively.

#### 4.2.1.2 Experimental Results

Fig. 4.10 shows the experimental result for the balancing of the ball at the center of the plate. Since the experimental data are noisy and in order to make the figures clear, only $x$ and $y$ states are shown here.

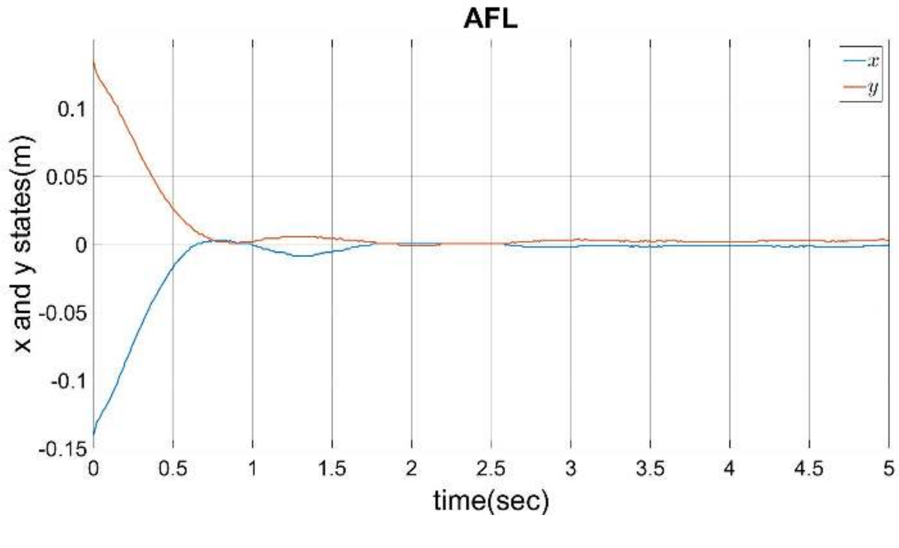

Figure 4.10: Experiment on using AFL in order to stabilize the ball at $(0, 0)$

As it is clear, the ball converges to the desired state $(-0.0064, 0.0026)$ within a small error. The initial conditions for the experiment are given below:

$$X = \begin{bmatrix} 0.00 & 0.06 & 0.00 & -0.03 & -0.14 & -6.43 & 0.13 & 6.14 \end{bmatrix}^T \tag{4.32}$$

Fig. 4.11 shows another stabilization experiment where the ball is supposed to be balanced at $(0.1, -0.1)$. As it is clear, AFL stabilizes the ball at the point $(0.073, 0.076)$. The amount of error in this experiment is 27.47% for the $x$ and 23.94% for the $y$. This error is caused by the approximations that have been made in the AFL design in Chapter 3. The initial conditions for this experiment was zero for all the states.

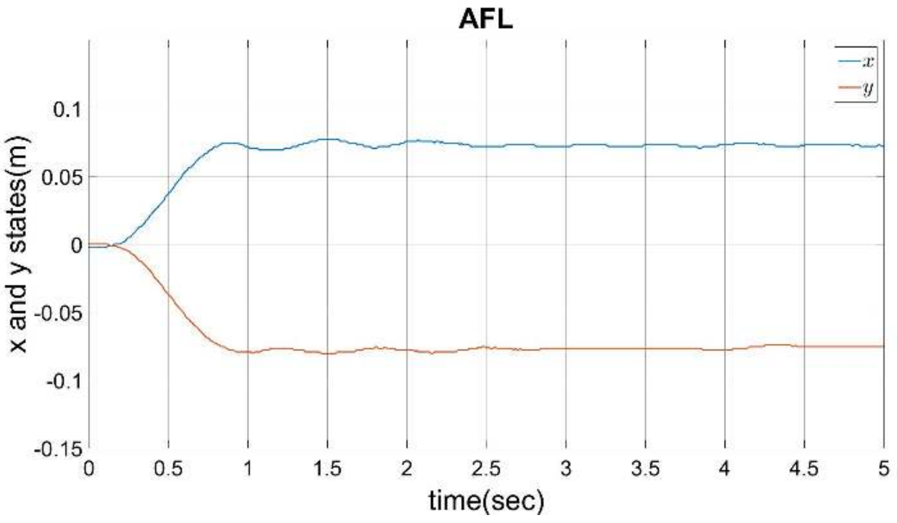

Figure 4.11: Experiment on using AFL in order to stabilize the ball at $(0.1, -0.1)$

The proposed controller which has the structure of AFL plus NNPID (AFLNNPID hereafter) in Fig. 4.12 balanced the ball at $(0.093, -0.093)$, reducing the error to 6.56% and 7.13% for the $x$ and the $y$ axes, respectively. A learning rate of $\eta = 0.3$ and momentum rate of $\alpha = 0.6$ are used in the experiment. The same initial conditions as in AFL experiment, zero for all the states, are used in this experiment.

It is interesting to investigate the change of PID gains and the control inputs over time. Figs. 4.13-4.16 show the change in the PID gains and the amount of control input from AFL and NNPID for the $x$ axis. It is clear from the Fig. 4.13 that the NN helped in adjusting the PID gains. It is worth mentioning that the amount of the torque input needed for the stabilization is small, so the NN updates the PID gains within a small range. It was mentioned before that the ball was balanced at an offset point from the desired location using only AFL. Fig. 4.15 shows that the NNPID sends the compensatory torque in the opposite direction to decrease the effect of the AFL control input. As a result, the ball finds the opportunity to move further under the influence of the gravitational force and get closer to the desired location. Finally, NN stops updating PID gains to fix the ball at the desired location. Fig. 4.16 shows the total control input from AFL and NNPID.

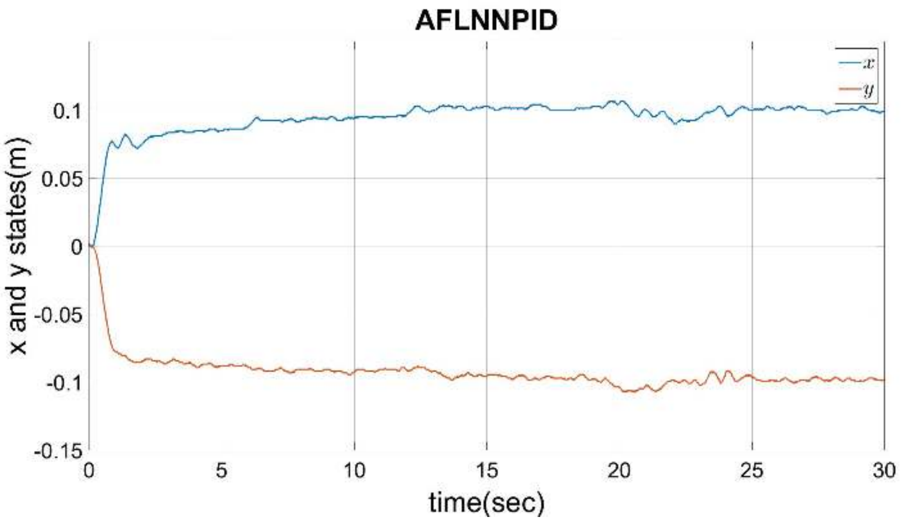

Figure 4.12: Experiment on using AFLNNPID in order to stabilize the ball at $(0.1, -0.1)$

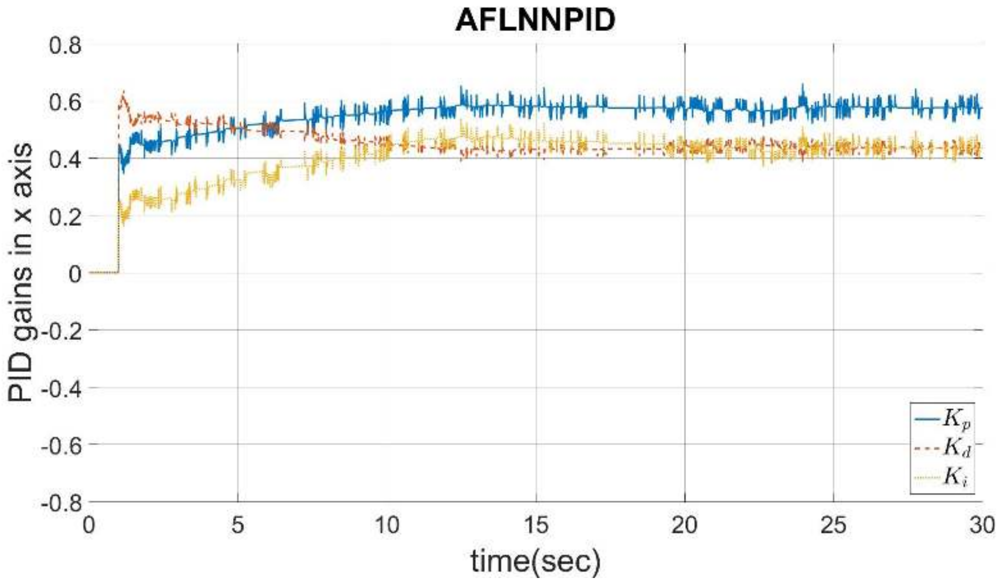

Figure 4.13: Experiment on PID gains in $x$ axis in order to balance the ball at $(0.1, -0.1)$

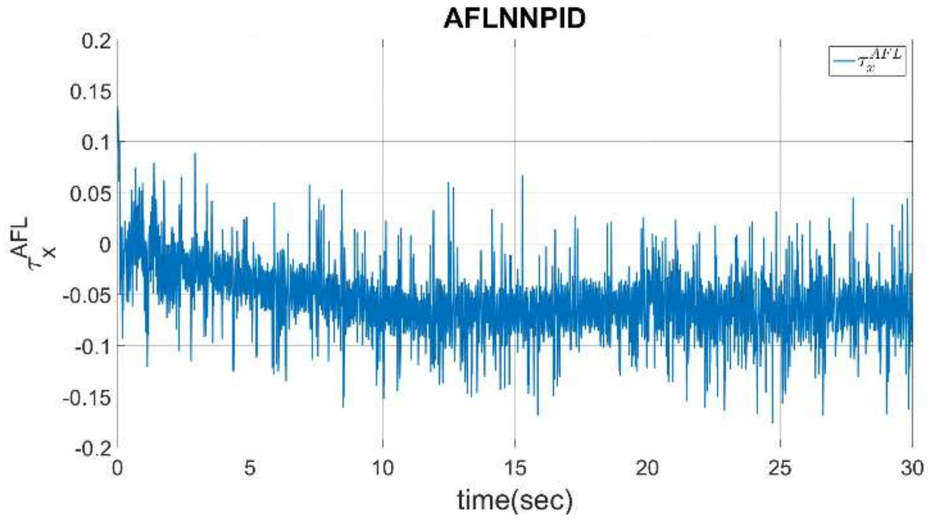

Figure 4.14: Experiment on AFL control input of AFLNNPID in $x$ axis in order to balance the ball at $(0.1, -0.1)$

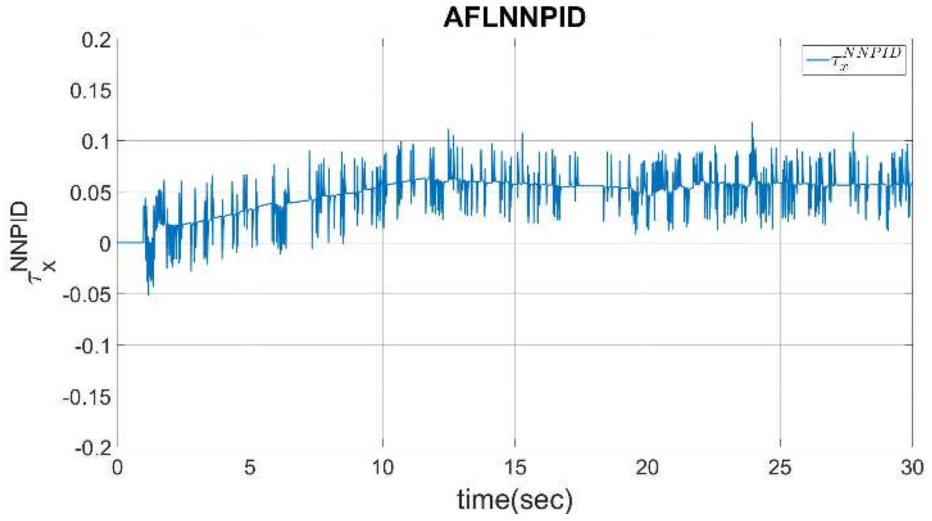

Figure 4.15: Experiment on NNPID control input of AFLNNPID in $x$ axis in order to balance the ball at $(0.1, -0.1)$

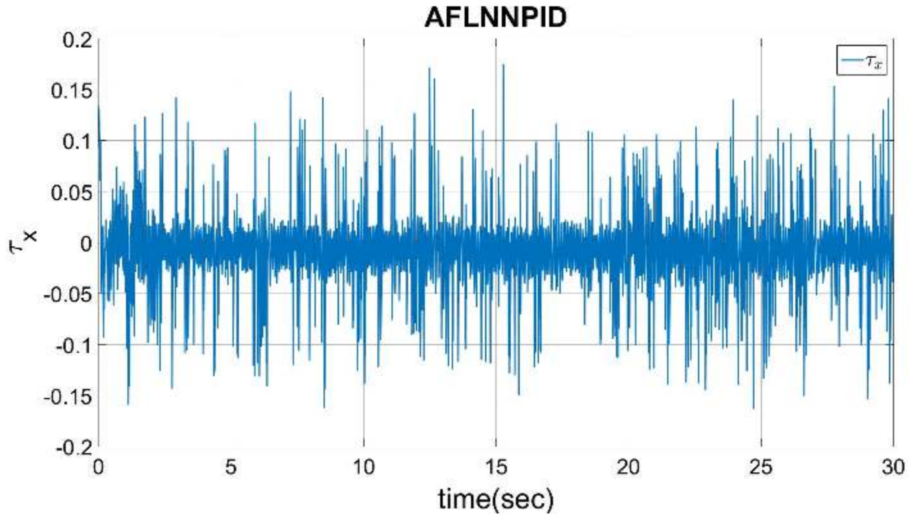

Figure 4.16: Experiment on total control input (AFLNNPID) in $x$ axis in order to balance the ball at $(0.1, -0.1)$

### 4.2.2 Trajectory Tracking

To demonstrate the performance of the proposed controller, a circular tracking of the ball is presented. The desired ball position for the trajectory tracking is given below:

$$x^d = R\sin(\omega t) \tag{4.33}$$

$$y^d = R\cos(\omega t) \tag{4.34}$$

$$\tag{4.35}$$

where $R$, $\omega$, and $t$ are radius of the circle, the circle angular velocity, and the time, respectively. The time derivatives of the desired trajectory are given by

$$\dot{x}^d = R\omega\cos(\omega t) \tag{4.36}$$

$$\ddot{x}^d = -R\omega^2\sin(\omega t) \tag{4.37}$$

$$\dddot{x}^d = -R\omega^3\cos(\omega t) \tag{4.38}$$

$$(x^d)^{(4)} = R\omega^4\sin(\omega t) \tag{4.39}$$

$$\dot{y}^d = -R\omega\sin(\omega t) \tag{4.40}$$

$$\ddot{y}^d = -R\omega^2\cos(\omega t) \tag{4.41}$$

$$\dddot{y}^d = R\omega^3\sin(\omega t) \tag{4.42}$$

$$(y^d)^{(4)} = R\omega^4\cos(\omega t) \tag{4.43}$$

#### 4.2.2.1 Simulation Results

The simulation result of using only AFL for the circular trajectory tracking with a radius of 0.08m and 2.0rad/s angular velocity is shown in Fig. 4.17. As mentioned before, since AFL is based on the decoupled simplified dynamic model, increasing radius or angular velocity would make the ball further deviate from the desired trajectory. Fig. 4.18 shows the simulated system response with a frequency of $\omega = 6$rad/s and radius of R = 0.06m. The ball is balanced at the radius of 0.0684m, which gives an error of 13.21%.

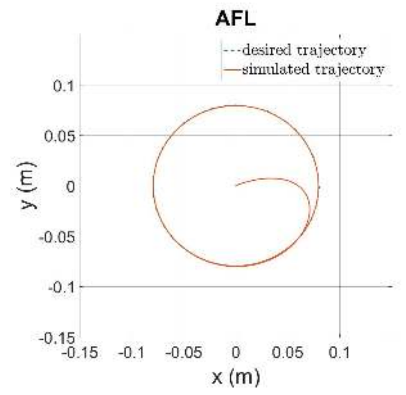

Figure 4.17: Simulation on using AFL for trajectory tracking ($\omega = 2.0$rad/s and R = 0.08m)

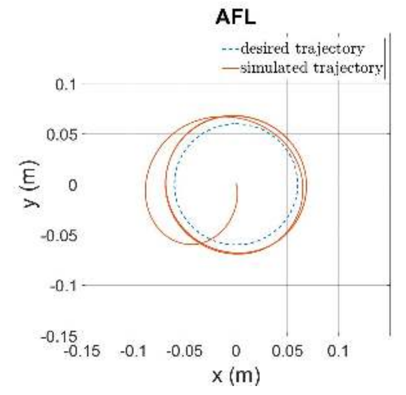

Figure 4.18: Simulation on using AFL for trajectory tracking ($\omega = 6.0$rad/s and R = 0.06m)

When the NNPID is added, it is able to reduce the error as shown in Fig. 4.19. The proposed controller made the ball to follow a trajectory with the radius of 0.0606m, which decreases the error to 0.97%. Figs. 4.20-4.23 show the results of this simulation for PID gains and control inputs for the $x$ axis. Since the ball follows a smaller trajectory in Fig. 4.18 by using only the AFL, it is necessary that the NNPID sends the compensatory torque input in the opposite direction to let the ball get closer to the desired trajectory. As a result of this action, the torque input to the plate will be decreased, and the ball will be pushed further to the circumference of the desired trajectory under the influence of the gravitational force. It is clear from the Fig. 4.22 that the NNPID sends the compensatory torque input in the opposite direction as the AFL, so the ball moves further away from the center of the desired trajectory. Consequently, it decreases the trajectory tracking error. Fig. 4.20 also verifies this conclusion because the NN added more emphasis on the $K_p$ gain along with subtle changes in $K_d$ and $K_i$. Total control input of the simulation is shown in Fig. 4.23.

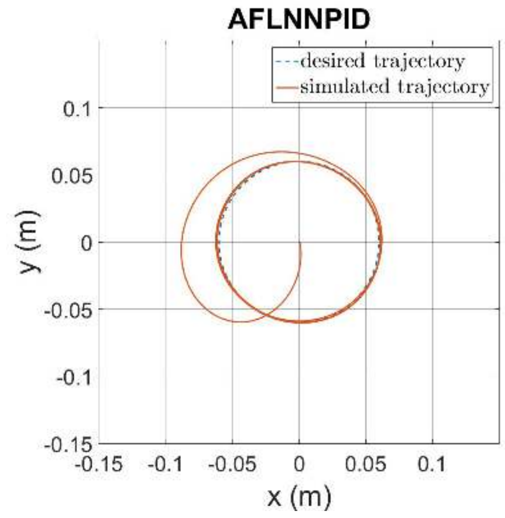

Figure 4.19: Simulation on using AFLNNPID for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m), simulation

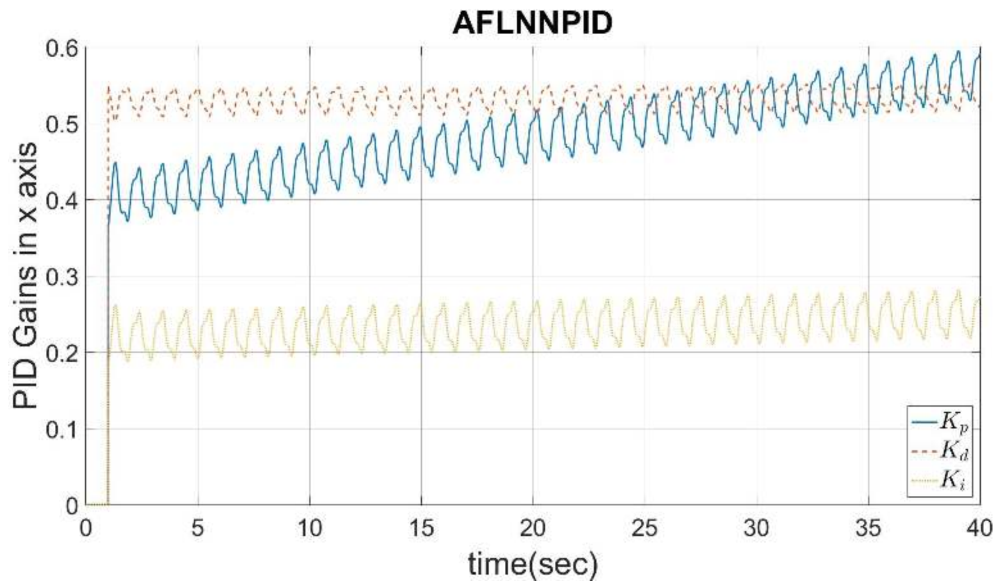

Figure 4.20: Simulation on PID gains in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

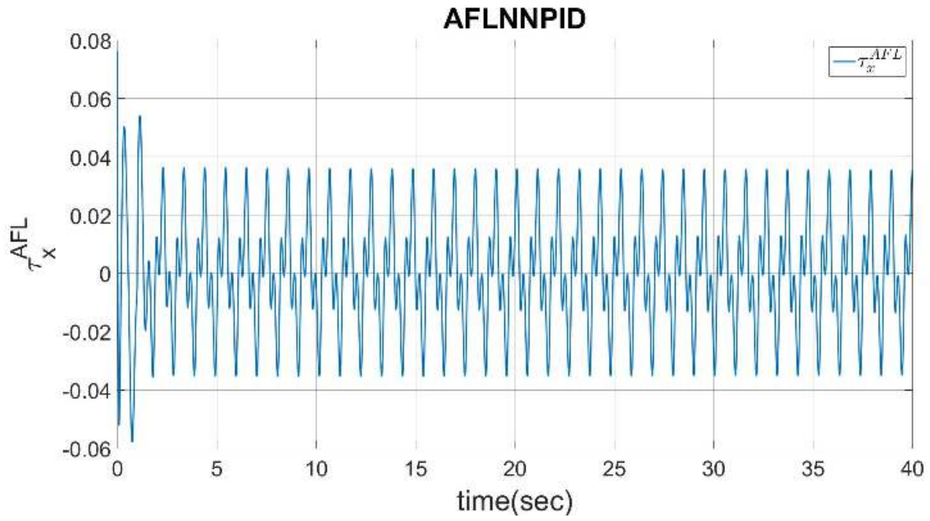

Figure 4.21: Simulation on AFL control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

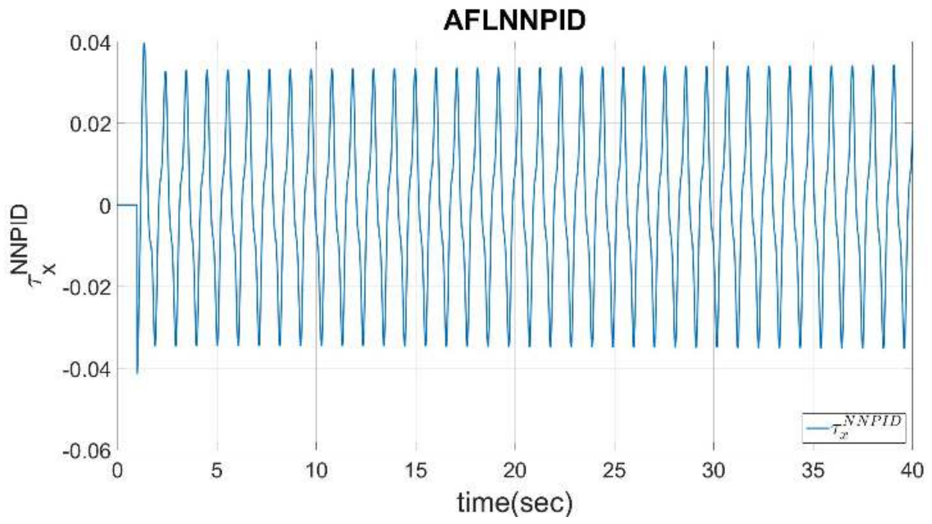

Figure 4.22: Simulation on NNPID control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

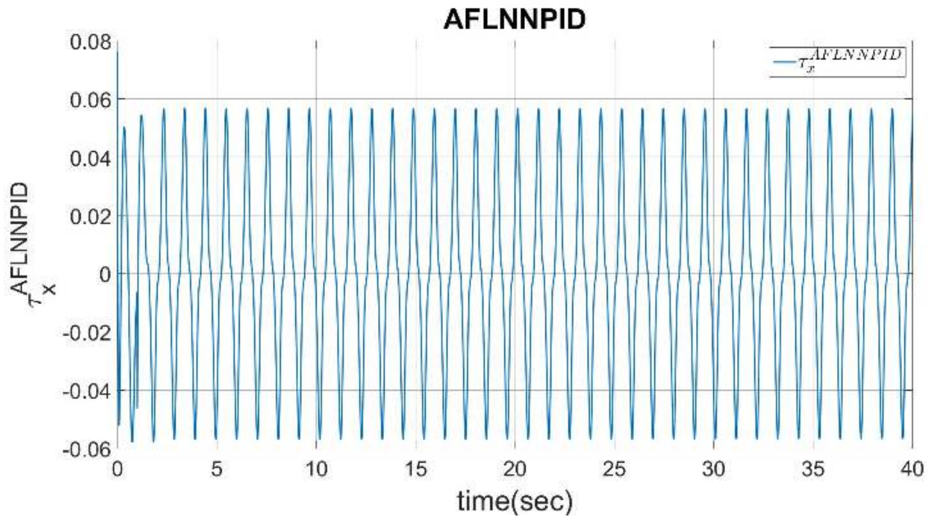

Figure 4.23: Simulation on total control input (AFLNNPID) in $x$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

#### 4.2.2.2 Experimental Results

Fig. 4.24 shows the experimental result of the ball following a circular trajectory with 0.08m radius and 2.0rad/s angular velocity. Next, a NN with $\eta = 0.3$ and $\alpha = 0.6$ as the learning and the momentum rates is used for the NNPID controller, Fig. 4.25. Experimental results show that NNPID controller decreases the error from 14.31% with the radius of 0.0685m in Fig. 4.24 to the error of 9.66% with the radius of 0.0723m in Fig. 4.25.

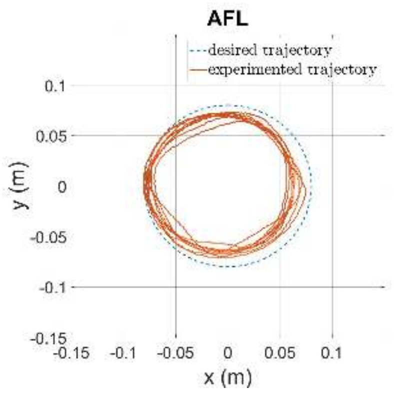

Figure 4.24: Experiment on using AFL for trajectory tracking ($\omega = 2.0$rad/s and R = 0.08m), experiment

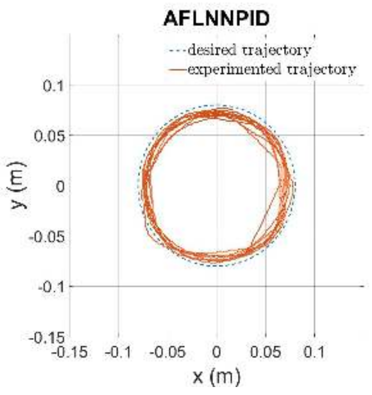

Figure 4.25: Experiment on using AFLNNPID for trajectory tracking ($\omega = 2.0$rad/s and R = 0.08m), experiment

Control inputs and PID gains history are provided in Figs. 4.26-4.29 for the $x$ axis. The $y$ axis results are provided in Appendix D. Looking at the Figs. 4.27 and 4.28 shows that the torque inputs from the NNPID is in the opposite direction of the AFL. For example, at $t = 5s$, the AFL sends almost $-0.05N.m$, but the NNPID counteracts that by sending almost $0.03N.m$ compensatory torque. As a result, the ball finds the opportunity to go further from the center of the circular trajectory under the influence of the gravitational force, and it decreases the trajectory tracking error. Fig. 4.26 shows that the NN adjusted PID gains online successfully. It shows that the NN tried to increase system response by decreasing $K_d$ gain while increasing trajectory tracking accuracy by increasing $K_p$ and $K_i$. Fig. 4.29 also shows the total control input used for the experiment.

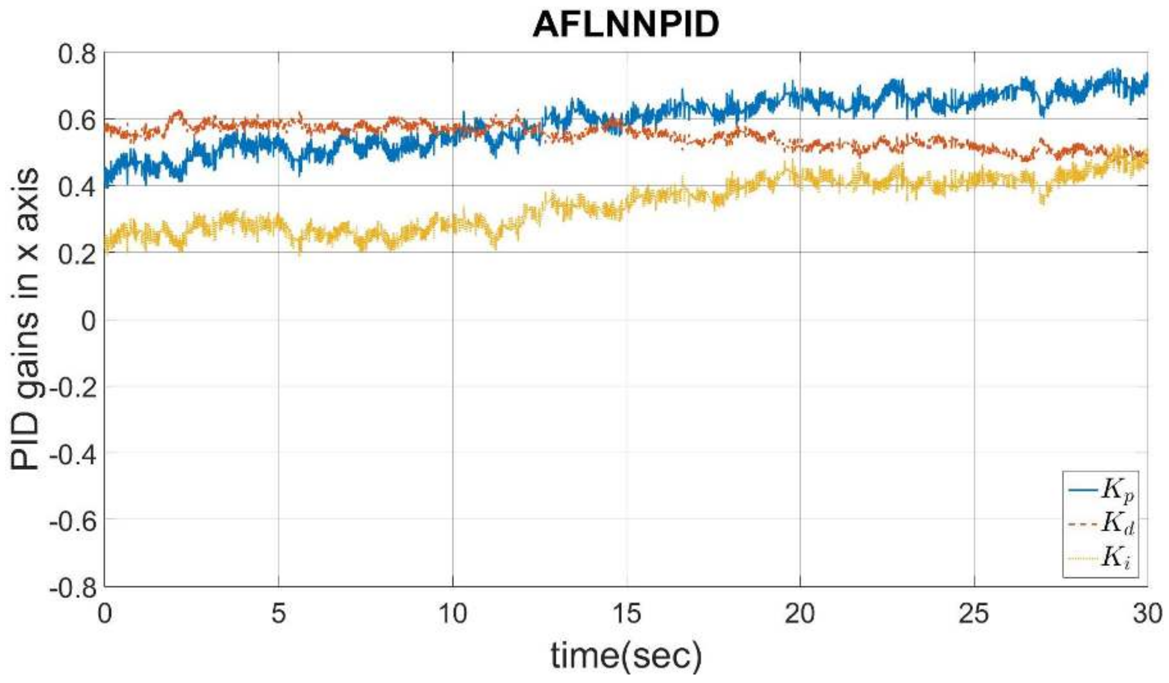

Figure 4.26: Experiment on PID gains in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

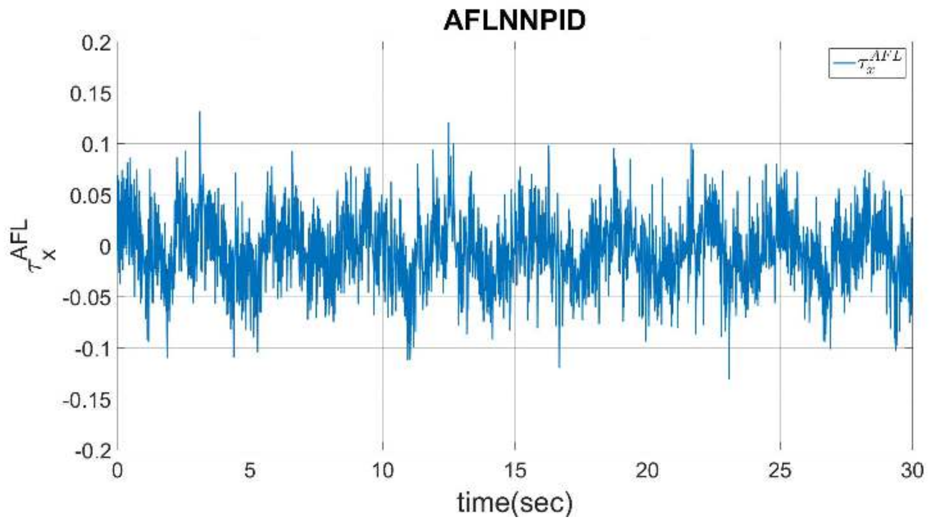

Figure 4.27: Experiment on AFL control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

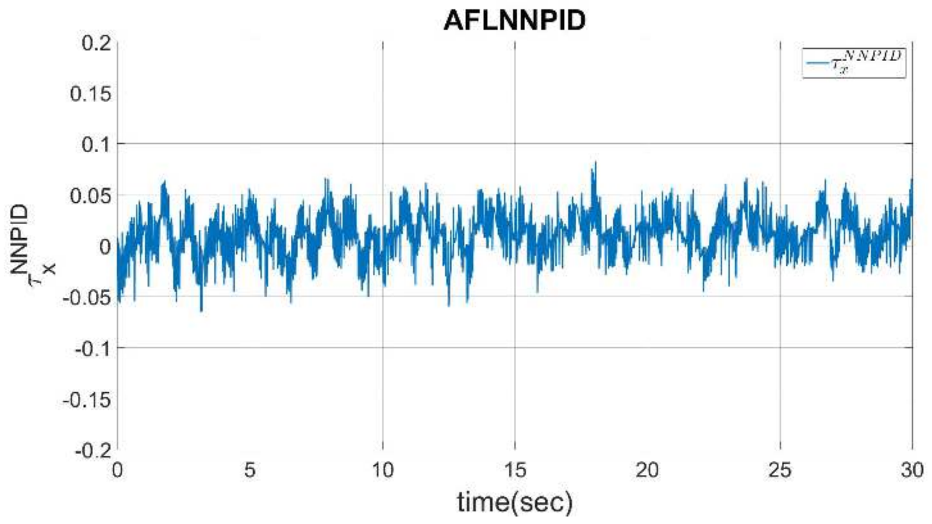

Figure 4.28: Experiment on NNPID control input of AFLNNPID in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

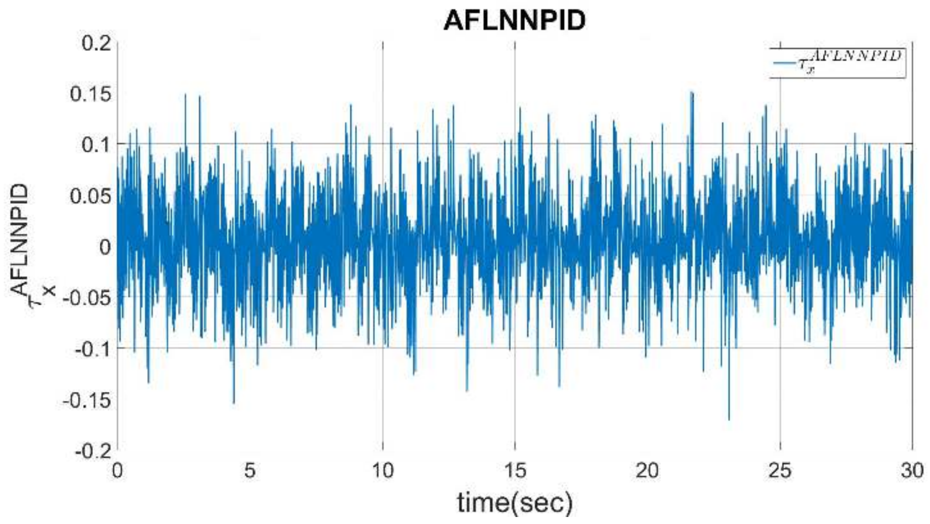

Figure 4.29: Experiment on total control input (AFLNNPID) in $x$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

# CHAPTER 5

# CONCLUSION

This thesis introduces a solution for the problem of nonlinear control design using neural network (NN hereafter). The controller consists of a base controller and a NN-based PID compensator. Since simplified dynamic model is used in the design of the base controller, control errors appear in the system response. The NN-based PID controller is added to the controller to reduce the error. The NN-based PID controller uses NN to generate PID gains. The online training feature of the neural network helps the PID controller to work with updated control gains. Next, the PID controller generates compensatory torque inputs. Finally, the output of NN-based PID controller is added to the output of the base controller. This way, the controller is supposed to decrease errors caused by simplifications and assumptions considered in the design of the base controller.

The proposed control method is verified on the ball-on-plate system for demonstration. It is shown that adding the NN-based PID controller can decrease the error in both stabilization and trajectory tracking. There is still some room for improvement in the design of the NN-based PID compensator in terms of the error reduction. One solution would be considering the plate angle and ball position together in the PID, and the neural network updates both of the PID gains. Neural network structure also needs to be changed accordingly.

The controller has some limitations. First of all, neural network needs to be trained offline. The second is the NN-based PID activation time. As mentioned before, NN-based PID comes into the picture when the base controller has settled the system's response. For the ball-on-plate system in this thesis, the NN-based PID is set to turn on one second after the starting point of the base controller, and the ball-on-plate system is controlled only by the base controller during the first second.

Finally, it would be helpful to design an extra neural network for estimating dynamic model of the system. In this case, one neural network is used in the controller and the second one is utilized as the system dynamic estimator. However, state derivatives of the system are noisy due to numerical derivation, so design of the neural network estimator that fits the noisy data is difficult. A possible solution in order to bypass the noisy data of the state derivatives would be using an observer. In that case, the observer predicts the value of the state derivatives using available position states, and the neural network dynamic estimator can be designed based on the position states and observer outputs. It is also recommended to use a better training algorithms like Levenberge-Marquardt, which has better convergence than backpropagation method. Finally, it would be a good idea to use a small, fast controller board, mentioned in Chapter 4, to make the entire system portable.

# REFERENCES

[1] Ming-tzu Ho, Yi-wei Tu, and Hao-shuan Lin, Controlling a ball and wheel system using full-state-feedback linearization [Focus on Education, IEEE Control Systems Magazine, vol. 29, no. 5, pp. 93101, Oct. 2009.

[2] J. C. Ryu, F. Ruggiero, and K. M. Lynch, Control of Nonprehensile Rolling Manipulation: Balancing a Disk on a Disk, IEEE Transactions on Robotics, vol. 29, no. 5, pp. 11521161, Oct. 2013.

[3] J. Hauser, S. Sastry, and P. Kokotovic, Nonlinear control via approximate input-output linearization: the ball and beam example, IEEE Transactions on Automatic Control, vol. 37, no. 3, pp. 392398, Mar. 1992.

[4] Y. Guo, D. J. Hill, and Z.-P. Jiang, Global nonlinear control of the ball and beam system, in Proceedings of 35th IEEE Conference on Decision and Control, 1996, vol. 3, pp. 28182823 vol.3.

[5] S. Y. Liu, Y. Rizal, and M. T. Ho, Stabilization of a ball and sphere system using feedback linearization and sliding mode control, in 2011 8th Asian Control Conference (ASCC), 2011, pp. 13341339.

[6] M.-T. Ho, Y. Rizal, and L.-M. Chu, Visual Servoing Tracking Control of a Ball and Plate System: Design, Implementation and Experimental Validation, International Journal of Advanced Robotic Systems, vol. 10, no. 7, p. 287, Jul. 2013.

[7] T. D. C. Thanh and K. K. Ahn, Nonlinear PID control to improve the control performance of 2 axes pneumatic artificial muscle manipulator using neural network, Mechatronics, vol. 16, no. 9, pp. 577587, Nov. 2006.

[8] K. K. Ahn and T. D. C. Thanh, Nonlinear PID control to improve the control performance of the pneumatic artificial muscle manipulator using neural network, Journal of Mechanical Science and Technology, vol. 19, no. 1, pp. 106115, Jan. 2005.

[9] . Eski and . Yldrm, Vibration control of vehicle active suspension system using a new robust neural network control system, Simulation Modelling Practice and Theory, vol. 17, no. 5, pp. 778793, May 2009.

[10] S. Cong and Y. Liang, PID-Like Neural Network Nonlinear Adaptive Control for Uncertain Multivariable Motion Control Systems, IEEE Transactions on Industrial Electronics, vol. 56, no. 10, pp. 38723879, Oct. 2009.

[11] S. Jung and S. S. Kim, Control Experiment of a Wheel-Driven Mobile Inverted Pendulum Using Neural Network, IEEE Transactions on Control Systems Technology, vol. 16, no. 2, pp. 297303, Mar. 2008.

[12] S. Jung and H. T. Cho, Decoupled Neural Network Reference Compensation Technique for a PD Controlled Two Degrees-of-Freedom Inverted Pendulum, International Journal of Control, vol. 2, no. 1, p. 9, 2004.

[13] S. Jung, H. T. Cho, and T. C. Hsia, Neural Network Control for Position Tracking of a Two-Axis Inverted Pendulum System: Experimental Studies, IEEE Transactions on Neural Networks, vol. 18, no. 4, pp. 10421048, Jul. 2007.

[14] T.-C. Chen and T.-T. Sheu, Model reference neural network controller for induction motor speed control, IEEE Transactions on Energy Conversion, vol. 17, no. 2, pp. 157163, Jun. 2002.

[15] S. S. Ge, C. C. Hang, and Tao Zhang, Adaptive neural network control of nonlinear systems by state and output feedback, IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), vol. 29, no. 6, pp. 818828, Dec. 1999.

[16] P. Baldi, Gradient descent learning algorithm overview: a general dynamical systems perspective, IEEE Transactions on Neural Networks, vol. 6, no. 1, pp. 182195, Jan. 1995.

# APPENDIX A

# EQUATIONS OF MOTION MATRICES

## A.1 Equations of Motion Matrices

The equations of motion of the ball-on-plate system are derived in Chapter 2. This appendix shows the matrices of the inertia, centrifugal and coriolis forces, potential, and external forces.

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

where elements are

$$\begin{aligned}
m_{11} &= -mx^2\cos^2(\theta_2) + I_b + I_{zp} + my^2 + mr^2\cos^2(\theta_2) + \cos^2(\theta_2)I_{xp} - I_{zp}\cos^2(\theta_2) + mx^2 \\
&\quad - 2mr\sin(\theta_2)x\cos(\theta_2) \\
m_{12} &= -my(\cos(\theta_2)x + \sin(\theta_2)r) \\
m_{13} &= -m\sin(\theta_2)y \\
m_{14} &= -\frac{-mr\sin(\theta_2)x + I_b\cos(\theta_2) + mr^2\cos(\theta_2)}{r} \\
m_{21} &= -my(\cos(\theta_2)x + \sin(\theta_2)r) \\
m_{22} &= mr^2 + I_{yp} + I_b + mx^2 \\
m_{23} &= \frac{I_b + mr^2}{r} \\
m_{24} &= 0 \\
m_{31} &= -m\sin(\theta_2)y
\end{aligned}$$

$$m_{32} = \frac{I_b + mr^2}{r}$$
$$m_{33} = \frac{I_b + mr^2}{r^2}$$
$$m_{34} = 0$$
$$m_{41} = -\frac{-mr\sin(\theta_2)x + I_b\cos(\theta_2) + mr^2\cos(\theta_2)}{r}$$
$$m_{42} = 0$$
$$m_{43} = 0$$
$$m_{44} = \frac{I_b + mr^2}{r^2}$$

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

where elements are

$$c_{11} = -2\cos(\theta_2)I_{xp}\sin(\theta_2)\dot{\theta}_2 - 2mr^2\cos(\theta_2)\sin(\theta_2)\dot{\theta}_2 + 2mx^2\cos(\theta_2)\sin(\theta_2)\dot{\theta}_2 +$$
$$2I_{zp}\cos(\theta_2)\sin(\theta_2)\dot{\theta}_2 - 2mr\sin(\theta_2)\dot{x}\cos(\theta_2) + 2my\dot{y} - 4mr\cos^2(\theta_2)\dot{\theta}_2 x - 2mx\cos^2(\theta_2)\dot{x}$$
$$+ 2mx\dot{x} + 2mr\dot{\theta}_2 x$$
$$c_{12} = \frac{-2mr\cos(\theta_2)\dot{x}y + I_b\sin(\theta_2)\dot{y}}{r} - \dot{\theta}_2 my(-\sin(\theta_2)x + \cos(\theta_2)r)$$
$$c_{13} = 0$$
$$c_{14} = 0$$

$$c_{21} = -\frac{\dot{y}(2mrx\cos(\theta_2) + 2mr^2\sin(\theta_2) + I_b\sin(\theta_2))}{r} + \dot{\theta}_1(\cos(\theta_2)I_{xp}\sin(\theta_2)$$

$$+ mr^2\cos(\theta_2)\sin(\theta_2) - mx^2\cos(\theta_2)\sin(\theta_2) - I_{zp}\cos(\theta_2)\sin(\theta_2) + 2mr\cos^2(\theta_2)x - mrx)$$

$$c_{22} = 2mx\dot{x}$$

$$c_{23} = 0$$

$$c_{24} = 0$$

$$c_{31} = -2m\sin(\theta_2)\dot{y} + \dot{\theta}_1 m\sin(\theta_2)(-\sin(\theta_2)x + \cos(\theta_2)r)$$

$$c_{32} = -mx\dot{\theta}_2$$

$$c_{33} = 0$$

$$c_{34} = 0$$

$$c_{41} = \frac{I_b\sin(\theta_2)\dot{\theta}_2 + 2mr^2\sin(\theta_2)\dot{\theta}_2 + 2mr\cos(\theta_2)\dot{\theta}_2 x + 2mr\sin(\theta_2)\dot{x}}{r} - ym\dot{\theta}_1$$

$$c_{42} = 0$$

$$c_{43} = 0$$

$$c_{44} = 0$$

$$G = \begin{bmatrix} -mg(-\sin(\theta_1)\sin(\theta_2)x - \cos(\theta_1)y + \cos(\theta_2)\sin(\theta_1)r) \\ -mg\cos(\theta_1)(\cos(\theta_2)x + \sin(\theta_2)r) \\ -mg\cos(\theta_1)\sin(\theta_2) \\ mg\sin(\theta_1) \end{bmatrix}$$

$$Q = \begin{bmatrix} \tau_1 & \tau_2 & 0 & 0 \end{bmatrix}^T$$

# APPENDIX B

# Y AXIS CONTROLLER

## B.1 Y Axis Controller

Controller design of the ball-on-beam system in $x$ axis is discussed in Chapter 3. This appendix discusses the controller design in $y$ axis.

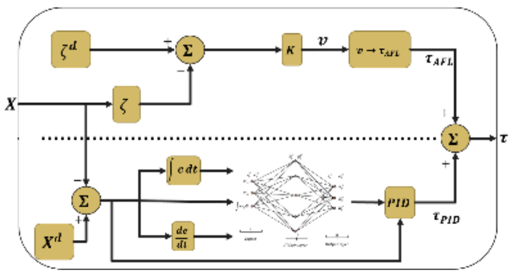

Figure B.1: Inside of controller block diagram ($y$ axis)

### B.1.1 Approximate Feedback Linearization

States of the ball-on-beam system in $y$ axis are given below:

$$X = \left[\theta_1, \dot{\theta}_1, y, \dot{y}\right]^T \tag{B.1}$$

Ball-on-beam equations of motion in $y$ axis are derived in Eqs. (2.25) and (2.28).

$$\dot{X} = \left[\dot{\theta}_1, \ddot{\theta}_1, \dot{y}, \ddot{y}\right]^T \tag{B.2}$$

$$g(x) = \begin{bmatrix} 0 & \dfrac{1}{I_b + I_p + my^2} & 0 & 0 \end{bmatrix}^T \tag{B.3}$$

$$u = \tau_1 \tag{B.4}$$

$$f(X) = \begin{bmatrix} \dot{\theta}_1 \\ \frac{1}{I_b + I_p + my^2}\left(mgr\sin(\theta_1) - mgy\cos(\theta_1) - 2my\dot{y}\dot{\theta}_1\right) \\ \dot{x} \\ \frac{1}{m + \frac{I_b}{r^2}}\left(my\dot{\theta}_1^2 - mg\sin(\theta_1)\right) \end{bmatrix} \tag{B.5}$$

(B.6)

Output of the ball-on-beam system is ball position itself.

$$h(X) = y \tag{B.7}$$

Transformed states of the ball-on-beam system in $y$ axis is given by

$$\zeta_1 = h(X) = y \tag{B.8}$$

$$\zeta_2 = \frac{d\zeta_1}{dt} = \dot{y} \tag{B.9}$$

$$\zeta_3 = \frac{d\zeta_2}{dt} = \ddot{y} = \frac{1}{m + \frac{I_b}{r^2}}\left(my\dot{\theta}_1^2 - mg\sin(\theta_1)\right) \tag{B.10}$$

$$\zeta_4 = \frac{d\zeta_3}{dt} = \frac{1}{m + \frac{I_b}{r^2}}\left(m\dot{y}\dot{\theta}_1^2 - mg\dot{\theta}_1\cos(\theta_1)\right) \tag{B.11}$$

where the term $2my\dot{\theta}_1\ddot{\theta}_1$ is ignored from the $\zeta_4$. Transformed control input can be obtained as below:

$$\dot{\zeta}_4 = \alpha + \beta\tau_1 \tag{B.12}$$

$$\alpha = A_y^2 m\dot{\theta}_1^2(my\dot{\theta}_1^2 - mg\sin(\theta_1)) + A_y mg\dot{\theta}_1^2\sin(\theta_1) + \tag{B.13}$$

$$A_yB_y(2m\dot{y}\dot{\theta}_1 - mg\cos(\theta_1))(-2my\dot{y}\dot{\theta}_1 - mgy\cos(\theta_1) + mgr\sin(\theta_1)) \tag{B.14}$$

$$\beta = A_y B_y \left[2m\dot{y}\dot{\theta}_1 - mg\cos(\theta_1)\right] \tag{B.15}$$

$$A_y = \frac{1}{m + \dfrac{I_b}{r^2}} \tag{B.16}$$

$$B_y = \frac{1}{my^2 + I_p + I_b} \tag{B.17}$$

### B.1.2 NN-Based PID Controller

NN-based PID controller is discussed in this part. PID controller in $y$ axis is given below:

$$\tau_{NN} = K_p(y^d - y) + K_d(\dot{y}^d - \dot{y}) + K_i\left[\int_0^t (y^d - y)dt\right] \tag{B.18}$$

Neural network in $y$ axis is shown in Fig. B.2.

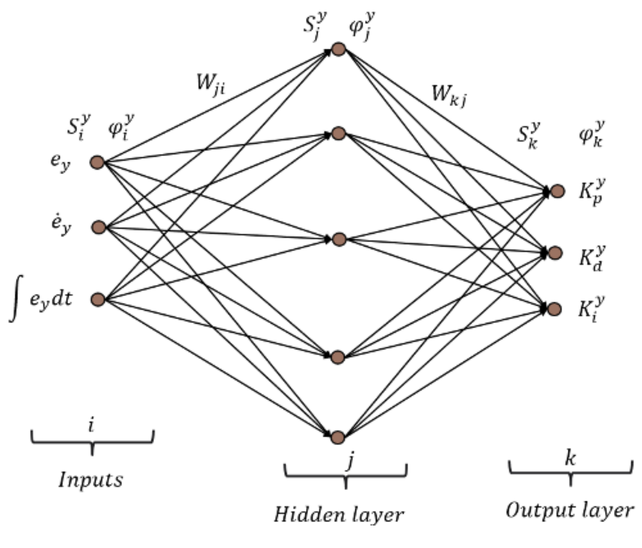

Figure B.2: Neural network structure ($y$ axis)

The inputs to the neural network are given below:

$$S_1 = y^d - y \tag{B.19}$$

$$S_2 = \dot{y}^d - \dot{y} \tag{B.20}$$

$$S_3 = \int_0^t (y^d - y)dt \tag{B.21}$$

A tangent sigmoid and a linear transfer functions are used for the hidden and the output layers, respectively.

$$f_{tansig}(s) = \frac{2}{1 + e^{-2s}} - 1 \tag{B.22}$$

$$f_{lin}(s) = s \tag{B.23}$$

The outputs of the neural network are given below:

$$\phi_1 = K_p \tag{B.24}$$

$$\phi_2 = K_d \tag{B.25}$$

$$\phi_3 = K_i \tag{B.26}$$

From Fig. B.1,

$$\tau_1 = \tau_{AFL} + \tau_{NN} \tag{B.27}$$

Neural network error function is given below:

$$E = \frac{1}{2}(y^d - y)^2 \tag{B.28}$$

Applying backpropagation on neural network results in updating equations:

$$\frac{\partial E}{\partial w_{kj}} = (\frac{\partial E}{\partial y})(\frac{\partial y}{\partial \tau_1})(\frac{\partial \tau_1}{\partial \tau_{NN}})(\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial w_{kj}}) \tag{B.29}$$

$$\frac{\partial E}{\partial b_k} = (\frac{\partial E}{\partial y})(\frac{\partial y}{\partial \tau_1})(\frac{\partial \tau_1}{\partial \tau_{NN}})(\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial b_k}) \tag{B.30}$$

$$\frac{\partial E}{\partial w_{ji}} = (\frac{\partial E}{\partial y})(\frac{\partial y}{\partial \tau_1})(\frac{\partial \tau_1}{\partial \tau_{NN}}) \left[ \sum_{k=1}^{3} (\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial \phi_j}) \right] (\frac{\partial \phi_j}{\partial S_j})(\frac{\partial S_j}{\partial w_{ji}}) \tag{B.31}$$

$$\frac{\partial E}{\partial b_j} = (\frac{\partial E}{\partial y})(\frac{\partial y}{\partial \tau_1})(\frac{\partial \tau_1}{\partial \tau_{NN}}) \left[ \sum_{k=1}^{3} (\frac{\partial \tau_{NN}}{\partial \phi_k})(\frac{\partial \phi_k}{\partial S_k})(\frac{\partial S_k}{\partial \phi_j}) \right] (\frac{\partial \phi_j}{\partial S_j})(\frac{\partial S_j}{\partial b_j}) \tag{B.32}$$

From Eq. (B.28),

$$\frac{\partial E}{\partial y} = -(y^d - y) \tag{B.33}$$

From Fig. 3.1,

$$\frac{\partial y}{\partial \tau_1} = sign(\frac{\Delta y}{\Delta \tau_1}) = sign(\frac{y - y_{prev}}{\tau_1 - \tau_{1prev}}) \tag{B.34}$$

From Eq. (B.27),

$$\frac{\partial \tau_1}{\partial \tau_{NN}} = 1 \tag{B.35}$$

From Eq. (B.18),

$$\frac{\partial \tau_{NN}}{\partial \phi_1} = y^d - y \tag{B.36}$$

$$\frac{\partial \tau_{NN}}{\partial \phi_2} = \dot{y}^d - \dot{y} \tag{B.37}$$

$$\frac{\partial \tau_{NN}}{\partial \phi_3} = \int_0^t (y^d - y) dt \tag{B.38}$$

(B.39)

From Eqs. (3.36)-(3.40),

$$\frac{\partial S_k}{\partial w_{kj}} = \phi_j \tag{B.40}$$

$$\frac{\partial S_k}{\partial b_k} = 1 \tag{B.41}$$

$$\frac{\partial S_k}{\partial \phi_j} = w_{kj} \tag{B.42}$$

$$\frac{\partial S_j}{\partial w_{ji}} = \phi_i \tag{B.43}$$

$$\frac{\partial S_j}{\partial b_j} = 1 \tag{B.44}$$

From Eqs. (B.22) and (B.23),

$$\frac{\partial \phi_k}{\partial S_k} = 1 \tag{B.45}$$

$$\frac{\partial \phi_j}{\partial S_j} = 1 - \phi_j^2 \tag{B.46}$$

(B.47)

# APPENDIX C

# EXPERIMENTAL SETUP

## C.1 Experimental Setup

Items used for the design of the ball-on-plate system are summarized in Table C.1. Figs. C.1-C.11 show the items of the mechanical setup in SolidWorks.

Table C.1: Mechanical setup items of the ball-on-plate system

| item | Model/Material | Quantity |
|---|---|---|
| DYNAMIXEL motor | XM430-W210-T | 2 |
| USB2Dynamixel | ... | 1 |
| Robotis power supply | ... | 1 |
| Motor frames | Aluminum | 2 |
| Rod end bearing | Steel | 2 |
| Rod end bolt | Carbon steel | 2 |
| Top plate | Acrylic | 1 |
| Base plate | Acrylic | 1 |
| Tube(connecting rod) | Aluminum | 1(3ft) |
| Center rod | Aluminum | 1(0.5ft) |
| Universal joint | Metal/Plastic | 3 |

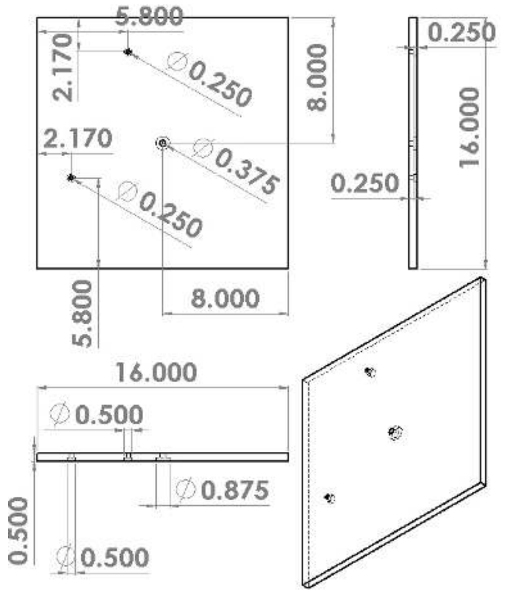

Figure C.1: Base plate designed in SolidWorks

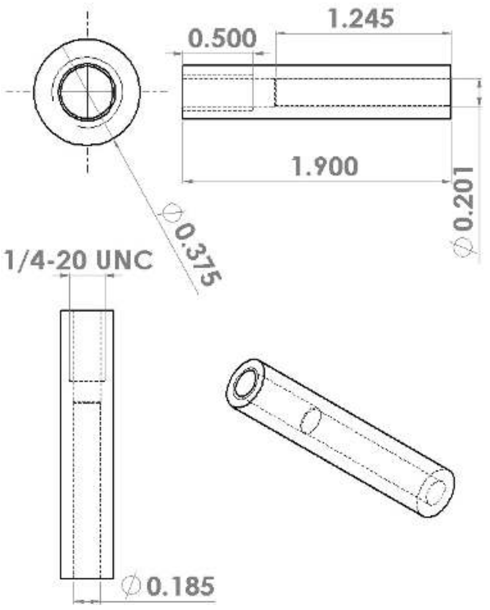

Figure C.2: Connecting rod of the linkage mechanism designed in SolidWorks

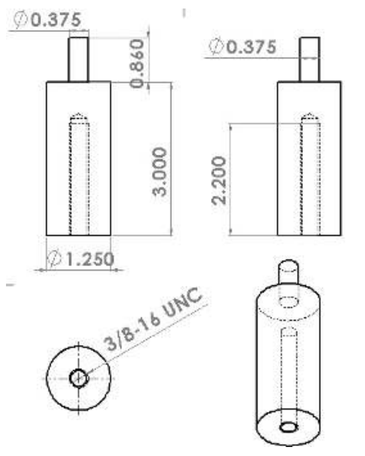

Figure C.3: Center rod designed in SolidWorks

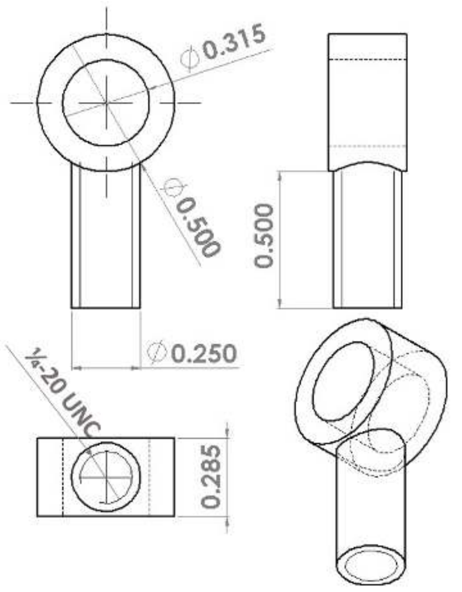

Figure C.4: Rod end of the linkage mechanism designed in SolidWorks

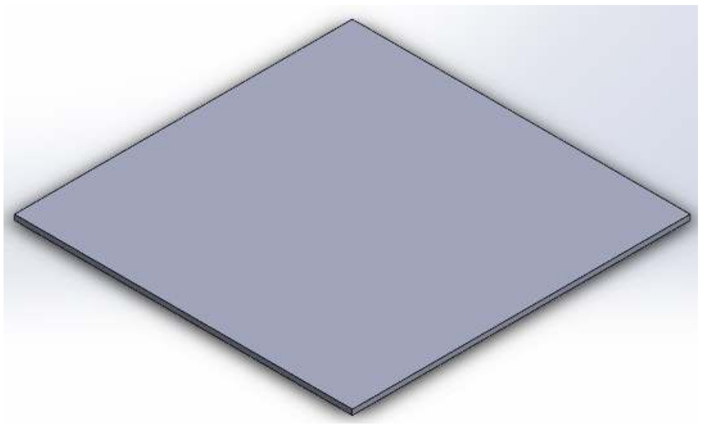

Figure C.5: Top plate (off-the-shelf item)

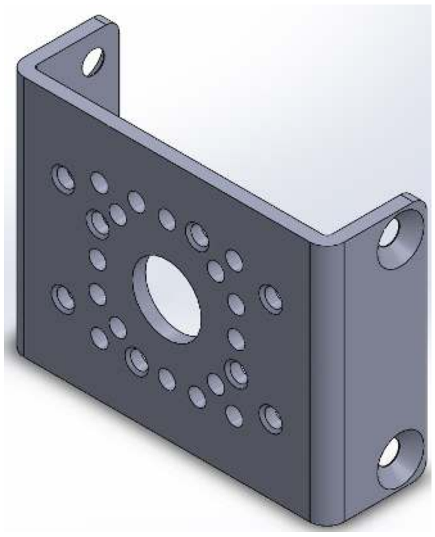

Figure C.6: Motor bottom bracket (off-the-shelf item)

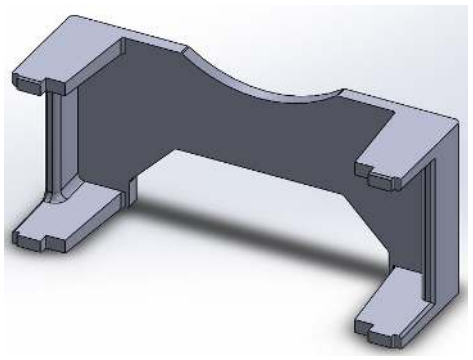

Figure C.7: Motor back frame (off-the-shelf item)

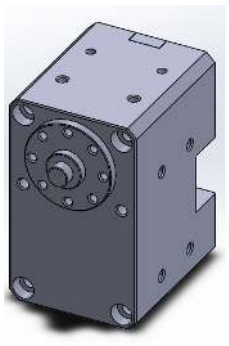

Figure C.8: Dynamixel motor (off-the-shelf item)

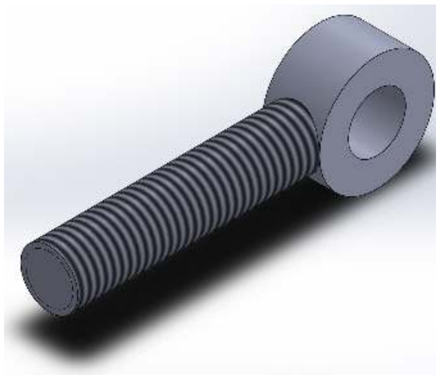

Figure C.9: Rod end attached to motor side bracket (off-the-shelf item)

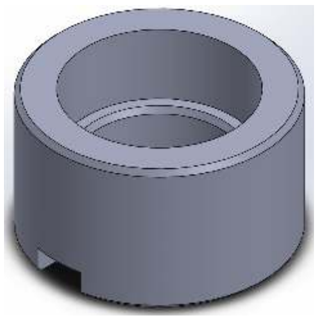

Figure C.10: Motor spacer (off-the-shelf item)

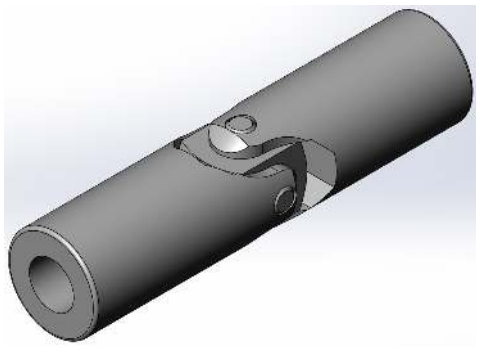

Figure C.11: Universal joint (off-the-shelf item)

# APPENDIX D

# Y AXIS RESULTS

## D.1 Experiment on Offset Point Stabilization

This section provides the output graphs for the PID gains and the control inputs for the $y$ axis where the ball is supposed to be balanced at $(0.1, -0.1)$.

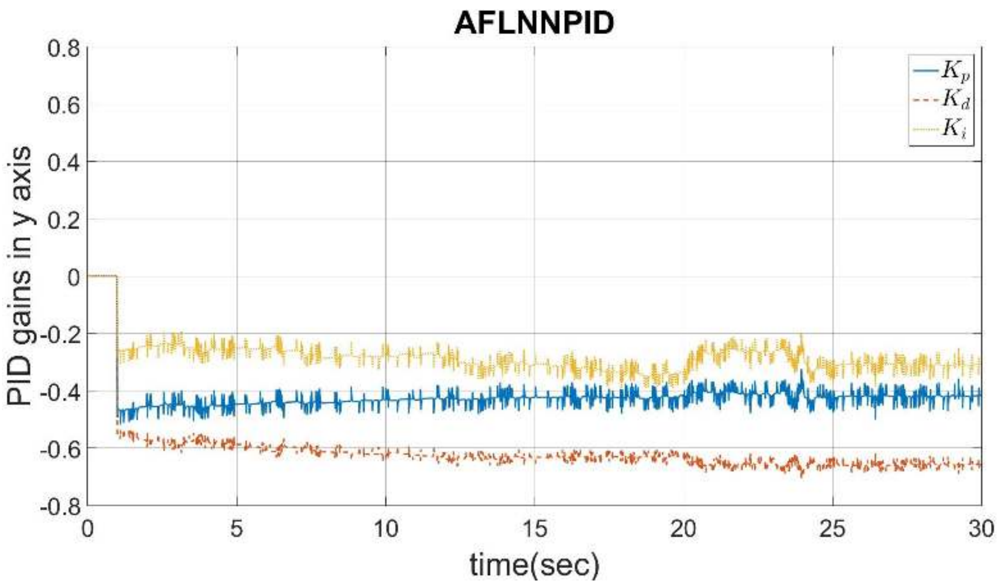

Figure D.1: Experiment on PID gains in $y$ axis in order to balance the ball at $(0.1, -0.1)$

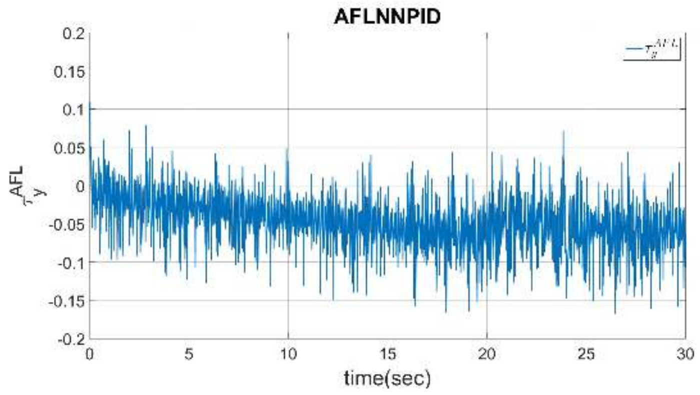

Figure D.2: Experiment on AFL control input of AFLNNPID in $y$ axis in order to balance the ball at $(0.1, -0.1)$

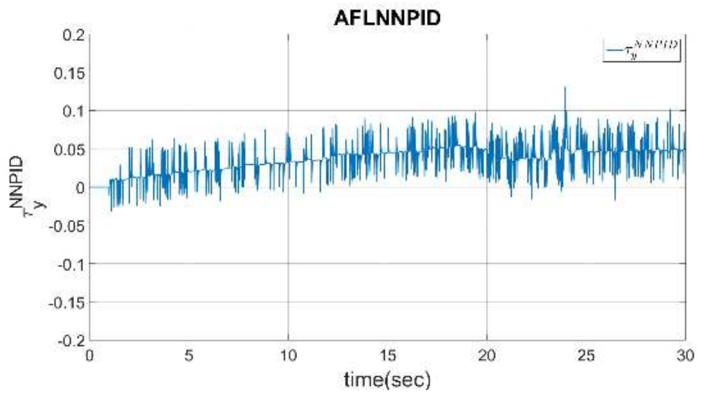

Figure D.3: Experiment on NNPID control input of AFLNNPID in $y$ axis in order to balance the ball at $(0.1, -0.1)$

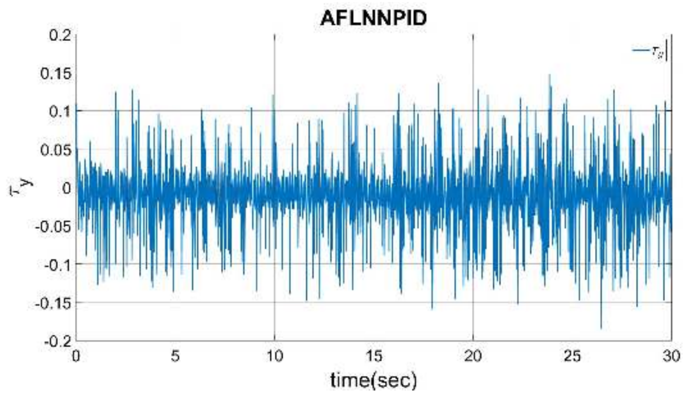

Figure D.4: Experiment on total control input (AFLNNPID) in $y$ axis in order to balance the ball at $(0.1, -0.1)$

## D.2 Simulation on Trajectory Tracking

The simulated output graphs of trajectory tracking for the change in PID gains and the control inputs are provided in this section. The ball is supposed to follow a circular trajectory with a radius of 0.06m and angular velocity of $\omega = 6$rad/s.

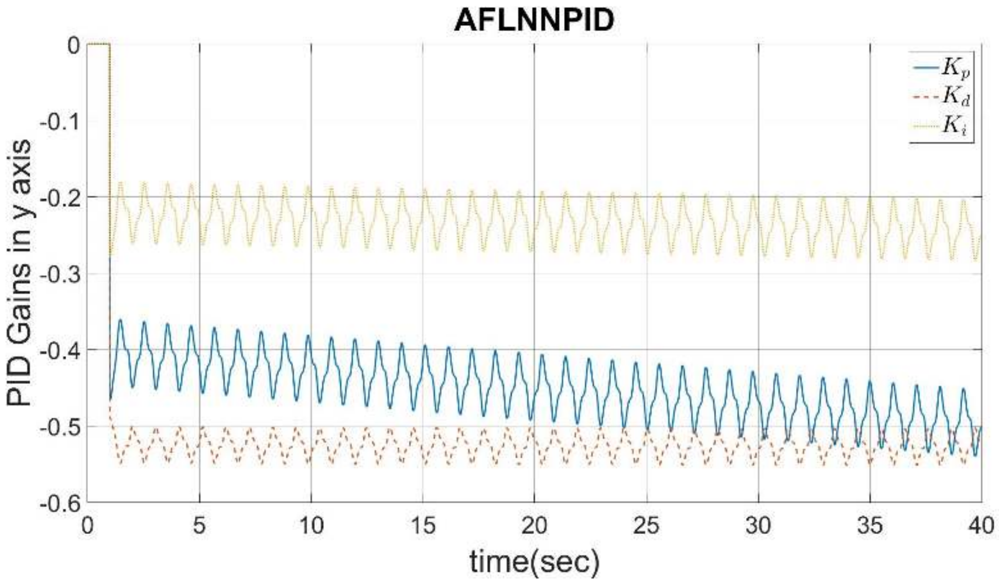

Figure D.5: Simulation on PID gains in $y$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

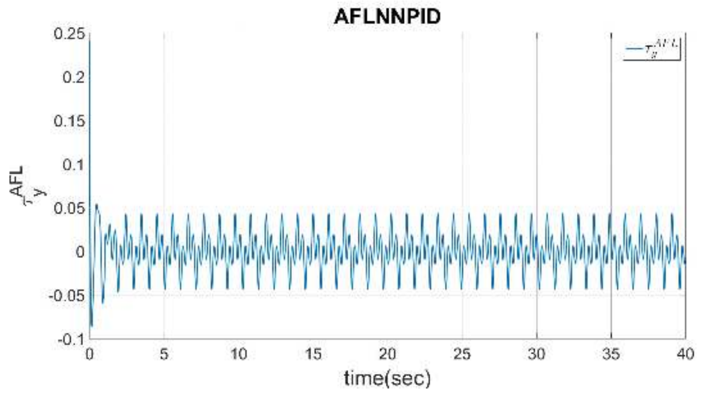

Figure D.6: Simulation on AFL control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

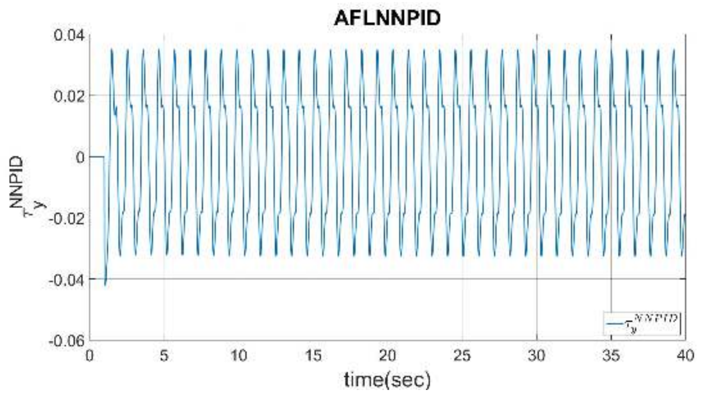

Figure D.7: Simulation on NNPID control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

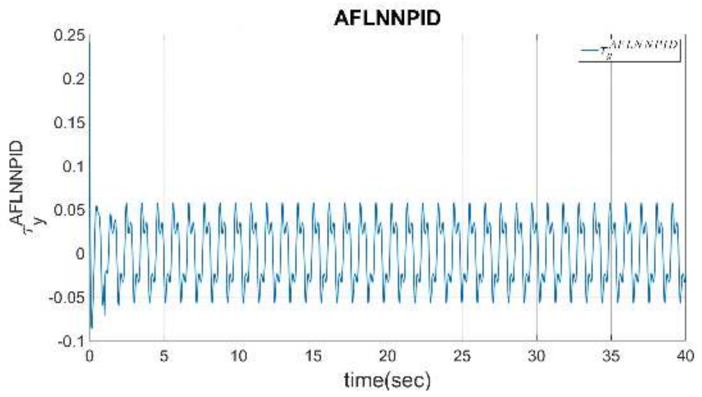

Figure D.8: Simulation on total control input (AFLNNPID) in $y$ axis for trajectory tracking ($\omega$ = 6.0rad/s and R = 0.06m)

## D.3 Experiment on Trajectory Tracking

The experimental results for the change in PID gains and control inputs are given in this section. The ball is supposed to follow a circular trajectory with radius of 0.08m and angular velocity of $\omega$ = 2rad/s.

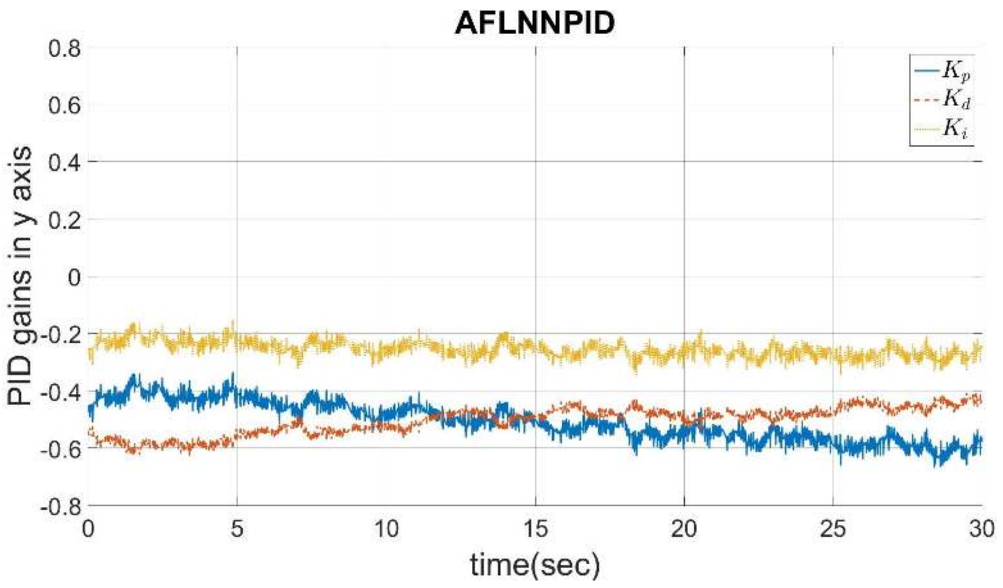

Figure D.9: Experiment on PID gains in $y$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

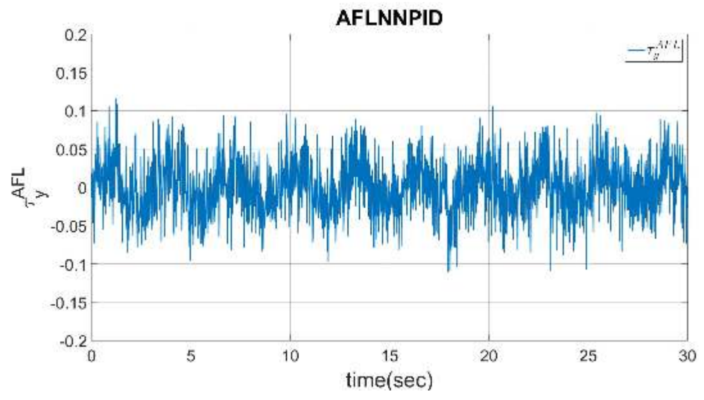

Figure D.10: Experiment on AFL control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

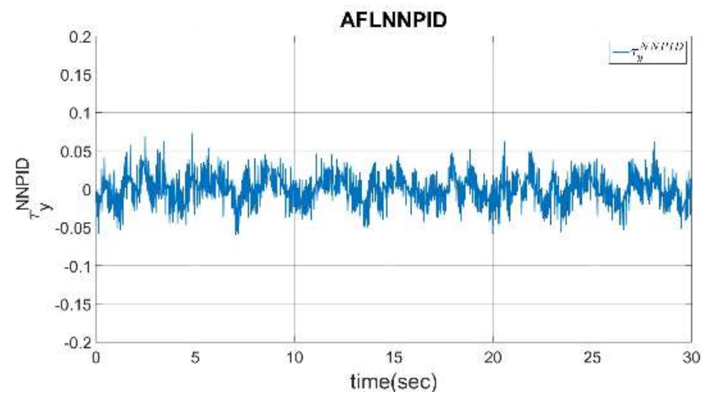

Figure D.11: Experiment on NNPID control input of AFLNNPID in $y$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)

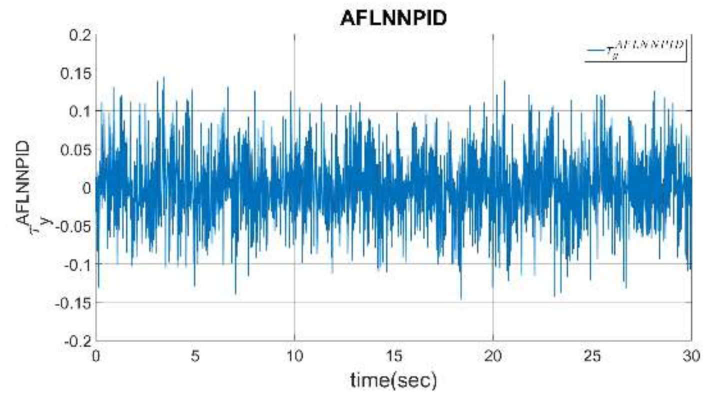

Figure D.12: Experiment on total control input (AFLNNPID) in $y$ axis for trajectory tracking ($\omega$ = 2.0rad/s and R = 0.08m)