Fall 2023

Online Aircraft System Identification Using a Novel Parameter Informed Reinforcement Learning Method

Nathan Schaff
Embry-Riddle Aeronautical University, schaffn@my.erau.edu

ONLINE AIRCRAFT SYSTEM IDENTIFICATION USING A NOVEL PARAMETER INFORMED REINFORCEMENT LEARNING METHOD

By

Nathan Michael Schaff

A Thesis Submitted to the Faculty of Embry-Riddle Aeronautical University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Aerospace Engineering

December 2023

Embry-Riddle Aeronautical University

Daytona Beach, Florida

ONLINE AIRCRAFT SYSTEM IDENTIFICATION USING A NOVEL

PARAMETER INFORMED REINFORCEMENT LEARNING METHOD

By

Nathan Michael Schaff

This Thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Richard Prazenica, Department of Aerospace Engineering, and has been approved by the members of the Thesis Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in the partial fulfillment of the requirements for the Degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE

Chair, Dr. Richard Prazenica	Member, Dr. Hever Moncayo
Member, Dr. K. Merve Dogan	
Graduate Program Coordinator, Dr. Daewon Kim	Date
Dean of the College of Engineering, Dr. James W. Gregory	Date
Associate Provost of Academic Support, Dr. Christopher Grant	Date

To my cats, Sean and Athena, who forced me to take breaks when I truly needed them

ACKNOWLEDGMENTS

Words cannot express the debt I owe to all those that have helped me through this thesis and its associated degree. I want to thank my incredible family and friends for giving me the love and confidence I needed to press through when I was sure there was not enough time left to finish this thesis. I have to thank my mother for leading by example both as a respected academic but also as a kind and loving person. To my father, who inspired me to become an engineer by taking me to every engineering museum in the area, wherever we were, I am forever grateful for the sense of fascination you gave me when looking at the world around me. I must also thank my sibling Chloe; without your encouragement, support, and curiosity I would not be where I am today. I must as well thank my incredible girlfriend who supported me through this thesis from beginning to end, through the ups and downs, and whose faith in me never wavered even when mine did.

Finally, I have to thank my advisor Dr Richard Prazenica who this thesis would not have been possible without. He allowed me to grow and learn independently when I could and gave me the support and guidance I needed when I could not. I also must thank Dr. K. Merve Dogan and Dr. Hever Moncayo, my committee members who provided valuable guidance and feedback.

ABSTRACT

This thesis presents the development and analysis of a novel method for training reinforcement learning neural networks for online aircraft system identification of multiple similar linear systems, such as all fixed wing aircraft. This approach, termed Parameter Informed Reinforcement Learning (PIRL), dictates that reinforcement learning neural networks should be trained using input and output trajectory/history data as is convention; however, the PIRL method also includes any known and relevant aircraft parameters, such as airspeed, altitude, center of gravity location and/or others. Through this, the PIRL Agent is better suited to identify novel/test-set aircraft.

First, the PIRL method is applied to mass-spring-damper systems with differing mass, spring constants, and damper constants. The reinforcement learning agent is trained using a random value for each constant within a fixed range. It is then tested over that same range as well as constants with a variation of three times the trained range. The effect of varying hyperparameters for the reinforcement learning agent was observed as well as the performance of the agent with added sensor noise and with reduced PIRL parameters. These initial studies show that PIRL is able to create accurate models within a short timeframe. They additionally demonstrate robustness to significant sensor noise.

Second, a linear fixed wing aircraft longitudinal flight model is used to evaluate the effectiveness of PIRL in the context of aircraft system identification. The reinforcement learning agent is provided with simulated flight test data generated using stability and control parameters obtained using the United States Air Force's Stability and Control Digital DATCOM. Nine aircraft are selected as training aircraft and one for testing. The agent is trained with each training episode comprising a randomly chosen aircraft from the set and its dynamics model is used to generate artificial online flight data. PIRL was evaluated with respect to its accuracy and speed of convergence and was found to generate models that are more accurate than those obtained using conventional reinforcement learning and extended Kalman filters.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

Figure **Page**

Figure **Page**

LIST OF TABLES

NOMENCLATURE

ω_0	MSD Natural Frequency
ζ	MSD Damping Ratio
δ_e	Change in Elevator Deflection (From Trim)
δ_t	Change in Throttle Input (From Trim)
θ	Pitch (From Trim)
θ_*	Longitudinal Pitch Trim Angle
A	System Matrix
A_T	Trial System Matrix
$Ahat$	Trial System Matrix (Label in Simulink)
B	Input Matrix
b	Damping Constant in MSD
B_T	Trial Input Matrix
$Bhat$	Trial Input Matrix (Label in Simulink)
C	Output Matrix
D	Feedthrough Matrix
dt	EKF Time Step Interval
E_N	Final Episode Number
F	Jacobian of A Matrix With Respect to X
g	Acceleration Due to Gravity

h	Altitude (From Trim)
I	Identity Matrix
K	Kalman Gain
k	EKF Time Step
k	Spring Constant in MSD
m	Mass of Mass in MSD
$M_{\delta e}$	Aircraft Input Derivative
M_q	Aircraft Stability Derivative
M_u	Aircraft Stability Derivative
M_w	Aircraft Stability Derivative
M_{PV}	Maximum Possible Value
P	Error Covariance Matrix
Q	Process Noise Covariance Matrix
q	Pitch Rate
R	Reward Value
R	Sensor Noise Covariance Matrix
RLO	Reinforcement Learning Agent Output
T_f	Episode Length/Final Time
T_s	Sample Time of RL Agent
U	Control Input Vector

u	Control Input Force in MSD
u	Horizontal Speed (From Trim) in LFM
u_*	Horizontal Trim Speed in LFM
U_{MV}	Unknown Matrix Value
w	Vertical Speed (From Trim) in LFM
w_*	Vertical Trim Speed in LFM
X	State Vector
x	Position of Mass in MSD
$X_{\delta e}$	Aircraft Input Derivative
$X_{\delta t}$	Aircraft Input Derivative
X_q	Aircraft Stability Derivative
X_u	Aircraft Stability Derivative
X_w	Aircraft Stability Derivative
Y	Output Vector
$Z_{\delta e}$	Aircraft Input Derivative
Z_q	Aircraft Stability Derivative
Z_u	Aircraft Stability Derivative
Z_w	Aircraft Stability Derivative
AI	Artificial Intelligence
CG	Center of Gravity

DDPG	Deep Deterministic Policy Gradient
EKF	Extended Kalman Filter
LFM	Longitudinal Flight Model
MSD	Mass-Spring-Damper
PINN	Physics Informed Neural Network
PIRL	Parameter Informed Reinforcement Learning
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RLS	Recursive Least Squares
SYSID	System Identification

1 Introduction

In the field of aviation, there exist scenarios in which generating a mathematical model of the aircraft quickly and accurately while the aircraft is currently flying is of significant value. This problem is known as Online Aircraft System Identification and its solutions have relevance in flight testing and fault tolerance. An aircraft being evaluated and tested must have its exact performance and abilities documented across a spectrum of flight conditions. This involves generating a number of environment-specific flight models, and the quicker these can be generated, the more time and cost efficient the flight testing is. If an aircraft were to suffer a fault of some kind, it may be of significant value to be able to generate a replacement model that more accurately represents the dynamics of the post-fault system. This problem of how to best generate quick and accurate flight models has motivated, in this thesis, the development of a novel implementation of reinforcement learning known as Parameter Informed Reinforcement Learning (PIRL)[1]. This algorithm differs from conventional reinforcement learning in that it augments the reinforcement learning agent's inputs with additional readily available relevant aircraft parameters, thereby enabling the agent to be applied to novel aircraft without retraining. It should be noted that while, theoretically this algorithm could be applied to online learning, this thesis concerns online application of the algorithm following offline training.

1.1 System Identification

System identification (SYSID) refers to the algorithms and methods developed to generate mathematical models for an unknown system based on information regarding the system's inputs and outputs. The field traces its origins to Lagrange's Harmonic analysis of planetary motion, wherein a more accurate understanding of planetary motion was sought [1]. SYSID plays a significant role in the aerospace industry, where specifically online identification of aircraft system parameters can reduce flight test costs as well as provide the ability to update the system model in response to changing flight conditions [2], fuel or payload expenditure [3], or failures [4].

[1]This is not to be confused with the related Physics Informed Reinforcement Learning

1.2 Reinforcement Learning

Reinforcement Learning (RL) refers to a subset of artificial intelligence (AI) and machine learning in which algorithms attempt to complete tasks and learn through rewards based on their performances. Development of RL can be traced to developments in optimal control as well as artificial intelligence, and the field was established as it is known today in the 1980s [5]. Reinforcement Learning functions by creating an Agent that receives input data relating to a given Environment. This Agent takes an action of some kind and is rewarded based on the effect of this action. This cycle can be seen in Figure 1.1. RL differs from supervised learning in that there is no correct answer for the Agent to find; rather, it must try different actions with the goal of maximizing its total reward. This makes RL a different type of artificial intelligence compared to most existing AI algorithms [5].

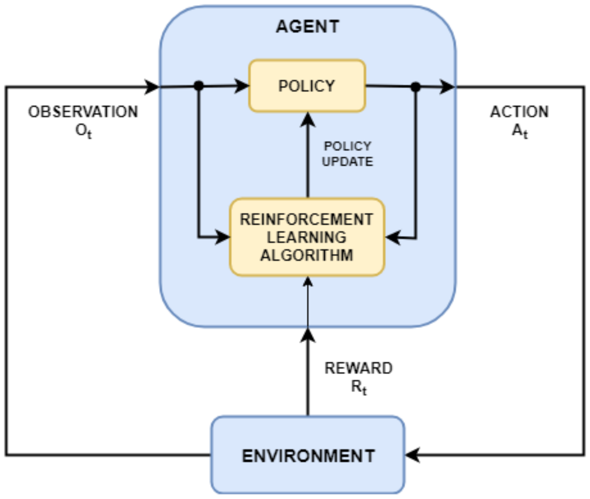


Figure 1.1 Reinforcement Learning Cycle [6]

1.3 Objectives and Methodology

The purpose of this thesis is to develop and investigate a novel implementation of Reinforcement Learning named Parameter Informed Reinforcement Learning (PIRL) and evaluate its effectiveness at rapid online SYSID. The PIRL algorithm is implemented and optimized using two different systems, a mass-spring-damper and a longitudinal flight model, and in each is tasked with determining system parameters. It is the purpose of this investigation to determine the effectiveness of PIRL and how it compares to other existing algorithms, namely, conventional reinforcement learning and extended Kalman filters.

1.4 Parameter Informed Reinforcement Learning (PIRL)

Parameter Informed Reinforcement Learning (PIRL) was developed from the hypothesis that an RL Agent trained for SYSID on a variety of systems that are similar to each other should be able to develop a general model of the parameters therein. This would allow an AI system to perform SYSID faster and more accurately than conventional methods. The performance of this system could theoretically then be improved if the AI was informed by any known system parameters that may be relevant. For example, knowledge of an aircraft's wingspan may allow the RL Agent to better identify parameters relating to control force and roll damping. This task can be solved using supervised as well as reinforcement learning; however, reinforcement learning was ultimately selected due to its ability to learn online.

1.5 Organization of Thesis

This thesis is organized into five sections, beginning with the introduction. Chapter Two provides a review of the literature related to this work. This includes existing SYSID methods as well as others' implementation of RL for SYSID. Chapter Three includes a detailed analysis of PIRL's effectiveness on a Mass-Spring-Damper system. This includes discussion of system dynamics, methods of RL implementation, effectiveness of the method, as well as an analysis of the effect of variations of hyperparameters. Chapter Four includes a similar discussion as Chapter Three, but the system used is a fixed wing longitudinal aircraft model. Chapter Five will summarize the conclusions of the results of PIRL applied to both systems as well as recommendations for future work.

2 Literature Review

The purpose of this chapter is to examine the current research into system identification methods and specifically reinforcement learning for the purpose of system identification.

2.1 State of System Identification

The concept of system identification refers to the creation of mathematical models of dynamical systems using the inputs and observed outputs of a given system [7]. The motivation for these methods is that, while mathematical models of dynamical systems can often be theoretically obtained through an understanding of the underlying physical relationships, this is often difficult for complex systems such as the dynamics of an aircraft, and as a result, these models often have a degree of error. An aircraft's dynamics are determined by the aircraft geometry, loading and aerodynamic environment, all of which make the task of developing simple mathematical relationships between the vehicle and its dynamics without test data difficult, if not impossible.

While only formally established in the 1950s, using data to create models of motion has been in practice since the 1770s when Lagrange sought models of planetary motion [1]. System identification can be performed with a number of different methods including the Extended Kalman Filter [8], Least Mean Square Methods [9], Model Reference Methods [10], Frequency-Based Methods [11], and Reinforcement Learning (RL) [12–15]. Each method possesses advantages and certain disadvantages over its counterparts.

This thesis concerns the development of algorithms that apply to a variety of similar systems and are able to perform system identification to a satisfactory extent on this collection of systems. For example, such an algorithm could be applied on all fixed wing aircraft, given their similar dynamics, instead of one aircraft alone. In the process of conducting this literature review, to the best of the author's knowledge, no research concerning the development or implementation of these algorithms was found in terms of performing SYSID on multiple related systems. Thus, this literature review concerns the application of system identification algorithms to single systems.

2.1.1 Extended Kalman Filter

One common method for performing SYSID is the Extended Kalman Filter (EKF). The EKF was originally designed for the state estimation of nonlinear systems, but it can be extended to system identification by expanding the dynamics of the system to include the unknown parameters as states themselves. These parameter states are then propagated via the EKF algorithm and the system is progressively identified. While the extension from a Kalman Filter to an Extended Kalman Filter no longer guarantees optimal state estimation, EKFs have been shown to be effective for system identification by a number of authors. For example, Bauer and Andrisani used the EKF to identify a short period aircraft system [8]. Nonomura, Shibata, and Takaki used an EKF both to denoise the data as well as perform system identification [16]. Abas et al. found the EKF to be a computationally effective and cost effective solution for identifying a quadcopter UAV model [17]. Extended Kalman Filters have some limitations in their implementations. Basappa and Jategaonkar found that, when using an EKF to estimate aircraft longitudinal and lateral dynamics, the EKF performs poorly with imperfect tuning information, specifically with regard to its noise covariances [18].

2.1.2 Recursive Least Squares Filter

Another common method for system identification is applying the Recursive Least Squares (RLS) Filter. RLS Filtering works by minimizing a weighted linear least squares cost function. The RLS Filter has been shown to be effective in performing system identification in a number of different environments. Choi et al. found that a linearized RLS was effective in identifying road-friction coefficients quickly and accurately [19]. RLS Filters can be modified or combined with other methods to suit specific applications. For instance, they can be modified and applied to identifying nonlinear systems. In their 2017 paper, Ding et al. reported that their implementation of an RLS Filter was effective at identifying nonlinear industrial systems and that their proposed implementation of the algorithm was more accurate than the generalized RLS method [20]. Additionally, the RLS method can be combined with a Kalman Filter to increase its accuracy as was done by De Souza et al. in their 2021 paper [21]. They found that using a Kalman Filter to determine the regressor matrix and parameter vector improved its performance over a conventional RLS . RLS methods can also be combined with artificial intelligence as was used by Li et al. when they combined RLS and neural networks to better refine vehicle-trajectory prediction to support autonomous vehicle operation [22].

RLS methods have been shown to be accurate in aerospace applications. Kamali, Pashilkar, and Raol, found in their 2006 study, that their implementation of RLS was accurate and, in contrast to the Kalman filter, did not require tuning. Additionally, their RLS algorithm was able to identify longitudinal and lateral parameters without calibrated angle of attack or sideslip angle measurements, making their implementation fault-tolerant [23].

2.1.3 Frequency Domain System Identification

While methods such as the Extended Kalman Filter typically are implemented with time domain data, system identification is often performed using frequency domain methods. Guo and Kareem found that frequency domain methods were effective in modeling the changing behavior of bridges and buildings when subjected to different environments [24]. In the aerospace field, frequency domain methods have been used to estimate the flight models of various aircraft. Morelli compared the use of frequency domain methods to other time domain methods and found that the frequency methods had improved prediction capability when applied to the Tu-144LL Supersonic Transport Aircraft [25]. Morelli's later 2020 paper *Practical Aspects of Frequency-Domain Approaches for Aircraft System Identification* demonstrated how using frequency domain solutions can reduce the dimensionality of the data analyzed and allow for simpler noise rejection and online parameter identification [26].

2.2 Reinforcement Learning

Reinforcement Learning is a method of problem solving derived from trial and error. It traces its roots to the fields of optimal control as well as artificial intelligence, and reinforcement learning was established as it is known today in the late 1980s [5]. RL works by having an 'Agent' that interacts with a given environment through actions. Based on the outcome of these actions, a reward of some kind is given to the Agent. This reward is defined by a mathematical function, and the RL Agent seeks to maximize this by performing better and better actions. Through this process, the RL Agent learns which actions, or series of actions, produce the highest reward and thus can solve complex problems. This makes RL a different type of artificial intelligence compared to most existing AI algorithms, and makes it uniquely suited to problems where a 'correct answer' cannot be easily defined [5].

Reinforcement learning, due to its general applicability, has seen new developments, especially in recent years. The type of RL Agent implemented in this thesis, Deep Deterministic Policy Gradient (DDPG), was created by Lilicrap et al. in 2015 [27]. This type of RL Agent allows for a continuous action space. Without this, RL algorithms would instead select their actions based on a number of discrete choices. This is sufficient for a number of common AI problems and allows reinforcement learning to be applied to classification or recommendation problems but it is unsuitable for system identification and other continuous problems.

DDPGs have since been applied to a number of different fields, including energy management [28], stock-trading/finance [29], path-planning [30], and system identification [31]. They have been shown to be accurate, robust to environmental disturbances, and to perform well with real-world systems and data.

2.2.1 Use of Reinforcement Learning in the Aerospace Industry

While not as ubiquitous as conventional supervised learning, reinforcement learning has been implemented in a number of aerospace industry applications. In Aoun's 2021 thesis, he demonstrated that reinforcement learning could be applied to the problem of fault-tolerance in quadcopter platforms [32]. The RL Agent was able to reject disturbances successfully if applied to a stable or stabilized environment. Reinforcement Learning has also been applied as a higher level control law selector and optimal launch time estimator in target-missile-defender problems. Shalumov demonstrated that reinforcement learning methods were shown to be close to optimal in this application based on the selected cost function [33]. Reinforcement Learning was shown in this paper to be especially effective against more difficult missiles that changed their control law to avoid interception. When combined with online applicability, this made reinforcement learning a useful asset for missile interception.

2.2.2 Use of Reinforcement Learning in System Identification

Reinforcement learning in general has often been used for system identification with much success; however, much of this research focuses exclusively on using the input and output histories of the system [12, 34]. Less research, if any, has been conducted on using RL for SYSID where the agent is informed by the system inputs and outputs as well as any parameters known to influence the dynamics of the system. In a mass-spring-damper, this could be the mass, spring constant, and/or damping constants. In an aircraft's short period (pitch) mode, this could be the Center of Gravity (CG) expressed in Per Cent Mean Aerodynamic Chord (MAC), or the aircraft's mass moment of inertia about the Y axis. These parameters help determine the dynamics of the system, and thus it is hypothesized that this Parameter Informed Reinforcement Learning (PIRL) method will allow the agent to rapidly create an accurate system model. Furthermore, no research appears to have been conducted regarding the use of AI or RL to a broad class of similar systems.

3 Environment 1: Mass-Spring-Damper

The Mass-Spring-Damper (MSD) is a physical representation of a linear damped oscillator. Given this fact, it is analogous to a number of simple one degree of freedom systems. For this reason, it has been chosen as the first system in which PIRL will be trialed. A simple system such as this is used to allow for the development and optimization of PIRL. While the algorithm is explicitly defined, many of the RL Agent's hyperparameters must be tuned for ideal results.

3.1 System Model

The physical MSD system can be seen in Figure 3.1.

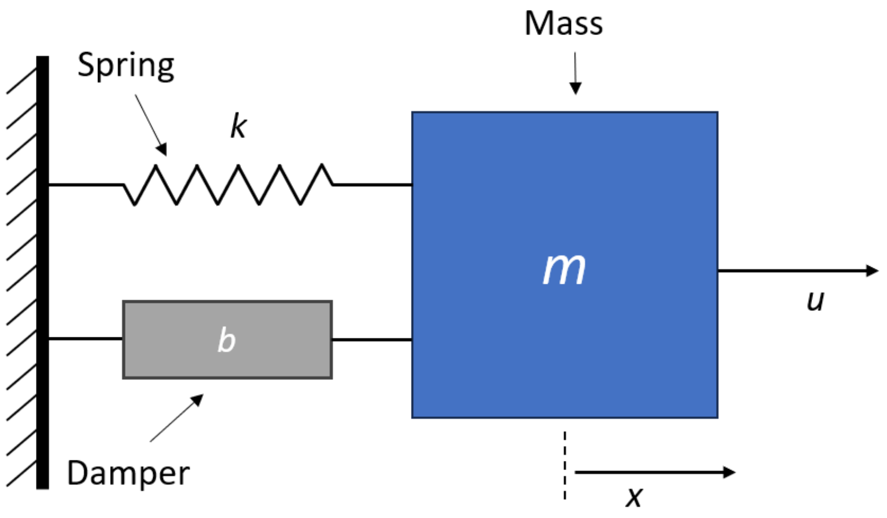


Figure 3.1 Mass-Spring-Damper

The associated dynamics of the mass-spring-damper system are given in Equation 3.1:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = u \tag{3.1}$$

where x is the position of the mass (in meters), k refers to the spring constant (in newtons/meter), b refers to the damping constant (in newton-seconds/meter), m refers to the mass (in kilograms), and u refers to the control input force (in Newtons). It should be noted that this does not refer to a specific system but rather the family of systems created by altering the mass, spring constant, and damper constant. Equation 3.1 can be written in state-space form in Equations 3.2 and 3.3:

$$\dot{X} = AX + BU \tag{3.2}$$

$$Y = CX + DU \tag{3.3}$$

where X is the state vector composed of the position and velocity, U refers to the control input vector, and Y refers to the measured output vector, which in this case corresponds to the position and velocity. A, B, C, and D are arbitrary matrices that determine the specific system. For this experiment, they are defined as follows in Equation 3.4:

$$A = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 1/m \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; D = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.4}$$

The RL Agent is tasked with finding the A and B matrices and does this by generating trial A and B matrices with specific values assigned as unknowns to be estimated. This implementation makes the PIRL Agent a physics-informed neural network. The trial system is defined in Equation 3.5:

$$A_T = \begin{bmatrix} 0 & 1 \\ A21 & A22 \end{bmatrix}, \; B_T = \begin{bmatrix} 0 \\ B21 \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; D = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.5}$$

where A21, A22, and B21 are unknowns in the system dynamics.

The three parameters of the true system (*m*, *k*, *b*) are chosen at random within a predetermined range. PIRL is based on the idea that an RL Agent can take an 'informed guess' at the parameters of the system as it has seen similar ones in the past. To represent this concept, each value is assigned as a random number between 0.9 and 1.1. These values remain consistent through each training episode and change between episodes. The effect this range has on the A and B matrices can be seen in Table 3.1. It should be noted that minimum and maximum values are with respect to their magnitudes. The effect of the parameter range on the system's natural frequency and damping ratio is shown in Table 3.2. These values were calculated with Equations 3.6 and 3.7.

Table 3.1 Effect of Parameter Range on A and B Matrices

Matrix Entry	Minimum Possible Value	Maximum Possible Value	Minimum Possible Per Cent Change	Maximum Possible Per Cent Change
A21	-0.8182	-1.2222	-18.18%	+22.22%
A22	-0.8182	-1.2222	-18.18%	+22.22%
B21	0.9091	1.1111	-9.09%	+11.11%

Table 3.2 Effect of Parameter Range on Natural Frequency and Damping Ratio

	Natural Frequency	Damping Ratio
No Variation Value	1	0.5
Minimum Possible Value	0.9045	0.4091
Maximum Possible Value	1.1055	0.6111
Minimum Possible Per Cent Change	-9.55%	-18.18%
Maximum Possible Per Cent Change	+10.55%	+22.22%

$$\omega_0 = \sqrt{\frac{k}{m}} \tag{3.6}$$

$$\zeta = \frac{b}{2\sqrt{mk}} \tag{3.7}$$

3.2 Implementation of Reinforcement Learning

The RL Agent's performance is impacted by a variety of factors including its learning method, architecture, input structure, output interpretation, and a wide range of tunable hyperparameters.

3.2.1 RL Agent Type (DDPG)

The RL Agent was chosen to be a Deep Deterministic Policy Gradient (DDPG). A DDPG is a model-free, actor-critic RL method. DDPGs differ from other learning methods in that they offer continuous outputs as opposed to discrete. This is ideal for SYSID wherein the outputs are continuous by design. DDPG networks also support continuous or discrete observation spaces, which is well suited to the task of time-series based SYSID.

3.2.2 Network Architecture

The Network Architecture is important to the effectiveness of the Agent. If the architecture is too simplistic/too small, the network may not have the required complexity to accurately create a policy function. If it is too complicated/too large, then the network may suffer from exceptionally long training times and overly burdensome computational loads. This would make any practical online implementation of the network difficult. With these two factors in mind, the critic network was constructed with a variety of fully connected layers of 20 nodes each with Rectified Linear Unit (ReLU) layers in between. The actor network was simpler with only a 3 node fully connected layer and a hyperbolic tangent layer. This structure was chosen as it was sufficiently complex for the task of system identification and the ReLU layers in particular would support noise-robustness.

3.2.3 Implementation of PIRL

Parameter Informed Reinforcement Learning is implemented within the RL Agent's observations. The chosen parameters that are deemed relevant to the system model and can be obtained easily are provided as additional inputs to the RL Agent at each time-step. For the MSD, the parameters have been chosen to be m, k, and b, or mass, spring-constant, and damper-constant. Given the system model in Equation 3.4, these parameters are sufficient to determine the system model and thus represent a best-case-scenario for the effectiveness of PIRL. It should be noted that any practical scenarios will not have such directly relevant information. For this reason, tests of PIRL's effectiveness with some variables missing are also conducted, as well as tests without PIRL altogether to test if PIRL is a contributing factor to the SYSID performance or just the RL Agent.

3.2.4 RL Agent Inputs

The RL Agent observes a state history (position and velocity of the mass) of the last second in 0.1 second intervals. It also observes an error history of the last second in 0.1 second intervals. The error history is calculated by propagating the last state through the trial matrices to create a trial current state. The error is the difference between this trial state and the true, measured state.

It should be noted that the trial system states are not propagated through the trial matrices again to generate the next error value. The current measured state is used instead. Otherwise, this would effectively be assigning the RL Agent the task of SYSID as well as to drive the system by altering its parameters to the true system if its guesses have caused it to deviate. This is outside the scope of SYSID. The error calculation method is illustrated in Figure 3.2.

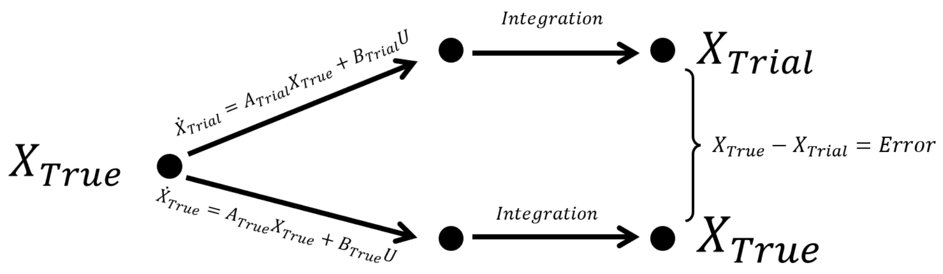

X_{True}
$\dot{X}_{Trial} = A_{Trial}X_{True} + B_{Trial}U$
$\dot{X}_{True} = A_{True}X_{True} + B_{True}U$
Integration
Integration
X_{Trial}
$X_{True} - X_{Trial} = Error$
X_{True}

Figure 3.2 Error Calculation Method

It should be noted that in a real system, state histories may not be obtainable and one must use output histories. In this work, it is assumed that the states are directly measured as outputs and thus the C matrices are set to be identity for the true and trial systems.

Finally, the PIRL inputs are added into the observations along with the control input. The RL Agent's Observations can be seen in Figure 3.3.

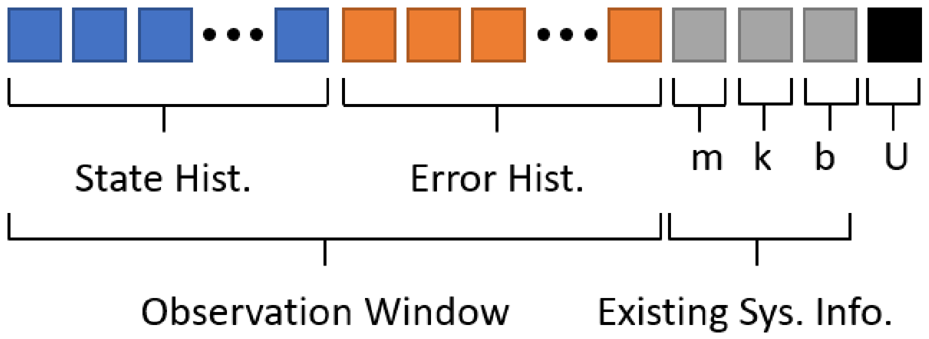


Figure 3.3 RL Agent's Observation

3.2.5 RL Agent Outputs

The RL Agent has a 3 x 1 array output, where the first term corresponds to A21, the second to A22, and the third to B21, as seen in Equation 3.5. This is then constructed in full A and B trial matrices, which are used to create the new observation and error observation windows. In this manner, the PIRL Agent takes the form of a physics-informed neural network (PINN). PINNs are neural networks that are able to leverage existing knowledge of physical systems, typically through some form of differential equation that the neural network is encoded with [35]. By making the output of the PIRL Agent values in a state-space dynamics representation, it is a PINN.

3.2.6 Reward Function

The Reward Function is the negative of the sum of the absolute values of the differences between the trial and true matrices. The specific reward calculation subsystem in the Simulink file can be seen in Figure 3.4. The Figure shows a Simulink implementation of Equation 3.8 where the summed differences of the unknown system values and the estimated values comprise the reward function signal. Note that values of the trial matrices are denoted in the Simulink with 'hat'. This reward was chosen as it penalizes any error between the true system and the trial system.

$$Reward = -(|B_{21} - B_{21T}| + |A_{21} - A_{21T}| + |A_{22} - A_{22T}|) \tag{3.8}$$

Consideration was made to a variety of other reward functions, such as to penalize the cumulative sum of the error between the trial system states and the true system states, or alternatively to include a time penalty to reward faster convergence to the true value. These were ultimately not implemented. For the first option, while this reward function carries the benefit that it theoretically allows for completely online continuous learning, it also has the drawback of not directly penalizing differences in the trial and true system. This could lead to the RL Agent outputting similar system dynamics but with very different state-space representations due to the lack of a uniqueness of this state-space representation. It also would mean that the RL Agent would not have direct feedback. This would likely slow the learning process. The second option was ultimately not implemented as there was already an existing penalty for inaccuracies in the current reward function; therefore, this addition would not be worth the added complexity in the system.

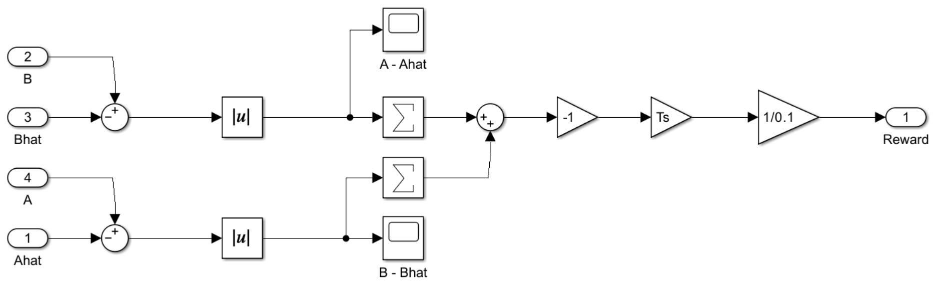


Figure 3.4 RL Agent's Reward Function

3.2.7 Other Hyperparameters

All other hyperparameters used to create the network have been listed in Table 3.3. Unless otherwise stated, all RL Agents were trained and implemented using these hyperparameters. Sample time, which refers to the rate at which the RL Agent is activated, was chosen to be 0.1 seconds as this was not computationally expensive and allowed for 500 feedback iterations every episode. The final time, which controls the duration of each episode, was chosen as 50 seconds as this would theoretically allow for any transient responses to decay fully. The target smooth factor, mini batch size, experience buffer length, critic learn rate and critic gradient threshold, actor learn rate and actor gradient threshold were all left at their default values for DDPGs. These values are all related to very specific tuning of the DDPG learning process and this degree of tuning was not ultimately required. Discount factor refers to how the RL Agent is rewarded during the episode. It affects whether the RL Agent will prioritize immediate rewards or long-term rewards. Noise variance and noise decay rate are tunable hyperparameters that affect the exploration versus exploitation of the RL Agent. They influence the noise that is added to RL Agent's output which ensures that the RL Agent explores different options. The maximum episode number is a hyperparameter that controls the maximum number of episodes the RL Agent will be trained with before training is completed.

Table 3.3 Hyperparameters used for RL Agent Training

Parameter	Purpose	Value
Sample Time	Rate at Which the Agent is Activated	0.1s
Final Time	Total Time for Each Training Episode	50s
Target Smooth Factor	Related to Overfitting	1e-3
Discount Factor	Priority for Immediate Rewards	0.95
Mini Batch Size	Related to Training	64
Experience Buffer Length	Related to Training	1e6
Noise Variance	Related to Exploration Vs. Exploitation	0.3
Noise Variance Decay Rate	Related to Exploration Vs. Exploitation	1e-5
Critic Learn Rate	Learning Rate for Critic Network	1e-3
Critic Gradient Threshold	Related to Learning Rate	1
Actor Learn Rate	Learning Rate for Actor Network	1e-3
Actor Gradient Threshold	Related to Learning Rate	1
Max Episodes	Maximum Training Episodes	8000

3.2.8 Simulink Implementation

The Simulink model used to implement PIRL on the MSD environment is shown in Figures 3.5 to 3.11. It should be noted that 'hat' denotes the trial systems and associated signals. Figure 3.5 shows the Simulink diagram in its entirety. It can be seen in the upper left that the random values for m, k, and b are generated, vertically concatenated, and sent to the observation window from the Unknown Parameter Extractor. This connection is the key distinction of PIRL compared to other RL systems. The A and B matrices formed from the sampled m, k, and b values are sent to the State Space Model as well as the Reward Function. In the State Space Model, they are used to simulate the true system, and in the Reward Function, they are compared to the RL Agent's trial matrices to create the reward.

The State Space Model receives the sampled A and B matrices, the control input from the Exciter, and the trial A and B matrices and outputs the Y and Yhat signals that correspond to the observed outputs from the true system and the trial system. These signals are then differenced to create an error signal. The error signal as well as the true state output are inputted into a multiplexer (Mux) port, which is then an input for the observation window Subsystem. This process creates the state and error data that the observation window Subsystem converts into state and error histories.

The Exciter Subsystem generates the control input. It is composed of the superposition of three sine waves. The three sinusoids are used to excite the system to generate sufficient outputs for identification. The control input signal 'U' is additionally used as an input to the observation window.

The RL Agent block can be seen in the bottom right of the figure. It receives as inputs the Obs signal from the Observation Window Subsystem, the Reward signal from the Reward Function Subsystem as well as the isdone signal from a timer. The isdone signal determines if the RL Agent has irrecoverably deviated from its intended purpose during an episode and that the episode should therefore be ended early. This behavior was not observed during any training and thus the isdone signal is used as a timer to end the simulation after 50 seconds.

The RL Agent outputs the A21, A22, and B21 values, which are demultiplexed and used as inputs for the Generating Trial Matrices Subsystem. These trial matrices are then connected to the State Space Model to generate the trial observation: Yhat. Finally, the Reward Function Subsystem receives the A, B, Ahat and Bhat signals as its inputs and from these generates the reward signal that is provided as an input to the RL Agent

Figure 3.6 shows the Unknown Parameter Extractor Subsystem. On the left is the x0 variable which is reset each episode with the mass, spring constant, and damping constant assigned as a random sample within the predetermined ranges. This variable is assigned through Simulink's Reinforcement Learning Environment's Reset Function. Each value is then extracted and assigned to an output that will be used in the Observation Window Subsystem. The necessary operations to convert the constants into the true values of A21, A22, and B21 are then executed and the matrices are concatenated with relevant predefined constants to create the true A and B matrices. These are then assigned to an output for use in the Reward Function and State Space Subsystem.

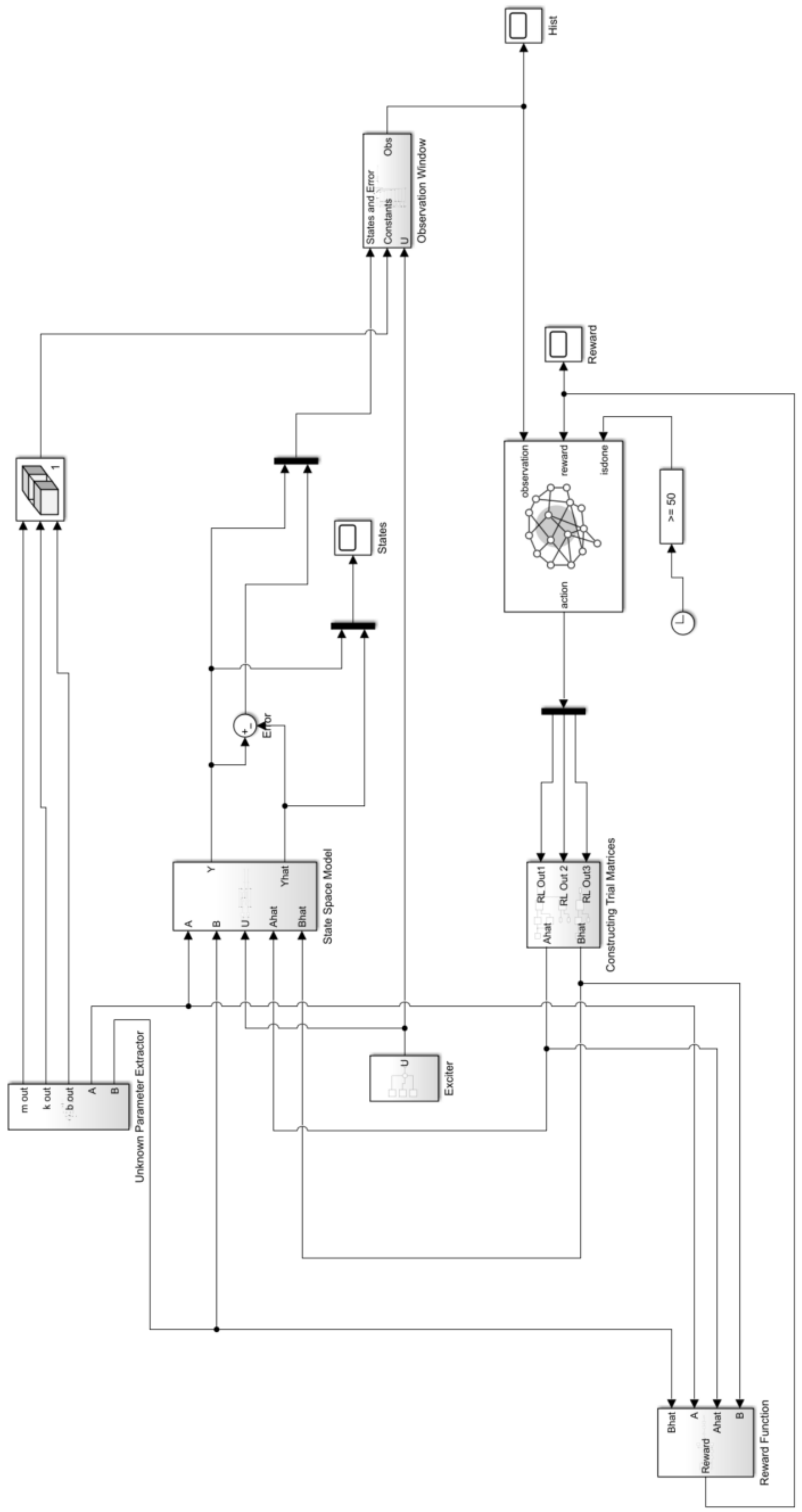


Figure 3.5 Overall Implementation of PIRL

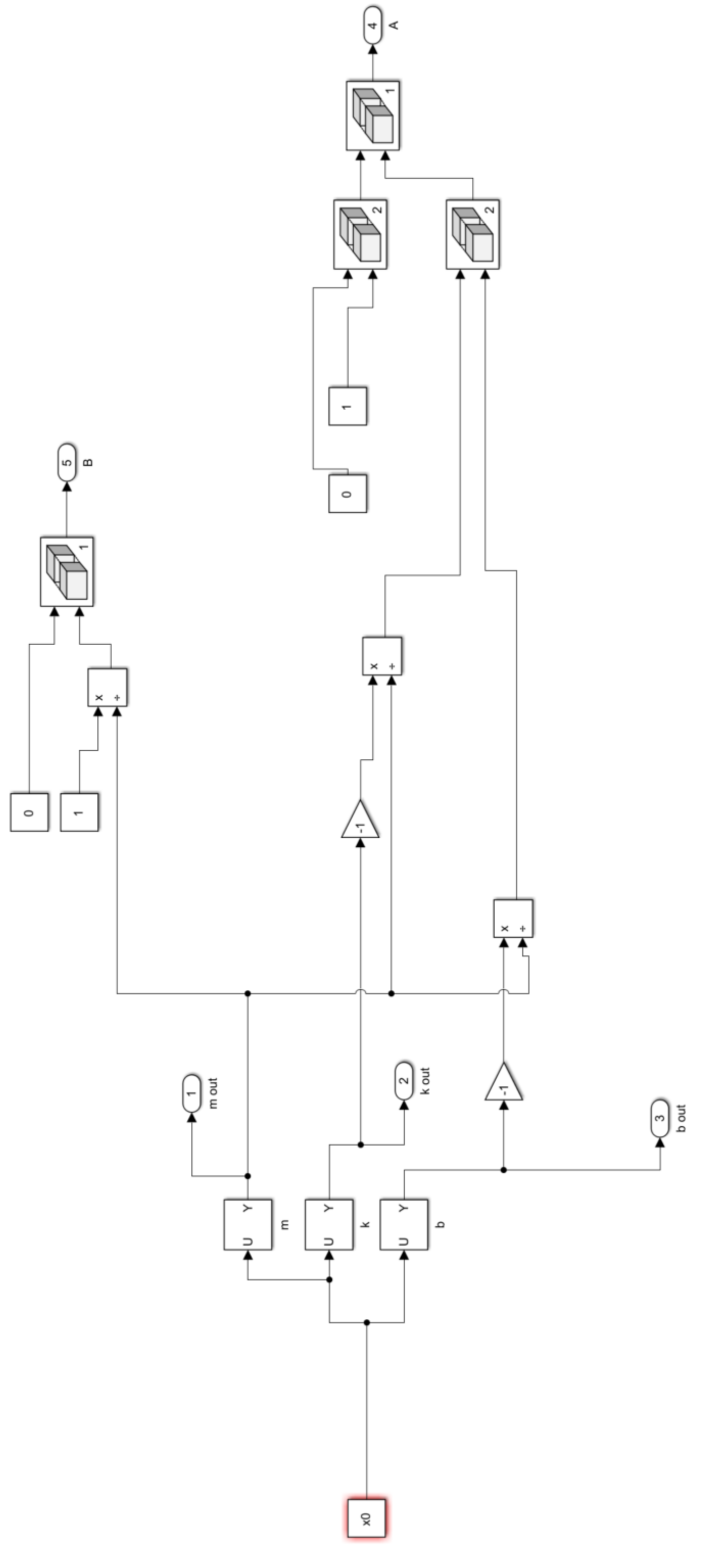


Figure 3.6 Unknown Parameter Extractor Subsystem

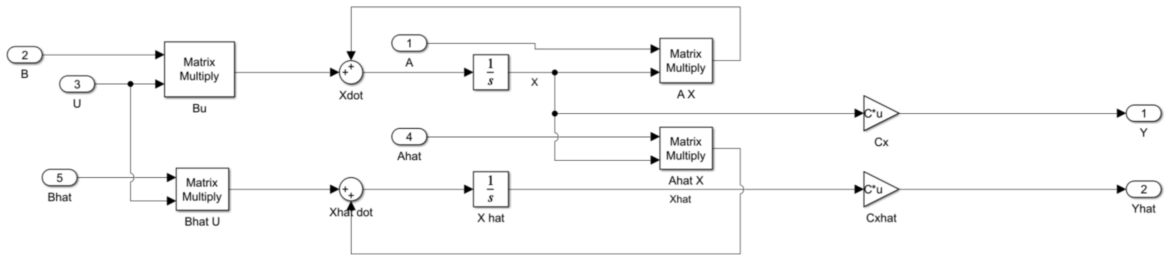


Figure 3.7 State Space Model Subsystem

The State Space Subsystem can be seen in Figure 3.7. It receives as inputs the A, B, Ahat, Bhat and U signals. These are then applied as is conventional for a state space model following Equations 3.2 and 3.3. The Y and Yhat signals are then assigned to outputs to be used by the Observation Window Subsystem

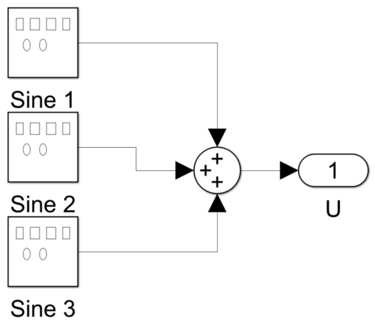


Figure 3.8 Exciter Subsystem

The excitation function is shown in Figure 3.8. The Exciter Subsystem generates the control inputs that excite the system with the frequencies necessary to allow the RL Agent to identify the unknown system matrices. Three sine functions have been chosen for the excitation function and can be seen to be summed and assigned an output. Each sine function has an amplitude of 1 with frequencies of 1, 2, and 3 Hz respectively. The control input is provided as an equation in Equation 3.9

$$U = sin(2\pi t) + sin(4\pi t) + sin(6\pi t) \tag{3.9}$$

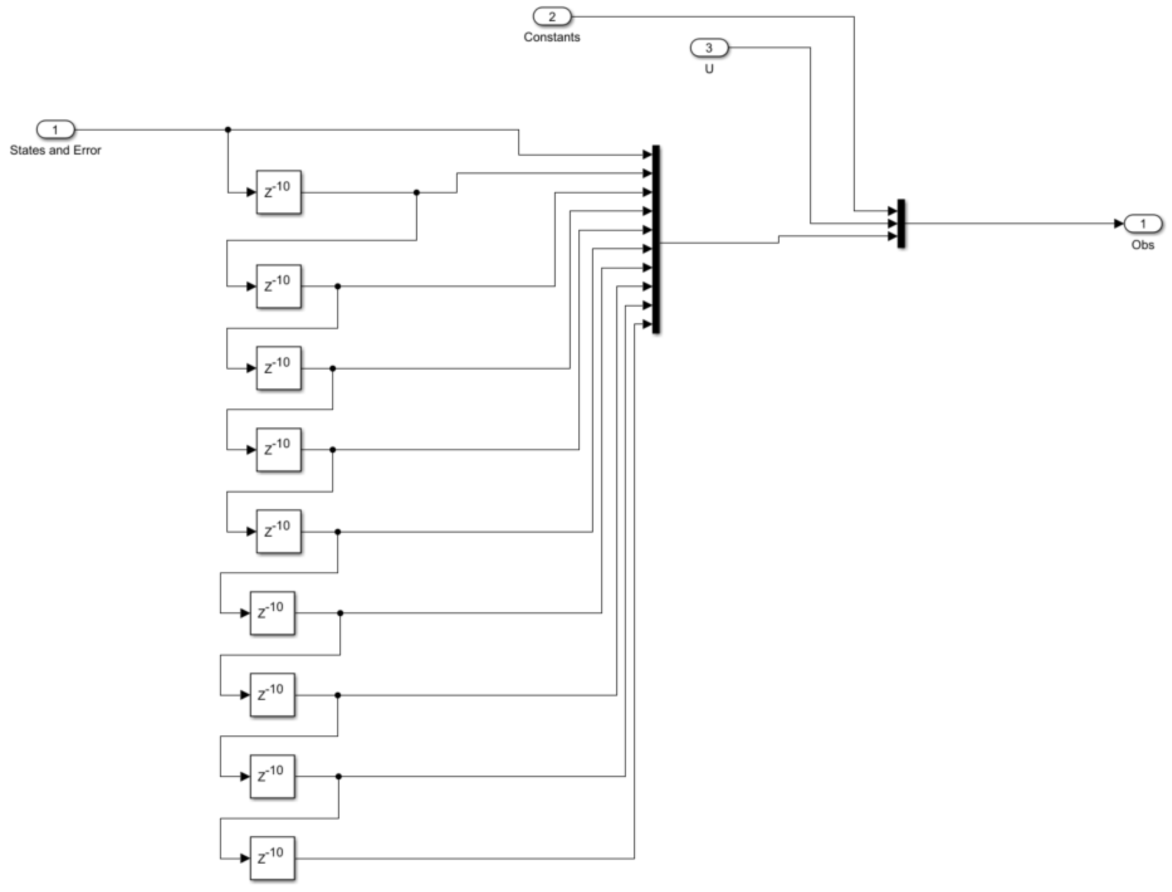


Figure 3.9 Observation Window Subsystem

Figure 3.9 shows the Observation Window Subsystem. This subsystem receives the parameters of PIRL, the control input, and state and error data as its inputs. The parameters and control inputs are multiplexed into the observation window as is, while the state and error data are propagated through a series of delay blocks. This creates a window of observation of the last second in 0.1 second intervals. The observation window is multiplexed with the parameters and control input and assigned to the Obs output. This output signal is then connected to the RL Agent.

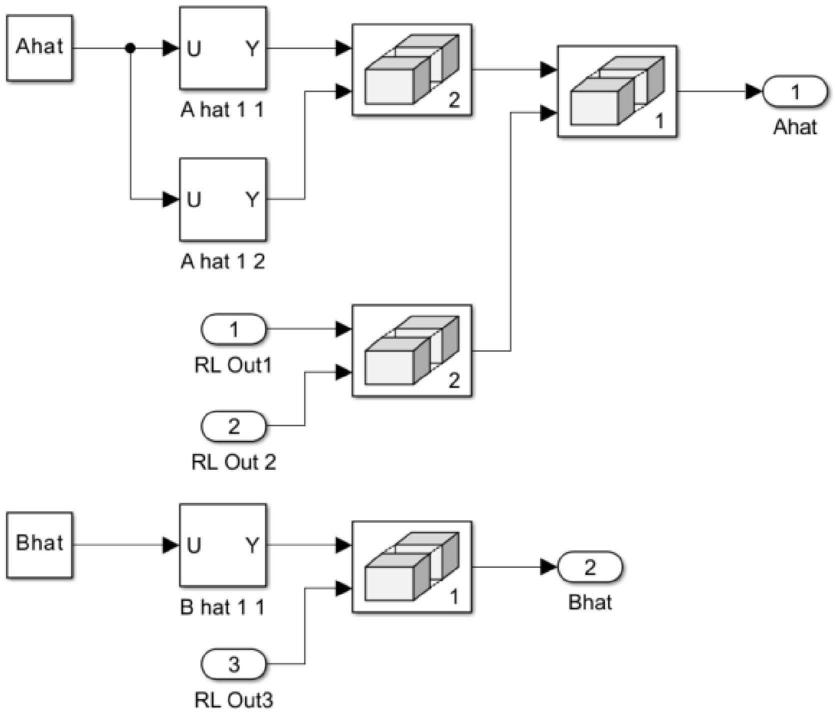


Figure 3.10 Constructing Trial Matrices Subsystem

The Constructing Trial Matrices Subsystem can be seen in Figure 3.10. The Subsystem has 3 inputs, all of which are the demultiplexed outputs from the RL Agent. To combine these outputs into the trial matrices, the RL Agent's first two outputs are concatenated vertically to create the bottom row of the A matrix while the top row is created by extracting it from the Ahat matrix that is stored in the program's memory. Bhat's top row follows a similar process. This initial Ahat and Bhat values can be seen in Equation 3.10. An alternative to this method would be to create the Ahat and Bhat matrices as concatenations of the RL Agent outputs and constants; however, the method shown was chosen for its ability to be easily applied to different, more complex, systems.

$$Ahat = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \ Bhat = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.10}$$

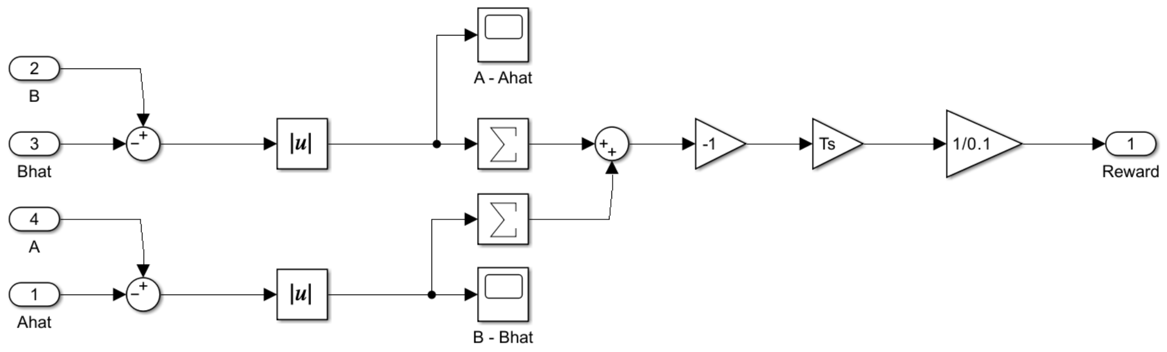


Figure 3.11 Reward Function Subsystem

The Reward Function Subsystem is shown in Figure 3.11. In this subsystem, it can be seen that the A and Ahat matrices, as well as the B and Bhat matrices, are differenced and the absolute value of these differences are squared and summed to create a total error. RL Agents require a reward function to increase in value with improvement so this sum signal is multiplied by a gain of negative one to ensure that increases in accuracy lead to a higher value. This signal is then fed through an integrator and differentiator block to prevent an algebraic loop from occurring. A memory cell or delay block would also serve this purpose.

3.2.9 Training and Testing Method

The quality of the results found in a test are limited by the quality of the test conducted. There are many ways to train and test an RL Agent and great care must be given to selecting an appropriate testing method or the effect that one seeks may not be captured. For many tests, the RL Agents were trained to reach an Episode Reward of -50 and then saved for later evaluation. This is termed the -50 Test. The test was effective for observing how the RL Agents learned and what variables impacted their learning rates; however, this test was ultimately unable to show the effect of changes to hyperparameters and other variables on RL Agent performance as the -50 threshold places a maximum limit on performance. This means that, even though changing a variable may have an impact on the maximum possible performance of an RL Agent, this effect will not be observed in the results. If changing a variable reduces performance significantly, it may not reach the -50 threshold altogether, and thus will give a crude understanding of the effect of the variable. This is illustrated in Figure 3.12.

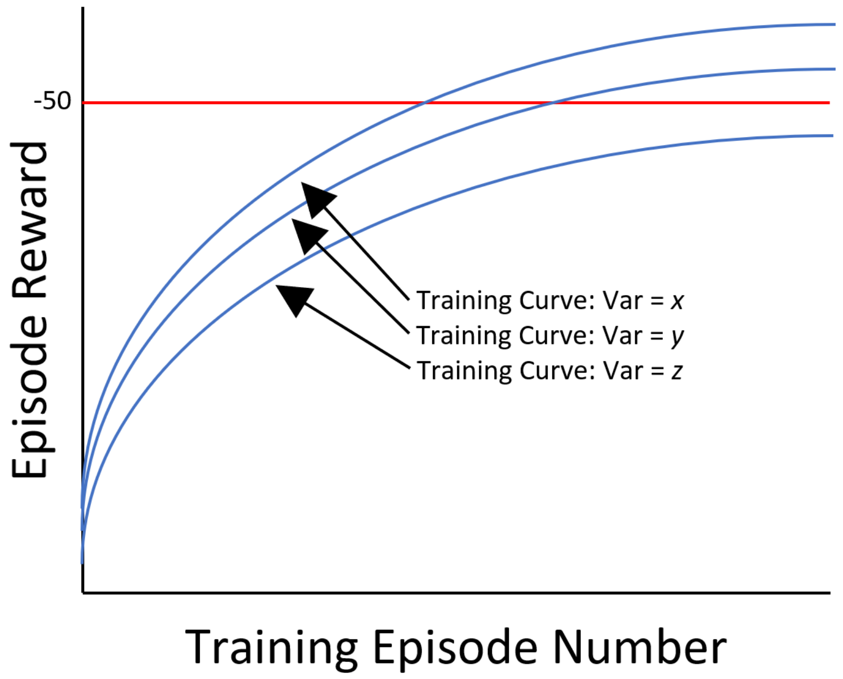


Figure 3.12 Training Curve Threshold and its Impact on Performance

For this reason, rather than applying a reward threshold, RL Agents were trained with a fixed number of episodes and the Agent that had the greatest average reward was used for testing. The number of episodes required for this was set to 8000 as this allowed all Agents to reach their maximums without training being ended prematurely. This test was found to be useful in observing the impact to performance that resulted from changes in a specific variable had.

For testing the RL Agents, the absolute value average error of each term in the A and B matrices was summed to create an average error per 50 second episode. The RL Agent was then provided with 100 randomly selected systems with parameters within the trained range and the average error from these systems was recorded. This was then repeated for 100 randomly selected systems with parameters selected from a range that was triple that of the original training data. This was done to observe if the RL Agent maintained effectiveness when presented with new data that were far outside the range of the training data.

3.3 Results

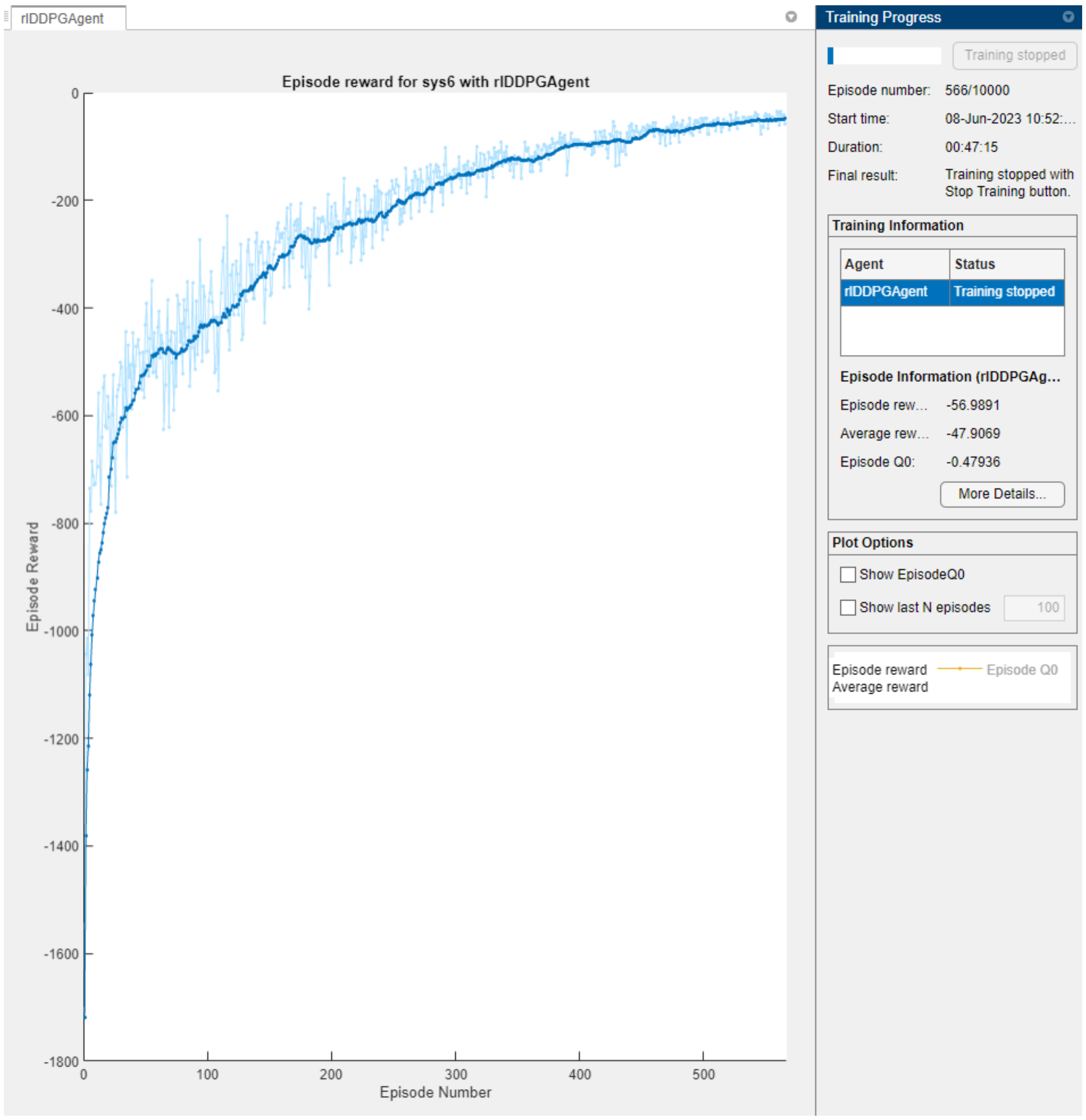
The results of the PIRL implementation for the mass-spring-damper environment are included in the following section. They are divided into three categories: Comparison to Existing Methods, Effect of Hyperparameters, and Effect of Environmental Variables. Performance Comparison of PIRL to Existing Systems provides insight into the performance of PIRL as a technology and how its performance compares to existing methods. Effect of Hyperparameters depicts how alterations to the parameters that affect RL Agent training affect the performance of PIRL. Finally, Effect of Environmental Variables shows the robustness of the RL Agent to changes in its environment.

3.3.1 Performance Comparison of PIRL to Existing Systems

This section concerns the comparison of PIRL to existing SYSID algorithms. Specifically, PIRL is analyzed in detail and its performance and effectiveness evaluated, and then this performance is compared to that of conventional reinforcement learning and an extended Kalman filter.

Performance of PIRL

The training curve of the PIRL Agent can be seen in Figure 3.13. The RL Agent rapidly identifies methods of improvement and follows them. The average curve follows a smooth logarithmic increase indicative of diminished returns; however, the network explores a wide range in the solution space as is evident by the larger variations in the episode reward curve.



Episode number: 566/10000
Start time: 08-Jun-2023 10:52:...
Duration: 00:47:15
Final result: Training stopped with Stop Training button.

Training Information

Agent	Status
rlDDPGAgent	Training stopped

Episode Information (rlDDPGAg...

Episode rew...	-56.9891
Average rew...	-47.9069
Episode Q0:	-0.47936

Plot Options

☐ Show EpisodeQ0

☐ Show last N episodes 100

Figure 3.13 PIRL - Training Curve

Performance on a Random System

The RL Agent was trained on an MSD with the mass, spring constant, and damping constant all being randomly assigned within fixed ranges. To first understand the effectiveness of PIRL on an MSD, the trained RL Agent was used to identify a random system within the unknown constant ranges over which it was originally trained. The results of this test can be seen in Figures 3.14 to 3.17. The sampled constants are given in Table 3.4 along with the Per Cent Variation of the sample with respect to the maximum possible deviation. The mass is roughly located within the middle of the sampled system range while the spring constant and damping constant are further to the maximum and minimum ranges respectively. The resulting A and B matrices are provided in Equation 3.11

Table 3.4 Sample Constants for MSD

Parameter	Value	Per Cent Variation
Mass (m)	0.9937	-6.3%
Spring Constant (k)	1.0824	82.4%
Damper Constant (b)	0.9208	-79.2%

$$A_{True} = \begin{bmatrix} 0 & 1 \\ -1.0893 & -0.9266 \end{bmatrix}, \; B_{True} = \begin{bmatrix} 0 \\ 1.0063 \end{bmatrix} \tag{3.11}$$

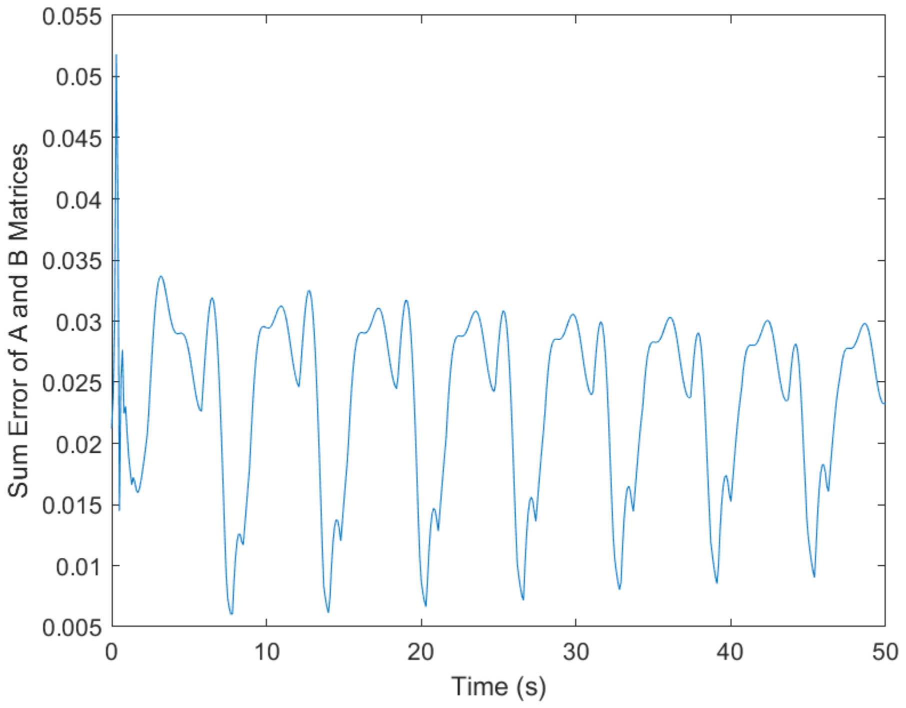


Figure 3.14 PIRL - Random Sample - Sum of Errors of A21, A22, and B21

Figure 3.14 shows the total summation of the errors of each unknown. From this Figure, it is evident that the accuracy of the RL agent exhibits a brief transient response that lasts roughly 1-2 seconds and then reaches a oscillatory pattern that decays gradually. The transient section of this curve reaches a maximum total error of 0.0518 and the oscillatory section has a maximum error of roughly 0.03 and a minimum error of approximately 0.007. It should be noted that this is not Per Cent error. The results indicate that when applied to the randomly selected system, the PIRL Agent does not seem to exhibit a significant convergence time and may converge to the true value if provided enough time. This is not fully demonstrated, though, as the error diminishes with time but does not fully reach zero.

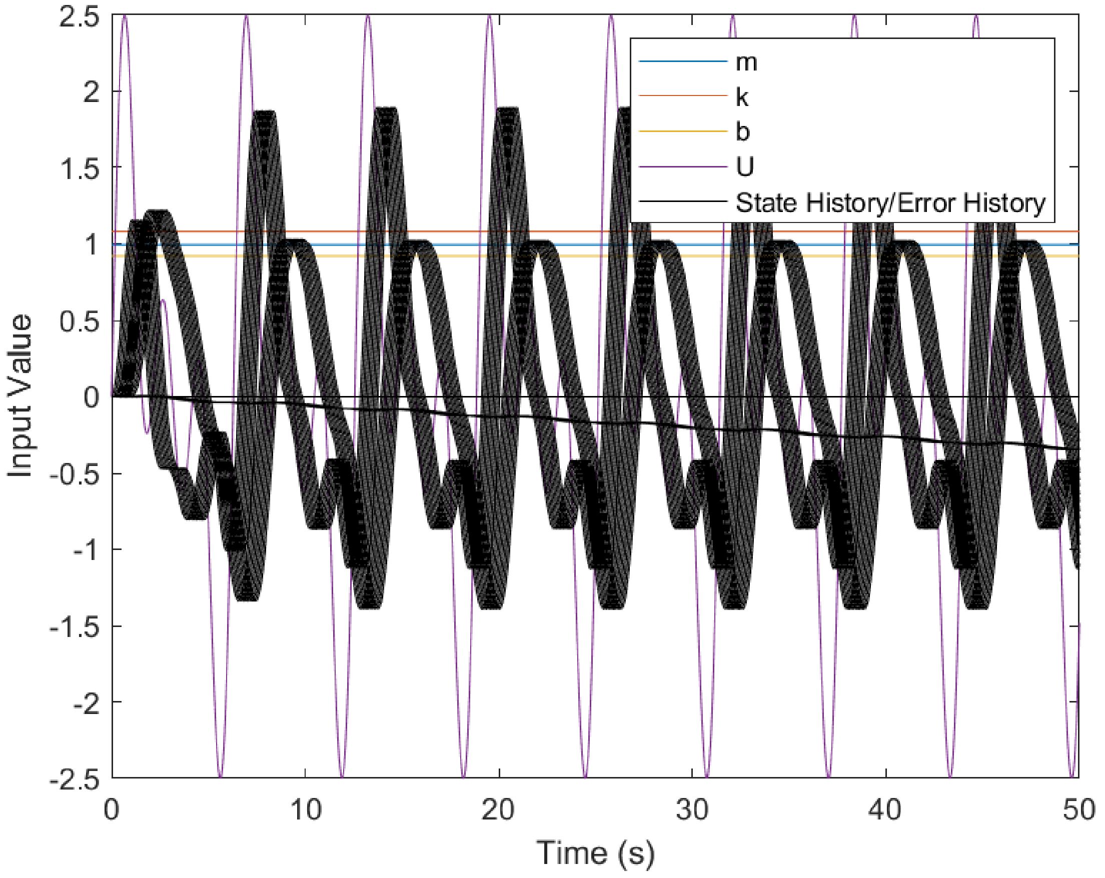


Figure 3.15 PIRL - Random Sample - Input Signals to RL Agent

The input signal histories to the RL Agent can be seen in Figure 3.15. The state and error histories are shown in black. This ribbon-like appearance is due to the observation window, wherein the multiple past samples are plotted in the same color. The larger oscillating ribbons represent the state history and the thicker ribbon that slowly decreases in value is the error. It can be seen that the error between the trial and true system does increase in magnitude slightly. This may be a sign of a possible divergence. It is of note that the system model consistently gets more accurate as evident in Figure 3.14; however, the ratio of these parameters with respect to each other are similarly important to an accurate system model and it is possible that the RL Agent may be converging on a more accurate model in the sense of the A and B matrices being close to the truth but in a manner that leads to a temporary greater error in the state histories. This can be seen in Figure 3.16 where the X2 trial state diverges slowly from the true state. It is possible that the RL Agent may slowly diverge from the system model with time; however this cannot be confirmed without further experimentation.

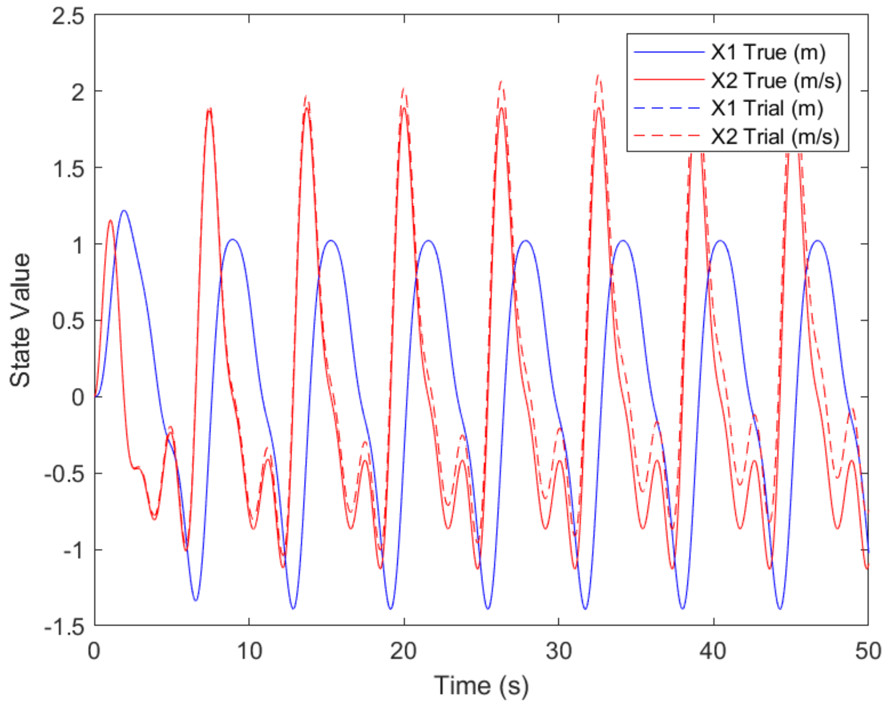


Figure 3.16 PIRL - Random Sample - History of True States and RL Agent Trial States

In Figure 3.16 one can see the time histories of the system's two states. It can be seen that the X1 trial state appears to be identical to the X1 true state. This is not due to the RL Agent's accuracy and is in fact due to the specific nature of the system as well as how the RL Agent has been implemented. The state-space form of an MSD will always have the exact same upper row of the A and B matrices. This means that, regardless of the A21, A22, and B21 values, if an MSD is initialized at a given position and velocity, the derivative of the initial position will always be defined exclusively by the initial velocity. The RL Agent receives as inputs the state history of the true system as well as the error between the true and trial systems. This error is not propagated through time, and at each time step, the trial system is initialized at the true system's current state. This is done to avoid divergence of the system during training. In addition, if the Agent is tasked with both system identification and error propagation through time, it would be required that the RL Agent drive the system to an identical state history by affecting the A and B matrices. For these reasons, at each time step, the true and trial systems have the same starting X vector and thus the X1 curve in both systems is defined through the experiment by the X2 curve of the true system.

Figure 3.17 presents the percentage error in the A21, A22 and B21 values estimated by the PIRL Agent. Similar to the trends seen in Figure 3.14, there exists a roughly 1 second long transient behavior from the RL Agent but it soon shifts to an oscillatory response in all unknowns. Errors for all unknowns remain below 3% consistently, and for B21 the error remains below 0.8% after the transient period. The A21 error does increase slightly during the 50 second episode. It is unclear why this occurs although it may be due to there being no explicit reward/penalty for convergence to one specific value or the lack thereof.

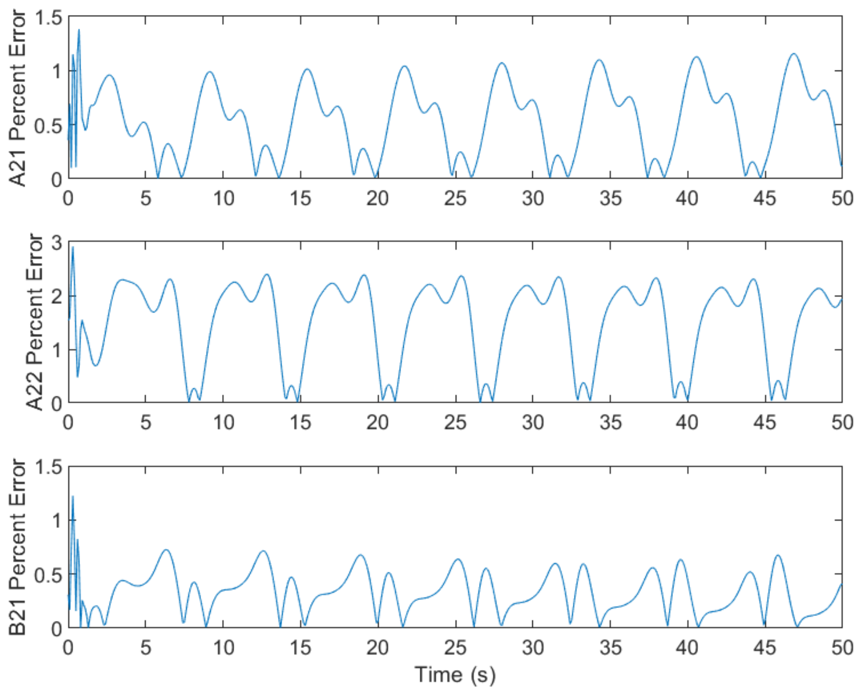


Figure 3.17 PIRL - Random Sample - Percentage Error of Trial and True Values in System Model

Performance of PIRL within Trained Ranges

In order to better understand the performance of the PIRL Agent it was implemented on 100 different randomized systems that were within its trained ranges. In this test, the Agent has not been exposed to these specific systems; however, it should be capable of identifying their parameters through its training. The results of this test can be seen in Table 3.5. Given the minimal transient response and extended periodic response, the error's respective time histories have been averaged. It is evident that the PIRL Agent is effectively identifying the systems with all errors below 3%. Note that the average total error is the average error of A21, A22, and B21 across each 50 second window, across all 100 samples, and then finally averaged together.

Table 3.5 PIRL Agent Performance of 100 Systems Within Training Range

Parameter	Per Cent Error
A21	0.8763%
A22	2.6953%
B21	1.3978%
Average Total Error	1.6565%

Effectiveness of PIRL Agent at Maximum Training Values

The PIRL Agent is trained with random samples within a fixed range. To better understand the performance of the system, it was implemented and evaluated with the unknown quantities being maximized. The results of this test can be seen in Figures 3.18 to 3.21. In this test, the mass, spring constant, and damping constant are all set to 1.1.

Table 3.6 PIRL RL Agent Performance at Maximum Parameter Values

Parameter	Per Cent Error
A21	0.4872%
A22	0.5198%
B21	9.3122%
Average Total Error	3.4397%

The average error from this test can be seen in Table 3.6. It is evident that, while the errors of A21 and A22 decrease, the B21 error increases sufficiently that the trial system is ultimately less accurate at this upper bound. Figure 3.18 illustrates that the error, much like the random sample system, exhibits a transient response and then an oscillatory response. The oscillatory pattern differs between the tests but appears to have a similar period. This pattern is likely due to the sinusoidal excitation inputs. The upper limit system, while exhibiting this similarity in pattern, does so at a greater magnitude of error. The maximum of the upper limit system is more than double that of the random sample system. The oscillatory pattern of the error is also roughly 3 times greater in magnitude in the upper boundary than the random sample. This supports the conclusion that the RL Agent may lose accuracy as unknown system constants depart from their central values of 1.

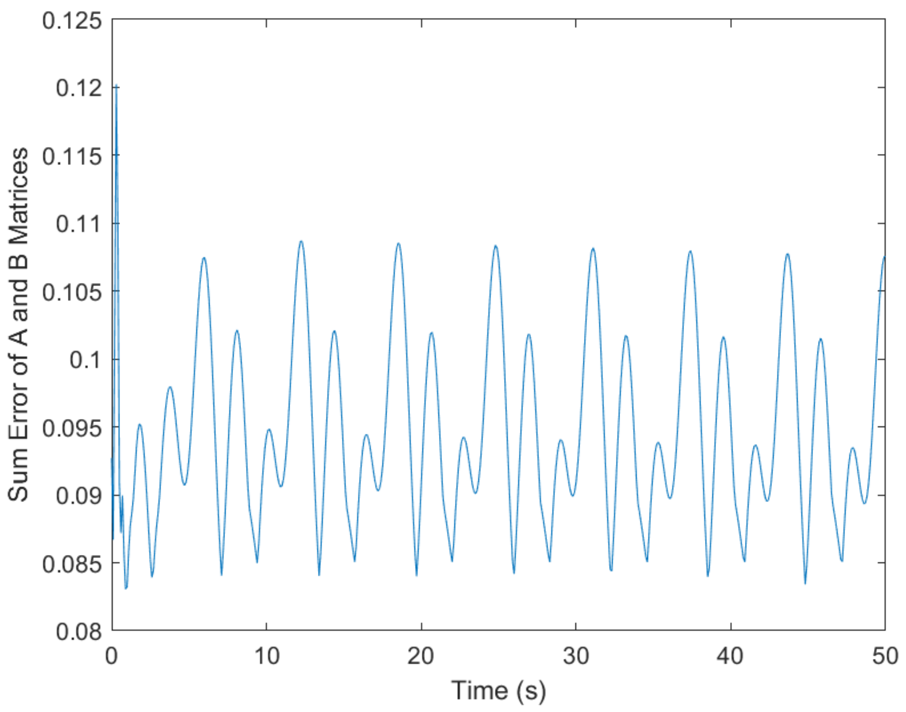


Figure 3.18 PIRL - Upper Limit - Sum of Errors of A21, A22, and B21

Figure 3.19 shows the RL Agent's inputs and it can be seen that state histories are roughly similar to those of the random system, which is to be expected given that the systems, while different in their physical parameters, are still fundamentally similar. In Figure 3.20 the state histories are shown and it is clear that the X2 error is much greater than that obtained for the random system and, while the trial states follows a similar trajectory to the true states, it has notably more error than in the random sample system.

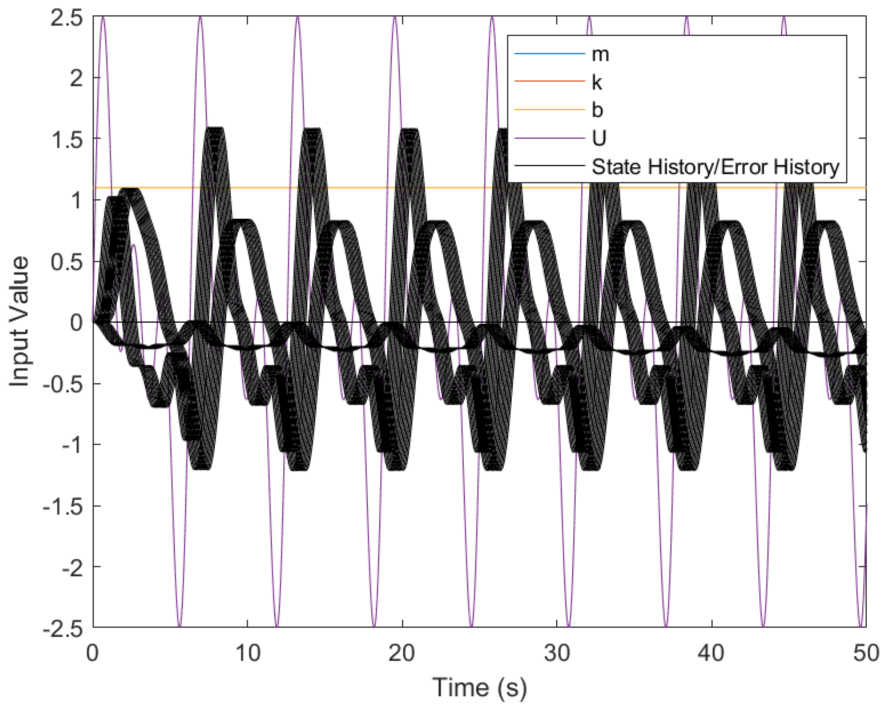


Figure 3.19 PIRL - Upper Limit - Input Signals to RL Agent

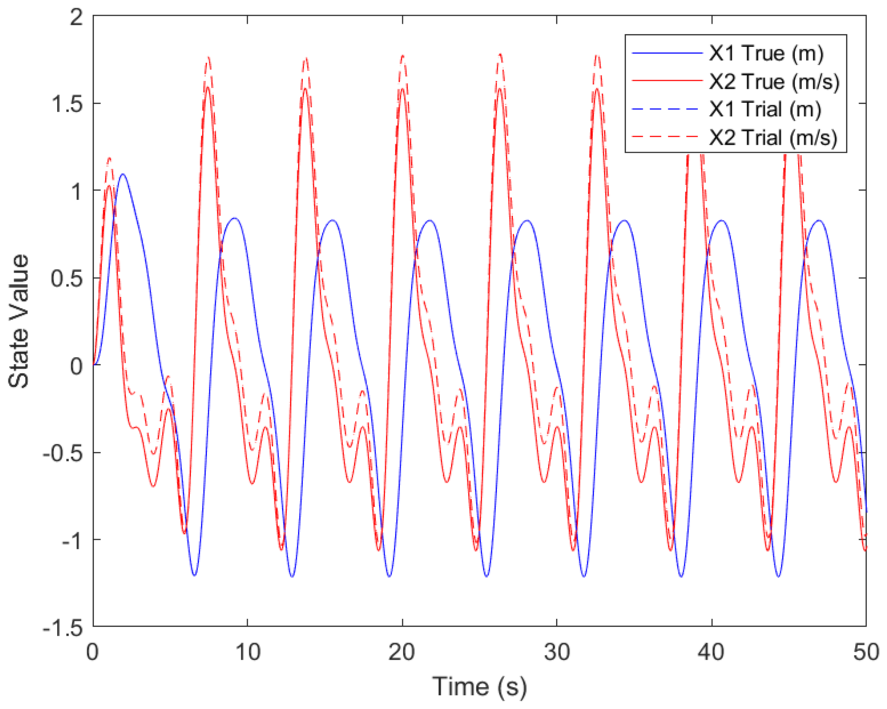


Figure 3.20 PIRL - Upper Limit - State History of True States and RL Agent Trial States

The percentage error histories are shown in Figure 3.21. The A21 and A22 errors are roughly similar between the upper limit and the random sample and are somewhat smaller in magnitude; however, the B21 error differs considerably and reaches a maximum error of roughly 10%.

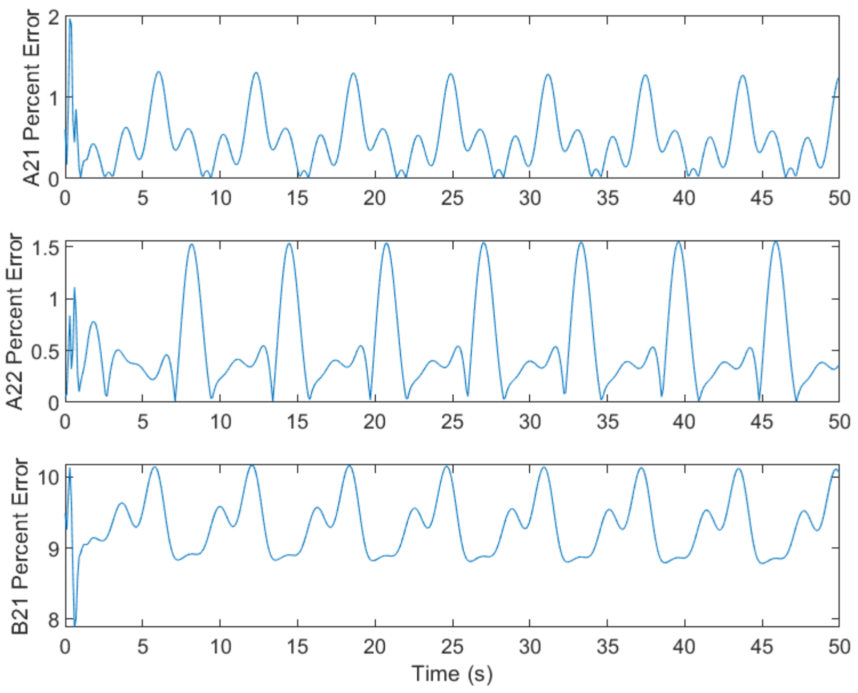


Figure 3.21 PIRL - Upper Limit - Percentage Error of Trial and True Values in System Model

Effectiveness of PIRL Agent at Minimum Training Values

Similarly to the upper limit test, the PIRL Agent was implemented on the system with all unknowns set to the lower limit. The results of this test are provided in Table 3.7. The effect of the minimum limit system is similar to that of the maximum system with slight changes in error from A21 and A22, but the predominant loss in accuracy coming from B21. The results of this test can be seen in Figures 3.22 to 3.25. It is evident from these Figures, that much like the previous tests, the average error of the PIRL Agent remains low; however, the B21 error specifically does increase and thus causes an increase in the summation error curve, the error history curve, and the difference between the true and trial states curves.

Table 3.7 PIRL Agent Performance at Minimum Parameter Values

Parameter	Per Cent Error
A21	0.3584%
A22	1.3657%
B21	8.9190%
Average Total Error	3.5477%

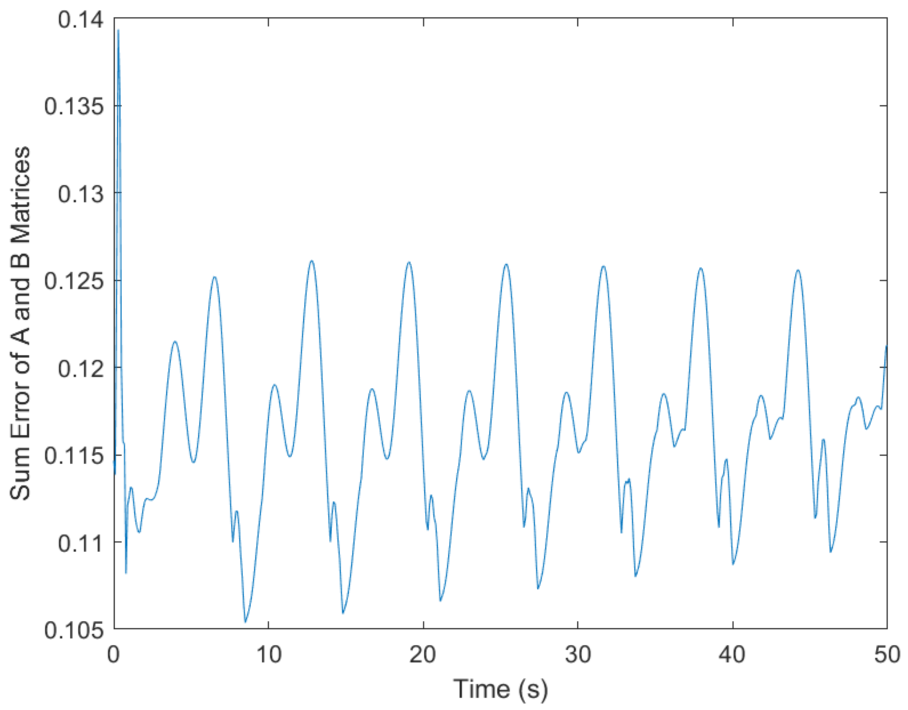


Figure 3.22 PIRL - Lower Limit - Sum of Errors of A21, A22, and B21

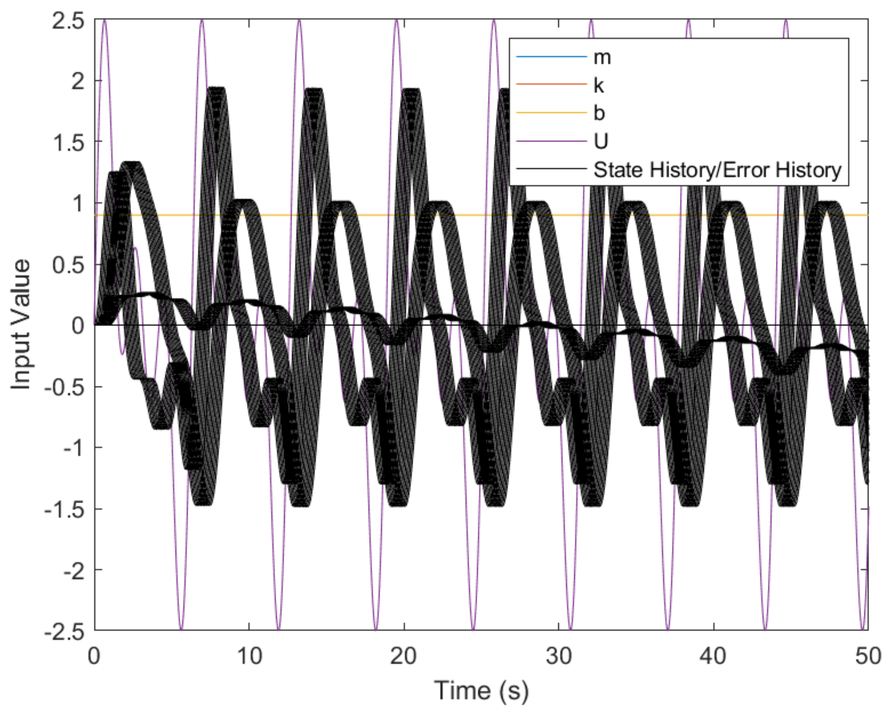


Figure 3.23 PIRL - Lower Limit - Input Signals to RL Agent

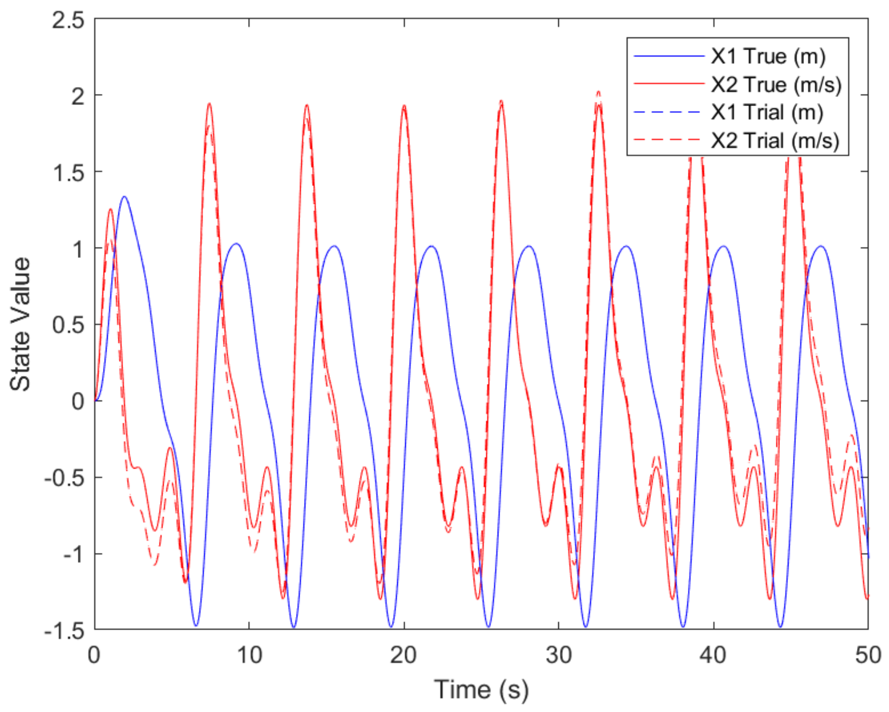


Figure 3.24 PIRL - Lower Limit - State History of True States and RL Agent Trial States

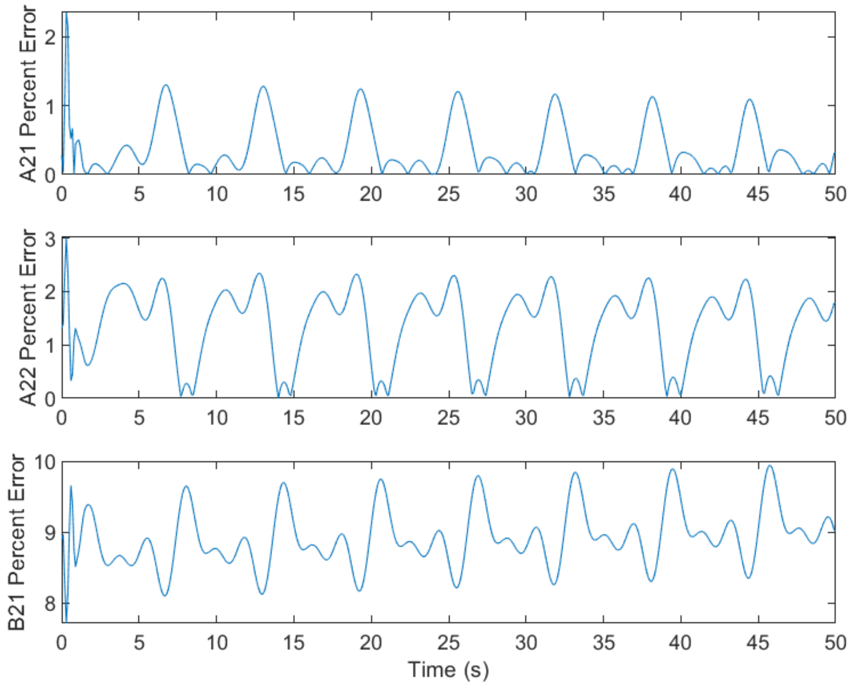


Figure 3.25 PIRL - Lower Limit - Percentage Error of Trial and True Values in System Model

It is evident from these Figures that, much like the maximum value test, when the unknown parameters are assigned at the limits of their training values, the percentage error increases. Despite this increase in error, the RL Agent continues to require little to no convergence time and the state histories of the true and trial systems remain closely aligned.

Effectiveness of PIRL Agent at Triple the Range - Upper Bound

The RL Agent was trained within a predefined bound of possible systems; however, if PIRL were to be implemented in a real system, there may exist models that are outside of the training ranges that the RL Agent has experienced. To better understand the effect of systems outside of the training range, the RL Agent was implemented on a system with all unknowns set to 1.3. This represents a system with an overall range of system values of 0.6 compared to the original range of 0.2. The average error of this test is shown in Table 3.8.

Table 3.8 PIRL Agent Performance at Triple Max Parameter Values

Parameter	Per Cent Error
A21	0.5645%
A22	1.6401%
B21	26.7047%
Average Total Error	9.6364%

The average error of this system is significantly higher than the RL Agent operating within its trained range. Figure 3.26 shows the significant increase in total error of this system. This greater error is apparent in the state histories in Figure 3.27 where the X2 trial history diverges significantly from the true state.

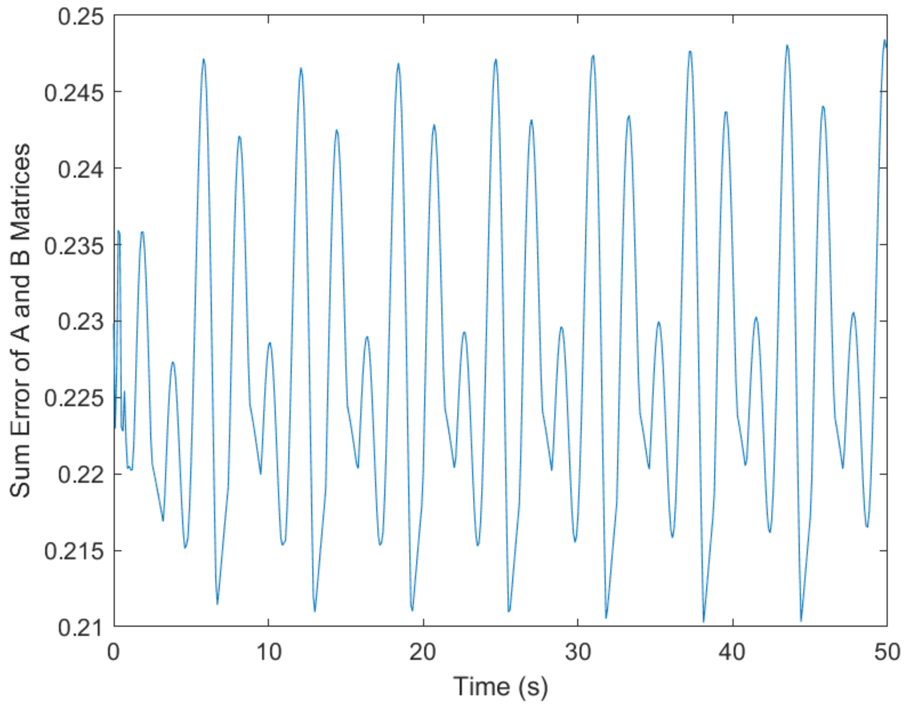


Figure 3.26 PIRL - Triple Range - Upper Limit - Sum of Errors of A21, A22, and B21

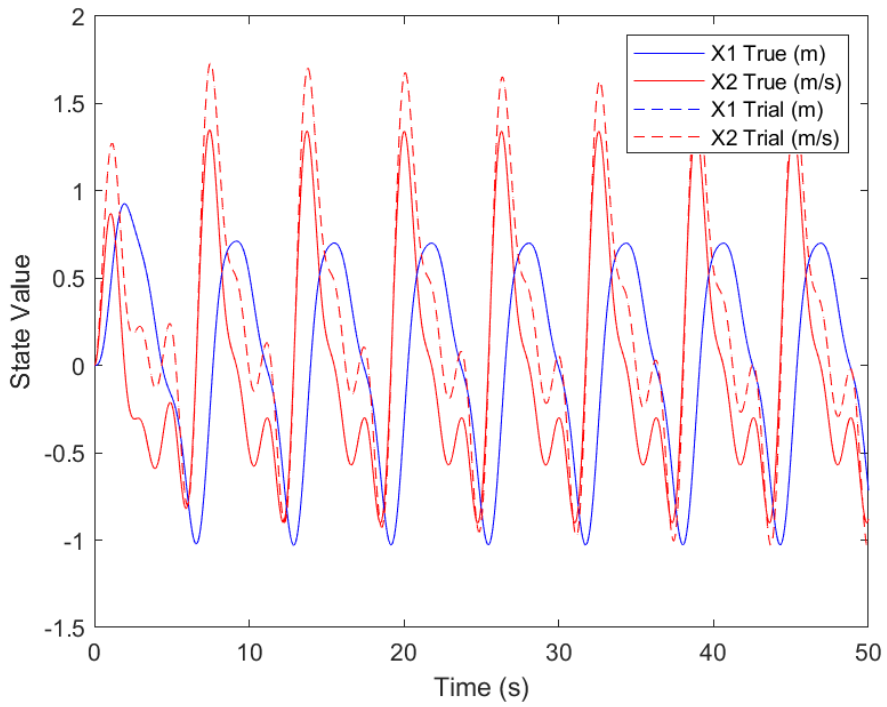


Figure 3.27 PIRL - Triple Range - Upper Limit - State History of True States and RL Agent Trial States

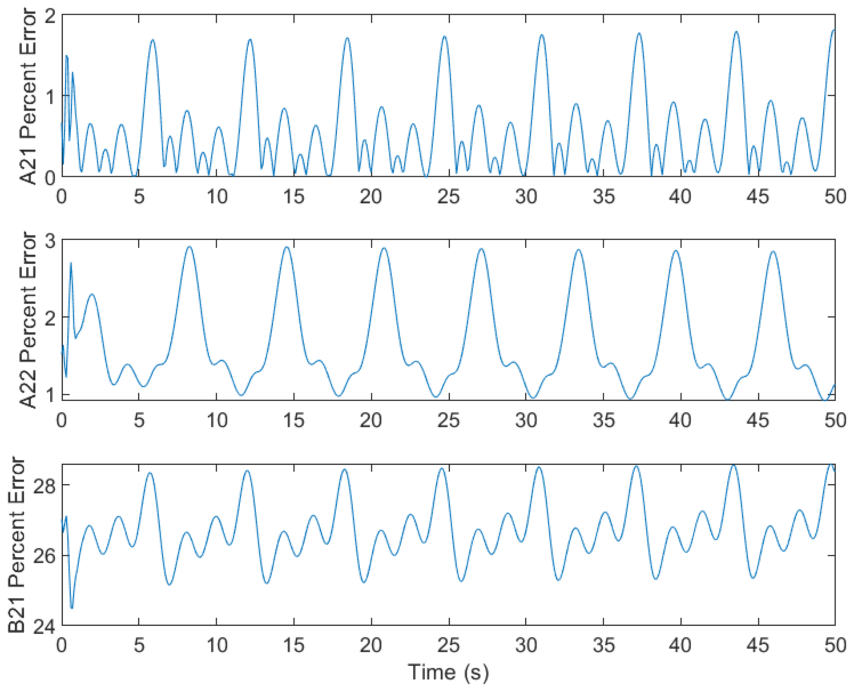


Figure 3.28 PIRL - Triple Range - Upper Limit - Percentage Error of Trial and True Values in System Model

The percentage error of each estimated term is illustrated in Figure 3.28. While the error of each variable exhibits the same transient and periodic pattern, the magnitude of the errors are significantly higher than any previous tests. This indicates that a significant departure from the training range will lead to a considerable decrease in the RL Agent's effectiveness. Furthermore, this increase in error may not manifest equally in all unknown parameters, as the error in B21 increases considerably, while the RL Agent remains able to determine the values of A21 and A22 with little to no difficulty.

Effectiveness of PIRL Agent at Triple the Range - Lower Bound

A similar test is conducted to observe the impact of the minimum limit of this increased range, the results of which can be seen in Table 3.9. For this test, all unknowns are set as 0.7 and it can be seen that the effect of this is similar to that of the upper limit test where the A21 and A22 values experience slight losses in accuracy, whereas the B21 value differs significantly leading to an ultimately higher average error between the trial system and the true system. The state history and percentage error history can be seen in Figures 3.29 and 3.30. It is evident in these figures that the lower bound system seems to result in X2 being consistently underestimated in value until 40 seconds. Additionally, the A21 and A22 errors remain relatively consistent but the B21 error increases with time.

Table 3.9 PIRL Agent Performance at Triple Minimum Parameter Values

Parameter	Per Cent Error
A21	0.7542%
A22	2.6025%
B21	27.9408%
Average Total Error	10.4325%

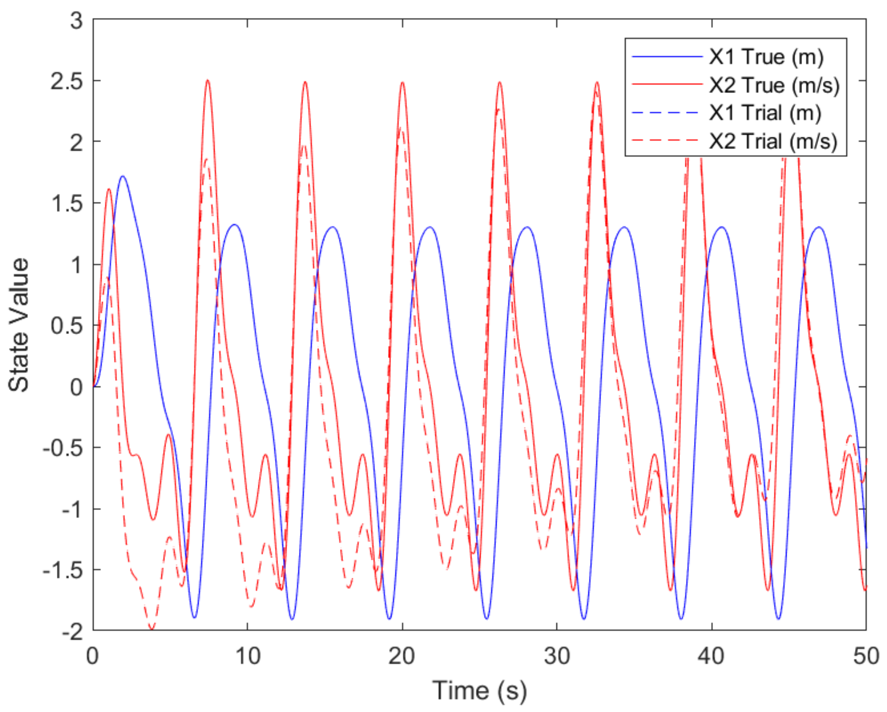


Figure 3.29 PIRL - Triple Range - Lower Limit - State History of True States and RL Agent Trial States

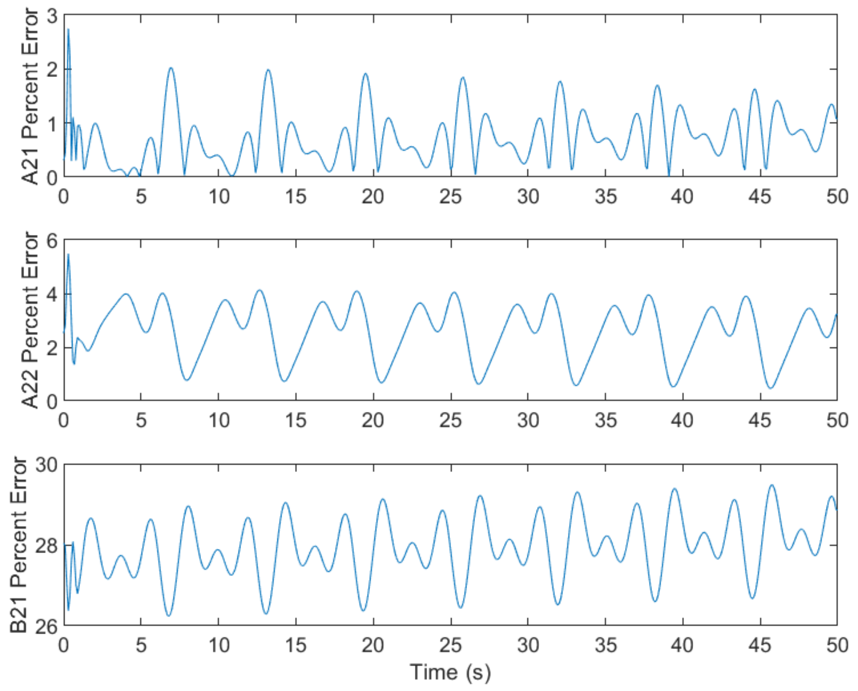


Figure 3.30 PIRL - Triple Range - Lower Limit - Percentage Error of Trial and True Values in System Model

Performance of Conventional Reinforcement Learning

The use of reinforcement learning for SYSID is not unique and it has been shown that it is an effective method. PIRL differs from the typical implementation of RL with its use of system parameters as additional inputs. To avoid misattributing the performance of an RL method to PIRL specifically, a second RL Agent was trained and evaluated with all characteristics being the same except that it lacked the parameters as inputs. The RL Agent was then evaluated over 100 different systems within trained ranges as well as with the triple size range. The PIRL Agent was implemented in the same manner for comparison. The results of these tests are shown in Table 3.10. The percentage error refers to the average error across the 50 second episode and across A21, A22, and B21.

Table 3.10 PIRL Agent Vs. Conventional RL Agent

Implementation	PIRL Per Cent Error	Conventional Per Cent Error
Trained Range	1.3543%	6.3386%
Triple Trained Range	5.078%	14.1924%

It is clear from these tests that the conventional RL Agent is effective within its trained range and loses effectiveness as it is exposed to a greater range than it has been trained on; however, the PIRL Agent consistently performs better than conventional RL by a factor of 2.7 or more. In the trained range PIRL performs better than conventional RL by a factor of 4.68. This is a strong indicator that PIRL positively impacts the RL Agent's performance in SYSID.

Comparison to an Extended Kalman Filter

In addition to evaluating how PIRL performs with regard to other non-PIRL RL Agents, it is also important to understand how PIRL performs when compared to more conventional methods. An Extended Kalman Filter is used as an existing method through which conclusions can be drawn regarding the PIRL Agent's performance as an independent method of performing system identification. It should be noted that this implementation of the extended Kalman filter is not the only one possible. This particular implementation was chosen at it requires the filter to perform the same task as the PIRL Agent.

Extended Kalman filters (EKFs) are used for a variety of purposes including filtering noise, state estimation and system identification. When appropriately tuned, they can perform these tasks well and in a computationally efficient manner. The EKF is similar to PIRL in that it relies on estimating states and then, by comparing the differences between these states, it can determine the true system. PIRL uses the error found between its trial system estimated state and the measurements but differs in that this is only one part of the many different inputs provided to the algorithm and, rather than explicitly estimating states, PIRL only indirectly estimates them by using the estimated system. An EKF is implemented for system identification by using the desired estimated system values (A21, A22, and B21) as states within the filter. These are then estimated by the EKF in the same manner as conventional states. The Extended Kalman filter is required, rather than the regular Kalman filter, as estimating the system values converts the system from linear to nonlinear. Before the EKF can be used, the system must be converted from continuous time to discrete time. This is done using Euler's method as shown in Equations 3.12 and 3.13. It should be noted that dt refers to the time step interval at which the EKF is implemented and not necessarily the rate at which the MSD is simulated. In this experiment though, they are the same; dt is equal to T_s.

$$\dot{X} = AX + BU \tag{3.12}$$

$$X_{k+1} = (Adt + I)X_k + BdtU_k \tag{3.13}$$

Following the continuous to discrete conversion, the desired values are added as states. This will make the system nonlinear and thus it must be represented as a series of functions as can be seen in Equation 3.14, where $\hat{x}_3$, $\hat{x}_4$, and $\hat{x}_5$ represent A21, A22, and B21 respectively. It should be noted that the ˆ denotes that each state is an estimate.

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \end{bmatrix}_{k+1} = f(\hat{X}_k, U_k) = \begin{Bmatrix} \hat{x}_{1k} + \hat{x}_{2k}dt \\ (\hat{x}_{3k}dt)\hat{x}_{1k} + (1 + \hat{x}_{4k}dt)\hat{x}_{2k} + \hat{x}_{5k}dtU_k \\ \hat{x}_{3k} \\ \hat{x}_{4k} \\ \hat{x}_{5k} \end{Bmatrix} \tag{3.14}$$

With the discrete dynamics defined in Equation 3.14, the following procedure given in Equations 3.15 to 3.19 is performed in order at each k time step. Note that $k + 1/k$ refers to intermediate values and that C has been augmented with zeroes to support the larger $\hat{X}$ vector. In this process, F refers to the Jacobian of A with respect to $\hat{X}$, while Q, R, P, and K refer to the process noise covariance matrix, the sensor noise covariance matrix, the estimation error covariance matrix and the Kalman gain respectively.

$$\hat{X}_{k+1/k} = f(\hat{X}_k, U_k) \tag{3.15}$$

$$P_{k+1/k} = F_k P_k F_k^T + Q_k \tag{3.16}$$

$$K_{k+1} = P_{k+1/k}C^T(CP_{k+1/k}C^T + R_k)^{-1} \tag{3.17}$$

$$\hat{X}_{k+1} = \hat{X}_{k+1/k} + K_{k+1}(Y_{k+1} - C\hat{X}_{k+1/k}) \tag{3.18}$$

$$P_{k+1} = (I - K_{k+1}C)P_{k+1/k} \tag{3.19}$$

In Figure 3.31, the EKF is applied to the MSD system and the estimations of the two states can be seen to be very close to the true states observed. For this example, the mass, spring constant, and damper constant are all equal to one with Gaussian noise added to the observations with a signal to noise ratio of 40.

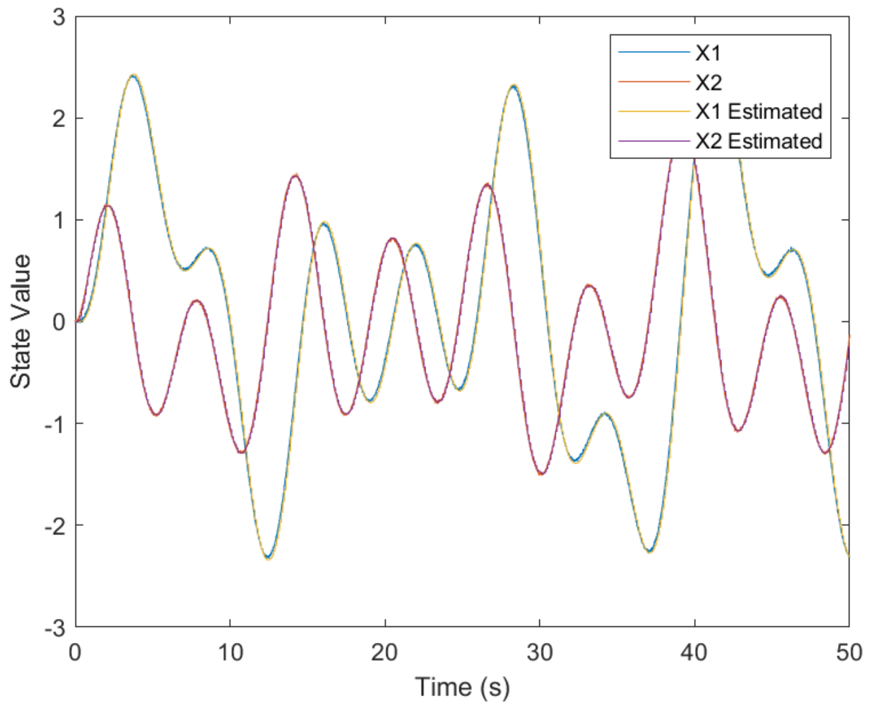


Figure 3.31 Extended Kalman Filter State Estimation Example

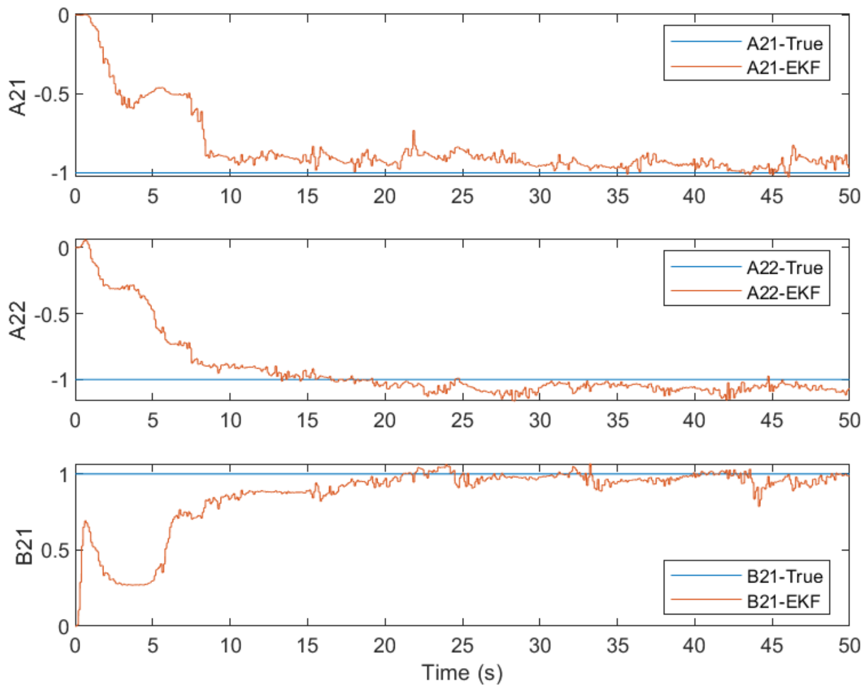


Figure 3.32 Extended Kalman Filter System Identification Example

In Figure 3.32, one can see the estimations of the A21, A22, and B21 values as estimated by the EKF. It is evident from this figure that the EKF has little to no error in the estimated states, but it suffers in accuracy for system identification with convergence times of up to 20 seconds for B21. The average error of each term across the 50 seconds can be seen in Table 3.11; however, a more accurate determination of average error would be to use the values once the EKF converged and these results can be seen in Table 3.12. When only the last 25 seconds are used, the error diminished significantly, with an average percentage error per term of only 3.58%. This is still higher than the average error of the PIRL Agent across the trained range.

Table 3.11 EKF Average Error Example

Parameter	Per Cent Error
A21	16.2739%
A22	15.1577%
B21	13.5061%
Average Total Error	14.9792%

Table 3.12 EKF Average Error (Converged) Example

Parameter	Per Cent Error
A21	5.8274%
A22	7.9625%
B21	3.5786%
Average Total Error	5.7895%

To better understand how the EKF compares to PIRL, the EKF was tasked with the same test as PIRL: to identify 100 systems randomly selected from the PIRL training range and the PIRL testing range. It should be noted that the EKF is not being tuned for each system, much like how PIRL cannot be altered for each system. The EKF is tuned to perform for a mass, spring, constant, and damper constant of one, thereby keeping it tuned for the average expected system constants. The results of the EKF have been compared with those of PIRL and only the last 25 seconds of the EKF data has been used to avoid a negative bias caused by the EKF's convergence time.

Table 3.13 Comparison of EKF to PIRL

Method	Average Error Trained Range	Average Error Triple Range
EKF	5.6114%	5.7974%
PIRL	0.6766%	3.9348%

The results of this test can be seen in Table 3.13. It is clear from these data that using PIRL leads to a significant reduction of average error in the trained range as well as a minor one in the testing range. This indicates that PIRL may have an advantage over the EKF for this specific application of system identification. When combined with the minimal convergence time exhibited by PIRL it is evident that in this environment, PIRL performs well as a novel method for system identification.

3.3.2 Effect of Hyperparameters

An RL Agent's training is influenced by a number of different hyperparameters. These may or may not impact the performance of the Agents. To better understand the relationship between each hyperparameter and the performance of the Agents, the sample time, neural network architecture, discount factor, noise variance and decay rate, and observation window length and interval were all altered individually and the effects of these variations were observed and analyzed.

Effect of Higher Episode Reward Threshold

An RL Agent can be trained with a fixed reward threshold in which, upon having reached this reward, training is completed. This was not the method used to train the RL Agents as it is not as effective as selecting the highest average reward value from a fixed number of episodes. It remains relevant to understand the effect of altering this threshold as it provides insight into the relationship between the performance observed in training and testing. The results from this test can be seen in Table 3.14. It should be noted that the -50 result comes from a different RL Agent but from the same training regime than previous results and thus it has performed differently. The Episode Number refers to the number of episodes required to reach the threshold.

Table 3.14 Effect of Episode Reward Threshold

Training Reward Cutoff	Average Error Trained Range	Average Error Triple Range	Episode Number
-500	12.6121%	23.9918%	8
-450	8.5146%	19.9860%	23
-400	6.1651%	17.8416%	62
-350	7.3524%	17.4945%	65
-300	6.4235%	17.7951%	94
-250	2.7437	8.6438%	144
-200	3.0812%	7.6651%	196
-150	2.1201	5.8566%	273
-100	0.8579%	4.6517%	349
-90	0.9982%	3.8319%	364
-80	1.2165%	5.7065%	392
-70	1.4015%	6.2514%	420
-60	1.3061%	5.1746%	431
-50	0.7694%	4.3294%	479
-40	0.5436%	3.7581%	495
-30	0.6015%	3.9229%	554
-20	0.7036%	3.7934%	695

In Table 3.14, it can be seen that, by increasing the required episode reward, the resultant accuracy in both the trained range and the triple trained range increases significantly. There does exist an asymptotic point roughly at -50 where further increases do not significantly improve performance and in some cases decrease it. These decreases in performance are not likely due to the increased reward limitation and are more likely the cause of the random initialization of the RL Agent during training. Given these data, it can be concluded that -50 is a reasonable reward limit, as further increases do not provide significant gains in performance and the number of training episodes required to reach this limit increases significantly. The results of Table 3.14 have been plotted in Figures 3.33 and 3.34. From Figure 3.33, it can be seen that the error seems to follow an approximate exponential decay curve where the first increases to the reward threshold lead to much larger improvements than the last increases. In Figure 3.34, it can be seen that increases in the threshold require an approximately exponential number of training episodes to reach them. While RL Agents in this thesis were not trained using a fixed cutoff (with the exception of this test), this test grants insight into how the training process may be shortened. The 8000 training episode method used in this thesis took several hours to complete on a high end consumer PC, while achieving a -50 reward took roughly 20-30 minutes to obtain[1].

In Figure 3.33, there exists two diversions from the decaying exponential trend at roughly -350 and -70. It is unclear what causes these deviations. One possible explanation is that these reward values may correspond to specific systems or small ranges of systems that, if the RL Agent is provided with them, creates a specifically high reward value, thereby inflating the actual performance of the RL Agent on all systems. If there existed an artificially ideal system, then it would most likely be evident in the required number of episodes to reach the threshold as the RL Agent would stop training sooner in this case. This is not evident in Figure 3.34 as the required number of training episodes follows very closely with an exponential trend.

It is also possible that these diversions from the trend may be caused by noise. Each RL Agent is initialized randomly and evaluated differently, and this may be the cause; however, it is notable that the diversions in Figure 3.33 are representative of multiple points, which is uncharacteristic of random noise.

[1]Computer Specifications: 11th Gen Intel(R) Core(TM) i7-11700KF @ 3.60GHz 3.60 GHz, Corsair Vengeance Pro DDR4 4×16GB (2x16GB) 3600MHz CL18 Intel XMP

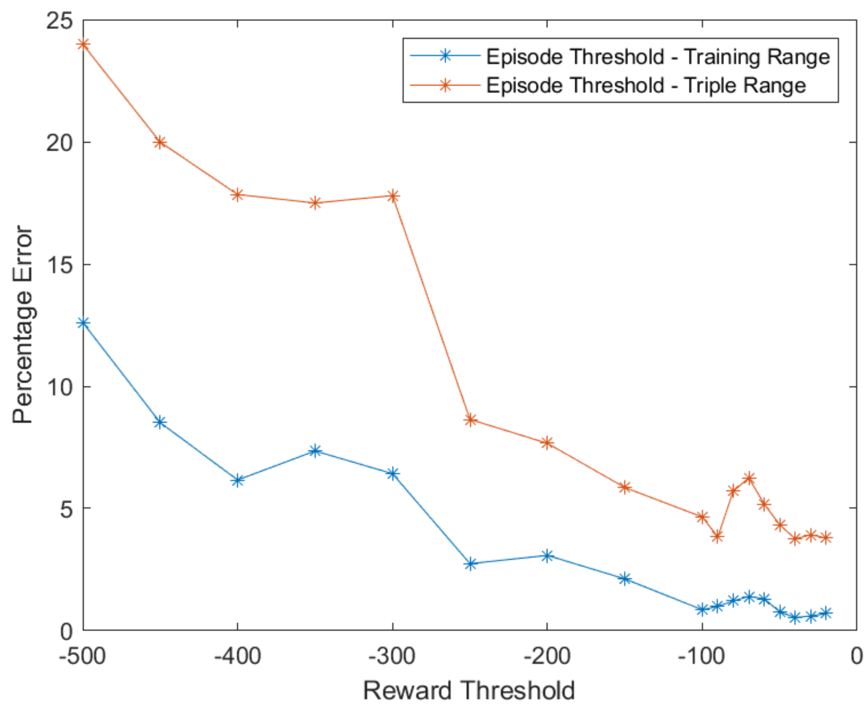


Figure 3.33 Episode Reward Threshold and Performance

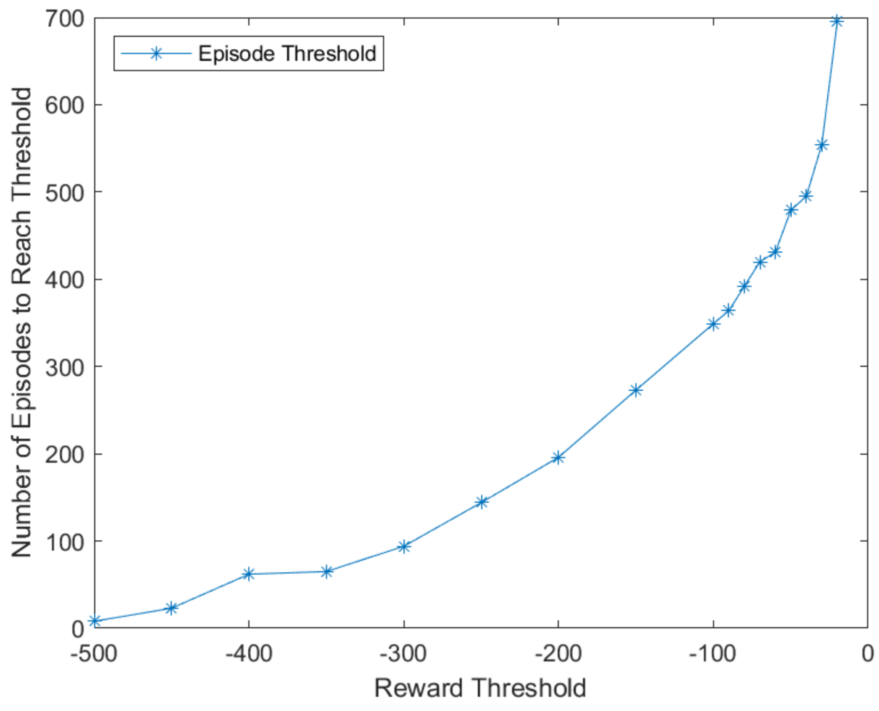


Figure 3.34 Episode Reward Threshold and Required Training Episodes

Effect of Higher Average Reward Threshold

To reduce the impact of an RL Agent being well suited through probability to a particular system, the same experiment was conducted with the average reward threshold used as opposed to the episode reward threshold. The average reward is a moving average of the reward of the last 20 episodes. Using this as the threshold should reduce the variability in the performance data. The results of this experiment can be seen in Table 3.15. From these data, it is evident that, similarly to the episode reward threshold test, increasing the average reward threshold corresponds to a decrease in the average percentage error in both the trained range and the triple trained range. Additionally, the episode number required to attain this threshold increases with the threshold; however, the episode number associated with each reward is significantly greater for the average reward experiment than the episode reward experiment.

Table 3.15 Effect of Average Reward Threshold

Training Reward Cutoff	Average Error Trained Range	Average Error Triple Range	Episode Number
-500	6.2646%	18.0490%	69
-450	6.1709%	18.7714%	95
-400	5.4421%	15.6981%	121
-350	4.3253%	13.9219%	144
-300	3.2476%	10.5674%	176
-250	3.1278%	10.6297%	210
-200	1.4579%	5.6982%	256
-150	2.2682%	7.9674%	302
-100	0.5929%	4.5089%	375
-90	1.3552%	5.1157%	418
-80	1.3526%	5.3207%	440
-70	0.8448%	5.0686%	460
-60	0.8177%	3.6122%	494
-50	0.6258%	3.7113%	527
-40	1.3507%	5.3390%	582
-30	0.6034%	3.3893%	634
-20	0.5697%	3.5536%	711

To better understand the effect of reward threshold on the performance of the RL Agent, the data from Table 3.15 have been graphed and shown in Figures 3.35 and 3.36. The results from Table 3.14 have also been included in these figures for comparison.

Figure 3.35 illustrates that, across both the trained range and triple range, increasing the reward leads to a decrease in the RL Agent's error. This is true for usage of the episode threshold or the average threshold. Given that the average threshold requires a more consistent performance by the RL Agent, it generally performs better than the episode threshold. There are a few exceptions to this rule, which may be attributed to noise or the lessened effects of particularly convenient systems for the RL Agent.

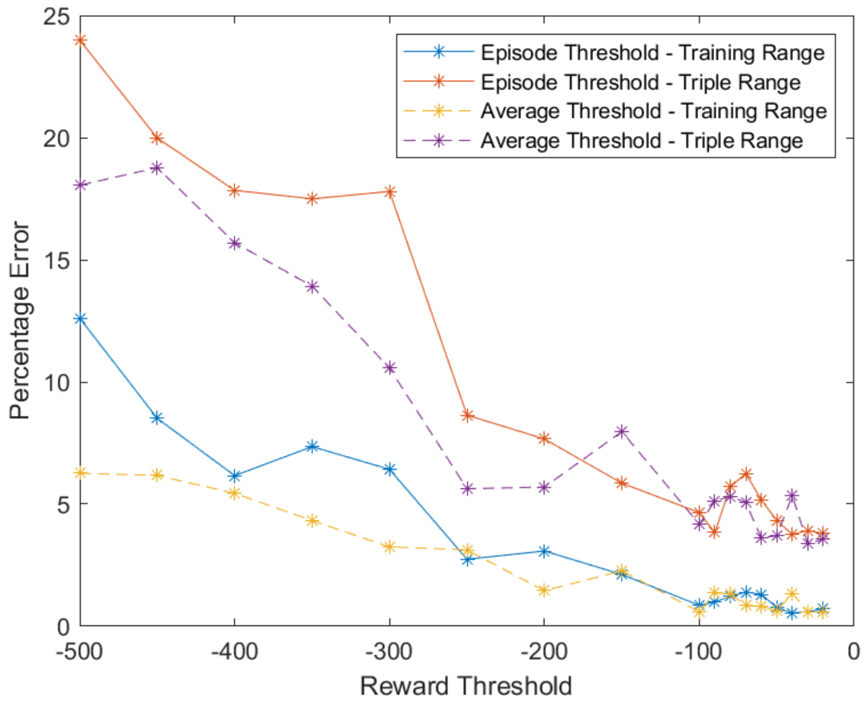


Figure 3.35 Average Reward and Episode Reward Threshold Performance

Figure 3.36 illustrates that using an average reward as opposed to episode reward effectively increases the number of training episodes provided to the RL Agent. Changing from episode reward to average reward not only requires the RL Agent to be sufficiently accurate (on average) across a given window of episodes, it also, due to this requirement, gives the RL Agent a greater time in which to improve. This two-fold benefit shows that the RL Agent performs better when training is evaluated with respect to the average reward rather than the episode reward. This may be beneficial if training to a fixed cutoff and desiring consistent performance, but as seen in Figure 3.36, it does incur an increased computation cost.

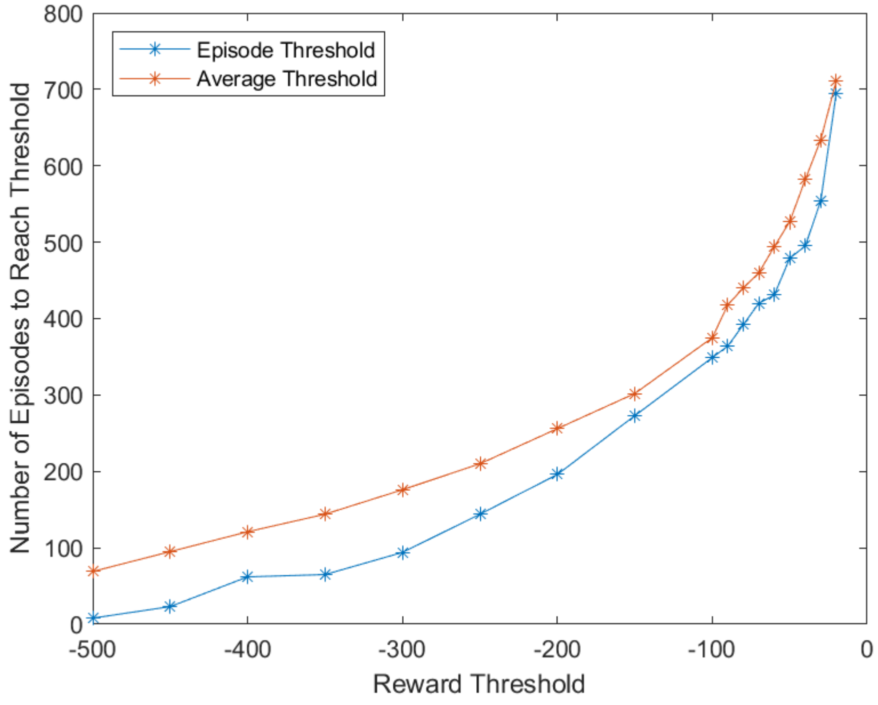


Figure 3.36 Average Reward and Episode Reward Threshold Required Training Episodes

Sample Time

The RL Agent has a separate sample time from the simulation's sample time. The RL Agent sample time refers to how often the RL Agent is activated during the simulation. Each activation gives a new A and B matrix. The reason for these sample times being different is to provide more control over the computational requirements. An RL Agent that is trained offline every 0.01 seconds during its training episodes will have a significant training time. Even with a sufficiently powerful computer or sufficient time, a complex AI of any kind being evaluated every 0.01 seconds may be prohibitively inefficient in an online or embedded environment. For this reason, for the MSD environment, the RL Agent is evaluated every 0.1 seconds.

It is hypothesized that, if the sample time is increased, the RL Agent will be able to complete each episode faster, but because an RL Agent receives feedback on its Actions every sample, an increase in the RL Agent's sample time will decrease its learning rate per episode. This will lead to larger final episode numbers but similar performances or even better performances given that the greater sample time systems have been exposed to more systems. Therefore, the total number of samples is expected to stay roughly the same.

To better understand this relationship, the sample time in training and testing was varied and the impact that this had on performance, final episode number, and the total number of samples was observed. The total samples value for each sample time was calculated using Equation 3.20, where E_N is the final episode number, T_f is the final episode time, and T_s is the sample time. For this test, the RL Agent was required to reach an Episode Reward of -50. This was chosen over the previously discussed method of evaluating the effect of variables to illustrate how sample time does not impact the training rate despite the AI being implemented more or less frequently.

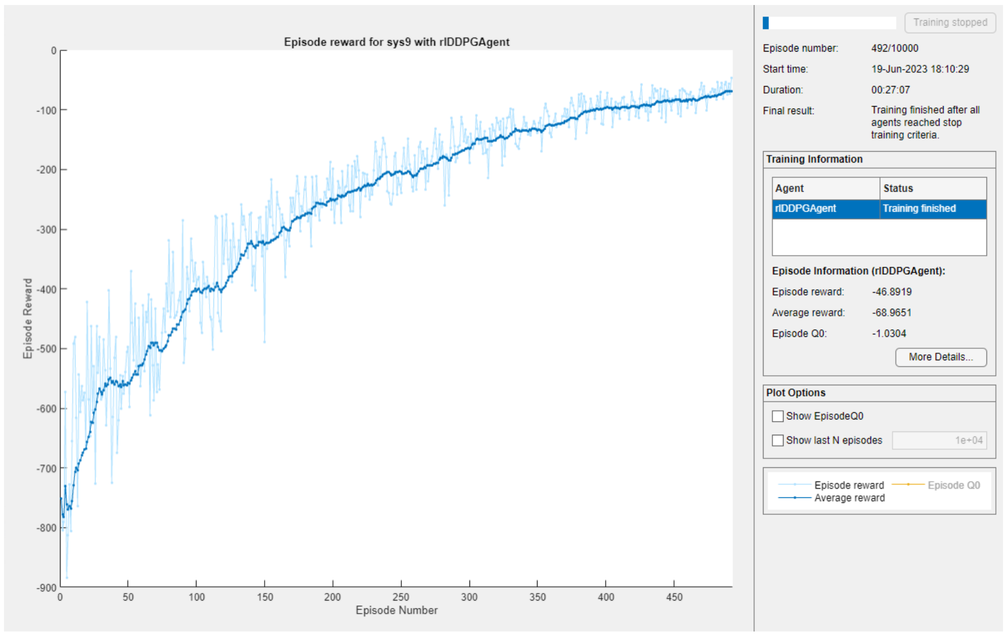
$$TotalSamples = \frac{E_N \times T_f}{T_s} \tag{3.20}$$

Table 3.16 Effect of Sample Time

Sample Time (s)	Average Error Trained Range	Average Error Triple Range	Episode Number	Total Samples
0.5	2.4899%	7.3761%	713	71,300
0.4	3.2230%	9.5613%	737	92,125
0.3	2.5934%	8.7263%	744	124,000
0.2	1.2903%	5.4141%	647	161,750
0.1	1.2637%	5.2865%	511	255,500

The results of this experiment can be seen in Table 3.16. It is clear that these data do not follow the hypothesized trend, with increases in sample time leading to a significant decrease in performance. Episode number is altered, but not by the substantial amount that a decrease in feedback by 1/5th (as is the case from 0.1s to 0.5s) would indicate. There is additionally no clear trend in either direction. Finally, increases in sample time lead to decreases in the total samples. This is contrary to the hypothesis that they would stay approximately the same.

To understand the reason for these contrary results, the effect of sample time must be analyzed more closely. The training curve for the 0.1 second sample time is shown in Figure 3.37, and the training curve for the 0.5 second sample time is shown in Figure 3.38.



Episode number:	492/10000
Start time:	19-Jun-2023 18:10:29
Duration:	00:27:07
Final result:	Training finished after all agents reached stop training criteria.

Training Information

Agent	Status
rlDDPGAgent	Training finished

Episode Information (rlDDPGAgent):

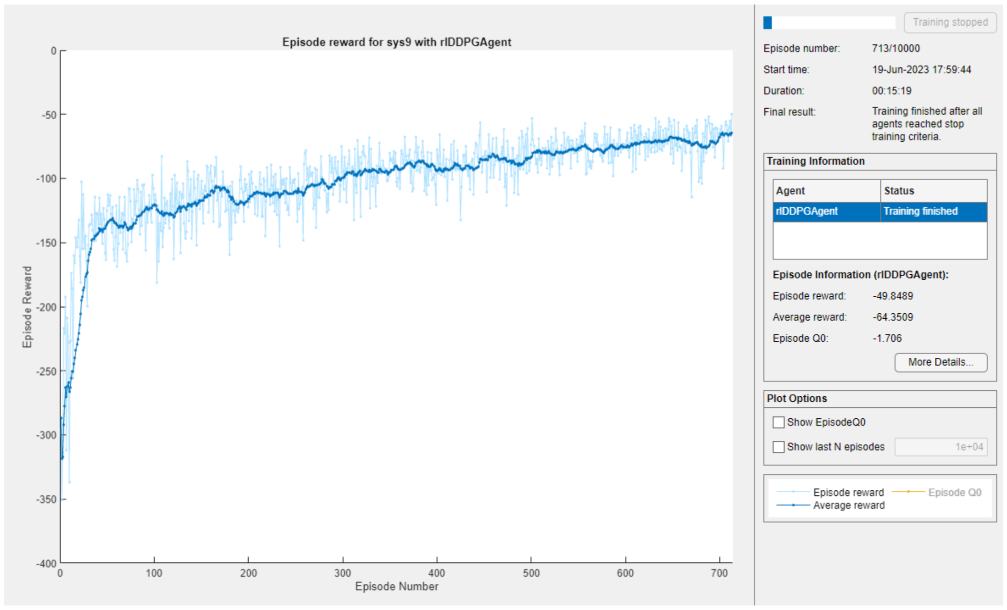
Episode reward:	-46.8919
Average reward:	-68.9651
Episode Q0:	-1.0304

Plot Options

☐ Show EpisodeQ0

☐ Show last N episodes 1e+04

Figure 3.37 0.1 Second Sample Time Training Curve



Episode number:	713/10000
Start time:	19-Jun-2023 17:59:44
Duration:	00:15:19
Final result:	Training finished after all agents reached stop training criteria.

Training Information

Agent	Status
rlDDPGAgent	Training finished

Episode Information (rlDDPGAgent):

Episode reward:	-49.8489
Average reward:	-64.3509
Episode Q0:	-1.706

Plot Options

☐ Show EpisodeQ0

☐ Show last N episodes 1e+04

Figure 3.38 0.5 Second Sample Time Training Curve

There exists a difference of note between the two curves. The 0.5s curve begins at a reward of roughly -320 whereas the 0.1s curve begins closer to -800. This would indicate that the decreased sample time is leading to an artificial increase in reward as new, untrained RL Agents should be broadly similar in capability. The Sum of Errors of the two Agents can be seen in Figures 3.39 and 3.40. Both Agents were provided with the same system where all constants were in the middle of the possible trained values. From these figures, it is evident that the error in the estimated A and B matrices of the 0.5s system is consistently greater than that of the 0.1s system, which is contrary to the reward curves seen previously. The error for the 0.5s system should be significantly lower as its training curve is consistently higher than that of the 0.1s system.

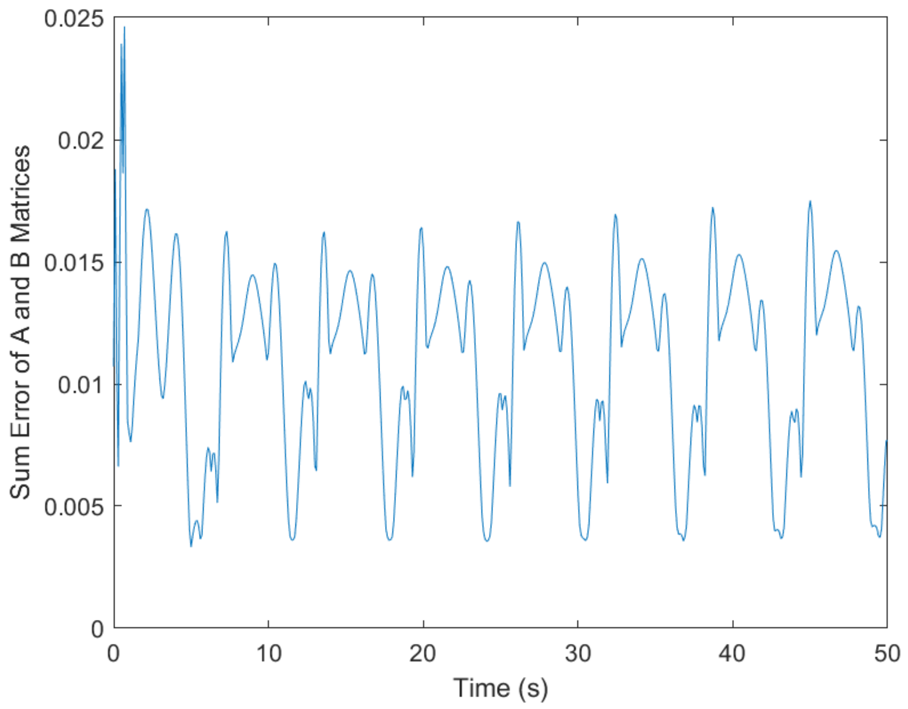


Figure 3.39 0.1 Second Sum of Errors

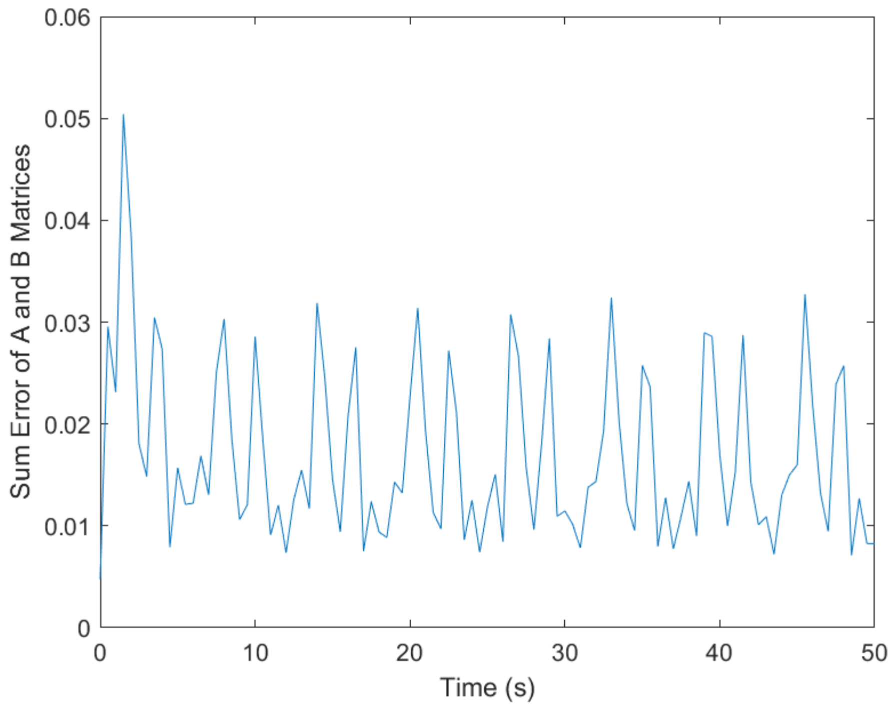


Figure 3.40 0.5 Second Sum of Errors

This trend is also evident in the reward curves of a single episode, as seen in Figures 3.41 and 3.42. The reward of the 0.5s system is once again indicating a less accurate RL Agent, which is contrary to the underlying reward curves that generated it. The reason for this trend lies in the specific calculation of the reward function. When the RL Agent is training, every action is evaluated with a specific reward that corresponds to the result of that action. The reward is the negative of the summation of differences between the A and B matrices and their respective true values. This is independent of sample time; however, the reward curve is summed at the end of each episode and this is the value that appears at each data point in Figures 3.37 and 3.38. If the sample time is decreased, the reward per RL Agent activation is not scaled to account for this and the total reward decreases significantly, meaning that it appears in the training curve to perform worse and thus is trained on more episodes to reach the chosen threshold. If the sample time is increased, the reward is summed less often and thus the total episode reward increases. This has the effect of ending training earlier for higher sample time systems, making them less accurate as they were trained on fewer systems.

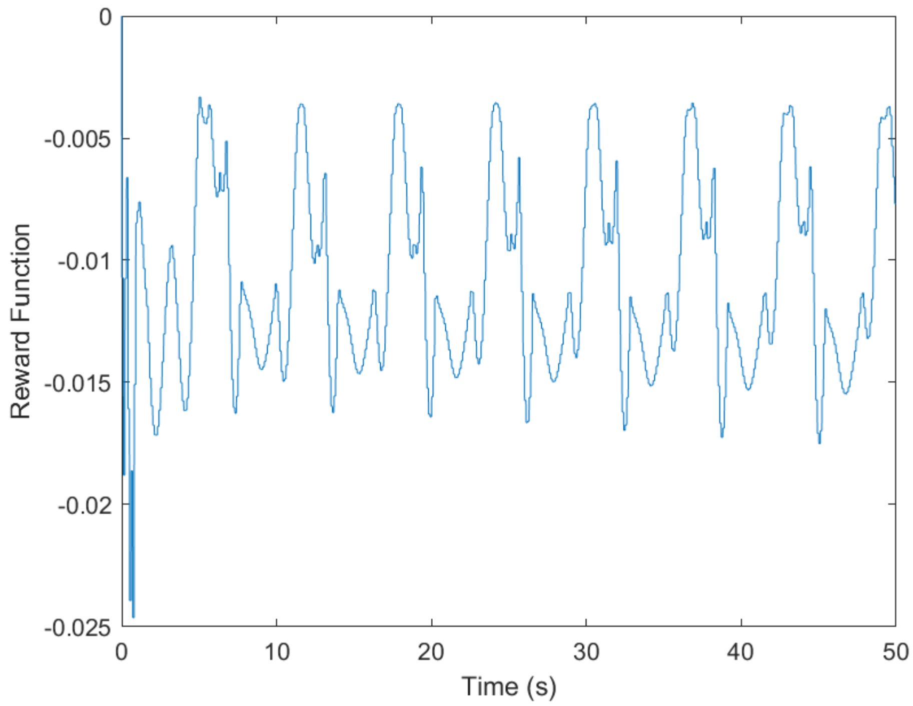


Figure 3.41 0.1 Second Reward Function

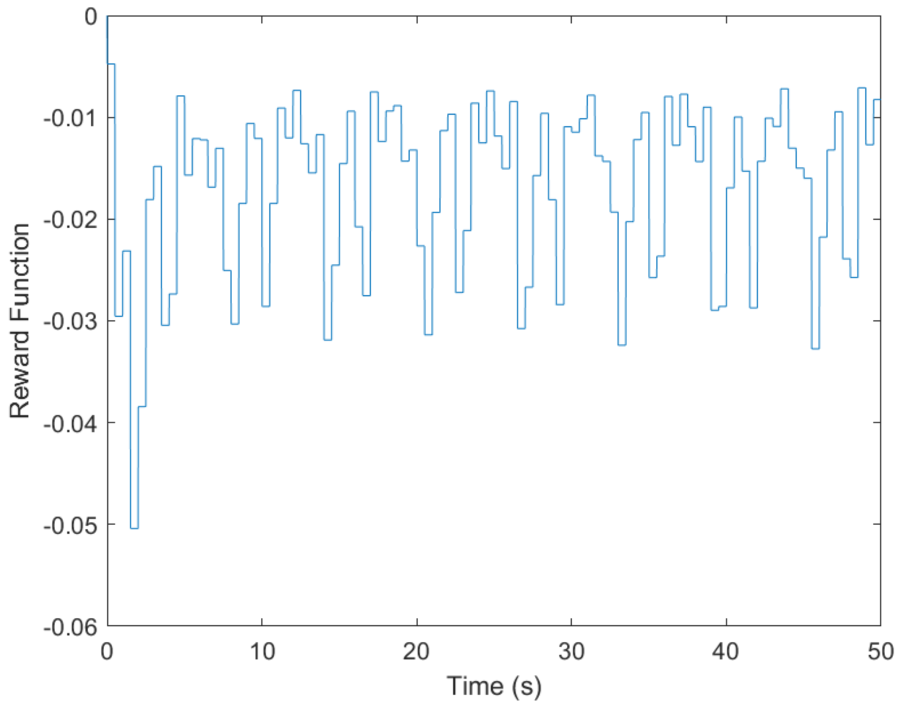


Figure 3.42 0.5 Second Reward Function

This is the reason that the larger sampling time artificially results in a less accurate RL Agent. If sample time is used to scale the reward, as seen in Figure 3.43, then the performance of the RL Agent is no longer impacted by sample time. This modified reward function is the Simulink implementation of Equation 3.21. This can be seen in Table 3.17. The total number of samples all are within the 220,000 to 250,000 range illustrating that sample time does not influence the total amount of feedback required to train an RL Agent, but it does change the total number of episodes required.

$$R = -(|B_{21} - B_{21T}| + |A_{21} - A_{21T}| + |A_{22} - A_{22T}|)\frac{T_s}{0.1} \tag{3.21}$$

It should be noted that an extra gain term was added to the modified reward function. The reason for this gain is to maintain the -50 reward cutoff as a metric of comparable performance. Scaling the error with sample time does prevent the reward from being artificially inflated as sample time is increased; however, without the added gain, the -50 threshold that was tuned for a non-scaled reward with a sample time of 0.1s would no longer produce RL Agents of the same capabilities. The gain ensures that this reward threshold and its corresponding performance remain consistent across these experiments. This specific test illustrates the significance of the reward function as some may produce different results for different hyperparameters. These experiments also show that, if the RL Agent must be implemented on sub-optimal hardware, the RL Agent can be trained to perform well at greater, more computationally efficient sample times.

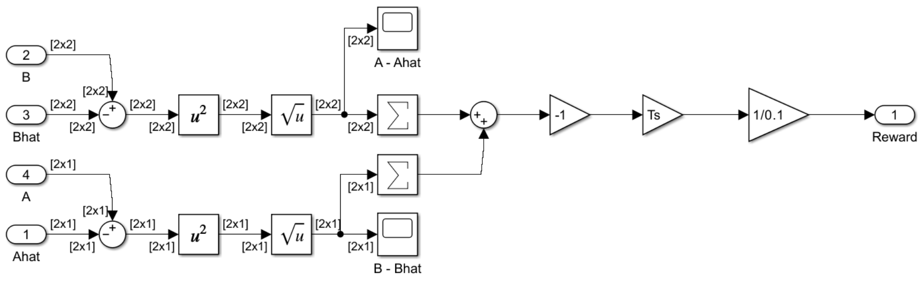


Figure 3.43 Modified Reward Function

Table 3.17 Effect of Sample Time with Modified Reward Function

Sample Time (s)	Average Error Trained Range	Average Error Triple Range	Episode Number	Total Samples
0.5	2.2857%	7.4036%	2295	229,500
0.4	1.7079%	6.1961%	1788	223,500
0.3	1.5595%	5.3913%	1457	242,833
0.2	2.7554%	10.1835%	891	222,750
0.1	1.2812%	5.1646%	493	246,500

While the sample time may have a minimal impact on the number of samples required to reach a specific Episode Reward, to better understand the effect sample time has on the performance of the RL Agent, the RL Agents were trained using a fixed number of episodes and the episode with the highest reward was tested. The results of this experiment are summarized in Table 3.18. It is clear from these data that the sample time has minimal impact on the effectiveness of the network. This finding is relevant in that it implies that the implementation of PIRL may not require significant computational power. Therefore, as long as the sample time is consistent across training and testing, the RL Agent should perform well even with greater sample times/lower sampling rates that are far less computationally intensive.

Table 3.18 Effect of Sample Time with Maximum Performance Test

Sample Time (s)	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Episode	Maximum Reward
0.5	1.0350%	4.8302%	4645	-13.0059
0.4	0.4903%	3.2599%	4812	-5.2994
0.3	1.7901%	6.0911%	3558	-19.8745
0.2	0.5371%	4.4834%	3799	-5.4391
0.1	1.1218%	5.0378%	6789	-13.4506

One other specific aspect in which the performance of the RL Agent is not impacted by sample time is the rate of convergence to the final A and B matrix values. It would appear in Figures 3.44 and 3.45, where the RL Agents have been trained with the -50 Test, that the 0.5s sample time converges slower; however, Figures 3.46 to 3.53, which show the sum of errors across a wider range of m, k, and b values, show that ultimately the sample time has limited to no effect of convergence rate. This trend was found to be consistent across different RL Agents that were trained with the same properties.

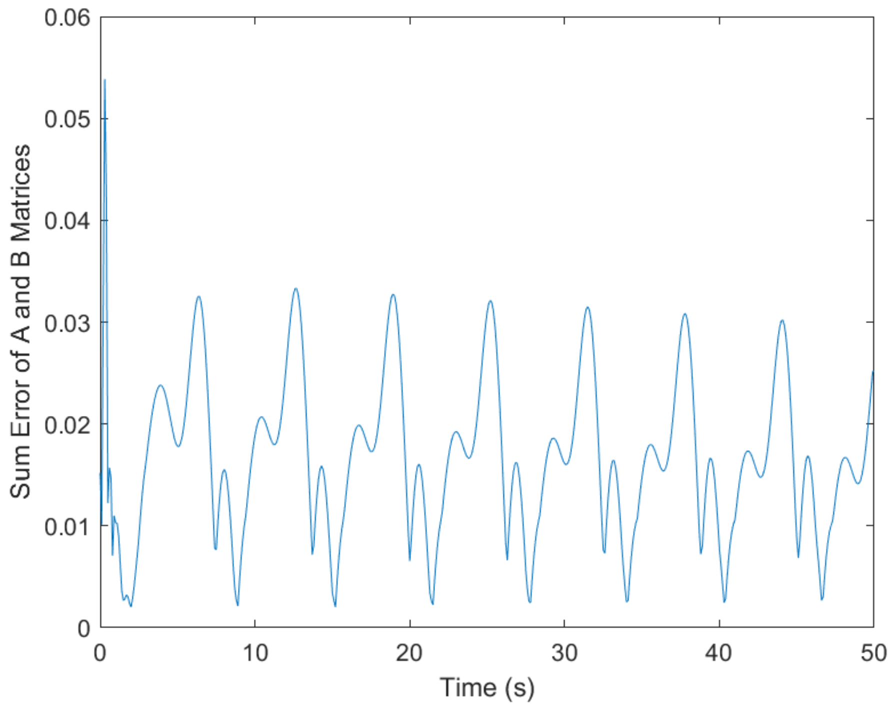


Figure 3.44 0.1 Second Sum of Errors (Modified Reward Function)

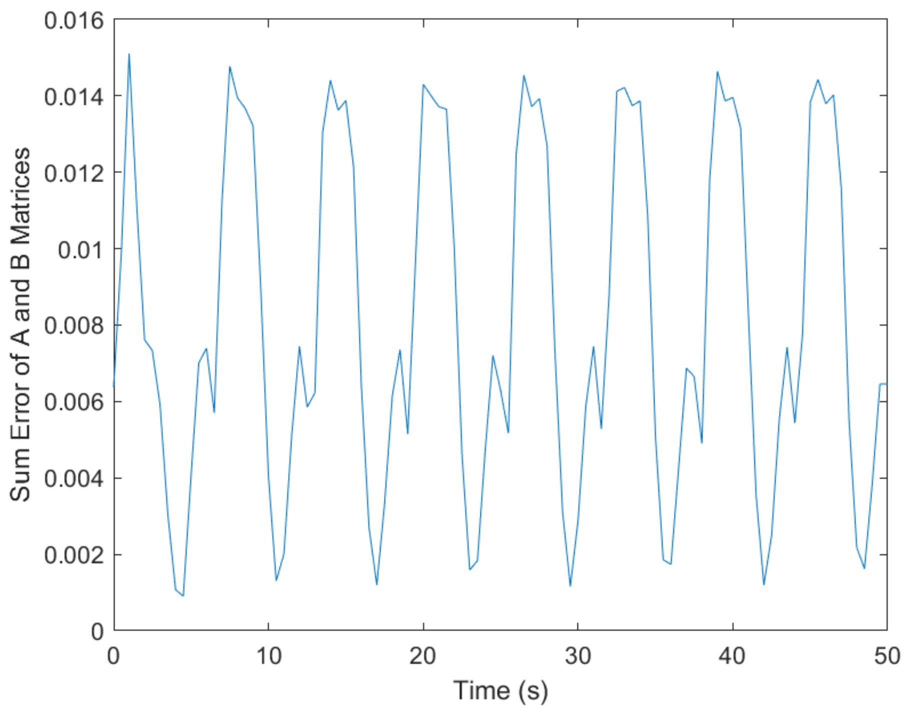


Figure 3.45 0.5 Second Sum of Errors (Modified Reward Function)

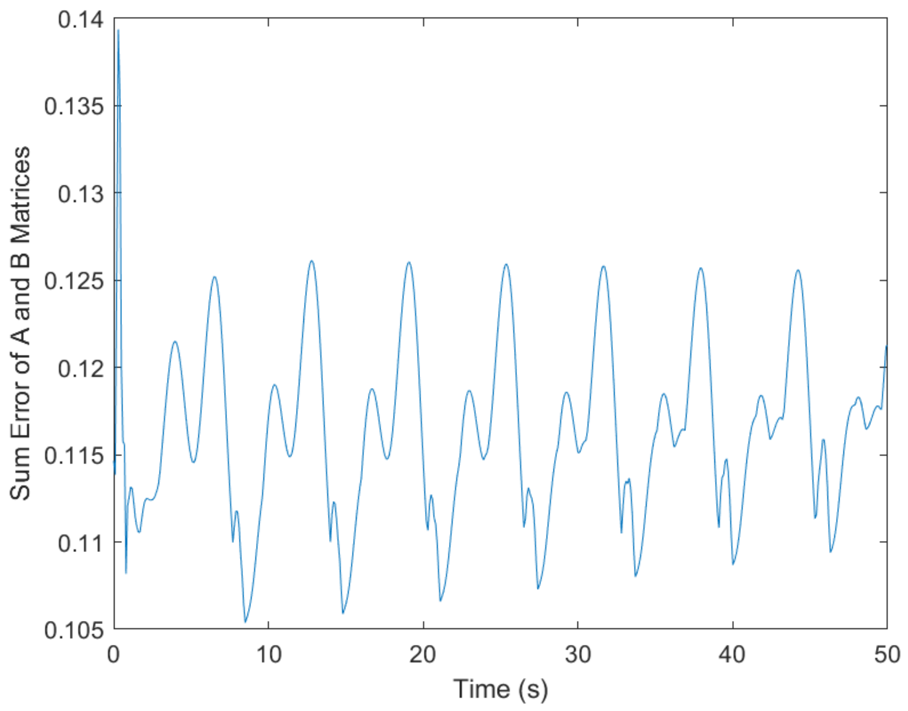


Figure 3.46 0.1 Second Sum of Errors (Modified Reward Function, Low Constant Values)

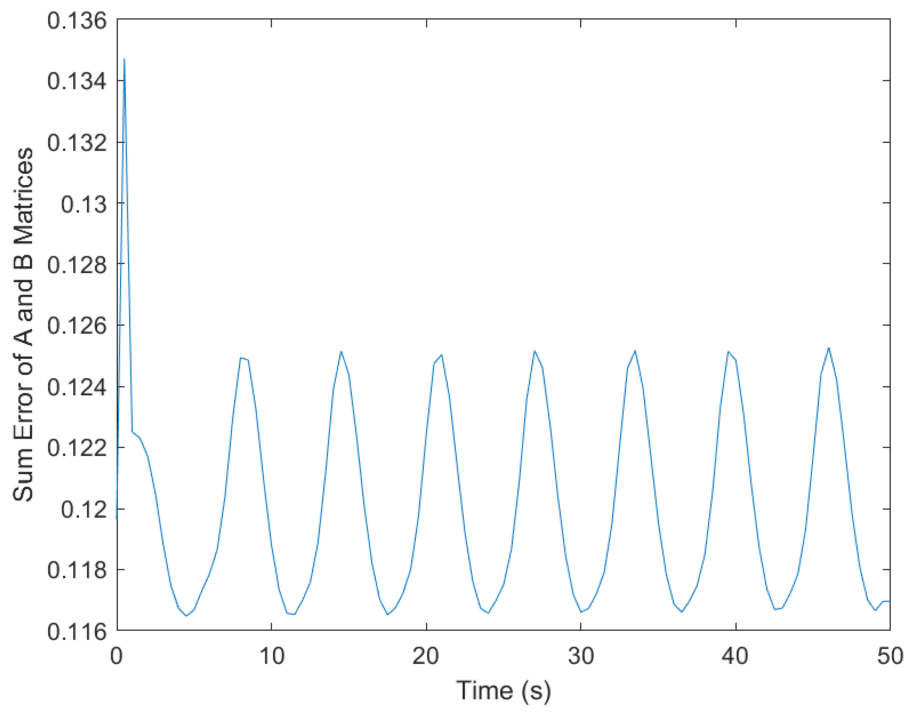


Figure 3.47 0.5 Second Sum of Errors (Modified Reward Function, Low Constant Values)

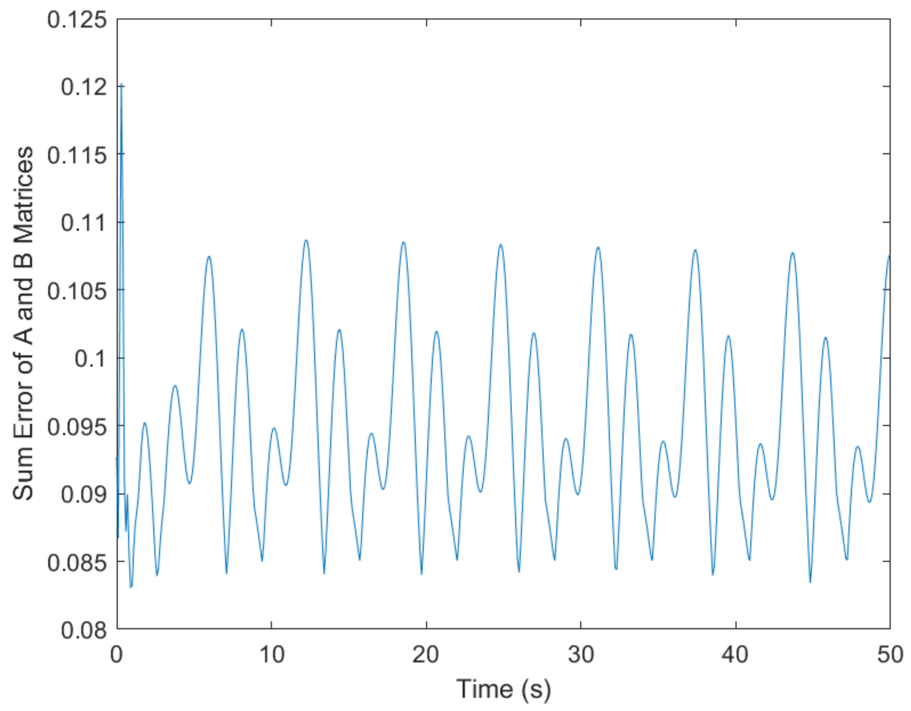


Figure 3.48 0.1 Second Sum of Errors (Modified Reward Function, High Constant Values)

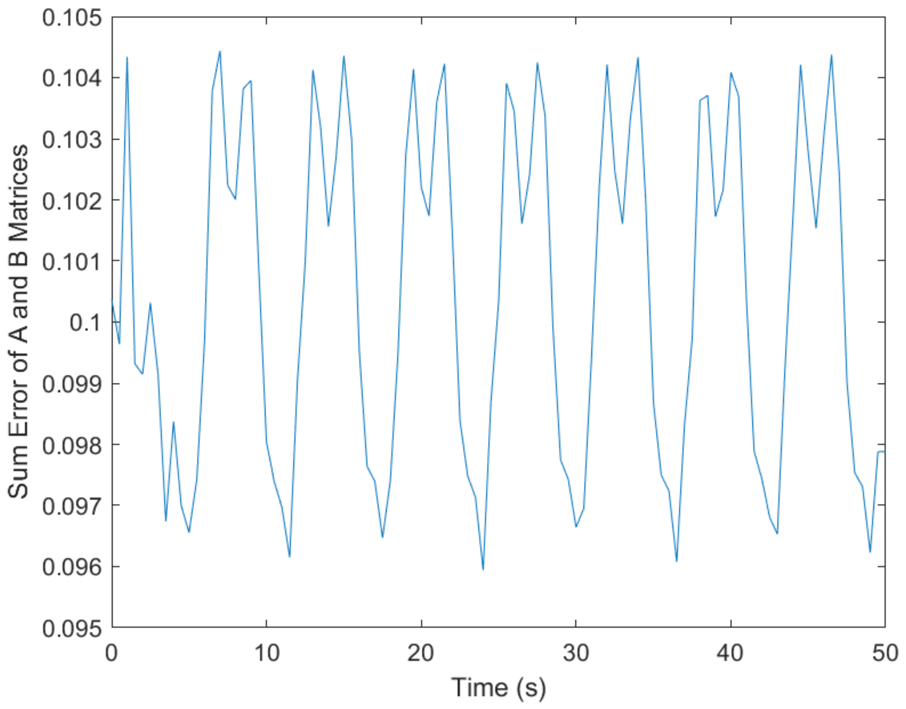


Figure 3.49 0.5 Second Sum of Errors (Modified Reward Function, High Constant Values)

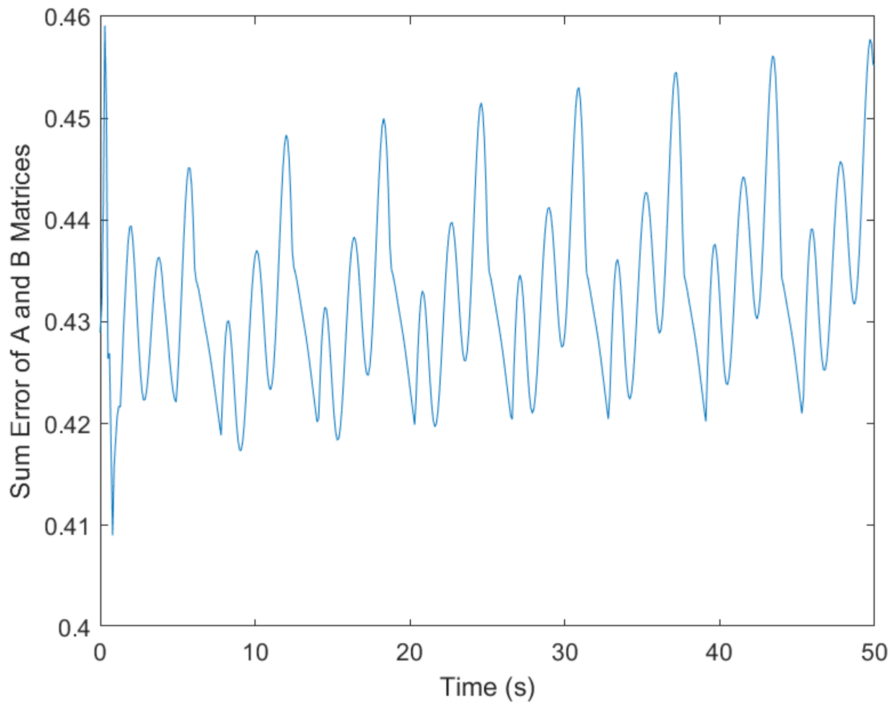


Figure 3.50 0.1 Second Sum of Errors (Modified Reward Function, Triple Low Constant Values)

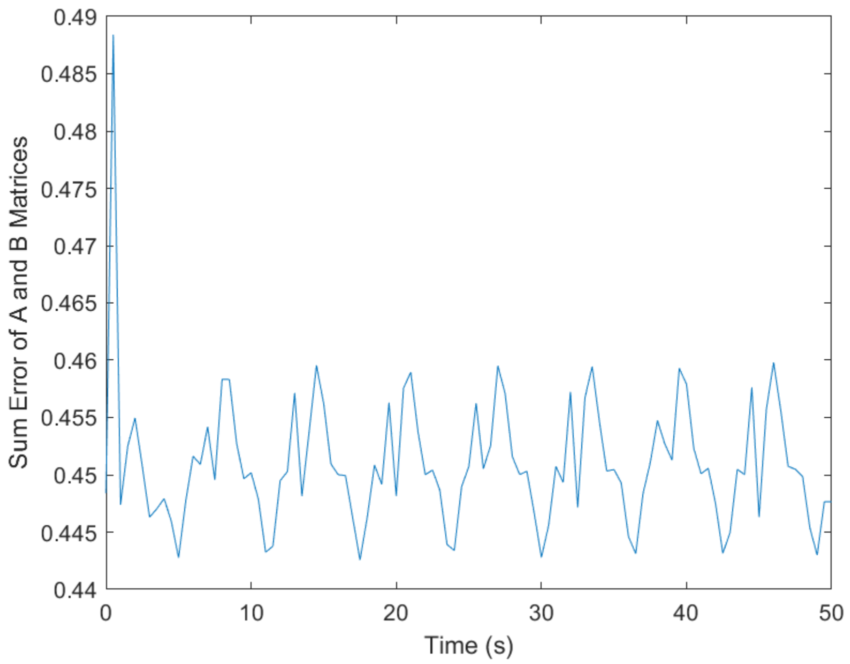


Figure 3.51 0.5 Second Sum of Errors (Modified Reward Function, Triple Low Constant Values)

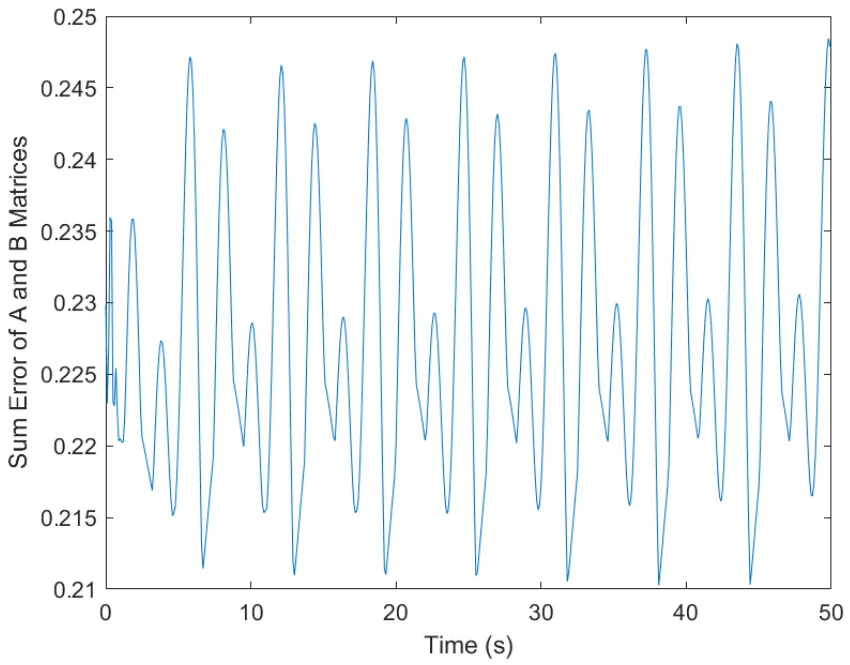


Figure 3.52 0.1 Second Sum of Errors (Modified Reward Function, Triple High Constant Values)

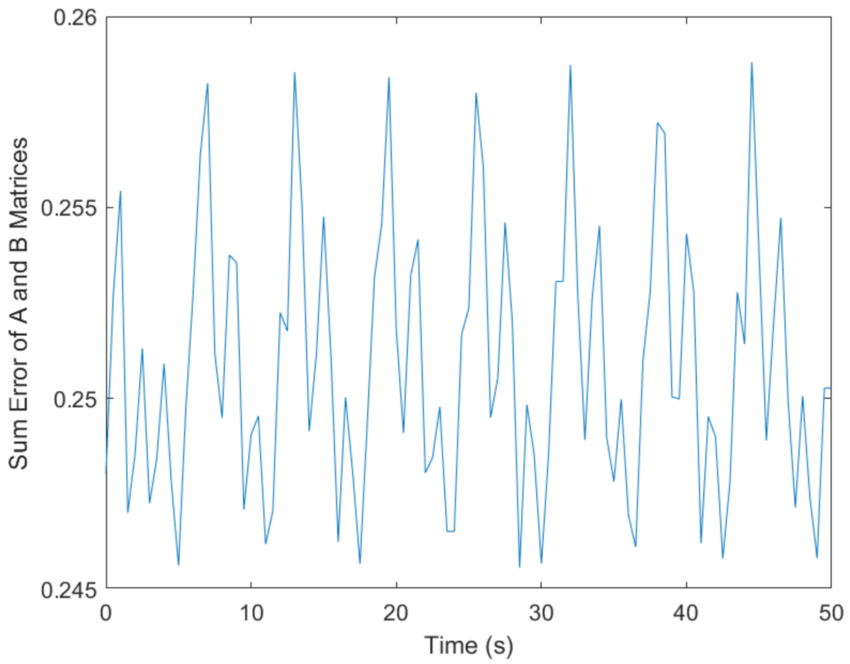


Figure 3.53 0.5 Second Sum of Errors (Modified Reward Function, Triple High Constant Values)

The most probable explanation for the minimal effect of sample time on rate of convergence is that the RL Agent does not converge to the final result in the same manner as a typical numerical algorithm, but rather it converges immediately upon access to the input data. The initial transient responses may in fact be due to the delay blocks in Simulink not being fully activated and thus the data being inaccurate. These blocks start at zero until sufficient time has passed that they are reassigned. For this test, this ’activation delay’ for the RL Agent is 1 second, and it can be seen in Figure 3.44 that the major deviations of the transient stop after 1 second.

Neural Network Architecture

The RL Agent’s primary ability to perform system identification lies in 3 hidden layers of the neural network architecture. These layers were assigned as 20 neurons each as this was found to be sufficient to achieve satisfactory results; however, there may exist superior configurations. To investigate this possibility, the number of neurons in each layer was varied and the effect on performance was observed. The results from this test are summarized in Table 3.19.

It can be seen in Table 3.19 that the neuron count per layer plays a significant role in the performance of the RL Agent at neuron counts less than 5. When the neuron count is below this value, the average error for both the trained range and the triple range criteria increases considerably, with the error of the trained range for 1 neuron being over 200 times larger than the trained range error with 5 neurons. When the neuron count is increased past 5 neurons per layer, the percentage accuracy does not change significantly. For example, the performance of an Agent with 100 neurons is not superior to one with 15 neurons.

The existence of the 5 neuron threshold is likely due to the RL Agent no longer having the complexity to effectively map the value function. This is similar in nature to the findings of Kon and Plaskota [36]. A neural network, when performing a task, has a lower bound on the number of neurons required to successfully complete that task. It is probable that 5 neurons per layer is this threshold for this environment as performance decreases significantly when neuron count is below it. The data in Table 3.19 have been shown in Figure 3.54 to more clearly illustrate the impact that this lower bound has on performance. A cropped version of Figure 3.54 is shown in Figure 3.55 to show that, beyond this lower bound of five neurons, there is no clear benefit to the addition of more neurons per layer.

Table 3.19 Effect of Neuron Architecture

Neurons Per Layer	Average Error Trained Range	Average Error Triple Range	Max. Perf. Episode	Maximum Reward
1	117.4523%	115.4125%	26	-1705.3
2	48.7983%	53.7361%	4807	-624.1528
3	14.3001%	24.0776%	669	-183.5785
4	1.4915%	6.1445%	2160	-16.8640
5	0.5637%	3.1982%	1244	-7.1807
6	0.9322%	4.7644%	5758	-11.7201
7	0.67552%	3.9369%	2051	-7.9873
8	0.8600%	4.2747%	7576	-11.1436
9	0.6022%	4.2262%	1198	-6.4388
10	0.6572%	4.4907%	1760	-7.1178
15	0.51017%	3.5466%	1084	-7.4278
20	0.7801%	4.0621%	4728	-10.2685
25	0.6659%	4.4961%	1180	-8.9347
30	0.6689%	3.4032%	7984	-8.7131
35	0.6866%	4.2270%	1084	-7.8227
40	1.3066%	5.4706%	1471	-14.6719
45	0.6801%	2.8981%	4853	-7.6499
50	0.8145%	3.7938%	1301	-11.5376
55	0.5403%	2.8906%	7969	-7.5695
60	0.6588%	3.5952%	1106	-9.4253
65	1.0880%	6.4320%	1937	-9.2964
70	0.8757%	3.5096%	976	-10.4806
75	1.4684%	5.1845%	2160	-15.5131
80	0.8060%	3.5366%	1059	-10.2402
85	0.7472%	3.9555%	1340	-8.6140
90	0.6272%	3.7322%	6780	-8.8105
95	0.5371%	2.6908%	5197	-7.8741
100	0.7341%	3.5323%	5017	-8.8868

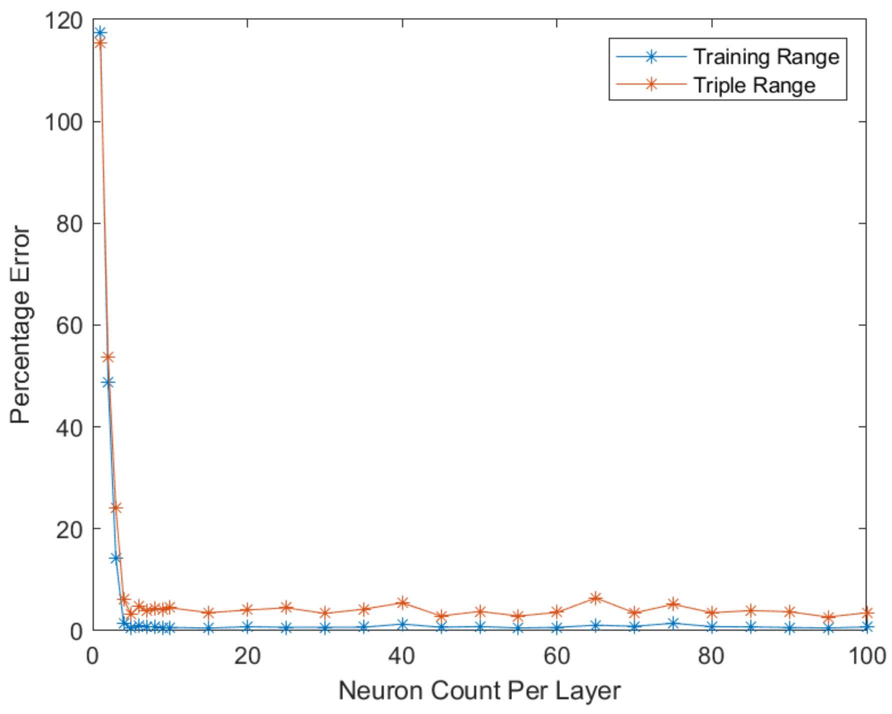


Figure 3.54 Effect of Neuron Count Per Layer on Performance

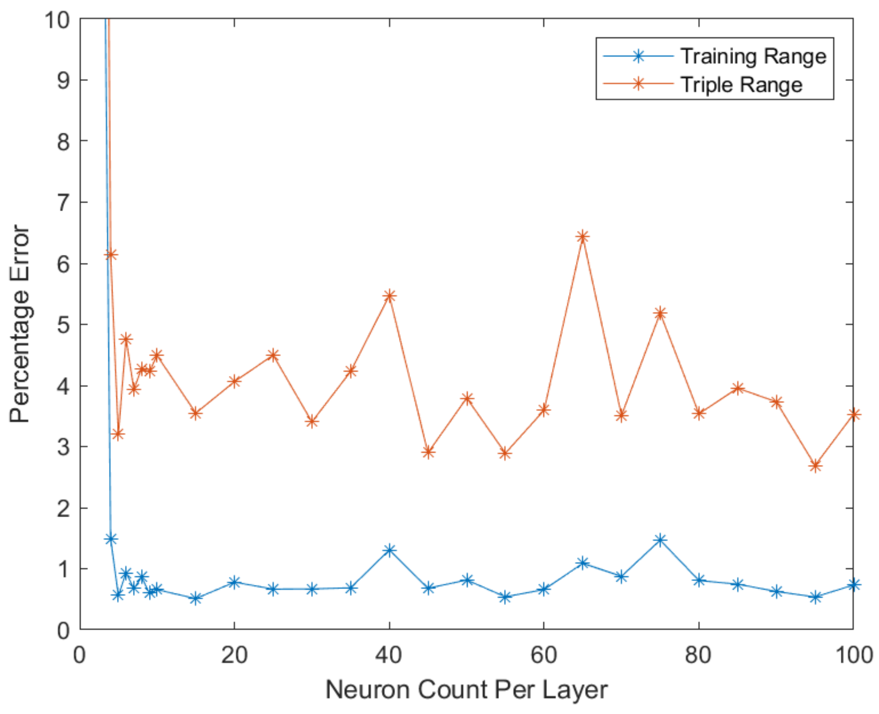


Figure 3.55 Effect of Neuron Count Per Layer on Performance - Reduced Axes

While increasing the neuron count in these hidden layers beyond 5 neurons did not impact performance, it did increase the complexity and computational requirements of the training and testing of the RL Agents. This increased the time required to train the Agents and may impact the ability for the Agents to be implemented in real time on a physical system. Thus for practical implementation, this test shows that once the minimum neuron count has been reached, the neuron count should not be increased further with the understanding that this will improve performance. It will only make the system less practical for implementation by increasing computational requirements.

Discount Factor

For a Reinforcement Learning Agent, the discount factor is a hyperparameter that affects whether the Agent prioritizes immediate rewards or delayed rewards. A discount factor of 1 will not impact the reward values while a value of less than one will guide the RL Agent to prioritize immediate rewards. This can be useful in tasks where early costs can lead to greater rewards later. For this task, the benefit of prioritizing immediate reward is unlikely to have any meaningful effect as the nature of the problem does not change with time. Any benefits caused by a quick convergence to the true A and B matrices should be adequately addressed by the reward function.

This expected trend is shown to be true in Table 3.20, shown visually in Figure 3.56. The discount factor has little impact on overall performance within the trained range as well as within the triple range. This is most likely due to the simplistic nature of the problem. The RL Agent's task is to reduce the difference between its estimated A and B matrices and the true A and B matrices as much as possible. There likely exist very few to no RL Agent 'decisions' with a short term cost causing a greater long term reward. Environments where these decisions are present is why the discount factor exists as a hyperparameter as altering the discount factor would provide the ability to prioritize the longer term reward and thus impact the accuracy.

Table 3.20 Discount Factor

Discount Factor	Average Error Trained Range	Average Error Triple Range	Max. Perf. Episode	Maximum Reward
1.0	1.0173%	20.8341%	392	-163.2839
0.95	0.8823%	4.0335%	1223	-9.4629
0.90	1.1260%	5.1486%	3704	-12.6133
0.85	0.5761%	2.8806%	5767	-7.9729
0.80	0.4712%	4.0550%	1191	-6.7614
0.75	0.6659%	3.4846%	1249	-7.2161
0.70	0.6055%	4.6472%	1122	-7.6588
0.65	1.3153%	5.4576%	3573	-13.7100
0.60	0.6199%	3.2766%	1352	-6.4159
0.55	0.5351%	3.9454%	1253	-6.3396
0.50	0.5429%	3.6746%	1201	-5.7554
0.45	0.4605%	3.0664%	1288	-5.3871
0.40	0.5437%	4.1352%	1411	-6.3960
0.35	1.0883%	4.3580%	4413	-11.7137
0.30	0.8922%	3.9381%	6522	-9.0571

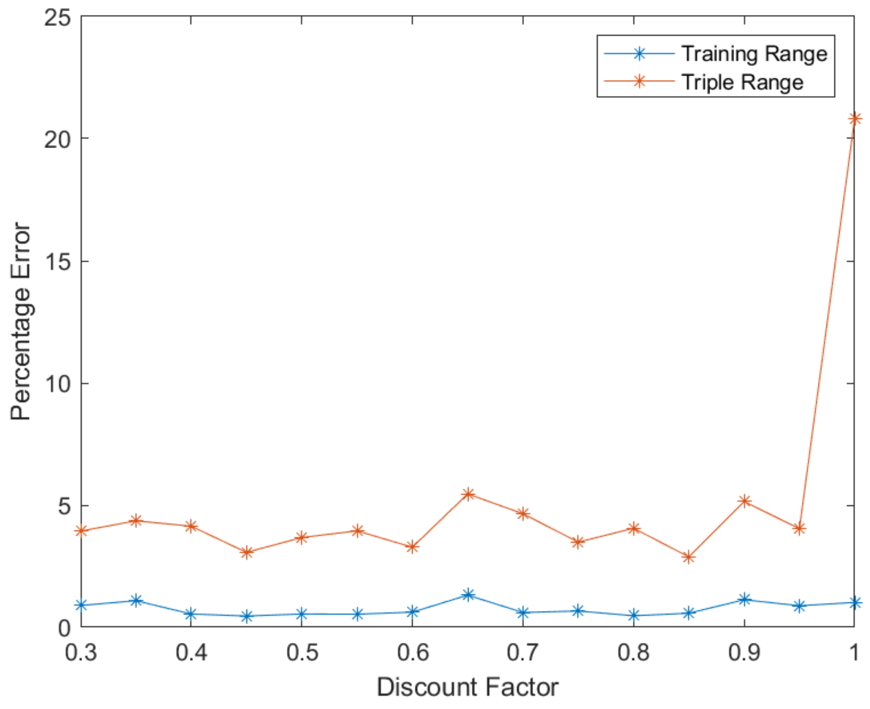


Figure 3.56 Effect of Discount Factor on Performance

Noise Variance and Noise Decay Rate

When learning, RL Agents rely on a concept known as exploration vs. exploitation. An RL Agent must explore possible value functions to find the optimum but must also exploit what it has encountered once it has found one that is ideal or close to ideal. This is accomplished on a mathematical level for DDPG networks with noise within the neural network's actions. The RL Agent's actions are the product of the network itself and the input data that it receives; however, before the output is enacted on the environment, noise is added into the output thereby causing the Agent to explore a range of different options. This noise influences the RL Agent's exploration vs. exploitation where higher noise corresponds to a greater priority placed on exploration. In order for the RL Agent to ultimately converge on one value function, the impact of this noise decays with training.

Altering this noise may impact the training and thus the performance of the Agent. The presence of noise may prevent the RL Agent from converging on non-ideal local minima and force the RL Agent to explore more, thereby finding the globally optimal strategy. The RL Agent's noise is defined by two hyperparameters, the variance and the decay rate. Given that, changes to the noise variance and its decay rate have been made and the effect of performance observed. These were both varied in their values along powers of 2 to observe the impact that they have on performance.

In the first test, the noise variance was altered while the decay rate was maintained at its value of 10^{-5}. The results of this test are shown in Table 3.21 where it can be seen that alterations to the noise variance do not appear to impact performance significantly. This would indicate that the RL Agent is able to find the ideal or near-ideal policy without requiring a substantial amount of exploring. For MSD SYSID, this is plausible as the problem is not complex. For this system, noise variance does not appear to be of great significance; however, on other more complex systems, it may be more important. The data from Table 3.21 have been presented in Figure 3.57.

Table 3.21 Effect of Noise Variance

Noise Variance	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Episode	Maximum Reward
2^3	0.8324%	5.1515%	2847	-11.6906
2^2	0.5745%	3.5302%	5731	-8.7419
2^1	1.4163%	5.9187%	2522	-16.3637
2^0	1.2094%	4.4433%	2398	-11.2355
2^{-1}	1.4394%	5.6702%	2017	-13.6594
2^{-2}	0.9292%	4.5371%	2130	-11.6052
2^{-3}	1.1403%	3.9651%	3028	-14.1251
2^{-4}	0.7343%	3.839%	4328	-9.1452
2^{-5}	0.7316%	3.4283%	4569	-9.9087

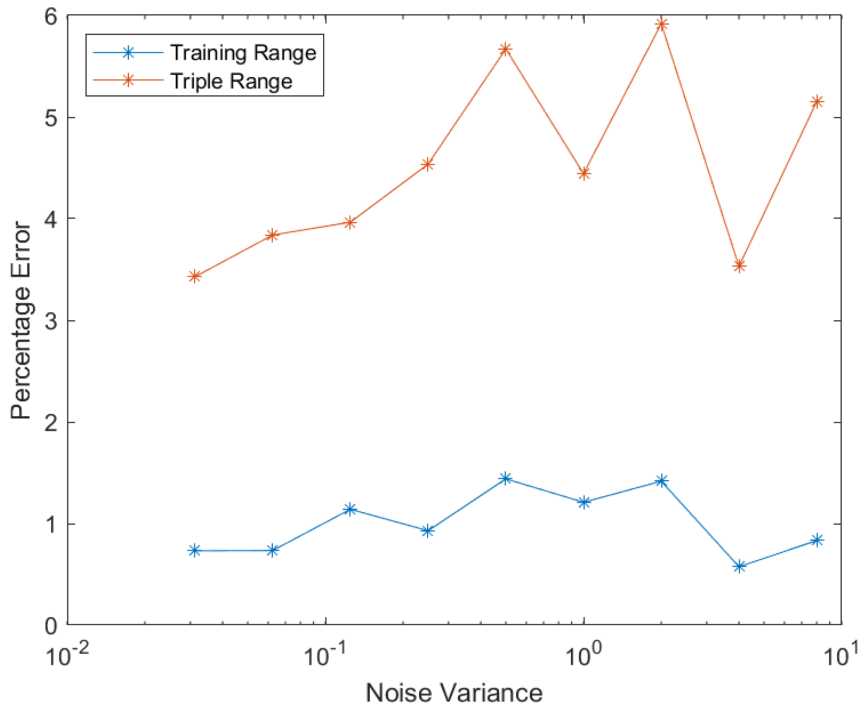


Figure 3.57 Noise Variance: Impact on Performance

When noise variance is fixed, the noise decay rate determines the importance of exploitation for the RL Agent. As the RL Agent trains, the decay rate reduces the impact of the random noise and thus leads the RL Agent to converge to a fixed value function. The effect of varying the noise decay rate can be seen in Table 3.22. Similarly to the variance, the decay rate has been assigned to a range of powers of 2. It can be seen that the decay rate does not impact the performance of the RL Agent significantly. The decay rate of 2^0 does seem to increase the error notably, but given the error seen in the 2^{-10} value, it may be an outlier.

Table 3.22 Effect of Noise Decay Rate

Noise Decay Rate	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Episode	Maximum Reward
2^0	2.6002%	6.2303%	591	-37.7748
2^{-1}	0.7687%	3.7818%	4758	-9.9919
2^{-2}	1.5014%	3.4068%	5121	-19.3172
2^{-3}	0.8370%	3.9105%	6306	-13.8286
2^{-4}	1.3111%	3.4721%	5381	-17.0258
2^{-5}	1.3141%	4.3113%	3520	-18.0116
2^{-6}	1.1650%	3.4753%	2271	-14.0532
2^{-7}	0.9891%	4.2281%	4624	-13.6886
2^{-8}	0.6866%	4.1839%	6743	-8.4960
2^{-9}	1.1222%	3.8615%	4271	-13.7597
2^{-10}	1.2996%	5.5162%	5254	-17.5571
2^{-11}	0.8189%	3.7462%	7599	-10.7555
2^{-12}	0.6833%	4.0995%	6873	-8.1551
2^{-13}	1.1199%	4.0457%	960	-15.2008
2^{-14}	1.0782%	3.1354%	7292	-15.1065
2^{-15}	0.7720%	3.8647%	5360	-10.4623
2^{-16}	0.5991%	3.7781%	782	-9.2520
2^{-17}	0.6677%	4.5512%	1510	-9.1095
2^{-18}	0.6521%	3.6303%	5599	-9.8897
2^{-19}	0.7108%	3.6303%	6219	-9.6393
2^{-20}	0.9917%	4.3146%	6653	-27.8691

The effect of varying decay rate is best seen in Figure 3.58. The performance seems to not be significantly impacted by the decay rate after the 2^0 value. The results of this test indicate that decay rate is not a significant factor in optimizing RL Agent performance. It should be noted that decreasing the decay rate may lead to an increase in episodes required to reach a specific reward (average or episode) as it will decrease the rate at which the RL Agent converges on the final value function. This will increase the required computation time for training with no significant impact on performance and thus should be avoided.

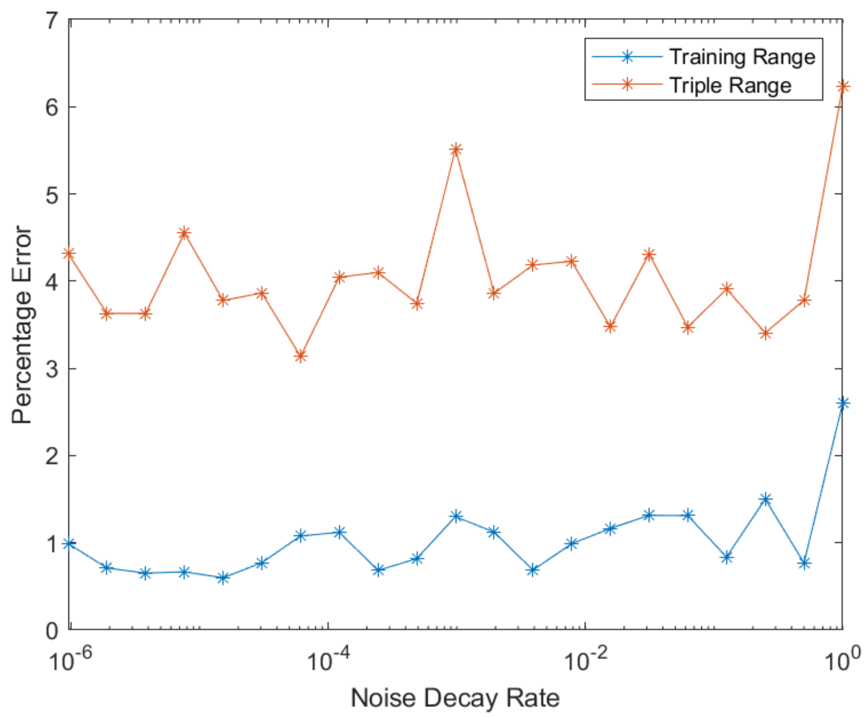


Figure 3.58 Effect of Noise Decay Rate on Performance

Observation Window Length and Interval

The RL Agents receive as input data an observation window of the current states as well as the past states of the system. The observation window is defined by 2 parameters, the number of data points or length, and the interval between each data point. Altering both of these variable influences the data that the RL Agent is exposed to and thus may have an impact on the RL Agent's performance. For this reason, the length and interval were altered and the effects on performance measured.

The difference between the original Observation Window Simulink Diagram and the augmented Observation Window Simulink Diagram can be seen in Figures 3.59 and 3.60. The Augmented version has 64 total data points as opposed to the original 10. It is hypothesized that the addition of more data points should allow for greater performance to be achieved as the RL Agent will have access to more information about the system at each sample time.

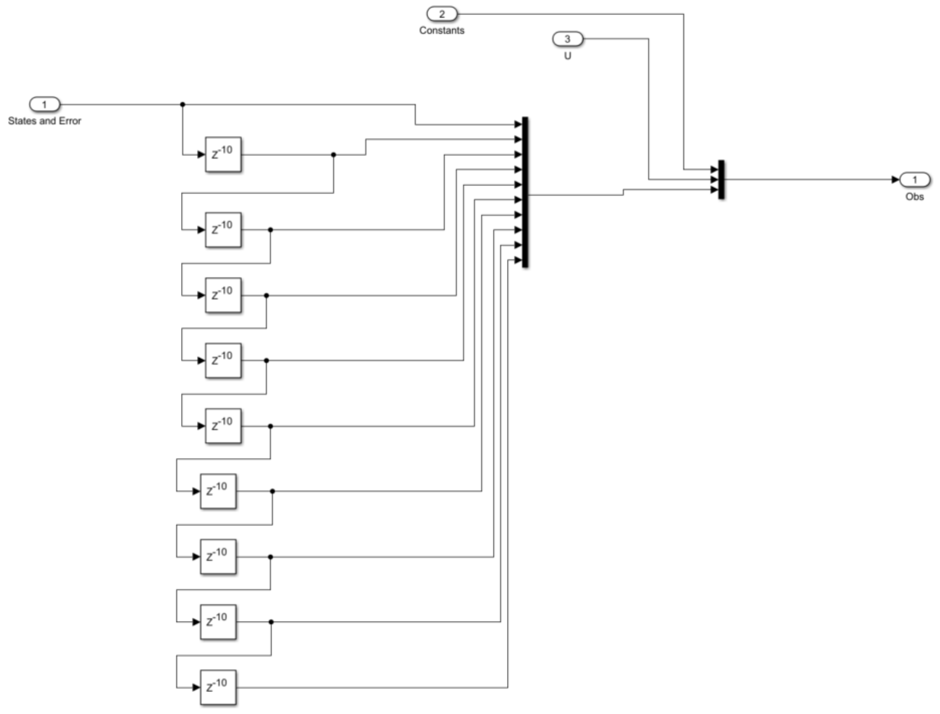


Figure 3.59 Original Observation Window

The increase of input data can be seen in Figures 3.61 and 3.62. It should be noted that the augmented Observation Window in Figure 3.62 corresponds to an untrained RL Agent and thus its error data are not reflective of that of a fully trained RL Agent. These error data are the clustered data points with a ribbon-like appearance that diverges. It can be seen across these figures that the standard Observation Window encapsulates a much smaller period of time than the 64 Length Augmented Window.

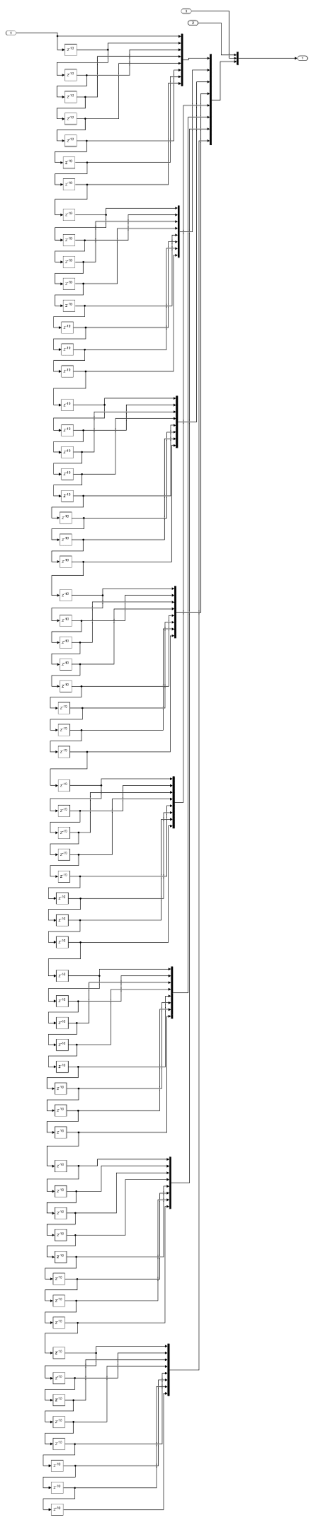


Figure 3.60 64 Length Augmented Observation Window

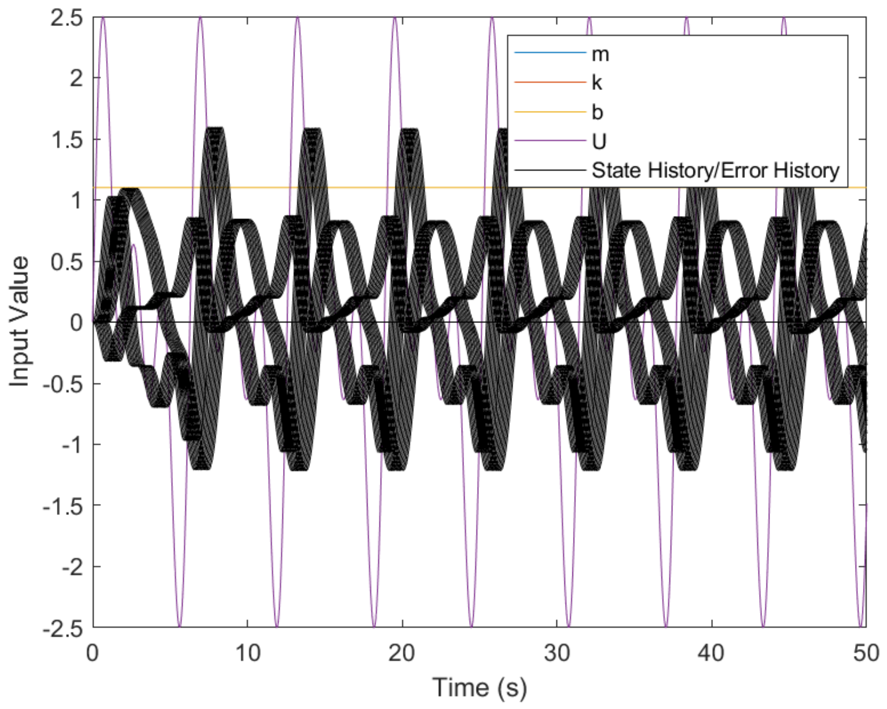


Figure 3.61 Original Observation Window Input Data

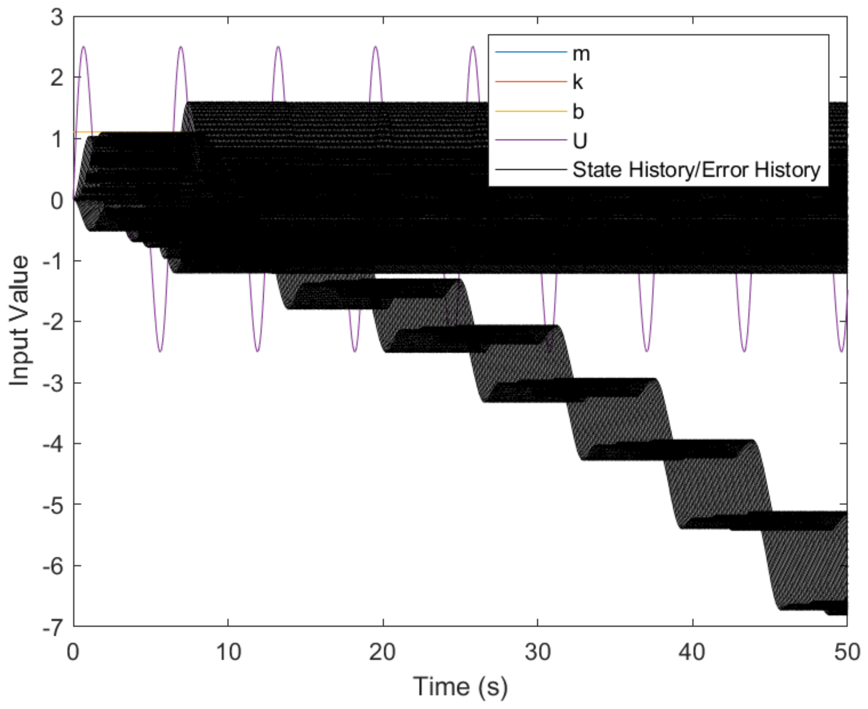


Figure 3.62 64 Length Augmented Observation Window Input Data

Increasing the size of the Observation Window does not appear to have a positive effect of RL Agent accuracy, as is evidenced in Table 3.23 where both Reductions and Augmentations to the RL Agent's Observation Window Length from its original value of 10 appear to have little to no impact on accuracy. When graphed in Figure 3.63, it can be seen that increases to the Observation Window Length appear to increase the average percentage error in both the trained range and the triple range. This runs contrary to the hypothesis that an increase in information provided to the RL Agent will increase its accuracy. These data would indicate that the Observation Window may add unnecessary information to the RL Agent that cause it to reach sub-optimal results.

Table 3.23 Effect of Observation Window Length

Observation Window Length	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Ep.	Maximum Reward
1	0.4906%	2.9154%	2204	-6.5528
2	0.5303%	3.2579%	4252	-6.4982
4	0.5044%	4.3880%	1281	-5.7308
8	0.7738%	3.5927%	5024	-9.5890
16	0.5731%	2.5453%	7619	-6.0819
32	0.8947%	4.6454%	1644	-12.3833
64	1.7296%	6.5237%	1268	-21.1884

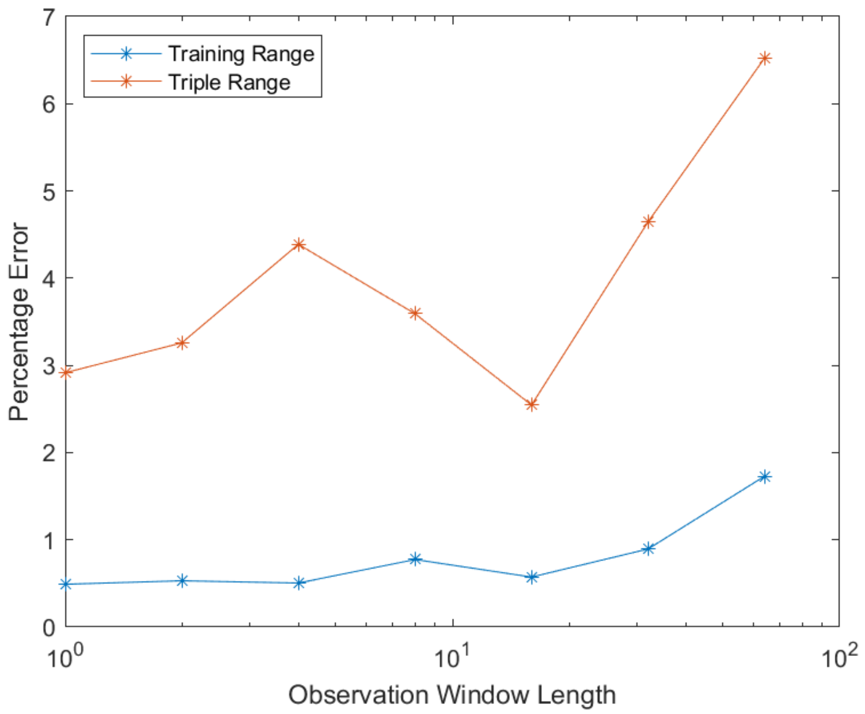


Figure 3.63 Effect of Observation Window Length on Accuracy

The observation window can also be altered by changing the interval between the delay cells. This allows the observation window to span a greater time period without increasing the computation requirements. To accomplish this, the Simulink implementation was altered from Figure 3.59 to 3.64.

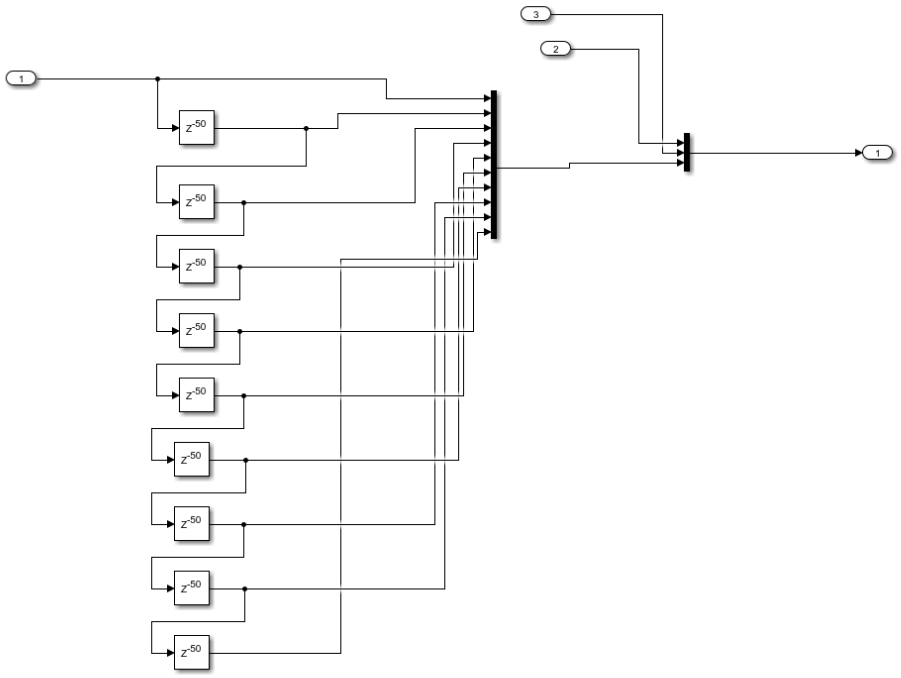


Figure 3.64 Simulink for 0.5 s Interval

The effect of altering the interval can be seen in Figure 3.65 and Table 3.24. In Figure 3.65 it can be seen that the interval allows the window to cover a larger window of time but with fewer data points. This is a different effect than that of the observation window length. Changing the interval alters the sampling frequency of the observation window much like using a different data rate on a sensor. The data provided in Table 3.24 indicate that there is no significant impact caused by changes to the observation window interval. This is consistent with the findings from the observation window length test in which more information did not increase accuracy. It would seem that the observation window is not of significant importance or value to the RL Agent and may not be required at all. This is likely due to the influence of PIRL negating the positive benefits of the observation window. When the RL Agent is able to directly observe the specific quantities that govern the system, it has little reason to place value on the state histories that only indirectly provide the same information.

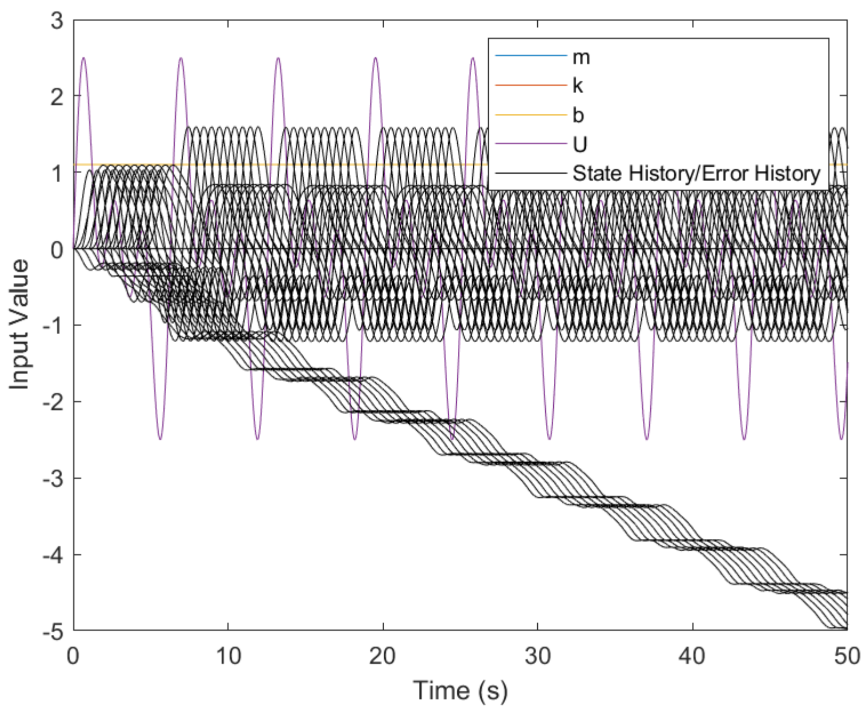


Figure 3.65 0.5 s Interval Observation Window Input Data

Table 3.24 Effect of Observation Window Interval

Observation Window Interval (s)	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Ep.	Maximum Reward
0.1	0.6369%	3.8204%	1164	-7.1106
0.2	1.4699%	4.9993%	1419	-14.1313
0.3	1.2924%	5.0317%	1623	-14.3071
0.4	0.5064%	3.7922%	1162	-6.6261
0.5	1.2831%	5.3159%	1528	-15.0392

3.3.3 Effect of Environmental Variables

An RL Agent's performance may be influenced by changes to its environment. Such changes, such as the range of the variation of the constants of the mass-spring-damper, the effect of sensor noise, and the effect of reduced parameters, could have a significant impact on the ability of the RL Agent to identify the system. For this reason, tests were conducted to observe the effects of these changes in environment.

Effect of Range of Variation

For all previous tests, each parameter has been altered by a maximum of plus or minus 10% (or 0.1) during training. During testing, this is kept consistent for the trained range criterion and then tripled for the triple range criterion. The purpose of the triple range criterion is to determine if the RL Agent is effective when exposed to data outside of the range of what it has observed. The 0.1 range was chosen as it encompassed a variety of systems; however, in a practical application of this system, the RL Agent may be applied to systems with significantly more variation. For this reason, the range of variation was expanded and the effect on percentage error observed. The results of this test are summarized in Table 3.25. It is evident from these data that the RL Agent is able to maintain substantial accuracy within its trained range. The triple range increases with error more significantly as is expected given that the triple range is increasing at triple the rate of the trained range. Despite this, the RL Agent still maintains less than 10% error. These data speak to the effectiveness of PIRL for SYSID.

Table 3.25 Effect of Greater Variation in Parameters

Variation Limit	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Ep.	Maximum Reward
0.0	0.1298%	0.1298%	4609	-1.6913
0.1	0.8914%	2.7442%	870	-13.8380
0.2	0.6389%	4.1848%	1126	-7.9597
0.3	0.9592%	7.7145%	2080	-9.3449

To better understand the influence of PIRL on the RL Agent, the test was repeated without PIRL and the results of this test are presented in Table 3.26. From these results, much like the previous test, it is evident that the percentage error increases as the range of variation increases. In the trained and triple range, PIRL outperforms the non-PIRL/conventional RL Agent at all data points, except 0.0 (no variation).

Table 3.26 Effect of Greater Variation in Parameters - Conventional RL

Variation Limit	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Ep.	Maximum Reward
0.0	0.06999%	0.6999%	2330	-1.5953
0.1	3.0791%	9.2511%	5366	-35.1741
0.2	5.5971%	16.7283%	6691	-68.3756
0.3	9.9655%	27.2288%	714	-112.6828

Effect of Noise on performance

If PIRL were to be implemented on a real dynamical system, it is most likely that there would be noise in the input data. This would occur inconsistently as parameter data would not have noise, as they would be derived from easily known and unchanging quantities. The input data also would not feature noise as they would correspond to a pre-defined sinusoidal input. The only input with noise would be that of the system states/observations and the error between the RL Agent's trial system's states/observations. To simulate the effect of noise, an AWGN (Add White Gaussian Noise) block was added into the RL Agent's states/error signal, given a random seed, and set to a constant predefined signal to noise ratio as measured in decibels (dB). This signal to noise ratio (SNR) was then varied from 40 dB to 10 dB. The impact of the 40 dB noise can be seen in Figure 3.66. It should be noted that this Figure was generated with only a partially trained RL Agent and thus the diverging error signal is not reflective of a fully trained RL Agent. In the Figure, the impact of the noise is minimal and the signal (the state trajectories) is clearly evident.

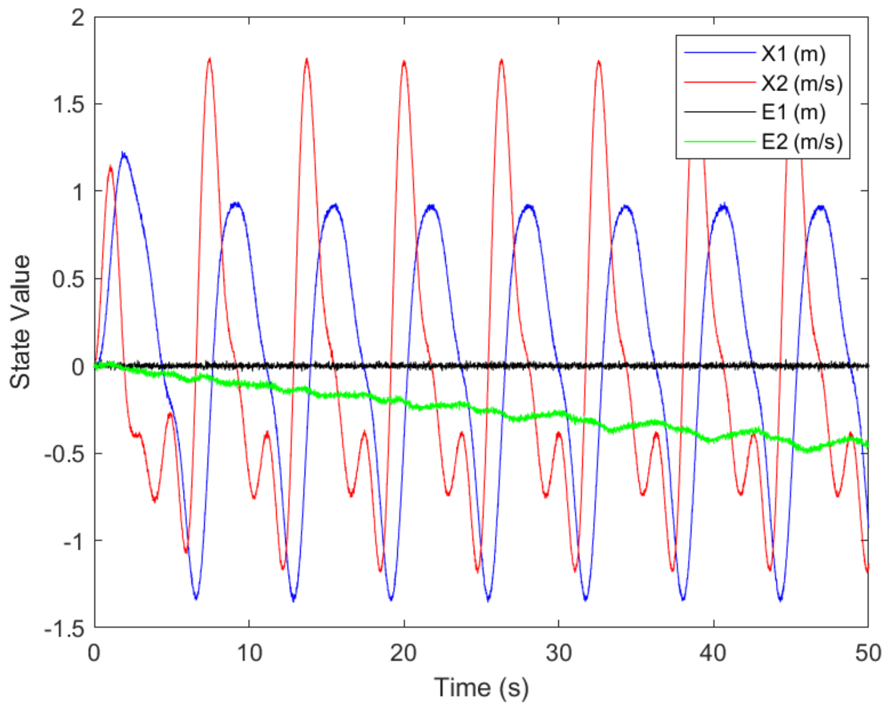


Figure 3.66 SNR 40dB

Figure 3.67 shows the effect of a 10 dB SNR. At this increased level of noise, the state and error trajectories are much more difficult to discern. It is hypothesized that this should cause the RL Agent to see a decrease in performance and an increase in error.

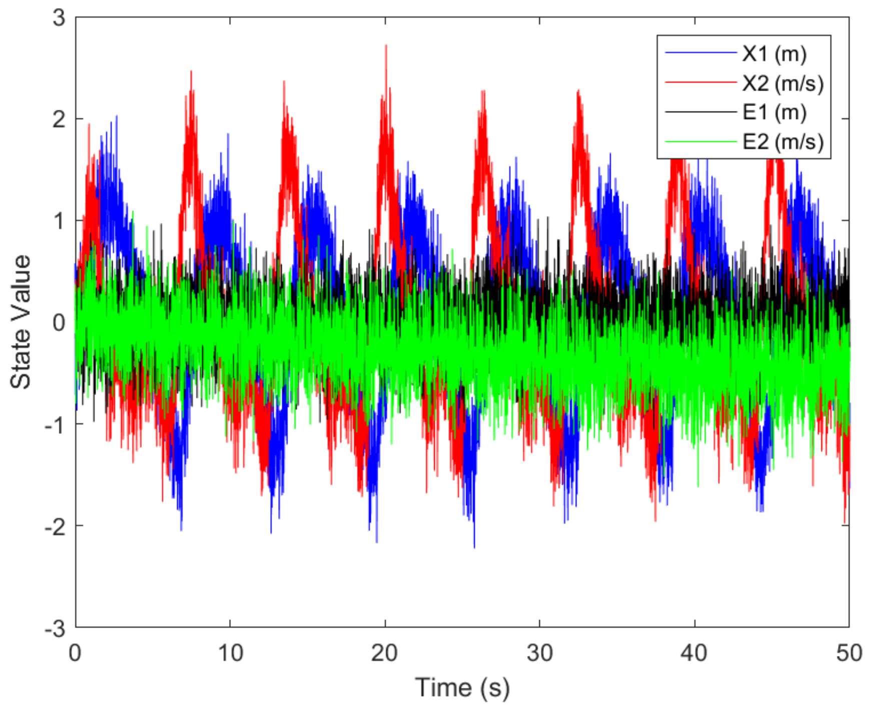


Figure 3.67 SNR 10dB

The results from the test disprove this hypothesis, as is evident in Table 3.27. The addition of Gaussian white noise has no discernible impact on the accuracy of the RL Agent at any tested noise level. The error with added noise is not notably different from the tests without noise. This is most likely due to the influence of PIRL. The RL Agent is able to be informed by the state trajectories, but ultimately does not require them as the parameters contain within them sufficient information to fully define the system. This may imply that the RL Agent may experience a decrease in performance caused by the presence of noise if any of these parameters are missing or if the relationship between parameters and system is sufficiently complex as to not be fully encapsulated by the RL Agent's neural network.

Table 3.27 Effect of Sensor Noise

Signal to Noise Ratio (dB)	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Ep.	Maximum Reward
40	0.6766%	3.9348%	4301	-8.5777
30	1.2457%	4.6325%	2149	-13.6742
20	0.8576%	4.8430%	1425	-10.2590
10	0.7229%	3.4888%	4741	-8.3063

Reduced Parameters

For the studies performed in this chapter, the RL Agent has been informed by the mass, spring-constant, and damper-constant. The objective of this investigation was to provide a preliminary understanding of the performance of PIRL and its possible benefits. In these initial tests, the RL Agent is provided with a close to ideal set of inputs. This may not be possible on other systems where the full scope of the system may not be captured by easily known parameters. In order to better understand how PIRL's performance is impacted by imperfect parameters, the parameters that were provided to the RL Agent were altered as can be seen in Table 3.28.

Table 3.28 Effect of Reduced Parameters for PIRL

Parameters	Average Error Trained Range	Average Error Triple Range	Maximum Perf. Ep.	Maximum Reward
m,k,b	0.7080%	5.3145%	1146	-8.3616
k,b	5.4208%	16.7332%	1146	-56.2411
m,b	2.2583%	7.8231%	7335	-23.2263
m,k	2.2310%	8.2912%	1216	-24.3703
m	3.7323%	12.9809%	3487	-37.1458
k	6.2881%	18.1904%	1718	-61.8511
b	5.9873%	19.0423%	900	-60.7773
None	6.2181%	16.5749%	2040	-70.6657

It can be seen in Table 3.28 that removing parameters has a significant effect on performance. It can also be seen that certain parameters had a greater effect than others. When mass is removed, the performance decreases to roughly similar to the conventional RL result and is in fact slightly less accurate. When only the damping constant is provided, PIRL seems to be less accurate than without any information although this may be due to statistical noise. This has the greatest effect on the triple range criterion. These data would indicate that the mass is the most important factor in accurately determining the system. Given that the mass directly impacts A21, A22, and B21, this is expected. The impact of knowledge of the spring or damper constant appears to be roughly similar. This demonstrates that, as would be expected, the value of a parameter to the RL Agent may be closely related to the relevance that parameter has to the A and B matrices.

3.4 Conclusion

Implementation of PIRL in the MSD environment has illustrated a number of key attributes of PIRL with regard to its use in system identification. PIRL has been shown to be effective at performing system identification both within its trained range as well as within the triple range criterion. It has been shown to produce more accurate models than an EKF and it does this with a convergence time of only a few seconds at most. Additionally, PIRL performs better than conventional RL models on the MSD system, which supports the hypothesis that additional parameters provides a better RL performance. This chapter has also outlined a rough guide regarding which hyperparameters have the most significant role in the performance of the Agents and thus which ones to concern oneself with when optimizing the RL implementation. Notably, noise variance, noise decay rate, discount factor, and observation window length and intervals all had minimal impacts, while architecture exhibited a clear lower bound, and sample time was relevant mostly to the design of the reward function and training threshold. PIRL was found to perform well even when the range of variation was increased, and while there did exist a divergent relationship between variation range and accuracy, PIRL continued to outperform conventional RL under these conditions. Finally, PIRL was shown to maintain some effectiveness even when some of the parameters were not provided to the Agent. There was an increase in error when parameters were reduced, but the addition of any of the chosen parameters always led to a decrease in average error in the trained range and mostly decreased the error in the triple range (some trials saw a slight increase in error). Additionally, this test demonstrated that some parameters will have a greater impact on reducing error than others. Mass was found to be the most significant parameter and this is likely due to it being a factor in all unknown system variables. This chapter ultimately illustrates that PIRL has merit as a novel system identification algorithm and appears to perform better than EKFs and conventional RLs at the role of identifying the A and B matrices from a range of similar mass-spring-damper systems.

4 Environment 2: Longitudinal Aircraft Model

An aircraft's motion can be simplified into two separate decoupled and linearized dynamical systems: lateral and longitudinal. Barring significant changes from the linearized state, these equations define the dynamic behavior of an aircraft reasonably well.

The longitudinal system encompasses forward speed, vertical speed, pitch, pitch rate, and sometimes altitude. The system is linearized from the aircraft's trim conditions and thus all states represent differences with respect to their trim conditions. The Longitudinal Flight Model (LFM) was chosen as it is an increase in complexity from the MSD and is more relevant to the target application of SYSID for fixed wing aircraft.

4.1 System Model

The state-space model of the LFM is given in Equation 4.1 [37]:

$$\begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} X_u & X_w & X_q & -g\cos\theta_* & 0 \\ Z_u & Z_w & Z_q & -g\sin\theta_* & 0 \\ M_u & M_w & M_q & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -\sin\theta_* & -\cos\theta_* & 0 & u_*\sin\theta_* + w_*\cos\theta_* & 0 \end{bmatrix} \begin{bmatrix} u \\ w \\ q \\ \theta \\ h \end{bmatrix} + \begin{bmatrix} X_{\delta e} & X_{\delta t} \\ Z_{\delta e} & 0 \\ M_{\delta e} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_e \\ \delta_t \end{bmatrix} \tag{4.1}$$

where δ_e and δ_t are elevator and throttle inputs, u is the forward velocity, w is the vertical speed, q is the pitch rate, θ is the pitch angle, h is the altitude, $X_u, X_w, X_q, Z_u, Z_w, Z_q, M_u, M_w$ are the aircraft stability derivatives, and g is the acceleration due to gravity. It should be noted that asterisks denote the trim conditions and that all states are differences measured from the trim conditions. For example, a value for u of -10 indicates that the aircraft is flying at -10 ft/s slower than trim and not 10ft/s backwards.

For the purposes of this environment, some of the complexity of the model has been removed. For Environment 2, altitude as a state and thrust control have been removed. Altitude was removed as all relevant altitude A and B terms could be determined by knowledge of the trim conditions and thus would not be relevant to the RL Agent's task. Thrust control was removed to simplify the model as Environment 2 serves only as a further proof of concept. The reduced system can be seen in Equation 4.2.

$$\begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} X_u & X_w & X_q & -g\cos\theta_* \\ Z_u & Z_w & Z_q & -g\sin\theta_* \\ M_u & M_w & M_q & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ w \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} X_{\delta e} \\ Z_{\delta e} \\ M_{\delta e} \\ 0 \end{bmatrix} \begin{bmatrix} \delta_e \end{bmatrix} \tag{4.2}$$

The RL Agent is tasked with finding certain terms in the A and B matrices. These values are $A22$, $A23$, and $A33$ and $B1$, $B2$, and $B3$. These specific values were selected as they could not be determined through knowledge of the trim conditions such as the case for $A14$ and $A24$. Additionally, $A22$, $A23$, and $A33$ were the largest, most influential values in their respective states in the training data. Initially, the goal was to determine the values of the upper left 3x3 square of the A matrix and the upper 3 values of the B matrix, but this was reduced because of its significant increase in complexity from the MSD environment and the difficulty vs benefit of determining some of the less impactful derivatives.

4.2 Training and Testing Data

Unlike the previous environment, the A and B matrices of the LFM cannot be assigned randomly along fixed ranges. The values of each matrix are dependent on the aerodynamics of the aircraft and thus must be generated through some form of simulator or through real-world data. Given the large quantity of data required for training an RL Agent and the expense of collecting it, obtaining sufficient data can be one of the largest challenges of AI systems implemented on aircraft. For this environment and for both training and testing data, the A and B matrices were created using the United States Air Force Stability and Control Digital Datcom.

Digital Datcom was created by the United States Air Force in 1978 at Wright-Patterson Air Force Base in Dayton, Ohio. It is a computer program that performs the procedures found in the

USAF Stability and Control DATCOM [38]. In this regard, it serves as a non-CFD based calculator for determining aerodynamic coefficients. The software is computationally inexpensive to run and thus it was chosen to create the training and testing data for Environment 2.

For training, nine aircraft were used that would represent a wide variety of different aircraft, thus ideally forcing the RL Agent to perform well for any given aircraft. The test aircraft was selected as its overall size placed it roughly in the middle of the training set and would provide the RL Agent ample opportunity to perform. Training and testing aircraft are shown in Table 4.1 along with relevant size and performance parameters. Note that some values are approximations when true values could not be obtained.

Table 4.1 Training and Testing Aircraft - General Information

Training				
Manufacturer	Model	Weight (lbs)	Mom. In (I_Y) ($slugft^2$)	Wingspan (ft)
Beechcraft	99	10000[39]	15148[40]	46[39]
Boeing	737	115000[41]	2500000	112[41]
Beechcraft	T-34C	4000[42]	1700	33[42]
Cessna	182	2950[43]	1500	26[43]
Cessna	Citation I	10000[44]	25000	47[44]
Cessna	Citation II	8000[45]	23216[46]	52[45]
Ryan	Navion	2750[47]	2772[48]	33[47]
North American	P-51	8000[49]	23000	37[49]
Northrop	T38	10500[50]	24000[50]	25[50]
Testing				
Learjet	35	14740[51]	18800	39[51]

To obtain the training and testing data, each aircraft was simulated using DATCOM at a range of flight conditions. Three loops were used to this effect, with the first being a range of Mach numbers, the second being a range of flight path angles, and the last being a range of altitudes. The Mach numbers were given boundaries defined by reasonable airspeeds for the aircraft at its given altitude, the flight path angles were given boundaries of 0 degrees to 20 or 15 degrees depending on the aircraft's capability, and finally the altitude was given boundaries that the aircraft could reasonably perform. Any flight conditions that led to abnormal and likely erroneous information were discarded by filtering based on the uncontrolled longitudinal stability eigenvalues. If an aircraft had a maximum real eigenvalue of greater than 0.3, it was discarded, as aircraft with eigenvalues greater than this diverged before the training episode could be completed. These stability data were generated and saved. The resultant training data were comprised of 28,507 flight conditions across the set of training aircraft, and the testing data were comprised of 2048 different flight conditions across its one aircraft.

4.3 Implementation of Reinforcement Learning

Similar to Environment 1, the RL Agent's performance is impacted by several factors including its learning method, architecture, input structure, output interpretation, and a wide range of tunable hyperparameters.

4.3.1 Learning Method (DDPG)

The RL Agent was again chosen to be a Deep Deterministic Policy Gradient (DDPG) Agent. This is a model-free, actor-critic RL method, unique in that it offers continuous outputs as opposed to discrete. This is optimal for SYSID as the outputs of the RL Agent are continuous values in the A and B matrices. DDPG networks also support continuous or discrete observation spaces, which is well suited to the task of time-series based SYSID.

4.3.2 Network Architecture

The Network Architecture is important to the effectiveness of the Agent. As was illustrated in Environment 1, there can exist a lower bound on the RL Agent hidden layer size. Without this necessary number of neurons, performance of the Agent decreases significantly. To avoid this outcome, the architecture and size of the hidden layers were altered significantly. The Critic and Actor networks are each composed of 7 hidden layers, starting at 500 neurons and gradually decreasing until it reaches 20 neurons. Each layer is interspersed with a hyperbolic tangent layer to ensure network stability. The MSD Actor originally was interspersed with reLu layers, but this was altered to hyperbolic tangent layers to allow for the propagation of negative values that would be required for the output layer.

4.3.3 Implementation of PIRL

Parameter Informed Reinforcement Learning is again implemented within the RL Agent's observations. The chosen parameters that are deemed relevant to the system model and can be obtained easily are added in to the RL Agent's inputs at each time-step. Unlike the MSD, the LFM parameters span sufficiently large ranges that their values are normalized between 0 and 1. The parameters were selected as Altitude (ft), Mach Number, Flight Speed (ft/s), Moment of Inertia (I_Y) (slug ft^2), Angle of Attack (rad), Pitch Angle (rad), Flight-Path-Angle (rad), Position of Center of Gravity (ft), Position of Center of Gravity Measured from Mean Aerodynamic Chord, Wing Surface Area, and Aircraft Weight. These were all selected as they embody possible effects caused by changes from one aircraft to another or from one flight condition to another.

4.3.4 RL Agent Inputs

Identically to the MSD, for the LFM, the RL Agent observes a history of the states in 0.1 second intervals. It also observes a history of the errors in the same manner. The error history is calculated by observing the differences in the true system states and the trial system states. It should again be noted that the trial system states are not fed back through the system as that would effectively be assigning the RL Agent the task of SYSID as well as driving the system by altering its parameters to the true system if its estimates have caused it to deviate. This is outside the scope of SYSID. The error calculation method is illustrated in Figure 4.1.

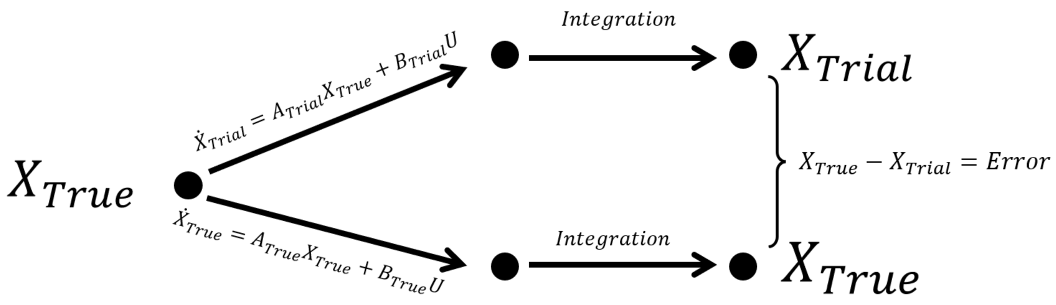

X_{True}
$\dot{X}_{Trial} = A_{Trial}X_{True} + B_{Trial}U$
$\dot{X}_{True} = A_{True}X_{True} + B_{True}U$
$Integration$
$Integration$
X_{Trial}
X_{True}
$X_{True} - X_{Trial} = Error$

Figure 4.1 Error Calculation Method

It should be noted that, in a real system, state histories may not be obtainable and one must use output histories or a state estimator. This is not considered for this system and thus the C matrices are assumed to be identity for the true and trial systems.

Finally, the PIRL inputs are added into the observations along with the control input. The RL Agent's Observations can be seen in Figure 4.2

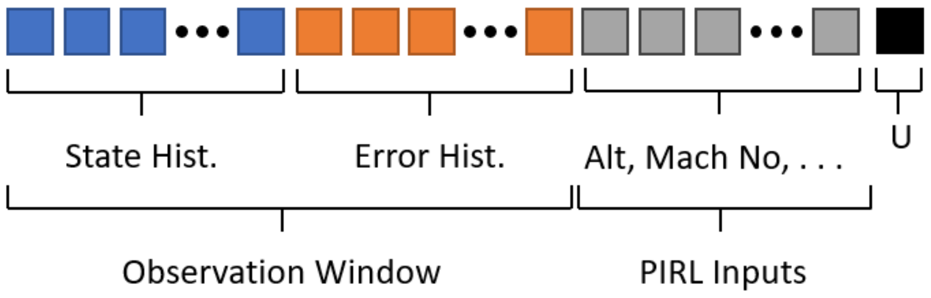


Figure 4.2 RL Agent's Observation

4.3.5 RL Agent Outputs

To perform SYSID, the RL Agent was tasked with finding $A22$, $A23$, $A33$ and $B1$, $B2$, and $B3$ from the A and B matrices of Equation 4.2. To determine these values, the RL Agent had an output of 6 values, as shown in Equation 4.3:

$$RLAgentOutput = \begin{bmatrix} A22_{Power} \\ A23_{Power} \\ A33_{Power} \\ B1_{Power} \\ B2_{Power} \\ B3_{Power} \end{bmatrix} \tag{4.3}$$

It should be noted that each RL output is denoted with a Power Subscript. This is due to the data transformation that was performed to allow the RL Agent to span the wide range of values that each unknown could possess. This transformation can be seen in general in Equation 4.4 where U_{MV} refers to the unknown value in the matrix (e.g. $A22$), M_{PV} refers to the maximum possible value that an unknown could possess (from the training data), and RLO refers to the RL Agent Output. An example is shown in Equation 4.5, where $A22$ is a transformation of the $A22_{Power}$ RL Agent output. This transformation allows the required AI outputs to lie within roughly -1 to 1 and logarithmically cover the full possible span of data. It should be noted that this transformation was a requirement for an effective Agent as conventional data normalization would often render the AI unable to estimate values that were near zero as scaling would reduce them to below that which could have a significant impact on the Agent. This method requires prior knowledge of the sign of each term.

$$U_{MV} = M_{PV}^{RLO} \tag{4.4}$$

$$A22 = A22_{max}^{A22_{Power}} \tag{4.5}$$

4.3.6 Reward Function

The Reward Function was altered for the LFM system to support the greater variation of parameters. The previous reward function was effective for the MSD because all unknowns were generally of equivalent value, whereas a percentage error reward function has to be used for the LFM. Without this, the RL Agent would prioritize certain values in the A and B matrices, which would lead to poor accuracy. The new reward function can be seen in Figure 4.3. Within the custom Matlab function, a percentage error of each desired term is calculated and the negative average of these percentages is the reward output signal. This signal is then scaled with the sample time to prevent the issues encountered in the MSD system and scaled with a hyperbolic tangent function and summed with one half[1]. The purpose of this transformation is to allow the critic network a close range with which to predict the value of each action. A memory block is also included to prevent algebraic loops from occurring. This delays the reward signal by 0.01 seconds, but given the sample time for the RL Agent of 0.1 seconds, this is not a factor in the RL Agent's learning ability. This is performing the same operations as shown in Equation 4.6

$$R = tanh(-T_s \frac{\frac{A_{22}-A_{22T}}{A_{22}} + \frac{A_{23}-A_{23T}}{A_{23}} + \frac{A_{33}-A_{33T}}{A_{33}} + \frac{B_1-B_{1T}}{B_1} + \frac{B_2-B_{2T}}{B_2} + \frac{B_3-B_{3T}}{B_3}}{6}) + 0.5 \quad (4.6)$$

One important note regarding the reward function is that the $B1$ term often possesses a value near zero which, even with a small absolute error, could lead to percentage errors diverging, leading to instability in Agent training. For this reason, if the $B1$ value is less than 0.5, which occurs in the training set in 10% of all datasets and never in the testing set, absolute error is used instead of percentage error. Additionally, this Reward function requires the prior knowledge of the true A and B matrices. This prohibits online learning for the RL Agent. To support online learning, the RL Agent should be implemented with a reward function more closely tied to the error in state histories and not to the error in the specific A and B matrices. This is outside the scope of this thesis, but is likely an area of worthwhile future research.

[1]This value was included to ensure that the reward in each time step was between 0 and 2; however, a value of 0.5 would move values between -0.5 and 1.5. This was found after data collection. To determine if this error would have an impact on the results, the training was repeated with the corrected value. When no changes to performance were observed across 100 random test database samples, the original data were used.

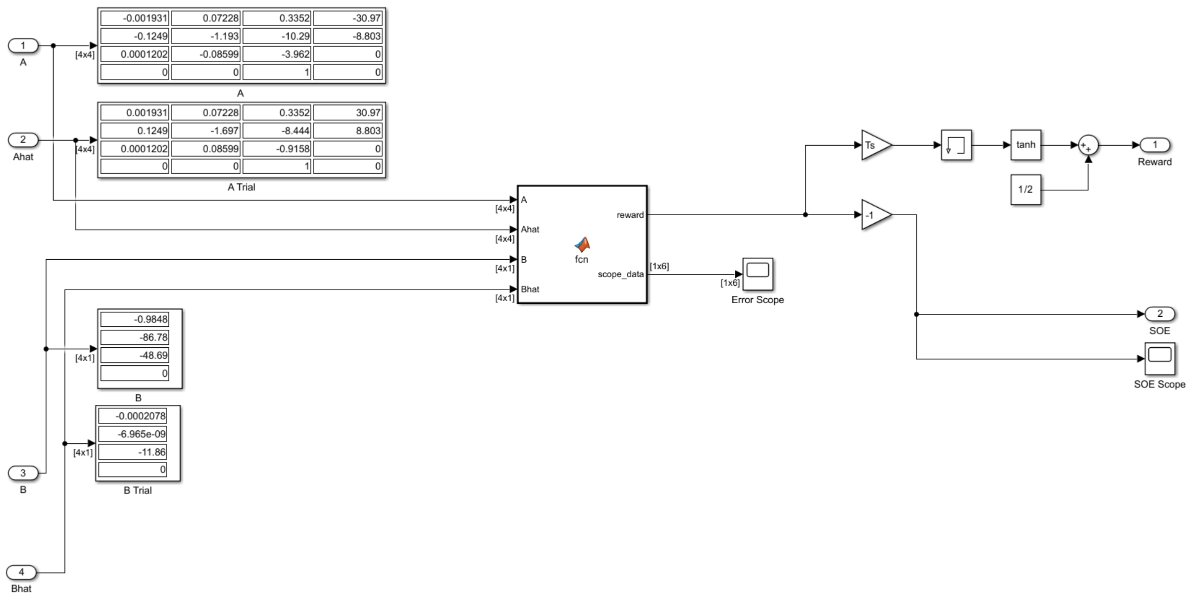


Figure 4.3 RL Agent's Reward Function

4.3.7 Other Hyperparameters

All other hyperparameters used to create the network are listed in Table 4.2. Unless otherwise stated, all RL Agents were trained and implemented using these hyperparameters.

Table 4.2 Hyperparameters used for RL Agent Training

Parameter	Purpose	Value
Sample Time	Rate at Which the Agent is Activated	0.1s
Final Time	Total Time for Each Training Episode	50s
Target Smooth Factor	Related to Overfitting	1e-3
Discount Factor	Priority for Immediate Rewards	0.95
Mini Batch Size	Related to Training	64
Experience Buffer Length	Related to Training	1e6
Noise Variance	Related to Exploration V. Exploitation	0.1
Noise Variance Decay Rate	Related to Exploration V. Exploitation	1e-5
Critic Learn Rate	Learning Rate for Critic Network	1e-3
Critic Gradient Threshold	Related to Learning Rate	1
Actor Learn Rate	Learning Rate for Actor Network	1e-4
Actor Gradient Threshold	Related to Learning Rate	1
Max Episodes	Maximum Training Episodes	100,000

4.3.8 Simulink Implementation

The Simulink File used to implement PIRL for the LFM is shown in Figures 4.4 to 4.9. It should be noted that ’hat’ denotes the trial systems and associated signals.

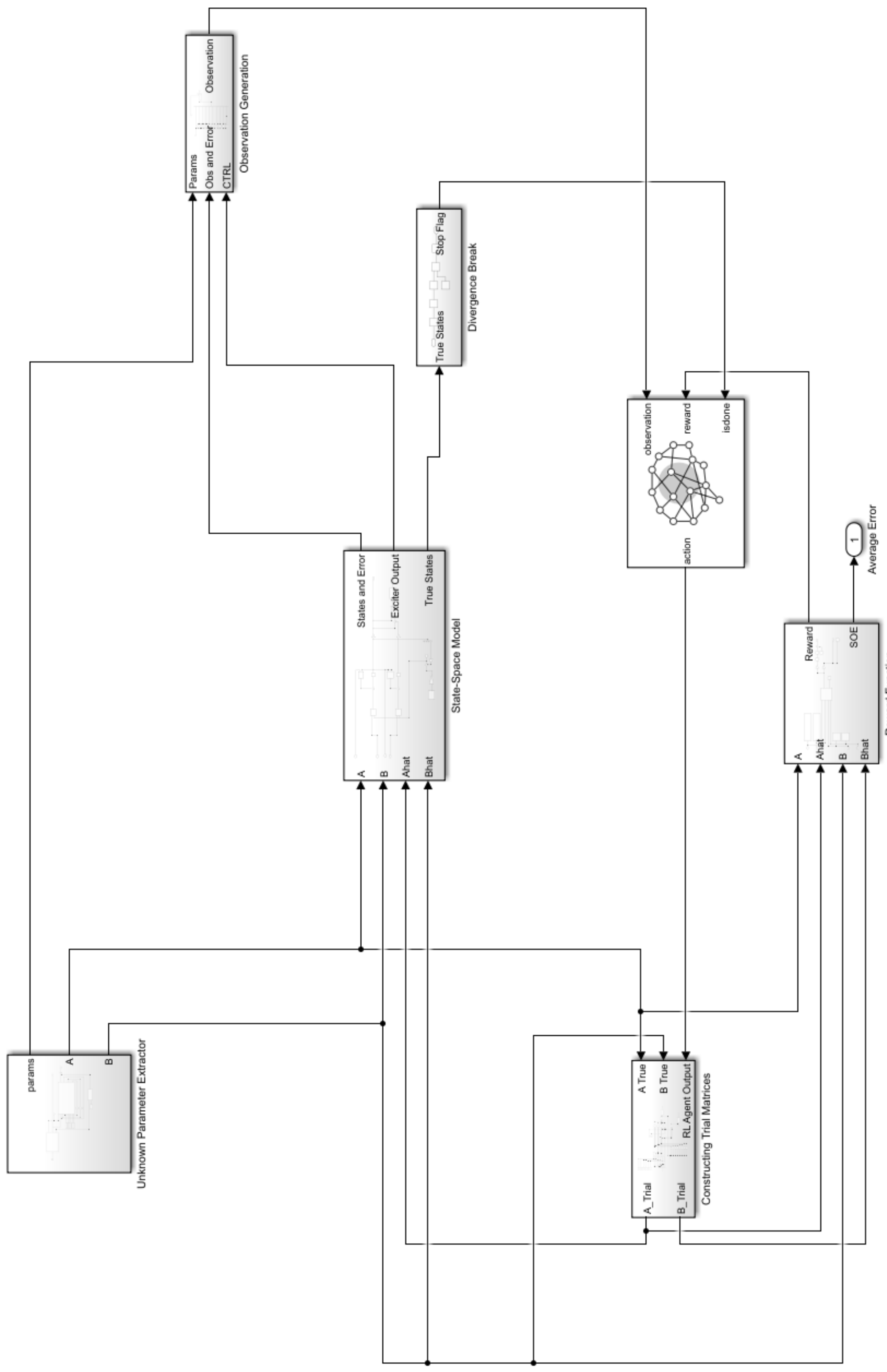


Figure 4.4 Overall Implementation of PIRL

Figure 4.4 shows the overall Simulink implementation. It features the following subsystems: Unknown Parameter Extractor, Observation Generation, State-Space Model, Divergence Break, Constructing Trial Matrices, and Reward Function. It additionally features the RL Agent Block as well as the average error output, which is used to determine the accuracy of the trial systems. The Unknown Parameter subsystem extracts the A and B matrices and PIRL parameters for the RL Agent and State-Space to use. Similarly to the MSD, all A and B values and the PIRL parameters must be stored in a single vector assigned as the initial conditions for the system. The State-Space Model subsystem uses the true A and B matrices as well as the trial A and B matrices to determine the state and error histories. These histories, along with the PIRL parameters and the Exciter Output, are used by the Observation Generation Subsystem to generate the RL Agent's Observations. The Divergence Break Subsystem uses the true values of the states to end the simulation early if it is found to be unstable and diverging. The Constructing Trial Matrices Subsystem converts the output of the RL Agent to full A and B trial matrices by including any relevant constants or true A and B values that are required. The Reward Function calculates the reward given to the RL Agent based on the accuracy of its trial A and B matrices, and finally, the RL Agent block performs the task of SYSID by using its observations and rewards to best find the value of A and B.

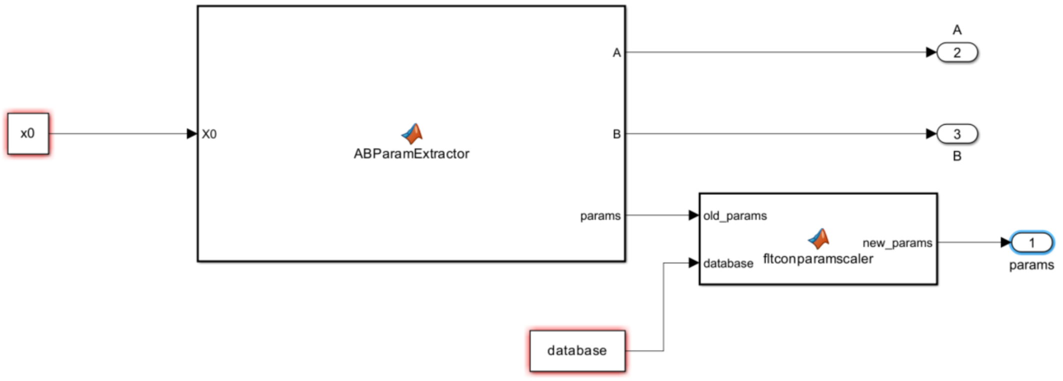


Figure 4.5 Unknown Parameter Extractor Subsystem

Figure 4.5 depicts the Unknown Parameter Extractor Subsystem. On the left is the $x0$, variable which is reset each episode with a new flight condition selected at random. The values of the A and B matrices, as well as the remaining PIRL parameters, are extracted from the $x0$ signal. The A and B matrices are then directly outputted to be used elsewhere in the system. The flight condition parameters are scaled through the *fltconparamscaler* function that transforms them to values between zero and one. These are then outputted to be used by the RL Agent.

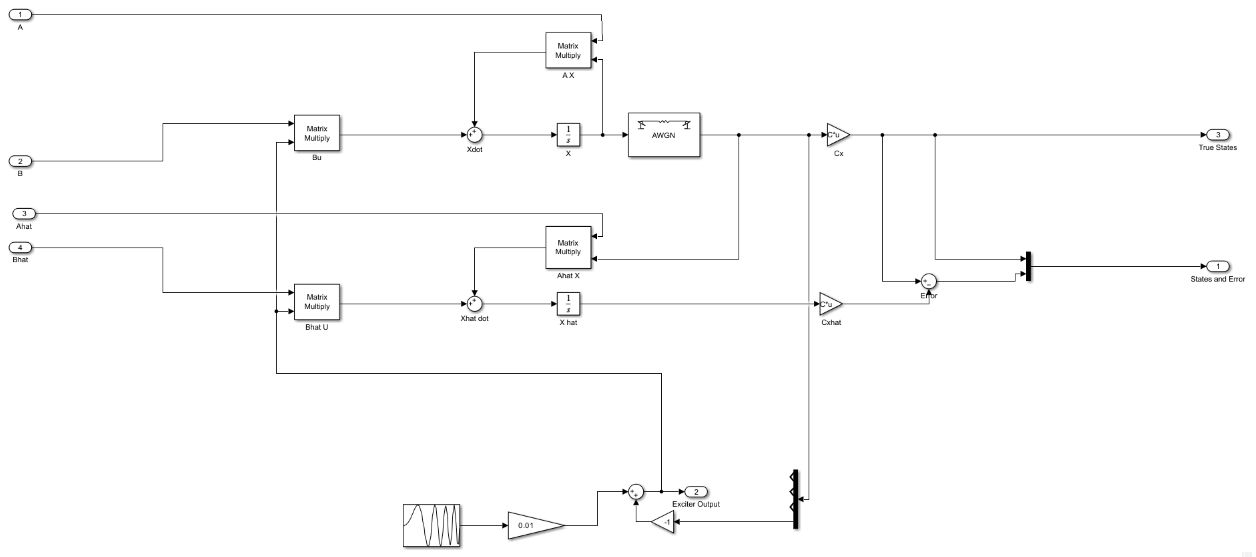


Figure 4.6 State Space Model Subsystem

The State Space Subsystem can be seen in Figure 4.6. Similar to the MSD, it receives as inputs the A, B, Ahat, and Bhat signals. Unlike the MSD Environment, the excitor is no longer its own subsystem and instead has been replaced by a chirp signal that excites the system across a spectrum of frequencies starting at 0.01 Hz and ending at 1 Hz. This was chosen to allow the RL Agent to observe the system when excited at a range of different frequencies and thus gain further insight into the separate long and short period modes that exist within the LFM. The Chirp signal has a gain applied that reduces its impact to a maximum elevator deflection of 0.01 radians, or roughly half a degree. This input is combined with that of a gain multiplying the angle state signal to create a proportional controller that maintains stability across the 50 second episode. The states are propagated exactly as in the MSD environment using a linear, time-invariant model which is numerically integrated to reach the next state. Unlike the MSD, Gaussian noise is added to the true states therein simulating measurement noise. This noise added signal is then used to generate the predicted states from the trial matrix to create the error signal. The measured state and error signals are multiplexed together and outputted. The true states are also used as outputs and are used in the Divergence Break Subsystem.

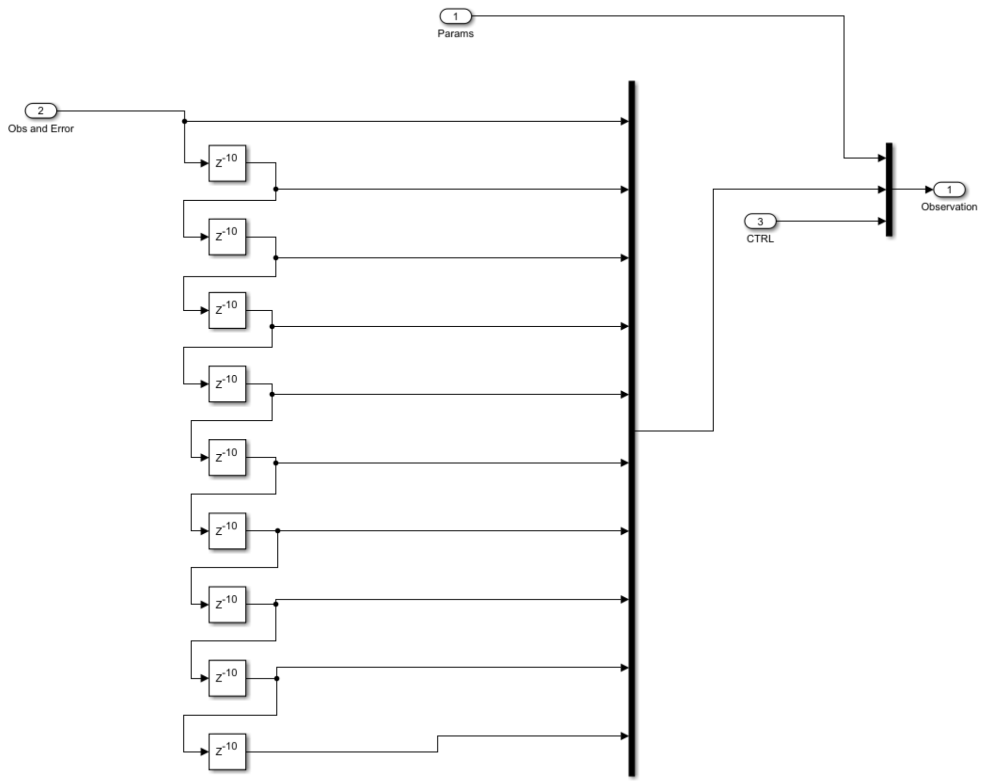


Figure 4.7 Observation Generation Subsystem

Figure 4.7 shows the Observation Generation Subsystem. This subsystem, like the MSD environment, receives the parameters of PIRL, the control input, and state and error data as its inputs. The parameters and control inputs are multiplexed directly into the observation signal, while the state and error data are propagated through a series of delay blocks. This creates a window of observation of the last second in 0.1 seconds intervals. The observation window is multiplexed with the parameters and control input and assigned to the Observation output. This output signal is then connected to the RL Agent.

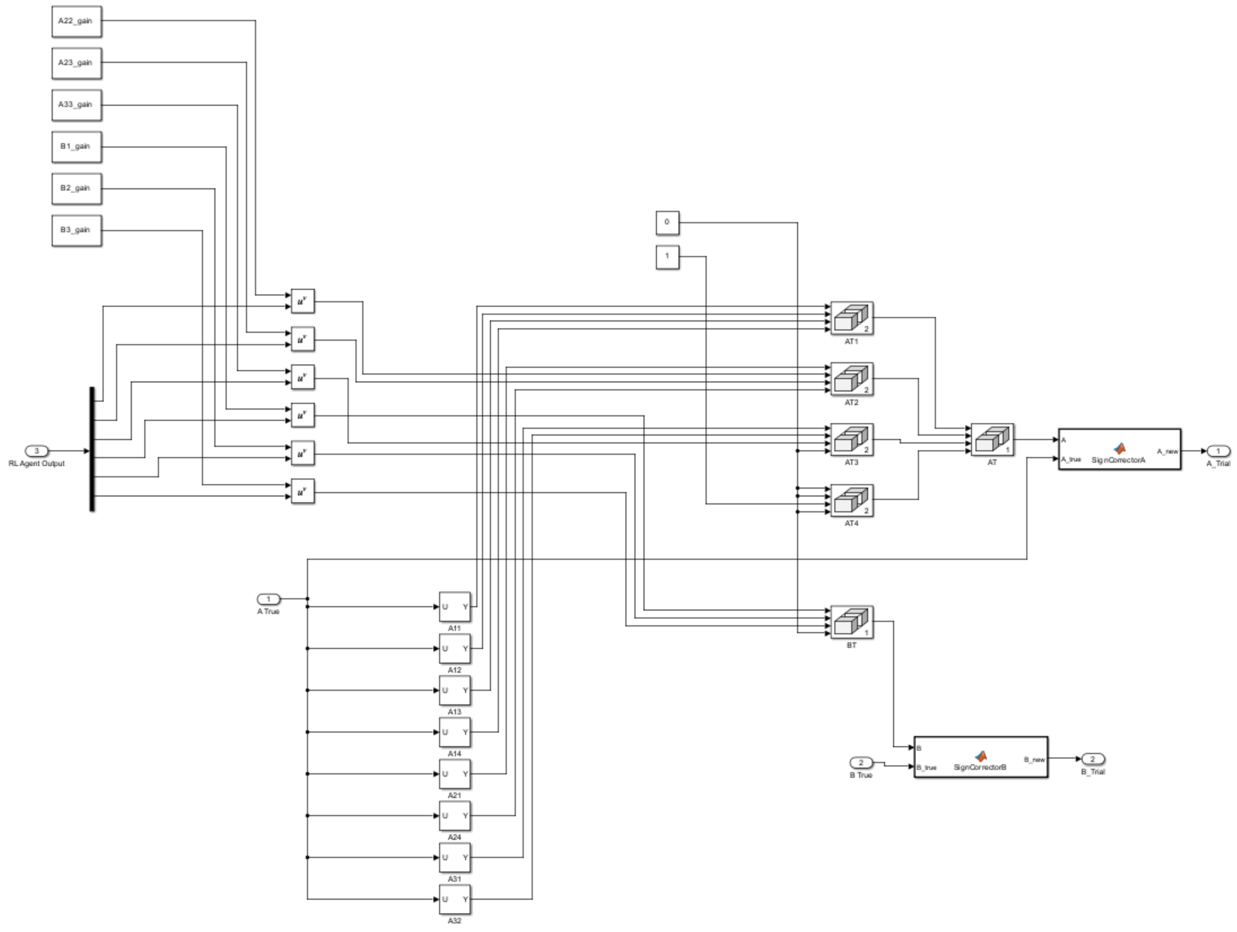


Figure 4.8 Constructing Trial Matrices Subsystem

The Constructing Trial Matrices Subsystem can be seen in Figure 4.8. This system differs in appearance significantly from the MSD but functionally serves the same purpose. The RL Agent Output is decomposed into its requisite multipliers, which are applied to the maximum values denoted A22_Gain, etc. These values are then used in combination with the true values of the other sections of the A and B matrices (constants used when available) and the trial matrix signals are completed. Each trial matrix has a sign correction function applied to since, as previously noted, the RL Agent operates under the assumption that the sign of each unknown is known.

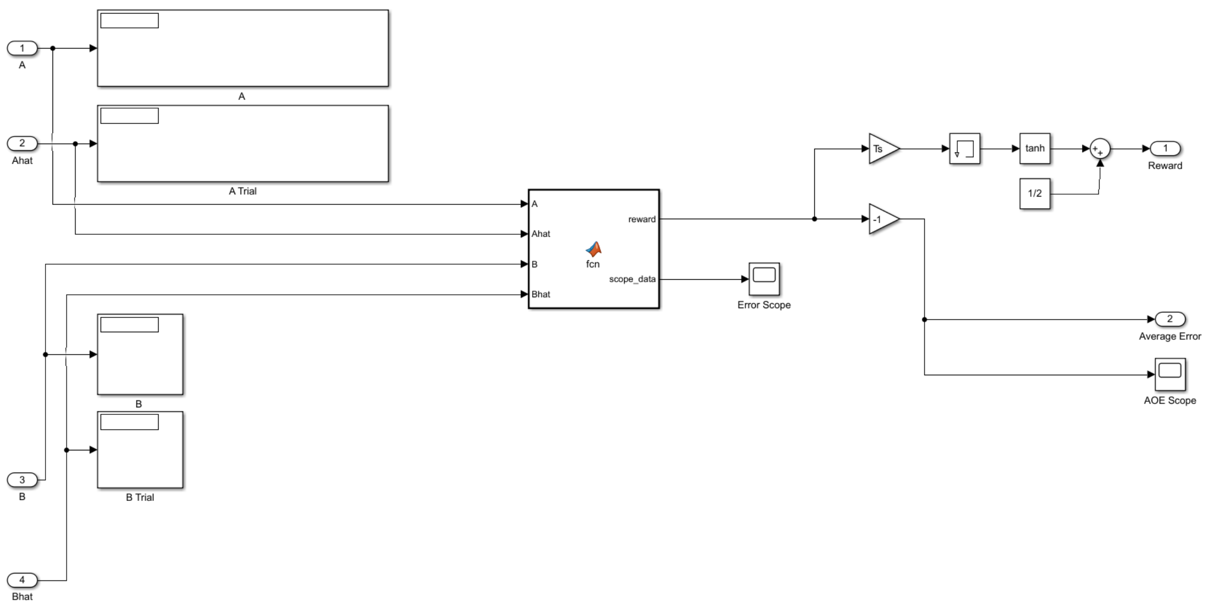


Figure 4.9 Reward Function Subsystem

The Reward Function Subsystem is shown in Figure 4.9. In this subsystem, it can be seen that the A and Ahat matrices and B and Bhat matrices are inputted into a custom Matlab function. This custom Matlab function calculates the percentage error of the differences of only the values that the RL Agent is estimating. It then averages these errors and outputs the negative of this value. The signal is processed through a memory cell to prevent an algebraic loop and then scaled between -1 and 1 through the hyperbolic tangent block. This signal is then outputted. Additionally, it can be seen in the Figure that four display blocks are used to show the current values of A, Ahat, and B and Bhat. This is used to observe the accuracy of each variable during training and testing. This can also be seen via the Error Scope, which plots each individual error with time. Finally, the AOE scope displays the average errors of each term with time. This signal is also used as an output, which is relevant for collecting average error data during testing.

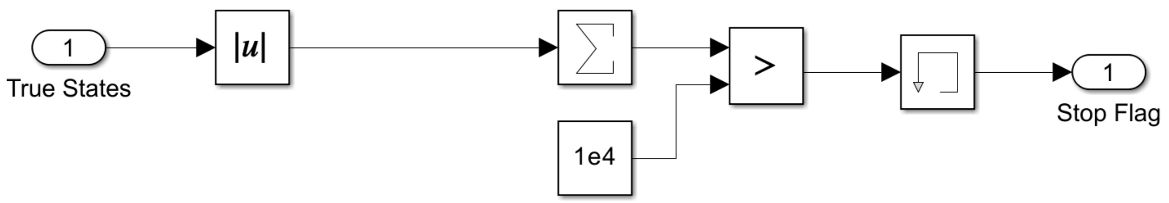


Figure 4.10 Divergence Break Subsystem

The Divergence Break Subsystem, shown in Figure 4.10, exists to flag and terminate the training for any system which may be highly unstable. This system is rarely, if ever, used as the P controller within the State-Space Model Subsystem maintains either stability or at least bounded states for every flight condition in the training and testing data sets.

4.3.9 Training and Testing Method

In order to ensure that only the most high-performing RL Agents were used, the testing method for the LFM was altered from the MSD. Instead of selecting the highest performing Agent based on the highest Average Reward, the RL Agent was set to learn over a course of 100,000 episodes, ensuring that it would reach an asymptotic level of performance. During training, each episode's RL Agent was saved, and roughly every 500-1000 Episodes, an Agent would be selected and evaluated. Following the completion of the training, the sample Agent that had the lowest percentage error would be selected as the fully trained candidate Agent.

This Agent would then be evaluated by tasking it with performing SYSID on a random 100 samples of different flight conditions selected from the test aircraft database. This would demonstrate if the RL Agent could truly perform system identification on a new, unseen aircraft and to what extent it could minimize the error between the true and trial systems. During this test, the average error of the RL Agent's trial system of the 50 second episode would be calculated and then averaged with the other 100 trials.

4.4 Results

The results from the Longitudinal Flight Model environment are included in the following section. Unlike the MSD, PIRL is only analyzed in its baseline configuration and in comparison with other existing methods, with no evaluation of the effect of varying hyperparameters. The computational requirements of training LFM networks prohibit the evaluation of the performance impact of other hyperparameters and environmental factors. The Performance of PIRL section documents how PIRL performs on the LFM as well as how changes in the application of PIRL impact performance. The Comparison to Existing Systems section is composed of analysis of PIRL's performance when compared to conventional Reinforcement Learning and an Extended Kalman Filter.

4.4.1 RL Agent Training Curve

The training curve of the PIRL RL Agent can be seen in Figure 4.11. This curve differs significantly from the training curve of the MSD. The LFM curve is bounded between -25 and 150 due to the hyperbolic tangent present in the reward function and the reward function being the negative sum of the average percentage error. Initially, the RL Agent improves and with a few dips in performance, it reaches a rough average reward (denoted by the dark blue curve) asymptote of 10-15. The RL Agent is unable to improve past this threshold despite its continued training. Additionally, the RL Agent has a number of moments in which the performance suddenly decreases for a period of several hundred episodes causing the average reward curve to be reduced to -5 or lower. These sudden changes end as soon as they begin and performance returns to the previous same level. It is unclear what causes this change although it may be due to the RL Agent exploring an unproductive path to SYSID and then returning to its original method.

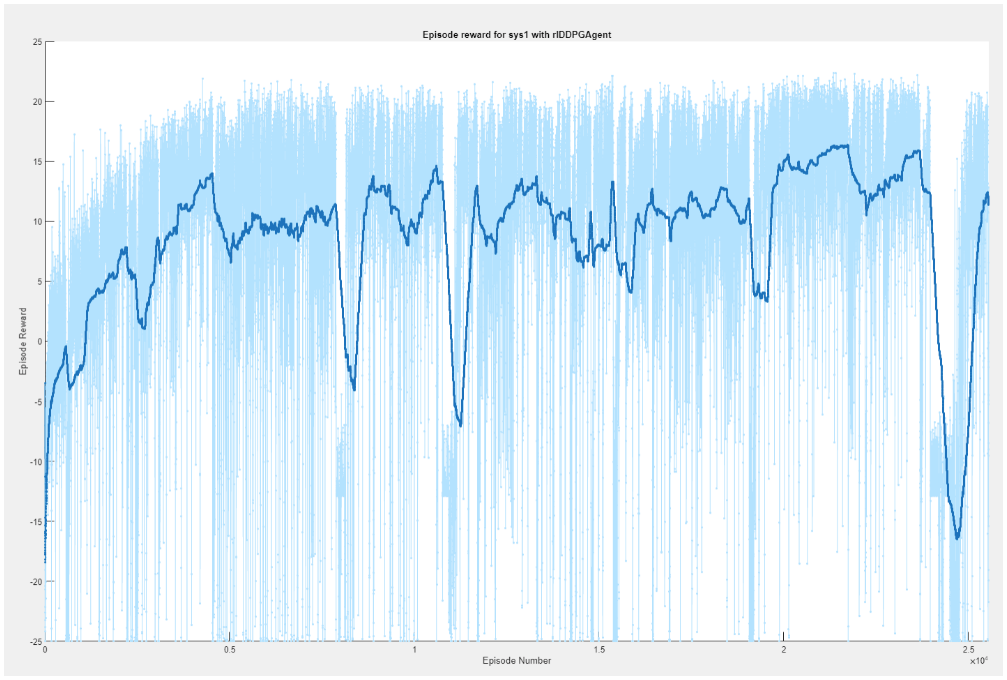


Figure 4.11 PIRL - Training Curve

4.4.2 Performance on a Random System

For an initial understanding of the effectiveness of the PIRL Agent, it was applied to a randomly selected flight condition, chosen from the test database. This flight condition is summarized in Table 4.3. It is also compared as a percentage difference to the average value of this flight condition in the training set. This is to provide an indicator of how different this condition is from the training data provided to the PIRL algorithm. It can be seen in the Table that this random flight condition is roughly similar to the average system encountered in training with regard to its Altitude, Mach Number, Airspeed, Pitch Angle, Flight Path Angle, and Weight. It is significantly different from the average training condition with regard to its angle of attack. This is less due to the particular angle caused by the flight condition and more due to a trait in how the Agent was trained. To ensure that Digital DATCOM could find an appropriate trim condition in higher altitudes where higher airspeeds would be required, the airspeeds at low altitudes were occasionally above what would be expected for conventional flight and thus in many cases the angle of attack at low altitudes was negative. This caused the average angle of attack to be near zero and thus the percentage error value to be very large. The randomly chosen system has the A and B matrices shown in Equation 4.7.

Table 4.3 Random Sample Flight Condition

Flight Condition	Value	Per Cent Variation
Altitude (ft)	16400	16.97%
Mach No.	0.5	7.83%
Flight Speed (ft/s)	525.78	3.045%
Angle of Attack (Deg)	1.5799	3471%
Pitch Angle (Deg)	9.5799	-4.60%
Flight Path Angle (Deg)	8	-20.6629%
Weight (lbs)	14740	-0.1470%
Moment of Inertia (slug ft^2)	18800	-90.45%

$$A_{True} = \begin{bmatrix} -0.0058 & 0.0370 & 0.2151 & -31.7512 \\ -0.1303 & -1.1165 & -7.8104 & -5.3575 \\ -0.0001054 & -0.08359 & -2.838 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \; B_{True} = \begin{bmatrix} -6.5911 \\ -75.0933 \\ -38.3456 \\ 0 \end{bmatrix} \tag{4.7}$$

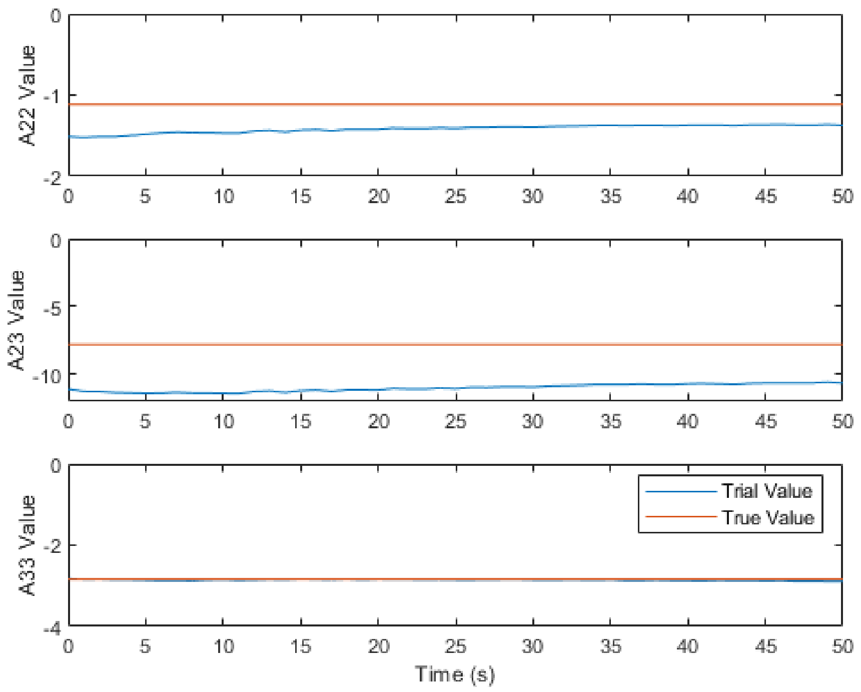


Figure 4.12 Random Flight Condition - Trial A Values

In Figure 4.12, the trial values of the A matrix are depicted along with the desired 'true' values. The values for the B matrix are shown in Figure 4.13. In Figure 4.12, it is shown that the A22 value follows its true value with a gradually reducing error of roughly 27%. A23 also gradually decreases in its error starting at roughly 40%. A33 has essentially no error and retains this throughout the 50 second window. In Figure 4.13, the B1 trial curve shows a gradual increase in error but remains at roughly 30%. B2, much like A23, reaches the true value and remains there effectively immediately. B3 exhibits the largest error with a trial value of zero. This occurs due to how the RL Agent's outputs have been used. If the Agent outputs a low positive or negative number, it is used in Equation 4.4 and will output a trial value very near zero. Since the Agent is penalized with percentage error, the RL Agent may output very near zero trial values as this incurs a maximum penalty of 100%, whereas overshooting a value lacks this maximum penalty. This conservative behavior may be caused by a lack of sufficient information to adequately predict a specific value. It also may be caused by insufficient training, but given the training curve shown in Figure 4.11, this is unlikely as the curve seems to show the RL Agent reaching a performance ceiling early that is not breached despite a significant training period.

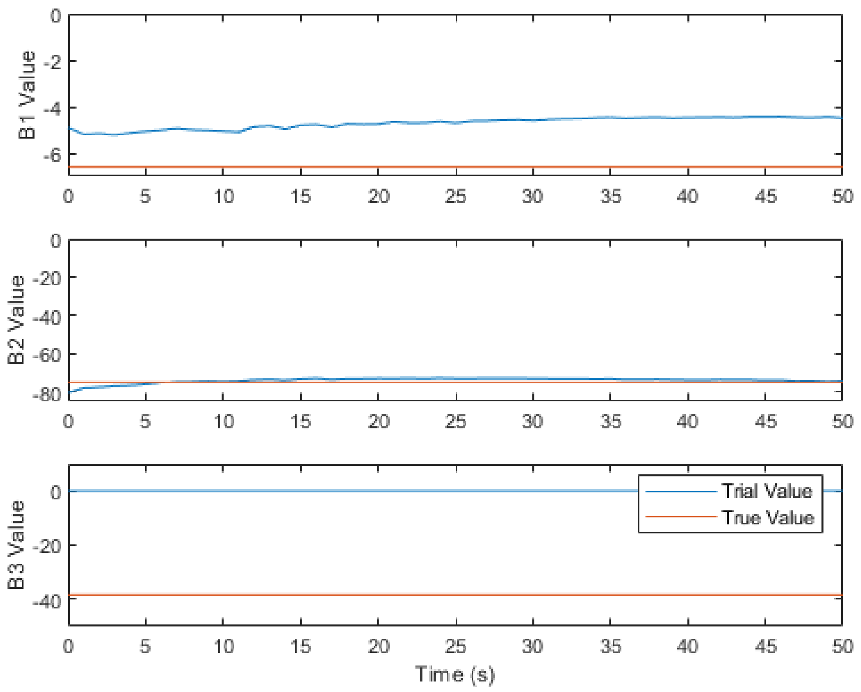


Figure 4.13 Random Flight Condition - Trial B Values

4.4.3 Performance on 100 Systems

Similarly to the MSD, the LFM PIRL Agent was implemented across 100 randomly selected flight conditions and the average percentage error across each system was recorded. The results of these tests have been provided in Table 4.4. In the table, it is evident that the RL Agent is able to estimate the values of A22 and A23 reasonably well, with A33 and B2 being estimated somewhat well. B1 is not estimated well, and the RL Agent fails to estimate B3 at all. The differences in these average errors could be caused by a number of different factors.

Table 4.4 PIRL RL Agent Percentage Accuracy Across 100 Randomly Selected Flight Conditions

Variable	PIRL Per Cent Error
A22	15.36%
A23	16.79%
A33	30.49%
B1	53.13%
B2	38.25%
B3	99.99%
Average	42.34%

One possible cause of these estimation errors is that the RL Agent lacks the neural capacity to fully and accurately estimate these values. This is not deemed likely, as during the training process, the neuron count of each layer, as well as the number of layers, changed extensively and was expanding significantly from the MSD Environment, and yet no noticeable improvements to performance were observed.

Another possible cause could lie in the variation of the data. To determine if this may be the cause, the Coefficients of Variance were calculated for both the training set and the test set. These data are shown in Table 4.5. The percentage error and Coefficients of Variance do appear to follow a similar trend with larger coefficients of variance leading to larger percentage errors. This would indicate that the variation of the variables has an impact on the RL Agent's ability to predict them, with greater variations leading to greater inaccuracies.

Table 4.5 Coefficients of Variance in Testing and Training Data

Variable	PIRL Per Cent Error	Training Coefficient of Variance	Testing Coefficient of Variance
A22	15.36%	0.5656	0.2050
A23	16.79%	0.5546	0.2211
A33	30.49%	0.7230	0.2212
B1	53.13%	1.2017	0.2960
B2	38.25%	0.9083	0.2674
B3	99.99%	0.9663	0.2755

The percentage accuracy of each unknown may also be impacted by how closely it is correlated with any given input parameter. If an unknown were to be closely correlated to a parameter, the RL Agent would have little difficulty in predicting its value, whereas if its value had little to no relation to any parameters, the RL Agent would have to obtain this information from the state and error histories, which may not be as easily interpretable.

4.4.4 Comparison to Existing Systems

It is important to understand the accuracy of PIRL, but this accuracy lacks context without an exploration of the effectiveness of other methods. This section concerns the performance of conventional Reinforcement Learning applied to this environment and an extended Kalman filter. These results are compared to those of PIRL to demonstrate the effectiveness of this novel method.

Performance of Conventional Reinforcement Learning

Reinforcement Learning has been applied to system identification previously with success. This application has, to the knowledge of the author, never included any PIRL additions or similar such algorithms. To avoid misattributing the performance of an RL method to PIRL specifically, a second RL Agent was trained and evaluated with all characteristics being the same except that it lacked the parameters as inputs. The RL Agent was then evaluated over 100 different systems randomly selected from the testing dataset. The first apparent difference between the two RL Agents can be seen in the training curve shown in Figure 4.14. These results show that, while initially the RL Agent learns well and improves both in episode reward and in average reward, this progress quickly reaches an asymptote and the AI can no longer improve. This asymptote is at roughly -4 for the average reward. This is indicative that, without the influence of PIRL, the RL Agent is unable to complete the task of system identification. When implemented with the test database, the percentage error leads one to reach a similar conclusion. The results of these tests are shown in Table 4.6. From this Table, it is clear that conventional Reinforcement Learning is, in this environment, less effective in performing the task of system identification than PIRL.

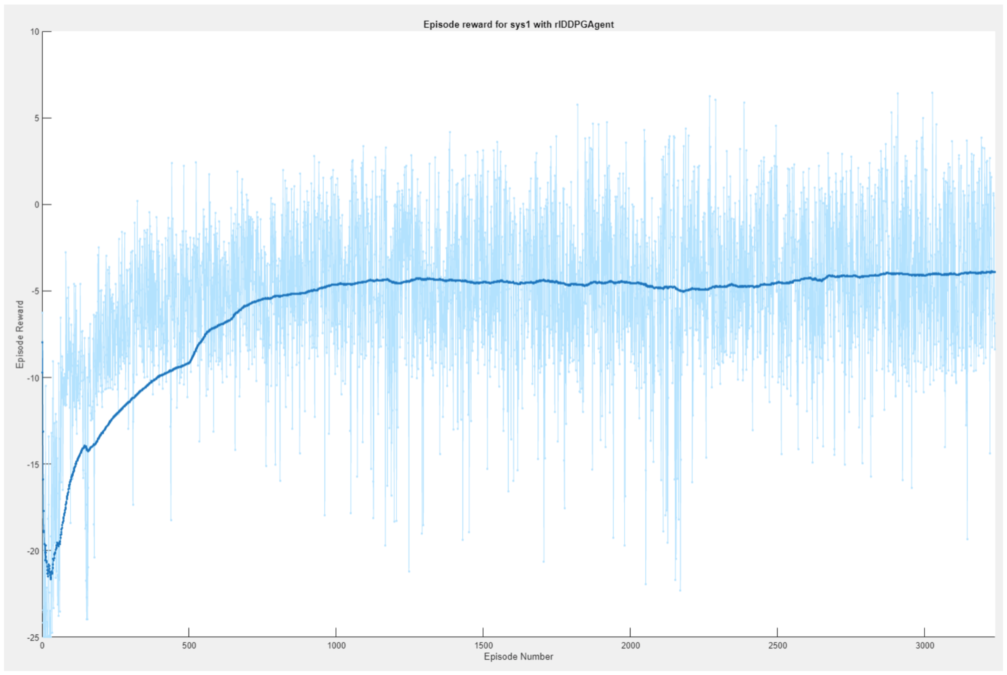


Figure 4.14 Conventional Reinforcement Learning Training Curve

Table 4.6 PIRL Agent Vs. Conventional RL Agent

Variable	PIRL Per Cent Error	Conventional Per Cent Error
A22	15.36%	38.23%
A23	16.79%	39.86%
A33	30.49%	78.63%
B1	53.13%	85.90%
B2	38.25%	99.06%
B3	99.99%	99.99%
Average	42.34%	73.61%

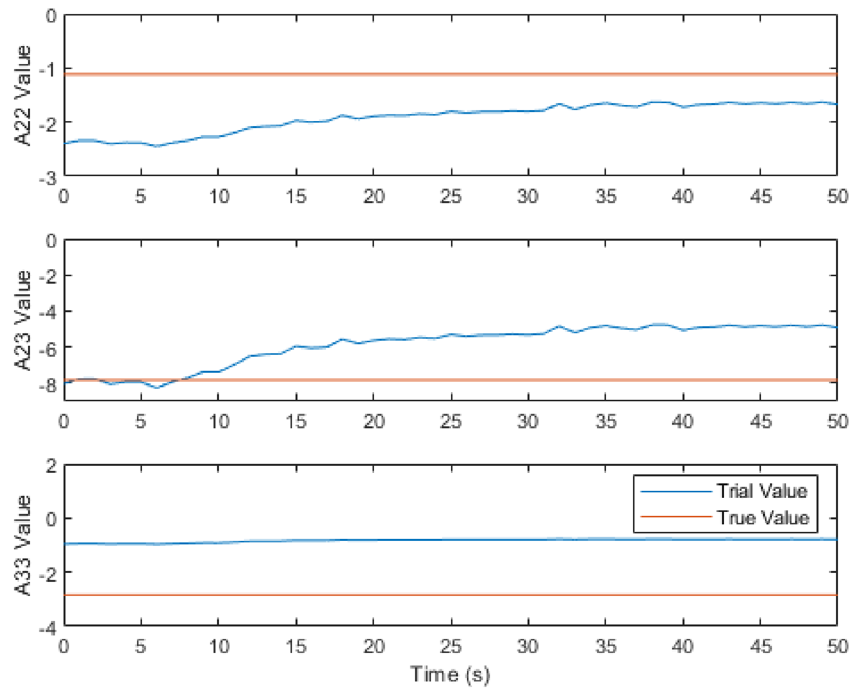


Figure 4.15 Error Trajectories of Non-PIRL Agent on a Randomly Selected Test Database System (A Matrix Values)

The trajectories of the errors of the non-PIRL Agent can be seen in Figures 4.15 and 4.16. In these figures, it is apparent that, even with sufficient training to lead to an asymptote in trained performance, the conventional RL Agent is fundamentally unable to identify the system. It is important to once again note that this is not due to the RL Agent being unable to identify a given system; rather it is being tasked with identifying over 28,000 different systems. This is a much more difficult endeavor, and the conventional RL Agent is unable to perform this task.

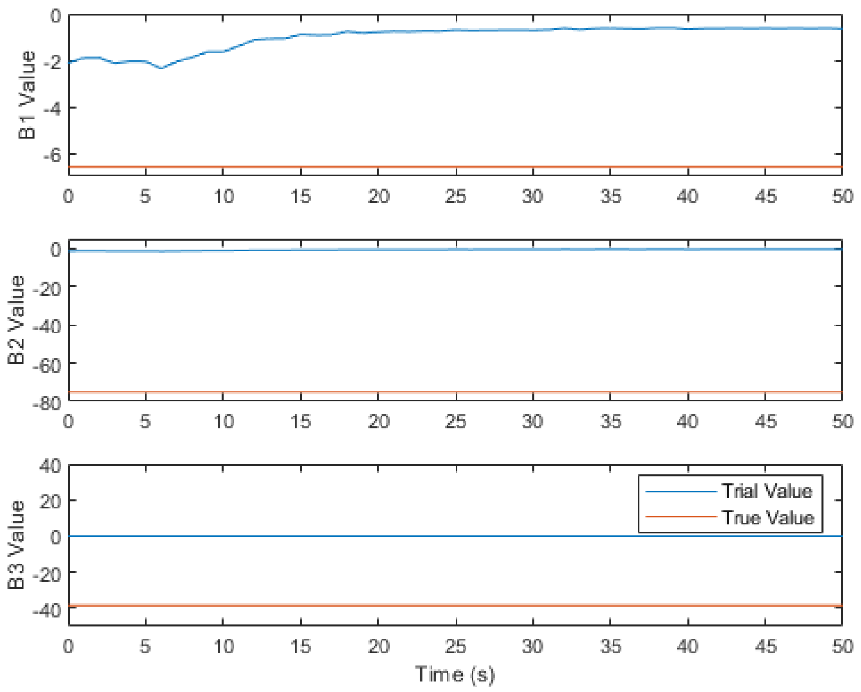


Figure 4.16 Error Trajectories of Non-PIRL Agent on a Randomly Selected Test Database System (B Matrix Values)

It is clear from these tests that the conventional RL Agent may have been effective in the MSD when the variation of the unknown system A and B values were more closely bounded, but in the LFM, it is almost entirely unable to perform SYSID on the test set. The PIRL Agent performs better by 31.27%, which is a clear indication of the benefit of the PIRL method.

Comparison to an Extended Kalman Filter

Similarly to the MSD, it is important to understand how PIRL performs when compared to more conventional methods. An extended Kalman filter is used as an existing method through which conclusions can be drawn regarding the Agent's performance as an independent method of performing system identification. It should be noted that the specific implementation used requires the EKF to perform the exact task as the PIRL Agent, but other implementations that leverage knowledge of the parameters to specific unknown matrix values may be used.

Extended Kalman filters (EKFs) are used for filtering noise, state estimation and system identification. When appropriately tuned, they can perform this task well and in a computationally efficient manner. The EKF, much like PIRL, relies on estimating states to estimate the unknown A and B values of the system. PIRL uses the error found between its trial system estimated state and the measurements, but differs in that this is only one of the many different inputs to the algorithm. Furthermore, the states are not estimated directly, but only indirectly by using the estimated system. An EKF is implemented for system identification by using the desired estimated system values (A22, A23, A33, and B1, B2, and B3) as states. These are then estimated as states by the EKF. The Extended Kalman filter is required over the regular Kalman filter as using the system values converts the system from linear to nonlinear. Before the EKF can be used, the system must be converted from continuous time to discrete. This is done in the same manner as the MSD using Euler's method and is shown Equations 4.8 and 4.9. It should be noted that dt refers to the time step at which the EKF is implemented and not necessarily the rate at which the LFM is simulated. In this experiment, dt is equal to 0.01 seconds.

$$\dot{X} = AX + BU \tag{4.8}$$

$$X_{k+1} = (Adt + I)X_k + BdtU_k \tag{4.9}$$

Following the continuous to discrete conversion, the desired values are added as states. This will make the system nonlinear and thus the augmented dynamics model is given as a series of functions shown in Equation 4.10 where $\hat{x}_5$, $\hat{x}_6$, $\hat{x}_7$, $\hat{x}_8$, $\hat{x}_9$ and $\hat{x}_{10}$ represent A22, A23, A33, B1, B2 and B3 respectively. It should be noted that the "$\hat{}$" denotes that each state is an estimate.

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \hat{x}_6 \\ \hat{x}_7 \\ \hat{x}_8 \\ \hat{x}_9 \\ \hat{x}_{10} \end{bmatrix}_{k+1} = f(\hat{X}_k, U_k) = \begin{Bmatrix} \hat{x}_{1k} + dt(X_u \hat{x}_{1k} + X_w \hat{x}_{2k} + X_q \hat{x}_{3k} - g\cos(\theta_*)\hat{x}_4 + u\hat{x}_{8k}) \\ \hat{x}_{2k} + dt(Z_u \hat{x}_{1k} + \hat{x}_{5k}\hat{x}_{2k} + \hat{x}_{6k}\hat{x}_{3k} - g\sin(\theta_*)\hat{x}_4 + u\hat{x}_{9k}) \\ \hat{x}_{3k} + dt(M_u \hat{x}_{1k} + M_w \hat{x}_{2k} + \hat{x}_{7k}\hat{x}_{3k} + u\hat{x}_{10k}) \\ \hat{x}_{4k} + dt\hat{x}_{3k} \\ \hat{x}_{5k} \\ \hat{x}_{6k} \\ \hat{x}_{7k} \\ \hat{x}_{8k} \\ \hat{x}_{9k} \\ \hat{x}_{10k} \end{Bmatrix} \tag{4.10}$$

With this dynamics model, the EKF procedure is given in Equations 4.11 to 4.15, which are performed in order at each k time step. Note that $k + 1/k$ refers to intermediate values and that C has been augmented with zeroes to support the larger $\hat{X}$ vector. In this process, F refers to the Jacobian of A with respect to $\hat{X}$, while Q, R, P, and K refer to the process noise covariance matrix, the sensor noise covariance matrix, the estimation error covariance matrix and the Kalman gain respectively.

$$\hat{X}_{k+1/k} = f(\hat{X}_k, U_k) \tag{4.11}$$

$$P_{k+1/k} = F_k P_k F_k^T + Q_k \tag{4.12}$$

$$K_{k+1} = P_{k+1/k} C^T (C P_{k+1/k} C^T + R_k)^{-1} \tag{4.13}$$

$$\hat{X}_{k+1} = \hat{X}_{k+1/k} + K_{k+1}(Y_{k+1} - C\hat{X}_{k+1/k}) \quad (4.14)$$

$$P_{k+1} = (I - K_{k+1}C)P_{k+1/k} \quad (4.15)$$

The results of this test can be seen in Table 4.7, where the EKF was applied to 100 randomly selected flight conditions. It is clear from these results that the extended Kalman filter is less effective than PIRL. The EKF manages to predict some values moderately well, but it is completely unable to estimate other values, such as B2. This could be caused by a number of different factors, including the particular system on which it was tuned, or more complicated variables such as the persistence of excitation. No matter the cause, the EKF in this particular test seems unable to effectively perform system identification with an average percentage error of 73.75% compared to PIRL's 42.34%. This is a positive indicator of PIRL's effectiveness as a novel system identification algorithm.

Table 4.7 PIRL RL Agent Vs. Extended Kalman Filter

Variable	PIRL Per Cent Error	EKF Per Cent Error
A22	15.36%	23.19%
A23	16.79%	81.94%
A33	30.49%	44.33%
B1	53.13%	100.41%
B2	38.25%	99.17%
B3	99.99%	93.49%
Average	42.34%	73.75%

4.5 Conclusion

System identification with the longitudinal flight model has demonstrated the effectiveness of PIRL as a novel system identification algorithm for multiple similar systems. The results demonstrate that, while PIRL is imperfect and still requires further development to accurately identify all variables, it consistently outperforms existing methods such as conventional reinforcement learning and extended Kalman filters. In addition to its greater accuracy, PIRL also identifies the systems instantaneously or near instantaneously, as opposed to extended Kalman filters which have a significant convergence time. Due to computational constraints, an in-depth analysis of the effects of hyperparameters and environmental factors was not possible. This is an area in which further study is recommended, as PIRL in its current status appears to bear merit as a novel algorithm and further optimization is likely to lead to further reduced error.

This chapter has also demonstrated how flight conditions can be generated *en masse* for a number of different aircraft through the use of Digital DATCOM. This may bear relevance outside of PIRL and SYSID and be useful for research concerning the application of artificial intelligence and deep learning with aerospace engineering. To the author's knowledge, sufficiently large databases of aircraft dynamics and control data for training artificial intelligence models on multiple different platforms and in multiple different environments are exceedingly rare. With the code provided in the appendices of this thesis, further research may be conducted from usage of this database alone.

5 Conclusions and Future Research

In Aerospace Engineering, novel aircraft design requires the generation of mathematical models of the aircraft swiftly and accurately. Conventional mathematical system identification methods, such as the extended Kalman filter, require tuning on each application while artificial intelligence based methods require extensive training on their chosen platforms. To the author's knowledge, there has not yet been published research regarding a system identification method that can be trained on a selection of similar related platforms and implemented on a new one. In this thesis, this form of implementation is supported by the development of a novel form of Reinforcement Learning Application termed PIRL: Parameter Informed Reinforcement Learning. This chapter concerns the conclusions regarding the development of PIRL and its application to the problem of SYSID for multiple related environments.

5.1 Conclusions

PIRL was created based on the notion that an AI algorithm that is trained to perform System Identification on a variety of different platforms would be better suited to the task if it were provided information beyond the state and error trajectories that inform existing mathematical and artificial intelligence methods. By being informed by additional related information about each system, the artificial intelligence agent should be able to develop at least a general internal mathematical model of the relationship between these parameters and their role in the system's dynamics. For example, if an RL Agent is trained on two aircraft of similar class and weight but one possessed a larger moment arm for its elevators than the other, the RL Agent should be able to observe this difference and conclude that the coefficient determining the effect of elevator deflection would likely be larger for the former aircraft. Thus, it would be able to carry this knowledge forward into the implementation on a previously unseen aircraft. This task can be solved using supervised as well as reinforcement learning; however, reinforcement learning was ultimately selected due to its ability to train online if given a reward function based on the differences between state trajectories. This type of reward function was ultimately not implemented due to difficulties in RL Agent training.

5.1.1 Environment 1: Mass-Spring-Damper

The Mass-Spring-Damper model was crucial in developing the first insights of how PIRL was best implemented and its effectiveness when compared to existing systems and robustness to changes in its operating Environment. In this Environment, the RL Agent was tasked with identifying the unknown values in the A and B matrices of a Mass-Spring-Damper system. It was found that PIRL was an effective novel method of SYSID. It produced more accurate models than an Extended Kalman Filter and was able to generate these models with little to no convergence time, whereas the EKF requires 10-15 seconds to converge. In addition, when applied to the MSD, PIRL was shown to be more effective at SYSID than conventional Reinforcement Learning. This strongly suggests that the presence of the additional parameters causes lower average percentage errors in the trial systems. The Mass-Spring-Damper system, due to its simplicity, also allowed for a significant amount of work to be conducted with regard to how the RL Agents can be optimized. Many hyperparameters had little to no effect, while others were found to have unexpected effects. In summary: noise variance, noise decay rate, discount factor, and observation window length and intervals all had minimal impacts, while architecture exhibited a clear lower bound and sample time was relevant mostly to the design of the reward function and training threshold. Finally, the MSD Environment enabled evaluation of PIRL when certain environmental variables change such as the range of possible variation, the presence of sensor noise of varying Signal-to-Noise Ratios, and the effect of reduced parameters. It was determined in these experiments that the RL Agents that were trained with PIRL were more effective than their conventional RL counterparts. Ultimately, this environment allowed PIRL and its implementation to be better understood so that it could be applied more effectively in the Longitudinal Flight Model. It also demonstrated on a small-scale how PIRL was more effective than existing methods for System Identification of multiple related environments.

5.1.2 Environment 2: Longitudinal Flight Model

The Longitudinal Flight Model Environment allowed for a much more accurate and thus more challenging task for the PIRL Agents. In this environment, the PIRL Agents were tasked with identifying specific values from the A and B matrices of the longitudinal flight models of a Learjet 35 Aircraft at various different flight conditions after being trained on simulated flight data of several other aircraft at other flight conditions. PIRL was once again evaluated against a conventional Reinforcement Learning agent and an extended Kalman filter. PIRL was ultimately unable to fully identify this system and suffered from an average percentage error of roughly 40%; however, when compared to other existing methods, this percentage error was considerably lower than attained by the other algorithms. The extended Kalman filter and conventional Reinforcement Learning both were only able to achieve roughly 70% error. Due to the computational requirements of training these Agents, an in-depth analysis of the effects of hyperparameters and environmental factors was not undertaken.

In the process of applying PIRL to this Environment, a large database of different aircraft across different flight conditions was developed. This was done through extensive usage of the USAF's Stability and Control Digital DATCOM program that was able to rapidly generate stability data that could be used to generate the A and B matrices of a Longitudinal Flight Model. To the author's knowledge, there are little to no databases containing a significant number of aircraft flight models across a significant number of flight conditions. The program and DATCOM files featured in the appendices of this thesis may allow for more of these databases to be generated. These databases are critical for any AI research with application to multiple aircraft or to aircraft at multiple different flight conditions.

5.2 Future Research

This thesis concerns the development and evaluation of the Parameter Informed Reinforcement Learning Method to SYSID of multiple related environments. From this topic, a number of further research areas may be pursued. This thesis is only intended to provide the first implementation of PIRL and not a comprehensive analysis of its effectiveness. Additionally, this thesis examines an area in which to the author's knowledge, little to no research exists. This is the problem of SYSID for multiple related environments. Given this thesis' limited scope, the following avenues of future research are recommended:

- Investigate PIRL's effectiveness when applied to more complex systems.
- Investigate the effect of hyperparameters on PIRL's effectiveness in the Longitudinal Flight Model (or others).
- Investigate the usage of an error trajectory based reward function to enable online learning.
- Further develop the PIRL algorithm to enable use of any control input signal.
- Investigate use of PIRL with frequency-domain data.
- Expand the aircraft training and testing database and observe any differences to PIRL performance.
- Conduct a more comprehensive comparison between PIRL and other conventional SYSID methods.
- Explore PIRL's application in other non-SYSID applications.
- Explore PIRL's application in fault-tolerance applications.
- Investigate the relationship between PIRL Parameters and Trial Matrix Values to determine most effective parameters.
- Determine ideal methods for pre-processing PIRL/RL Agent inputs.
- Determine ideal methods for post-processing PIRL/RL Agent outputs.
- Explore PIRL's application to the tuning of extended Kalman filter through the RL Agent's output.
- Explore methods to develop large AI datasets of flight test or flight model information.
- Explore implementation of novel or existing algorithms to SYSID of multiple related environments.

REFERENCES

[1] Deistler, M., *System Identification and Time Series Analysis: Past, Present, and Future*, Springer-Verlag, 2008, pp. 97–109. https://doi.org/10.1007/3-540-48022-6_7.

[2] Federal Aviation Administration, "AC 61-107B: Aircraft Operations at Altitudes Above 25,000 Feet Mean Sea Level or Mach Numbers Greater Than .75," Tech. rep., Federal Aviation Administration, 2013.

[3] Cessna Aircraft Company, *Pilot's Operating Handbook - Cessna Model 172N*, Cessna Aircraft Company, 1977.

[4] Australian Transport Safety Bureau, "In-flight uncontained engine failure overhead Batam Island, Indonesia 4 November 2010 VH-OQA Airbus A380-842," Tech. rep., Australian Transport Safety Bureau, 2013.

[5] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, 2018.

[6] Mathworks, *What Is Reinforcement Learning?*, Mathworks, Natick, Massachusetts, USA, 2023.

[7] Ljung, L., *System Identification: An Overview*, Springer London, London, 2015, pp. 1443–1458. https://doi.org/10.1007/978-1-4471-5058-9_100, URL https://doi.org/10.1007/978-1-4471-5058-9_100.

[8] Bauer, J. E., and Andrisani, D., "Estimating short-period dynamics using an extended Kalman filter," Tech. rep., NASA, 1990.

[9] Gu, Y., Jin, J., and Mei, S., "l_0 Norm Constraint LMS Algorithm for Sparse System Identification," *IEEE Signal Processing Letters*, Vol. 16, No. 9, 2009, pp. 774–777. https://doi.org/10.1109/LSP.2009.2024736.

[10] Kaltenbacher, B., and Nguyen, T. T. N., "A model reference adaptive system approach for nonlinear online parameter identification," *arXiv.org*, 2021.

[11] Morelli, E. A., and Grauer, J. A., "Practical Aspects of Frequency-Domain Approaches for Aircraft System Identification," *Journal of Aircraft*, Vol. 57, No. 2, 2020, pp. 268–291. https://doi.org/10.2514/1.C035599, URL https://doi.org/10.2514/1.C035599.

[12] Martinsen, A. B., Lekkas, A. M., and Gros, S., "Combining system identification with reinforcement learning-based MPC," *IFAC-PapersOnLine*, Vol. 53, No. 2, 2020, pp. 8130–8135. https://doi.org/https://doi.org/10.1016/j.ifacol.2020.12.2294, URL https://www.sciencedirect.com/science/article/pii/S2405896320329542, 21st IFAC World Congress.

[13] Hu, J., Wang, Q., Ye, Y., and Tang, Y., "Toward Online Power System Model Identification: A Deep Reinforcement Learning Approach," *IEEE Transactions on Power Systems*, Vol. 38,

No. 3, 2023, pp. 2580–2593. https://doi.org/10.1109/TPWRS.2022.3180415.

[14] H., J. A. M., Vicente, O. F., Perez, S., Belfadil, A., Ibanez-Llano, C., Rondon, F. J. P., Valle, J. J., and Pelaz, J. A., "Reinforcement Learning in System Identification," , 2022.

[15] Perrusquía, A., and Yu, W., "Identification and optimal control of nonlinear systems using recurrent neural networks and reinforcement learning: An overview," *Neurocomputing (Amsterdam)*, Vol. 438, 2021, pp. 145–154.

[16] Nonomura, T., Shibata, H., and Takaki, R., "Extended-Kalman-filter-based dynamic mode decomposition for simultaneous system identification and denoising," *PloS one*, Vol. 14, No. 2, 2019.

[17] Kassim, A. M., Abas, N., Legowo, A., Ibrahim, Z., and Rahim, N., "Modeling and System Identification using Extended Kalman Filter for a Quadrotor System," *Applied Mechanics and Materials*, Vol. 313-314, 2013, pp. 976–981.

[18] Basappa, K., and Jategaonkar, R., "Evaluation of Recursive Methods for Aircraft Parameter Estimation," *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, 2004.

[19] Choi, M., Oh, J. J., and Choi, S. B., "Linearized Recursive Least Squares Methods for Real-Time Identification of Tire-Road Friction Coefficient," *IEEE transactions on vehicular technology*, Vol. 62, No. 7, 2013, pp. 2906–2918.

[20] Ding, F., Wang, Y., Dai, J., Li, Q., and Chen, Q., "A recursive least squares parameter estimation algorithm for output nonlinear autoregressive systems using the input–output data filtering," *Journal of the Franklin Institute*, Vol. 354, No. 15, 2017, pp. 6938–6955.

[21] De Souza, D. A., Batista, J. G., Vasconcelos, F. J. S., Dos Reis, L. L. N., Machado, G. F., Costa, J. R., Junior, J. N. N., Silva, J. L. N., Rios, C. S. N., and Junior, A. B. S., "Identification by Recursive Least Squares With Kalman Filter (RLS-KF) Applied to a Robotic Manipulator," *IEEE access*, Vol. 9, 2021, pp. 63779–63789.

[22] Li, S., Xue, Q., Shi, D., Li, X., and Zhang, W., "Recursive Least Squares Based Refinement Network for Vehicle Trajectory Prediction," *Electronics (Basel)*, Vol. 11, No. 12, 2022.

[23] Kamali, C., Pashilkar, A., and Raol, J., "Evaluation of Recursive Least Squares algorithm for parameter estimation in aircraft real time applications," *Aerospace Science and Technology*, Vol. 15, No. 3, 2011, pp. 165–174. https://doi.org/https://doi.org/10.1016/j.ast.2010.12.007, URL https://www.sciencedirect.com/science/article/pii/S1270963810001628.

[24] Guo, Y., and Kareem, A., "Non-stationary frequency domain system identification using time–frequency representations," *Mechanical Systems and Signal Processing*, Vol. 72-73, 2016, pp. 712–726.

[25] Morelli, E. A., "Low-Order Equivalent System Identification for the Tu-144LL Supersonic Transport Aircraft," *Journal of guidance, control, and dynamics*, Vol. 26, No. 2, 2003, pp.

354–362.

[26] Morelli, E. A., and Grauer, J. A., "Practical Aspects of Frequency-Domain Approaches for Aircraft System Identification," *Journal of Aircraft*, Vol. 57, No. 2, 2020, pp. 268–291.

[27] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," , 2019.

[28] Qiu, C., Hu, Y., Chen, Y., and Zeng, B., "Deep Deterministic Policy Gradient (DDPG)-Based Energy Harvesting Wireless Communications," *IEEE Internet of Things Journal*, Vol. 6, No. 5, 2019, pp. 8577–8588. https://doi.org/10.1109/JIOT.2019.2921159.

[29] Jia, Z., Gao, Q., and Peng, X., "LSTM-DDPG for Trading with Variable Positions," *Sensors (Basel, Switzerland)*, Vol. 21, No. 19, 2021, pp. 6571–.

[30] Guo, S., Zhang, X., Zheng, Y., and Du, Y., "An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning," *Sensors (Basel, Switzerland)*, Vol. 20, No. 2, 2020, pp. 426–.

[31] Zhu, M., Tian, K., Wen, Y.-Q., Cao, J.-N., and Huang, L., "Improved PER-DDPG based nonparametric modeling of ship dynamics with uncertainty," *Ocean Engineering*, Vol. 286, 2023, p. 115513. https://doi.org/https://doi.org/10.1016/j.oceaneng.2023.115513, URL https://www.sciencedirect.com/science/article/pii/S0029801823018978.

[32] Aoun, C. E., "Fault Tolerant Deep Reinforcement Learning for Aerospace Applications," Master's thesis, Embry-Riddle Aeronautical University, 2021.

[33] Shalumov, V., "Cooperative online Guide-Launch-Guide policy in a target-missile-defender engagement using deep reinforcement learning," *Aerospace science and technology*, Vol. 104, 2020, pp. 105996–.

[34] Liu, Z., Li, J., Wang, C., Yu, R., Chen, J., He, Y., and Sun, C., "System Identification Based on Generalized Orthonormal Basis Function for Unmanned Helicopters: A Reinforcement Learning Approach," *IEEE Transactions on Vehicular Technology*, Vol. 70, No. 2, 2021, pp. 1135–1145. https://doi.org/10.1109/TVT.2021.3051696.

[35] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," *CoRR*, Vol. abs/1711.10561, 2017. URL http://arxiv.org/abs/1711.10561.

[36] Kon, M., and Plaskota, L., "Information complexity of neural networks," *Neural networks*, Vol. 13, No. 3, 2000, pp. 365–375.

[37] Hajiyev, C., "LQR Controller with Kalman Estimator Applied to UAV Longitudinal Dynamics," *Positioning*, Vol. 04, 2013, pp. 36–41. https://doi.org/10.4236/pos.2013.41005.

[38] McDonnell Douglas Astronautics Co., *The USAF Stability and Control Digital DATCOM.*

Volume I. Users Manual, United States Air Force, 1978.

[39] Airliners.net, 2023. URL https://www.airliners.net/aircraft-data/beech-99-airliner/66.

[40] Scott, j., "Beech 99 Low-Speed Cruise Configuration," https://m-selig.ae.illinois.edu/apasim/Aircraft-uiuc/beech99-v1/aircraft.dat, 2000. Accessed: 2023-10-17.

[41] Airliners.net, 2023. URL https://www.airliners.net/aircraft-data/boeing-737-800900/96.

[42] NASA, 2023. URL https://www.nasa.gov/aeronautics/t-34c-mission-support-aircraft/.

[43] AOPA, 2023. URL https://www.aopa.org/go-fly/aircraft-and-ownership/aircraft-guide/aircraft/cessna-182.

[44] GlobalAir.com, 2023. URL https://www.globalair.com/aircraft-for-sale/specifications?specid=183.

[45] GlobalAir.com, 2023. URL https://www.globalair.com/aircraft-for-sale/specifications?specid=184.

[46] Horssen, L. V., de Visser, C. C., and Pool, D. M., "Aerodynamic Stall and Buffet Modeling for the Cessna Citation II Based on Flight Test Data," *2018 AIAA Modeling and Simulation Technologies Conference*, 2018.

[47] GlobalAir.com, 2023. URL https://www.globalair.com/aircraft-for-sale/specifications?specid=300.

[48] Suit, W. T., *Aerodynamic parameters of the Navion airplane extracted from flight*, Legacy CDMS, 1972.

[49] GlobalAir.com, 2023. URL https://www.globalair.com/aircraft-for-sale/specifications?specid=484.

[50] Nolan, I., Robert C, *Wing Rock Prediction Method for a High Performance Fighter Aircraft*, 1992.

[51] GlobalAir.com, 2023. URL https://www.globalair.com/aircraft-for-sale/specifications?specid=25.

A Programs Used For Environments

A.1 Mass Spring Damper Code

A.1.1 One-Time Training of Fixed Reward

```
clc

close all

clear


n_knowns = 3

n_states = 2


C = eye(2)


Ts = 0.1

Tf = 50

window_size = 10



obsInfo = rlNumericSpec([4*window_size + n_knowns+1,1])



actInfo = rlNumericSpec([3,1])


open_system("sys10")

set_param('sys10','GPUAcceleration','off')

env = rlSimulinkEnv("sys10","sys10/RL Agent",...

    obsInfo,actInfo);
```

```
env.ResetFcn = @(in) setVariable(in,'x0',[rand()*0.2+0.9,rand()
   *0.2 + 0.9,rand*0.2 + 0.9]); %m k b

% Observation path
obsPath = [
    featureInputLayer(obsInfo.Dimension(1),Name="obsInputLayer")
    fullyConnectedLayer(20)
    reluLayer
    fullyConnectedLayer(20,Name="obsPathOutLayer")];

% Action path
actPath = [
    featureInputLayer(actInfo.Dimension(1),Name="actInputLayer")
    fullyConnectedLayer(20,Name="actPathOutLayer")];

% Common path
commonPath = [
    additionLayer(2,Name="add")
    reluLayer
    fullyConnectedLayer(1,Name="CriticOutput")];

criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,obsPath);
criticNetwork = addLayers(criticNetwork,actPath);
criticNetwork = addLayers(criticNetwork,commonPath);

criticNetwork = connectLayers(criticNetwork, ...
    "obsPathOutLayer","add/in1");
criticNetwork = connectLayers(criticNetwork, ...
    "actPathOutLayer","add/in2");

criticNetwork = dlnetwork(criticNetwork);
summary(criticNetwork)

critic = rlQValueFunction(criticNetwork, ...
    obsInfo,actInfo, ...
    ObservationInputNames="obsInputLayer", ...
    ActionInputNames="actInputLayer");
getValue(critic, ...
    {rand(obsInfo.Dimension)}, ...
    {rand(actInfo.Dimension)})

actorNetwork = [
    featureInputLayer(obsInfo.Dimension(1))
    fullyConnectedLayer(3)
    tanhLayer
    fullyConnectedLayer(actInfo.Dimension(1))
    ];

actorNetwork = dlnetwork(actorNetwork);
summary(actorNetwork)

actor = rlContinuousDeterministicActor(actorNetwork,obsInfo,
   actInfo);
```

```
getAction(actor,{rand(obsInfo.Dimension)})
agent = rlDDPGAgent(actor,critic);

agent.SampleTime = Ts;

agent.AgentOptions.TargetSmoothFactor = 1e-3;
agent.AgentOptions.DiscountFactor = 0.95; %OG was 1
agent.AgentOptions.MiniBatchSize = 64;
agent.AgentOptions.ExperienceBufferLength = 1e6;



agent.AgentOptions.NoiseOptions.Variance = 0.3;
agent.AgentOptions.NoiseOptions.VarianceDecayRate = 1e-5;

agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e-03; %OG
   was 3
agent.AgentOptions.CriticOptimizerOptions.GradientThreshold = 1;
agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e-04;
agent.AgentOptions.ActorOptimizerOptions.GradientThreshold = 1;

getAction(agent,{rand(obsInfo.Dimension)})



trainOpts = rlTrainingOptions(...
    MaxEpisodes=8000, ...
    MaxStepsPerEpisode=ceil(Tf/Ts), ...
    ScoreAveragingWindowLength=20, ...
    Verbose=false, ...
    Plots="training-progress",...
    StopTrainingCriteria="EpisodeReward",...
    StopTrainingValue= -50, ...
    SaveAgentCriteria = "EpisodeReward", ...
    SaveAgentValue = -50, ...
    SaveAgentDirectory = "sys10Agents");



trainingStats = train(agent,env,trainOpts);
```

A.1.2 One-Time Execution RL Agent

```
%For getting averages of PIRL nets


clc;

clear;

close all;


window_size = 10;


n_knowns = 3;

n_states = 2;


Ahat = [0,1;0,0];

Bhat = [0;0];

Ts = 0.1;

Tf = 50;

C = eye(2);

obsInfo = rlNumericSpec([4*window_size+1,1]);% + n_knowns+1,1])



actInfo = rlNumericSpec([3,1]);


open_system('executor10_simulink.slx');


env = rlSimulinkEnv("executor10_simulink","executor10_simulink/RL
    Agent",...
    obsInfo,actInfo);
```

```
error_top = 0;
error_top_A21 = 0;
error_top_A22 = 0;
error_top_B21 = 0;
error_bottom = 0;

agent = load("sys10Agents\Agent495"); %Change for Each Run
agent = agent.saved_agent;
for i = 1:100
    x0 = [rand()*0.2+0.9,rand()*0.2 + 0.9,rand*0.2 + 0.9]; %m k b
    agent.SampleTime = Ts;

    [a] = sim('executor10_simulink.slx',50);
    t1 = 0:Ts:Tf;
    t2 = 0:0.01:Tf;
    b = a.yout{1}.Values.Data(1,:)';
    c = a.yout{2}.Values.Data;
    d = a.yout{3}.Values.Data;
    e = a.yout{4}.Values.Data;
    f = a.yout{5}.Values.Data;



    error = 100*abs((e(1,1:end)-f(1))./(f(1)));
    errorA21 = mean(error);

    error = 100*abs((e(2,1:end)-f(2))./(f(2)));
    errorA22 = mean(error);

    error = 100*abs((e(3,1:end)-f(3))./(f(3)));
    errorB21 = mean(error);

    error_top_A21 = error_top_A21 + errorA21;
    error_top_A22 = error_top_A22 + errorA22;
    error_top_B21 = error_top_B21 + errorB21;

    error_top = error_top + (errorA21+errorA22+errorB21)/3;
    error_bottom = error_bottom + 1;
    [i,error_top_A21/error_bottom,error_top_A22/error_bottom,
       error_top_B21/error_bottom,error_top/error_bottom]
    error_top/error_bottom;

end
```

A.1.3 Optimal RL Agent Training

```
%Run tests in one code - better training criterion
%CURRENTLY CONFIGURED FOR DISCOUNT FACTOR


clc
close all
clear




results = [];
var_range = 1:-0.05:0.3;
mkdir('sys16Agents')

for var_n = 1:length(var_range)



    cd sys16Agents
    directory = cellstr(ls);
    directory = directory(3:end);

    for i = 1:length(directory)
        file = directory(i);
        delete(file{1})
    end

    cd ..
```

```
var = var_range(var_n);
%Trainer

n_knowns = 3;
n_states = 2;

Ts = 0.1;
Tf = 50;
window_size = 10;

obsInfo = rlNumericSpec([4*window_size + n_knowns+1,1]);
actInfo = rlNumericSpec([3,1]);

open_system("sys16")
set_param('sys16','GPUAcceleration','on')
env = rlSimulinkEnv("sys16","sys16/RL Agent",...
    obsInfo,actInfo);

env.ResetFcn = @(in) setVariable(in,'x0',[rand()*0.2+0.9,rand
   ()*0.2 + 0.9,rand*0.2 + 0.9]); %m k b

% Observation path
obsPath = [
    featureInputLayer(obsInfo.Dimension(1),Name="
       obsInputLayer")
    fullyConnectedLayer(20)
    reluLayer
    fullyConnectedLayer(20,Name="obsPathOutLayer")];

% Action path
actPath = [
    featureInputLayer(actInfo.Dimension(1),Name="
       actInputLayer")
    fullyConnectedLayer(20,Name="actPathOutLayer")];

% Common path
commonPath = [
    additionLayer(2,Name="add")
    reluLayer
    fullyConnectedLayer(1,Name="CriticOutput")];

criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,obsPath);
criticNetwork = addLayers(criticNetwork,actPath);
criticNetwork = addLayers(criticNetwork,commonPath);

criticNetwork = connectLayers(criticNetwork, ...
    "obsPathOutLayer","add/in1");
criticNetwork = connectLayers(criticNetwork, ...
    "actPathOutLayer","add/in2");

criticNetwork = dlnetwork(criticNetwork);
summary(criticNetwork)
```

```
critic = rlQValueFunction(criticNetwork, ...
    obsInfo,actInfo, ...
    ObservationInputNames="obsInputLayer", ...
    ActionInputNames="actInputLayer");
getValue(critic, ...
    {rand(obsInfo.Dimension)}, ...
    {rand(actInfo.Dimension)})

actorNetwork = [
    featureInputLayer(obsInfo.Dimension(1))
    fullyConnectedLayer(3)
    tanhLayer
    fullyConnectedLayer(actInfo.Dimension(1))
    ];

actorNetwork = dlnetwork(actorNetwork);
summary(actorNetwork)

actor = rlContinuousDeterministicActor(actorNetwork,obsInfo,
   actInfo);

getAction(actor,{rand(obsInfo.Dimension)})
agent = rlDDPGAgent(actor,critic);

agent.SampleTime = Ts;
agent.AgentOptions.TargetSmoothFactor = 1e-3;
agent.AgentOptions.DiscountFactor = var; %OG was 1
agent.AgentOptions.MiniBatchSize = 64;
agent.AgentOptions.ExperienceBufferLength = 1e6;



agent.AgentOptions.NoiseOptions.Variance = 0.3;
agent.AgentOptions.NoiseOptions.VarianceDecayRate = 1e-5;

agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e-03;
   %OG was 3
agent.AgentOptions.CriticOptimizerOptions.GradientThreshold =
    1;
agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e-04;
agent.AgentOptions.ActorOptimizerOptions.GradientThreshold =
   1;

getAction(agent,{rand(obsInfo.Dimension)})

trainOpts = rlTrainingOptions(...
    MaxEpisodes= 8000, ...
    MaxStepsPerEpisode=ceil(Tf/Ts), ...
    ScoreAveragingWindowLength=20, ...
    Plots="none", ...
    Verbose=false, ...
    StopTrainingCriteria="EpisodeReward",...
    StopTrainingValue= 10, ...
    SaveAgentCriteria = "EpisodeReward", ...
    SaveAgentValue = -100000, ...
    SaveAgentDirectory = "sys16Agents");



results
trainingStats = train(agent,env,trainOpts);

average_reward_stats = trainingStats.AverageReward;
[M,I] = max(average_reward_stats);

close_system('sys16.slx')
open_system('executor16_simulink.slx');
error_top = 0;
error_top_A21 = 0;
error_top_A22 = 0;
error_top_B21 = 0;
error_bottom = 0;

cd sys16Agents
directory = cellstr(ls);
cd ..

agent = load("sys16Agents\Agent"+I);
agent = agent.saved_agent;

for i = 1:100
```

```
x0 = [rand()*0.2+0.9,rand()*0.2 + 0.9,rand*0.2 + 0.9]; %m
    k b

[a] = sim('executor16_simulink.slx',50);
t1 = 0:Ts:Tf;
t2 = 0:0.01:Tf;
b = a.yout{1}.Values.Data(1,:)';
c = a.yout{2}.Values.Data;
d = a.yout{3}.Values.Data;
e = a.yout{4}.Values.Data;
f = a.yout{5}.Values.Data;



error = 100*abs((e(1,1:end)-f(1))./(f(1)));
errorA21 = mean(error);

error = 100*abs((e(2,1:end)-f(2))./(f(2)));
errorA22 = mean(error);

error = 100*abs((e(3,1:end)-f(3))./(f(3)));
errorB21 = mean(error);

error_top_A21 = error_top_A21 + errorA21;
error_top_A22 = error_top_A22 + errorA22;
error_top_B21 = error_top_B21 + errorB21;

error_top = error_top + (errorA21+errorA22+errorB21)/3;
```

```
    error_bottom = error_bottom + 1;
    [i,error_top_A21/error_bottom,error_top_A22/error_bottom,
       error_top_B21/error_bottom,error_top/error_bottom];
    error_top/error_bottom;

end

std_range_error = error_top/error_bottom;

error_top = 0;
error_top_A21 = 0;
error_top_A22 = 0;
error_top_B21 = 0;
error_bottom = 0;

for i = 1:100
    x0 = [rand()*0.6+0.7,rand()*0.6 + 0.7,rand*0.6 + 0.7]; %m
        k b
    agent.SampleTime = Ts;

    [a] = sim('executor16_simulink.slx',50);
    t1 = 0:Ts:Tf;
    t2 = 0:0.01:Tf;
    b = a.yout{1}.Values.Data(1,:)';
    c = a.yout{2}.Values.Data;
    d = a.yout{3}.Values.Data;
    e = a.yout{4}.Values.Data;
    f = a.yout{5}.Values.Data;



    error = 100*abs((e(1,1:end)-f(1))./(f(1)));
    errorA21 = mean(error);

    error = 100*abs((e(2,1:end)-f(2))./(f(2)));
    errorA22 = mean(error);

    error = 100*abs((e(3,1:end)-f(3))./(f(3)));
    errorB21 = mean(error);

    error_top_A21 = error_top_A21 + errorA21;
    error_top_A22 = error_top_A22 + errorA22;
    error_top_B21 = error_top_B21 + errorB21;

    error_top = error_top + (errorA21+errorA22+errorB21)/3;
    error_bottom = error_bottom + 1;
    [i,error_top_A21/error_bottom,error_top_A22/error_bottom,
       error_top_B21/error_bottom,error_top/error_bottom];
    error_top/error_bottom;

end

trp_range_error = error_top/error_bottom;
close_system('executor16_simulink.slx');
results = [results;I,M,var,std_range_error,trp_range_error]
```

```
    writematrix(results,'sys16results.txt');


end
```

A.1.4 Optimal RL Agent Training - Range of Variation

```
%CURRENTLY CONFIGURED FOR RANGE OF VARIATION


clc

close all

clear





results = [];

var_range = 0:0.1:1.9;

mkdir('sys20Agents')


for var_n = 1:length(var_range)



    cd sys20Agents

    directory = cellstr(ls);

    directory = directory(3:end);


    for i = 1:length(directory)

        file = directory(i);

        delete(file{1})

    end


    cd ..
```

```
var = var_range(var_n);
%Trainer

n_knowns = 3;
n_states = 2;

Ts = 0.1;
Tf = 50;
window_size = 10;

obsInfo = rlNumericSpec([4*window_size + n_knowns+1,1]);



actInfo = rlNumericSpec([3,1]);

open_system("sys20")
set_param('sys20','GPUAcceleration','on')
env = rlSimulinkEnv("sys20","sys20/RL Agent",...
    obsInfo,actInfo);

env.ResetFcn = @(in) setVariable(in,'x0',[(rand()-0.5)*var
   +1,(rand()-0.5)*var + 1,(rand()-0.5)*var + 1]); %m k b

% Observation path
obsPath = [
    featureInputLayer(obsInfo.Dimension(1),Name="
       obsInputLayer")
    fullyConnectedLayer(20)
    reluLayer
    fullyConnectedLayer(20,Name="obsPathOutLayer")];

% Action path
actPath = [
    featureInputLayer(actInfo.Dimension(1),Name="
       actInputLayer")
    fullyConnectedLayer(20,Name="actPathOutLayer")];

% Common path
commonPath = [
    additionLayer(2,Name="add")
    reluLayer
    fullyConnectedLayer(1,Name="CriticOutput")];

criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,obsPath);
criticNetwork = addLayers(criticNetwork,actPath);
criticNetwork = addLayers(criticNetwork,commonPath);

criticNetwork = connectLayers(criticNetwork, ...
    "obsPathOutLayer","add/in1");
criticNetwork = connectLayers(criticNetwork, ...
    "actPathOutLayer","add/in2");

criticNetwork = dlnetwork(criticNetwork);
```

```
summary(criticNetwork)

critic = rlQValueFunction(criticNetwork, ...
    obsInfo,actInfo, ...
    ObservationInputNames="obsInputLayer", ...
    ActionInputNames="actInputLayer");
getValue(critic, ...
    {rand(obsInfo.Dimension)}, ...
    {rand(actInfo.Dimension)})

actorNetwork = [
    featureInputLayer(obsInfo.Dimension(1))
    fullyConnectedLayer(3)
    tanhLayer
    fullyConnectedLayer(actInfo.Dimension(1))
    ];

actorNetwork = dlnetwork(actorNetwork);
summary(actorNetwork)

actor = rlContinuousDeterministicActor(actorNetwork,obsInfo,
   actInfo);

getAction(actor,{rand(obsInfo.Dimension)})
agent = rlDDPGAgent(actor,critic);

agent.SampleTime = Ts;
agent.AgentOptions.TargetSmoothFactor = 1e-3;
agent.AgentOptions.DiscountFactor = 0.95;
agent.AgentOptions.MiniBatchSize = 64;
agent.AgentOptions.ExperienceBufferLength = 1e6;



agent.AgentOptions.NoiseOptions.Variance = 0.3;
agent.AgentOptions.NoiseOptions.VarianceDecayRate = 1e-5;

agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e-03;
agent.AgentOptions.CriticOptimizerOptions.GradientThreshold =
    1;
agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e-04;
agent.AgentOptions.ActorOptimizerOptions.GradientThreshold =
   1;

getAction(agent,{rand(obsInfo.Dimension)})

trainOpts = rlTrainingOptions(...
    MaxEpisodes=8000, ...
    MaxStepsPerEpisode=ceil(Tf/Ts), ...
    ScoreAveragingWindowLength=20, ...
    Plots="none", ...
    Verbose=false, ...
    StopTrainingCriteria="EpisodeReward",...
    StopTrainingValue= 10, ...
    SaveAgentCriteria = "EpisodeReward", ...
    SaveAgentValue = -100000, ...
    SaveAgentDirectory = "sys20Agents");

results
trainingStats = train(agent,env,trainOpts);

average_reward_stats = trainingStats.AverageReward;
[M,I] = max(average_reward_stats);

open_system('executor20_simulink.slx');
error_top = 0;
error_top_A21 = 0;
error_top_A22 = 0;
error_top_B21 = 0;
error_bottom = 0;

cd sys20Agents
directory = cellstr(ls);
cd ..

agent = load("sys20Agents\Agent"+I);
agent = agent.saved_agent;

for i = 1:100
    x0 = [(rand()-0.5)*var+1,(rand()-0.5)*var + 1,(rand()
       -0.5)*var + 1]; %m k b
```

```
[a] = sim('executor20_simulink.slx',50);
t1 = 0:Ts:Tf;
t2 = 0:0.01:Tf;
b = a.yout{1}.Values.Data(1,:)';
c = a.yout{2}.Values.Data;
d = a.yout{3}.Values.Data;
e = a.yout{4}.Values.Data;
f = a.yout{5}.Values.Data;



error = 100*abs((e(1,1:end)-f(1))./(f(1)));
errorA21 = mean(error);

error = 100*abs((e(2,1:end)-f(2))./(f(2)));
errorA22 = mean(error);

error = 100*abs((e(3,1:end)-f(3))./(f(3)));
errorB21 = mean(error);

error_top_A21 = error_top_A21 + errorA21;
error_top_A22 = error_top_A22 + errorA22;
error_top_B21 = error_top_B21 + errorB21;

error_top = error_top + (errorA21+errorA22+errorB21)/3;
error_bottom = error_bottom + 1;
[i,error_top_A21/error_bottom,error_top_A22/error_bottom,
            error_top_B21/error_bottom,error_top/error_bottom];
        error_top/error_bottom;

    end

    std_range_error = error_top/error_bottom;

    error_top = 0;
    error_top_A21 = 0;
    error_top_A22 = 0;
    error_top_B21 = 0;
    error_bottom = 0;

    for i = 1:100
        x0 = [(rand()-0.5)*3*var+1,(rand()-0.5)*3*var + 1,(rand()
            -0.5)*3*var + 1]; %m k b

        agent.SampleTime = Ts;

        [a] = sim('executor20_simulink.slx',50);
        t1 = 0:Ts:Tf;
        t2 = 0:0.01:Tf;
        b = a.yout{1}.Values.Data(1,:)';
        c = a.yout{2}.Values.Data;
        d = a.yout{3}.Values.Data;
        e = a.yout{4}.Values.Data;
        f = a.yout{5}.Values.Data;
        error = 100*abs((e(1,1:end)-f(1))./(f(1)));
        errorA21 = mean(error);

        error = 100*abs((e(2,1:end)-f(2))./(f(2)));
        errorA22 = mean(error);

        error = 100*abs((e(3,1:end)-f(3))./(f(3)));
        errorB21 = mean(error);

        error_top_A21 = error_top_A21 + errorA21;
        error_top_A22 = error_top_A22 + errorA22;
        error_top_B21 = error_top_B21 + errorB21;

        error_top = error_top + (errorA21+errorA22+errorB21)/3;
        error_bottom = error_bottom + 1;
        [i,error_top_A21/error_bottom,error_top_A22/error_bottom,
           error_top_B21/error_bottom,error_top/error_bottom];
        error_top/error_bottom;

    end

    trp_range_error = error_top/error_bottom;
    results = [results;I,M,var,std_range_error,trp_range_error]
    writematrix(results,'sys20results.txt');
```

```
end
```

A.1.5 Extended Kalman Filter Implementation

```
clc

clear

close all


error_array_2 = [];


for i = 1:100


    sim_t_s = 0.01;

    T_s = 0.1;

    T_f = 50;


    m=(rand()-0.5)*1*0.2 + 1;

    k=(rand()-0.5)*1*0.2 + 1;

    b=(rand()-0.5)*1*0.2 + 1;


    SNR = 40;


    X = [0;0];

    X_hat = [0;0;0;0;0];

    Y = eye(2)*X;




    A_true_1 = [0,1;-k/m,-b/m];

    B_true_1 = [0,1/m]';

    C = awgn(eye(2)*X,SNR);
```

```
A_true = A_true_1*sim_t_s + eye(2);
B_true = B_true_1*sim_t_s;
Y_hist = [];
X_hat_hist = [];
C = [1,0,0,0,0;0,1,0,0,0];

syms x1 x2 x3 x4 x5 U_sym
f1 = x1 + x2*T_s;
f2 = x1*x3*T_s+(1+x4*T_s)*x2 + x5*T_s*U_sym;
f3 = x3;
f4 = x4;
f5 = x5;
f = [f1;f2;f3;f4;f5];
P = [1000,0,0,0,0;...
    0,1000,0,0,0;...
    0,0,1000,0,0;...
    0,0,0,1000,0;...
    0,0,0,0,1000];
Q = [0.01,0,0,0,0;...
    0,0.01,0,0,0;...
    0,0,100,0,0;...
    0,0,0,100,0;...
    0,0,0,0,100];
R = [10,0;0,10];
A = jacobian(f,[x1;x2;x3;x4;x5]);
for t = 0:sim_t_s:T_f
```

```
X_hat_hist = [X_hat_hist,X_hat];
Y_hist = [Y_hist,Y];
U = sin(t) + sin(t/2) + sin(t/3);
X = A_true*X + B_true*U;
Y = awgn(eye(2)*X,SNR);

if rem(t,T_s) == 0

    x1 = X_hat(1);
    x2 = X_hat(2);
    x3 = X_hat(3);
    x4 = X_hat(4);
    x5 = X_hat(5);
    A = [1,T_s,0,0,0;...
        T_s*x3,T_s*x4+1,T_s*x1,T_s*x2,T_s*U;...
        0,0,1,0,0;...
        0,0,0,1,0;...
        0,0,0,0,1];

    X_hat(1) = x1 + x2*T_s;
    X_hat(2) = x1*x3*T_s+(1+x4*T_s)*x2 + x5*T_s*U;
    X_hat(3) = x3;
    X_hat(4) = x4;
    X_hat(5) = x5;



    P = A*P*A.' + Q;
```

```
            K = P*C.'*inv(C*P*C.'+R);
            X_hat = X_hat + K*(Y - C*X_hat);
            P = (eye(5)-K*C)*P;
            X_hat = double(X_hat);

        end


    end

    error_array = [abs((X_hat_hist(3,:)-A_true_1(2,1))./A_true_1
       (2,1));...
        abs((X_hat_hist(4,:)-A_true_1(2,2))./A_true_1(2,2));...
        abs((X_hat_hist(5,:)-B_true_1(2,1))./B_true_1(2,1))]*100;

    hoz_error = mean(error_array(:,end-2500:end),2);
    error = mean(hoz_error);
    error_array_2 = [error_array_2;error];
    [i,error,mean(error_array_2)]
end
```

A.2 Longitudinal Flight Model Code

A.2.1 DATCOM Data Generation and Storage Code

```
clc

clear

%Notes

database = load('database_test_2.txt');

%database = [];

STMACH = 0.85;

cd Datcom_2020\'AC_Models'

ls

directory = cellstr(ls);

directory = directory(3:end);

n_planes = length(directory);

for aircraft_n = 2

    %Loading in aircraft from selection

    aircraft = directory(aircraft_n);

    aircraft = {'Learjet.dat'};

    cd ..

    aircraft_file_n = sprintf('AC_models/%s',aircraft{1});

    for alt = 4000:400:20000

        for fpa = 0:2:15

            for mach_req = 0.5:0.05:0.7

                keep_trying = 1

                while keep_trying == 1

                    try

                        clc

                        alt
```

```
fpa
mach_req
location = cellstr(cd);
location = location{1};
if strcmp(location,'C:\Users\natha\
   OneDrive\Documents\School\Thesis\
   Experiment 2')
    cd Datcom_2020
elseif strcmp(location,'C:\Users\natha\
   OneDrive\Documents\School\Thesis\
   Experiment 2\Datcom_2020\Bin')
    cd ..
end
%Rewriting mach
opts = delimitedTextImportOptions("
   Delimiter","ignore");
raw_AC = readcell(aircraft_file_n,opts);
mach_row = raw_AC(2);
mach_row = mach_row{1};
new_mach = sprintf('%.1f,%s%.2f,',
   mach_req,'STMACH=',STMACH);
for i = 1:length(mach_row(23:end))
    mach_row(23+i) = ' ';
end
mach_row(24:23+length(new_mach)) =
   new_mach;
raw_AC(2) = {mach_row};
```

```
alt_row = raw_AC(3);
alt_row = alt_row{1};
new_alt = sprintf('%.1f,',alt);
for i = 1:length(alt_row(14:end))
    alt_row(14+i) = ' ';
end
alt_row(14:13+length(new_alt)) = new_alt;
raw_AC(3) = {alt_row};
row = raw_AC(1);
row = row{1};
new_file = sprintf('%s',row);
for i = 2:size(raw_AC)
    row = raw_AC(i);
    row = row{1};
    if strcmp(row(1),'$')
        new_file = sprintf('%s \n %s',
           new_file,row);
    elseif strcmp(row(1:4),'NACA')||
       strcmp(row(1:4),'DAMP')||strcmp(
       row(1:4),'DERI')||strcmp(row(1:4),
       'TRIM')||strcmp(row(1:4),'NEXT')
        new_file = sprintf('%s \n%s',
           new_file,row);
    else
        new_file = sprintf('%s\n  %s',
           new_file,row);
    end
```

```
end
writematrix(new_file,'Bin/for005.dat',"
   QuoteStrings","none")
%Running DATCOM and loading the input and
    output files
cd Bin
open('DD_32x.exe')
cd ..
cd ..
pause(2)
aero = datcomimport('Datcom_2020/Bin/
   for006.dat','off',0);
aero = aero{1};
raw = readcell('Datcom_2020/Bin/for006.
   dat');
raw_AC = readcell('Datcom_2020/Bin/for005
   .dat');
%Extracting Data from files, some are not
    in aero and must be pulled
%manually
Mach0 = aero.mach; % --
altitude = aero.alt; % --
altitude_m = altitude*0.3048;
[T_m, a_m, P_m, rho_m] = atmosisa(
   altitude_m);
rho = 0.00194032033*rho_m;
%   Initial Conditions
```

```
for index = 1:size(raw,1)
    row = raw(index,1);
    row = row{1};
    if length(row) >= 4
        first_entry_mach = row(1:4);
        if strcmp(first_entry_mach,'MACH'
           )
            Vt0_row = raw(index+3,1);
        end
    end
end
Vt0_row = strsplit(Vt0_row{1},' ');
Vt0 = Vt0_row(4);
Vt0 = str2num(Vt0{1});
for index = 1:size(raw,1)
    row = raw(index,1);
    row = row{1};
    if length(row) >= 4
        first_entry_mach = row(1:4);
        if strcmp(first_entry_mach,'MACH'
           )
            S_row = raw(index+3,1);
        end
    end
end
S_row = strsplit(S_row{1},' ');
S = S_row(8);
```

```
S = str2num(S{1});
q = 1/2 * rho * Vt0^2;
%   Initial Conditions
raw_AC2 = readcell('Datcom_2020/Bin/
   for005.dat',opts);
index = 0;
notFound = 1;
while notFound == 1
    index = index + 1;
    row = raw_AC2{index};
    pause(0.1)
    if length(row) >= 2
        if strcmp(row(1:2),'WT')
            notFound = 0;
        end
    end
end
W = str2num(row(strfind(row,'WT=')+3:
   strfind(row,'.')-1));
syms alpha0 D T CL CL_w de CM_e CL_e CM
syms CL positive
syms CD positive
eq1 = CM_e == de*((aero.dcm_sym(6)-aero.
   dcm_sym(5))/(aero.delta(6)-aero.delta
   (5)));
eq2 = CL_e == de*((aero.dcl_sym(6)-aero.
   dcl_sym(5))/(aero.delta(6)-aero.delta
   (5)));
eq3 = CL == alpha0 * aero.cla(2) + aero.
   cl(2) + CL_e;
eq4 = CD == alpha0 * ((aero.cd(3)-aero.cd
   (2))/(aero.alpha(3)-aero.alpha(2))) +
   aero.cd(2);
eq5 = CM == alpha0 * aero.cma(2) + aero.
   cm(2) + CM_e;
eq6 = T + (CL*q*S)*sind(alpha0) - (CD*q*S
   )*cosd(alpha0) - W*sind(alpha0+fpa) ==
    0;
eq7 = -(CL*q*S)*cosd(alpha0) - (CD*q*S)*
   sind(alpha0) + W*cosd(alpha0 + fpa) ==
    0;
eq8 = CM == 0;
Sol = vpasolve([eq1,eq2,eq3,eq4,eq5,eq6,
   eq7,eq8]);
dT0 = double(Sol.T);
alpha0 = double(Sol.alpha0);
de0 = double(Sol.de)
CL_trim = double(Sol.CL);
CD_trim = double(Sol.CD);
%Rewriting to trim conds.
%Nalpha to 1
raw_AC = readcell('Datcom_2020/Bin/for005
   .dat',opts);
raw_AC(4) = {sprintf('NALPHA=2.0,ALSCHD
   =%.2f,%.2f,',alpha0,alpha0+1)};
row = raw_AC(1);
row = row{1};
new_file = sprintf('%s',row);
for i = 2:size(raw_AC)
    row = raw_AC(i);
    row = row{1};
    if strcmp(row(1),'$')
        new_file = sprintf('%s \n %s',
           new_file,row);
    elseif strcmp(row(1:4),'NACA')||
       strcmp(row(1:4),'DAMP')||strcmp(
       row(1:4),'DERI')||strcmp(row(1:4),
       'TRIM')||strcmp(row(1:4),'NEXT')
        new_file = sprintf('%s \n%s',
           new_file,row);
    else
        new_file = sprintf('%s\n  %s',
           new_file,row);
    end
end
writematrix(new_file,'Datcom_2020/Bin/
   for005.dat',"QuoteStrings","none")
%Importing Trimmed Case
cd Datcom_2020/Bin
open('DD_32x.exe')
cd ..
cd ..
pause(2)
%aero = datcomimport('Datcom_2020/Bin/
   for006.dat');
%aero = aero{1};
raw = readcell('Datcom_2020/Bin/for006.
   dat',opts);
raw_AC = readcell('Datcom_2020/Bin/for005
   .dat');
inertias = readcell('Inertias.txt');

try
    index = 1;
    inertia_found = 0;
    while inertia_found == 0

        if strcmp(inertias{index,1},aero.
           case)
            Iy = inertias{index,2};
            inertia_found = 1;
        end
        index = index + 1;
    end
catch
    fprintf('No Inertia\n')
    return
end
```

```
Sw = S;     %--        %   Surface area
   of wing (ft^2)
%Getting MAC
index = 0;
notFound = 1;
while notFound == 1
    index = index + 1;
    row = raw_AC2(index);
    row = row{1};
    if length(row) >= 4
        if strcmp(row(1:4),'NACA')
            wing_data_row_n = index;
            notFound = 0;
        end
    end
end
row = raw_AC2(wing_data_row_n+1,1);
row = row{1};
start_point = strfind(row,'CHRDR=')+6;
end_point = strfind(row,',')-1;
end_point = end_point(1);
root_chord = str2num(row(start_point:
   end_point));
start_point = strfind(row,'CHRDTP=')+7;
end_point = strfind(row,',')-1;
end_point = end_point(2);
tip_chord = str2num(row(start_point:
   end_point));
cw = (root_chord + tip_chord)/2;
g = 32.2;      %--          %
   Acceleration due to gravity (ft/s^2)
m = W/g;       %--          %   Mass (
   slugs)
%   Aerodynamic Coefficients at Trim
   Condition
index = 1;
notFound = 1;
while notFound == 1 && index < size(raw
   ,1)
    row = raw(index,1);
    row = row{1};
    if length(row) >= 6
        first_entry_mach = row(1:6);
        if strcmp(first_entry_mach,'
           NUMBER')
            cla_row = raw(index+6,1);
            notFound = 0;
        end
    end
    if notFound == 1
        index = index + 1;
    end
```

```
end
cla_row = strsplit(cla_row{1},' ');
cla = cla_row(8);
cla = str2num(cla{1});
cl_row = raw(index+6,1);
cl_row = strsplit(cl_row{1},' ');
clslow = cl_row(3);
clslow = str2num(clslow{1});
cd1_row = raw(index+6,1);
cd1_row = strsplit(cd1_row{1},' ');
cd1 = cd1_row(2);
a1 = cd1_row(1);
cd1 = str2num(cd1{1});
a1 = str2num(a1{1});
cdslow = cd1;
cd2_row = raw(index+7,1);
cd2_row = strsplit(cd2_row{1},' ');
cd2 = cd2_row(2);
a2 = cd2_row(1);
cd2 = str2num(cd2{1});
a2 = str2num(a2{1});
CL_alpha = cla*180/pi;
CD_alpha = ((cd2-cd1/(a2-a1)))*180/pi; %
   --
cma_row = raw(index+6,1);
cma_row = strsplit(cma_row{1},' ');
cma = cma_row(9);
cma = str2num(cma{1});

Cm_alpha = cma*180/pi;

if raw{index+25,1} == 0

    index = index + 1;

end

clad_row = raw(index+25,1);

clad_row = strsplit(clad_row{1},' ');

clad = clad_row(4);

clad = str2num(clad{1});

CL_alphadot = clad*180/pi;

cmad_row = raw(index+25,1);

cmad_row = strsplit(cmad_row{1},' ');

cmad = cmad_row(5);

cmad = str2num(cmad{1});

Cm_alphadot = cmad(1)*180/pi;

clq_row = raw(index+25,1);

clq_row = strsplit(clq_row{1},' ');

clq = clq_row(2);

clq = str2num(clq{1});

CL_q = aero.clq(1)*180/pi;

cmq_row = raw(index+25,1);

cmq_row = strsplit(cmq_row{1},' ');

cmq = cmq_row(3);

cmq = str2num(cmq{1});

Cm_q = aero.cmq(1)*180/pi;


index = 1;
```

```
notFound = 1;
while notFound == 1 && index < size(raw
   ,1)
    row = raw(index,1);
    row = row{1};
    if length(row) >= 6
        first_entry_mach = row(1:6);
        if strcmp(first_entry_mach,'
           NUMBER')
            cmslow_row = raw(index+6,1);
            notFound = 0;
        end
    end
    if notFound == 1
        index = index + 1;
    end
end
cmslow_row = strsplit(cmslow_row{1},' ');
cmslow = cmslow_row(4);
cmslow = str2num(cmslow{1});
ndm_row_check = raw(index+45,1);
if strcmp(ndm_row_check{1}(1),'0')
    index = index + 1;
end
dcl_row = raw(index+45,1);

dcl_row = strsplit(dcl_row{1},' ');
```

```
dcl = dcl_row(2);
dcl = str2num(dcl{1});
dcm = dcl_row(3);
dcm = str2num(dcm{1});
dcd = dcl_row(5);
dcd = str2num(dcd{1});
%Rewriting mach
cd Datcom_2020
raw_AC = readcell('Bin/for005.dat',opts);
mach_row = raw_AC(2);
mach_row = mach_row{1};
new_mach = sprintf('%.1f,%s%.2f,',
   mach_req+0.1,'STMACH=',STMACH);
for i = 1:length(mach_row(23:end))
    mach_row(23+i) = ' ';
end
mach_row(24:23+length(new_mach)) =
   new_mach;
raw_AC(2) = {mach_row};
row = raw_AC(1);
row = row{1};
new_file = sprintf('%s',row);
for i = 2:size(raw_AC)
    row = raw_AC(i);
    row = row{1};
    if strcmp(row(1),'$')
        new_file = sprintf('%s \n %s',
           new_file,row);
    elseif strcmp(row(1:4),'NACA')||
       strcmp(row(1:4),'DAMP')||strcmp(
       row(1:4),'DERI')||strcmp(row(1:4),
       'TRIM')||strcmp(row(1:4),'NEXT')
        new_file = sprintf('%s \n%s',
           new_file,row);
    else
        new_file = sprintf('%s\n  %s',
           new_file,row);
    end
end
writematrix(new_file,'Bin/for005.dat',"
   QuoteStrings","none")
cd Bin
open('DD_32x.exe')
cd ..
cd ..
pause(2)
raw = readcell('Datcom_2020/Bin/for006.
   dat',opts);
index = 1;
notFound = 1;
while notFound == 1 && index < size(raw
   ,1)
    row = raw(index,1);
    row = row{1};
    if length(row) >= 6
        first_entry_mach = row(1:6);
        if strcmp(first_entry_mach,'
           NUMBER')
            clfast_row = raw(index+6,1);
            notFound = 0;
        end
    end
    if notFound == 1
        index = index + 1;
    end
end
clfast_row = strsplit(clfast_row{1},' ');
clfast = clfast_row(3);
clfast = str2num(clfast{1});
cdfast_row = raw(index+6,1);
cdfast_row = strsplit(cdfast_row{1},' ');
cdfast = cdfast_row(2);
cdfast = str2num(cdfast{1});
cmfast_row = raw(index+6,1);
cmfast_row = strsplit(cmfast_row{1},' ');
cmfast = cmfast_row(4);
cmfast = str2num(cmfast{1});
cd1_row = raw(index+6,1);
cd1_row = strsplit(cd1_row{1},' ');
cd1 = cd1_row(2);
a1 = cd1_row(1);
```

```
cd1 = str2num(cd1{1});
a1 = str2num(a1{1});
cd2_row = raw(index+7,1);
cd2_row = strsplit(cd2_row{1},' ');
cd2 = cd2_row(2);
a2 = cd2_row(1);
cd2 = str2num(cd2{1});
a2 = str2num(a2{1});
CL_alpha = cla*180/pi;
CD_alpha = ((cd2-cd1/(a2-a1)))*180/pi; %
   --
cma_row = raw(index+6,1);
cma_row = strsplit(cma_row{1},' ');
cma = cma_row(9);
cma = str2num(cma{1});
Cm_alpha = cma*180/pi;
CL_Mach = (clfast-clslow)/0.1;
CD_Mach = (cdfast-cdslow)/0.1;
Cm_Mach = (cmfast-cmslow)/0.1;
CL_de = (dcl)/(5)*180/pi; %--
Cm_de = (dcm)/(5)*180/pi; %--
CD_de = (dcd)/(5)*180/pi; % --
theta0 = vpa((alpha0+fpa)*pi/180,4);
alpha0 = vpa(alpha0*pi/180,4);
u0 = vpa(Vt0*cos(alpha0),4);
w0 = vpa(Vt0*sin(alpha0),4);
%  Compute Stability Derivatives
```

```
Qw = 0.5*rho*Vt0^2;
CD_u = Mach0*CD_Mach;
CL_u = Mach0*CL_Mach;
Cm_u = Mach0*Cm_Mach;
%   Stability Derivatives
D_u = (CD_u + 2*CD_trim)*Qw*Sw/(m*u0);
L_u = (CL_u + 2*CL_trim)*Qw*Sw/(m*u0);
M_u = Cm_u*(Qw*Sw*cw)/(Iy*u0);
D_w = (CD_alpha - CL_trim)*Qw*Sw/(m*u0);
L_w = (CL_alpha + CD_trim)*Qw*Sw/(m*u0);
M_w = Cm_alpha*(Qw*Sw*cw)/(Iy*u0);
M_alpha = M_w*u0;
L_wdot = CL_alphadot*(cw/(2*u0))*Qw*Sw/(m
   *u0);
M_wdot = Cm_alphadot*(cw/(2*u0))*(Qw*Sw*
   cw)/(Iy*u0);
M_adot = M_wdot*u0;
L_q = CL_q*(cw/(2*u0))*Qw*Sw/m;
M_q = Cm_q*(cw/(2*u0))*(Qw*Sw*cw)/Iy;
L_de = CL_de*Qw*Sw/m;
M_de = Cm_de*Qw*Sw*cw/Iy;
D_de = CD_de*Qw*Sw/m;
X_u = (-D_u*cos(alpha0) + L_u*sin(alpha0)
   );
Z_u = (-D_u*sin(alpha0) - L_u*cos(alpha0)
   );
X_w = (-D_w*cos(alpha0) + L_w*sin(alpha0)
   );
Z_w = (-D_w*sin(alpha0) - L_w*cos(alpha0)
   );
X_q = L_q*sin(alpha0);
Z_q = -L_q*cos(alpha0);
X_de = (-D_de*cos(alpha0) + -D_de*cos(
   alpha0));
Z_de = (-D_de*sin(alpha0) - L_de*cos(
   alpha0));
Z_wdot = -L_wdot*cos(alpha0);
%   Form the State Space Model
Mbar_wdot = M_wdot/(1 - Z_wdot);
Z_wdot = 0;
A_long = [X_u, X_w, X_q, -g*cos(theta0),
   0;
    Z_u, Z_w, Z_q,...
    -g*sin(theta0), 0;
    M_u, M_w, M_q,...
    0, 0;
    0, 0, 1, 0, 0;
    sin(theta0), -cos(theta0), 0, Vt0,
       0];
B_long = [X_de; Z_de; M_de;0; 0];
row = raw(36);
pos1 = strfind(row,'=');
pos1 = pos1{1};
pos1 = pos1(1);
```

```
pos2 = strfind(row,',');
pos2 = pos2{1};
pos2 = pos2(1);
row = row{1};
XCG = row(pos1+1:pos2-1);
XCG = str2num(XCG);
XW = str2num(row(strfind(row,'XW=')+3:
   strfind(row,',ZW')-1));
index = 0;
notFound = 1;
while notFound == 1
    index = index + 1;
    row = raw_AC(index);
    row = row{1};
    if length(row) >= 6
        if strcmp(row(1:6),'NACA-W')
            notFound = 0;
        end
    end
end
row = raw_AC(index+2);
row = row{1};
SAVSI = str2num(row(strfind(row,'SAVSI=')
   +6:strfind(row,',CHSTAT')-1));
row = raw_AC(index+1);
row = row{1};
SSPN = str2num(row(strfind(row,'SSPN=')
   +5:strfind(row,',SSPNE')-1));
tr = tip_chord/root_chord;
y_mac = 1/3 * ((1+2*tr)/(1+tr))*SSPN;
X_C_bar_t = y_mac * tand(SAVSI);
cg_wrt_c_bar = (XCG-(X_C_bar_t+XW))/aero.
   cbar;
alt = double(alt);
mach_req = double(mach_req);
Vt0 = double(Vt0);
Iy = double(Iy);
alpha0 = double(alpha0);
theta0 = double(theta0);
fpa = double(fpa);
XCG = double(XCG);
X_C_bar_t = double(X_C_bar_t);
S = double(S);
W = double(W);
params = [alt,mach_req,Vt0,Iy,alpha0,
   theta0,fpa,XCG,cg_wrt_c_bar,S,W];
%fprintf('%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t
   %.2f\t\t%.2f\t%.2f',params)
for index1 = 1:size(A_long,1)
    for index2 = 1:size(A_long,2)
        A_long_new(index1,index2) =
           double(A_long(index1,index2));
    end
end
```

```
        A_long = A_long_new;
        for index1 = 1:size(B_long,1)
            for index2 = size(B_long,2)
                B_long_new(index1,index2) =
                   double(B_long(index1,index2));
            end
        end
        %Hello dedicated reader
        %Congrats on reading this far
        %Hope you enjoyed the thesis! :)
        B_long = B_long_new;
        database = [database;A_long(1,1:end),
           A_long(2,1:end),...
            A_long(3,1:end),A_long(4,1:end),
               A_long(5,1:end),...
            B_long(1:end,1)',params];
        %file_n = sprintf('%s.txt',aircraft{1});

        delete database_test_2.txt
        writematrix(database,'database_test_2.txt
           ');
        A_long;
        cd Datcom_2020
        keep_trying = 0;
    catch
        keep_trying = 1;
    end
```

```
                end
            end
        end
    end
end
```

A.2.2 RL Agent Training Code

```
%Run tests in one code - better training criterion
%CURRENTLY CONFIGURED FOR NEW AI OUTPUT IMPLEMENTATION - USE WITH
%SYS1EXECUTOR

clc
close all
clear




continue_training = 0
I = 7328

database = importdata('..\database2.txt');
%database = database(1:10,1:end)
zero_sum = 0;
for i=1:size(database)
    if database(i,1) == 0
        zero_sum = zero_sum + 1;
    end
end

if zero_sum > 10
    print(Badimport)
end
```

```
A22_gain = max(abs(database(:,7)));
A22_gain_2 = min(abs(database(:,7)));
A23_gain = max(abs(database(:,8)));
A23_gain_2 = min(abs(database(:,8)));
A33_gain = max(abs(database(:,13)));
A33_gain_2 = min(abs(database(:,13)));
B1_gain = max(abs(database(:,26)));
B1_gain_2 = min(abs(database(:,26)));
B2_gain= max(abs(database(:,27)));
B2_gain_2 = min(abs(database(:,27)));
B3_gain = max(abs(database(:,28)));
B3_gain_2 = min(abs(database(:,28)));
results = [];
var_range = 20:-10:0;
mkdir('sys1Agents')

for var_n = 1

    var = var_range(var_n);
    %Trainer

    n_knowns = 11;
    n_states = 5;

    C = eye(4);

    Ahat = [0,1;0,0];
```

```
Bhat = [0;0];
Ts = 1;
Tf = 50;
window_size = 10;

%[a] = sim('sys3.slx',20)

obsInfo = rlNumericSpec([8*window_size + n_knowns+1+20,1]);



actInfo = rlNumericSpec([6,1]);

open_system("sys1")
%set_param('sys1','GPUAcceleration','on')
env = rlSimulinkEnv("sys1","sys1/RL Agent",...
    obsInfo,actInfo);

env.ResetFcn = @(in) setVariable(in,'x0',database(randi(size(
   database,1)),1:end));

% Observation path
obsPath = [
    featureInputLayer(obsInfo.Dimension(1),Name="
       obsInputLayer")
    fullyConnectedLayer(500)
    tanhLayer
    fullyConnectedLayer(400)
    tanhLayer
    fullyConnectedLayer(300)
    tanhLayer
    fullyConnectedLayer(200)
    tanhLayer
    fullyConnectedLayer(100)
    tanhLayer
    fullyConnectedLayer(50)
    tanhLayer
    fullyConnectedLayer(25)
    tanhLayer
    fullyConnectedLayer(20,Name="obsPathOutLayer")];

% Action path
actPath = [
    featureInputLayer(actInfo.Dimension(1),Name="
       actInputLayer")
    fullyConnectedLayer(20,Name="actPathOutLayer")];

% Common path
commonPath = [
    additionLayer(2,Name="add")
    reluLayer
    fullyConnectedLayer(1,Name="CriticOutput")];

criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,obsPath);
criticNetwork = addLayers(criticNetwork,actPath);
criticNetwork = addLayers(criticNetwork,commonPath);

criticNetwork = connectLayers(criticNetwork, ...
    "obsPathOutLayer","add/in1");
criticNetwork = connectLayers(criticNetwork, ...
    "actPathOutLayer","add/in2");

criticNetwork = dlnetwork(criticNetwork);
summary(criticNetwork)

critic = rlQValueFunction(criticNetwork, ...
    obsInfo,actInfo, ...
    ObservationInputNames="obsInputLayer", ...
    ActionInputNames="actInputLayer");
getValue(critic, ...
    {rand(obsInfo.Dimension)}, ...
    {rand(actInfo.Dimension)})

actorNetwork = [
    featureInputLayer(obsInfo.Dimension(1))
    fullyConnectedLayer(500)
    tanhLayer
    fullyConnectedLayer(400)
    tanhLayer
    fullyConnectedLayer(300)
    tanhLayer
    fullyConnectedLayer(200)
    tanhLayer
    fullyConnectedLayer(100)
    tanhLayer
    fullyConnectedLayer(50)
    tanhLayer
    fullyConnectedLayer(25)
    tanhLayer
    fullyConnectedLayer(actInfo.Dimension(1))
    ];

actorNetwork = dlnetwork(actorNetwork);
summary(actorNetwork)

actor = rlContinuousDeterministicActor(actorNetwork,obsInfo,
   actInfo);

getAction(actor,{rand(obsInfo.Dimension)})

if continue_training == 1
    cd sys1Agents
    directory = cellstr(ls);
    cd ..

    agent = load("sys1Agents\Agent"+I);
    agent = agent.saved_agent;
else
```

```
    cd sys1Agents
    directory = cellstr(ls);
    directory = directory(3:end);

    for i = 1:length(directory)
        file = directory(i);
        delete(file{1})
    end

    cd ..
    agent = rlDDPGAgent(actor,critic);
end

agent.SampleTime = Ts;

agent.AgentOptions.TargetSmoothFactor = 1e-3;
agent.AgentOptions.DiscountFactor = 0.95; %OG was 1
agent.AgentOptions.MiniBatchSize = 64;
agent.AgentOptions.ExperienceBufferLength = 1e6;



agent.AgentOptions.NoiseOptions.Variance = 0.1;
agent.AgentOptions.NoiseOptions.VarianceDecayRate = 1e-4;

agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e-03;
   %OG was 3
```

```
agent.AgentOptions.CriticOptimizerOptions.GradientThreshold =
    1;
agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e-04;
agent.AgentOptions.ActorOptimizerOptions.GradientThreshold =
   1;

getAction(agent,{rand(obsInfo.Dimension)})

trainOpts = rlTrainingOptions(...
    MaxEpisodes=100000, ...
    MaxStepsPerEpisode=ceil(Tf/Ts), ...
    ScoreAveragingWindowLength=500, ...
    Verbose=false, ...
    StopTrainingCriteria="EpisodeReward",...
    StopTrainingValue= 1000, ...
    SaveAgentCriteria = "EpisodeReward", ...
    SaveAgentValue = -100000, ...
    SaveAgentDirectory = "sys1Agents");



results


trainingStats = train(agent,env,trainOpts);



average_reward_stats = trainingStats.AverageReward;
[M,I] = max(average_reward_stats);
```

```
%close_system('sys19.slx')
%open_system('executor1_simulink.slx');
error_top = 0;
error_top_A21 = 0;
error_top_A22 = 0;
error_top_B21 = 0;
error_bottom = 0;

cd sys1Agents
directory = cellstr(ls);
cd ..

agent = load("sys1Agents\Agent"+I);
agent = agent.saved_agent;
database2 = load('..\test_database.txt');
%database = load('..\database2.txt');
%database = database(1:10,1:end)
error_array = [];
for i = 1:100
    x0 = database2(randi(size(database2,1)),1:end);
    %x0 = [rand()*0.6+0.7,rand()*0.6 + 0.7,rand*0.6 + 0.7]; %
       m k b

    %set_param('executor11_simulink','GPUAcceleration','on');
    %agent = load("Test 4 Agents\Agent476.mat");
```

```
        [a] = sim('sys1.slx',50);

        error = mean(a.yout{1}.Values.Data)*100

        error_array = [error_array;error];

    end


    %close_system('executor19_simulink.slx');

    results = [results;I,M,var,mean(error_array)]

    writematrix(results,'sys1results.txt');


end
```

A.2.3 RL Agent Testing Code

```
clc

clear

close all

I = 6000;

agent = load("Test10\sys1Agents\Agent"+I);

agent = agent.saved_agent;

database2 = load('database_test_2.txt');



Ts = 1;

%database = load('..\database2.txt');

%database = database(1:10,1:end)


rng(0)




database = importdata('database2.txt');

%database = database(1:10,1:end)

zero_sum = 0;

for i=1:size(database)

    if database(i,1) == 0

        zero_sum = zero_sum + 1;

    end

end
```

```
if zero_sum > 10
    print(Badimport)
end

A22_gain = max(abs(database(:,7)));
A23_gain = max(abs(database(:,8)));
A33_gain = max(abs(database(:,13)));
B1_gain = max(abs(database(:,26)));
B2_gain = max(abs(database(:,27)));
B3_gain = max(abs(database(:,28)));



n_knowns = 11;
n_states = 5;

C = eye(4);

Ahat = [0,1;0,0];
Bhat = [0;0];

Tf = 50;
window_size = 10;
error_array = [];
error_array_2 = [];
for i = 1:100
    x0 = database2(randi(size(database2,1)),1:end);
```

```
%x0 = [rand()*0.6+0.7,rand()*0.6 + 0.7,rand*0.6 + 0.7]; %m k
   b

%set_param('executor11_simulink','GPUAcceleration','on');
%agent = load("Test 4 Agents\Agent476.mat");

[a] = sim('agent_tester_sys1.slx',Tf);
error = mean(a.yout{1}.Values.Data)*100;



error_array = [error_array;error];
[error,mean(error_array),i];
%scatter(x0(31),error,'k.')
%hold on
%pause(0.0001)



A22_error = mean(abs((reshape(a.trialValsA.Data(2,2,:),[1,Tf
   +1])-a.trueA.Data(2,2))./abs(a.trueA.Data(2,2))))*100;
A23_error = mean(abs((reshape(a.trialValsA.Data(2,3,:),[1,Tf
   +1])-a.trueA.Data(2,3))./abs(a.trueA.Data(2,3))))*100;
A33_error = mean(abs((reshape(a.trialValsA.Data(3,3,:),[1,Tf
   +1])-a.trueA.Data(3,3))./abs(a.trueA.Data(3,3))))*100;
B1_error = mean(abs((reshape(a.trialValsB.Data(1,:),[1,Tf+1])
   -a.trueB.Data(1))./abs(a.trueB.Data(1))))*100;
B2_error = mean(abs((reshape(a.trialValsB.Data(2,:),[1,Tf+1])
        -a.trueB.Data(2))./abs(a.trueB.Data(2))))*100;
    B3_error = mean(abs((reshape(a.trialValsB.Data(3,:),[1,Tf+1])
        -a.trueB.Data(3))./abs(a.trueB.Data(3))))*100;
    error_array_2 = [error_array_2;A22_error,A23_error,A33_error,
        B1_error,B2_error,B3_error];

    fprintf('%d\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n',i,
        mean(error_array_2,1),mean(error_array))
end


figure(1)
subplot(3,1,1)
plot(0:1:Tf,reshape(a.trialValsA.Data(2,2,:),[1,Tf+1]))
hold on
plot(0:1:Tf,ones(size(0:1:Tf))*a.trueA.Data(2,2))
ylabel('A22 Value')
ylim([-3 0])
subplot(3,1,2)
plot(0:1:Tf,reshape(a.trialValsA.Data(2,3,:),[1,Tf+1]))
hold on
plot(0:1:Tf,ones(size(0:1:Tf))*a.trueA.Data(2,3))
ylabel('A23 Value')
ylim([-9 0])
subplot(3,1,3)
plot(0:1:Tf,reshape(a.trialValsA.Data(3,3,:),[1,Tf+1]))
hold on
```

```
plot(0:1:Tf,ones(size(0:1:Tf))*a.trueA.Data(3,3))
ylabel('A33 Value')
ylim([-4 2])
xlabel('Time (s)')
legend('Trial Value','True Value')

figure(2)
subplot(3,1,1)
plot(0:1:Tf,reshape(a.trialValsB.Data(1,:),[1,Tf+1]))
hold on
plot(0:1:Tf,ones(size(0:1:Tf))*a.trueB.Data(1))
ylabel('B1 Value')
ylim([-7 0])
subplot(3,1,2)
plot(0:1:Tf,reshape(a.trialValsB.Data(2,:),[1,Tf+1]))
hold on
plot(0:1:Tf,ones(size(0:1:Tf))*a.trueB.Data(2))
ylim([-80 5])
ylabel('B2 Value')
subplot(3,1,3)
plot(0:1:Tf,reshape(a.trialValsB.Data(3,:),[1,Tf+1]))
hold on
plot(0:1:Tf,ones(size(0:1:Tf))*a.trueB.Data(3))
ylim([-50 40])
ylabel('B3 Value')
xlabel('Time (s)')
legend('Trial Value','True Value')
```

B DATCOM Files

B.1 Training

B.1.1 Beechcraft B99

```
CASEID B99

 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,

  NALT=1.0,ALT=10000.0,

  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,

  WT=10000.0,

  LOOP=2.0$

 $OPTINS SREF=278.5,CBARR=6.42,BLREF=47.08$

 $SYNTHS XCG=18.9,ZCG=3.125,XW=17.0,ZW=0.00,ALIW=1.5,XH=35.0,ZH=3.96,

  ALIH=0.0,XV=31.88,ZV=1.86,XVF=31.88,VERTUP=.TRUE.$

 $SYNTHS HINAX=36.0$

$BODY NX=4.0,

   X =0.0,5.0,33.0,44.0,

   R =0.0,2.7,2.7,0.0$

NACA-W-5-23015

 $WGPLNF CHRDR=8.9,CHRDTP=2.94,SSPN=23.54,SSPNE=20.8,

  SAVSI=6.0,CHSTAT=0.25,TWISTA=3.0,DHDADI=4.0,TYPE=1.0$

NACA-H-6-64A010

 $HTPLNF CHRDR=5.0,CHRDTP=2.5,SSPN=9.42,SSPNE=9.22,

  SAVSI=10.0,CHSTAT=0.0,TWISTA=0.0,DHDADI=9.0,TYPE=1.0$

 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,

  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,

  CHRDFO=2.0,SPANFI=2.3,SPANFO=9.42,NTYPE=1.0$

DAMP

DERIV DEG

TRIM
```

```
NEXT CASE
```

B.1.2 Boeing 737

```
CASEID BOEING 737
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=115000.,
  LOOP=2.0$
 $OPTINS SREF=1329.9,CBARR=14.3,BLREF=93.0$
 $SYNTHS XCG=40.3,ZCG=0.0,XW=28.3,ZW=-1.4,ALIW=1.0,
  XH=76.6,ZH=6.2,ALIH=0.0,XV=70.1,ZV=8.14$
 $SYNTHS HINAX=80.0$
 $BODY NX=14.,
  BNOSE=2.,BTAIL=2.,BLA=20.0,
  X(1)=0.,1.38,4.83,6.90,8.97,13.8,27.6,55.2,
   65.6,69.0,75.9,82.8,89.7,90.4,
  ZU(1)=.69,2.07,3.45,4.38,5.87,6.90,8.28,
   8.28,8.28,8.28,7.94,7.59,7.50,6.9,
  ZL(1)=-.35,-1.73,-3.45,-3.80,-4.14,-4.49,-4.83,
   -4.83,-3.45,-2.76,-0.81,1.04,4.14,6.21,
  R(1)=.34,1.38,2.76,3.45,4.14,5.18,6.21,6.21,
   5.87,5.52,4.14,2.76,.69,0.0$
NACA-W-4-0012-25
 $WGPLNF CHRDR=23.8,CHRDTP=4.8,SSPN=46.9,SSPNE=40.0,
  SAVSI=29.0,CHSTAT=0.25,TWISTA=8.0,TYPE=1.0$
NACA-H-4-0012-25
 $HTPLNF CHRDR=12.4,CHRDTP=4.1,SSPN=17.6,SSPNE=15.87,CHSTAT=.25,
 TWISTA=0.,TYPE=1.,SAVSI=31.,DHDADI=9.$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,
  CHRDFO=2.,SPANFI=0.,SPANFO=17.6,NTYPE=1.0$
NACA-V-4-0012-25
 $VTPLNF CHRDR=17.8,CHRDTP=6.37,SAVSI=40.1,
  SSPN=20.3,SSPNOP=0.,SSPNE=20.3,CHSTAT=0.0,TWISTA=0.,TYPE= 1.$
DAMP
DERIV DEG
TRIM
NEXT CASE
```

B.1.3 Beechcraft T34C

```
CASEID Beechcraft T-34C aircraft
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=4000.0,
  LOOP=2.0$
 $OPTINS SREF=179.9,CBARR=5.42,BLREF=33.396$
 $SYNTHS XCG=10.24,ZCG=3.7,XW=6.02,ZW=-1.1,ALIW=3.81,
         XH=21.4,ZH=1.2,ALIH=0.0,XV=20.9,ZV=0.89$
 $SYNTHS HINAX=21.6$
 $BODY NX=16.0,ITYPE=1.0,METHOD=1.0,
  X=0.0,1.11,1.78,5.01,5.48,6.33,6.59,7.09,7.62,9.54,12.9,15.2,
  18.3,21.0,21.6,25.7,
  ZU=0.45,0.98,1.34,1.83,1.92,2.67,2.9,3.25,3.39,3.39,3.39,2.32,
  2.14,2.36,1.87,0.94,
  ZL=0.45,0.0,-1.38,-1.78,-1.78,-1.78,-1.78,-1.78,-1.78,-1.78,
  -1.43,-1.29,-0.76,-0.31,-0.22,0.94,
  R=0.0,0.49,1.4,1.78,1.83,2.21,2.29,2.5,2.61,2.61,2.61,1.8,
  1.4,1.38,1.07,0.0,$
NACA-W-5-23016
 $WGPLNF CHRDR=6.68,CHRDTP=3.92,SSPN=16.0,SSPNE=14.1,
  SAVSI=3.98,CHSTAT=0.25,TYPE=1.0$
NACA-H-4-0009
 $HTPLNF CHRDR=3.57,CHRDTP=2.45,SSPN=5.79,SSPNE=5.35,SAVSI=3.98,CHSTAT=0.0,
  DHDADI=7.21,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=1.0,
  CHRDFO=0.8,SPANFI=0.,SPANFO=5.79,NTYPE=1.0$
NACA-V-4-0009
 $VTPLNF SAVSI=7.5,CHSTAT=0.0,TYPE=1.0,CHRDTP=2.90,SSPNE=4.72,SSPN=5.53,
  CHRDR=4.46$
DAMP
DERIV DEG
TRIM
NEXT CASE
```

B.1.4 Cessna 182

```
CASEID CESSNA 182
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=2950.0,
  LOOP=2.0$
 $OPTINS SREF=174.0,CBARR=4.90,BLREF=36.0$
 $SYNTHS XCG=7.29,ZCG=3.25,XW=6.05,ZW=6.80,ALIW=1.5,
         XH=19.53,ZH=3.82,ALIH=0.0,XV=20.26,ZV=4.25$
 $SYNTHS HINAX=22.0$
 $BODY NX=20.0,ITYPE=2.0,METHOD=2.0,
  X=0.34,0.85,1.80,2.21,3.82,5.01,5.73,6.16,8.49,10.15,
    11.55,12.95,14.39,15.92,17.75,19.27,20.76,22.29,23.69,24.84,
  ZU=4.84,5.26,5.31,5.35,5.35,6.20,6.71,7.05,6.96,6.71,
     5.77,5.14,4.97,4.92,4.80,4.67,4.54,4.42,4.33,4.25,
  ZL=3.74,2.89,2.38,2.29,2.00,1.95,1.95,2.04,2.17,2.34,
     2.45,2.55,2.72,2.80,2.89,3.01,3.18,3.31,3.57,3.74,
  S=7.63,20.45,21.76,22.88,24.37,28.03,31.54,34.55,35.08,32.20,
   26.38,23.37,19.90,17.51,14.92,12.13,10.00,6.99,4.79,0.65,
  P=17.61,18.76,18.46,18.74,20.46,21.76,22.80,22.91,22.22,19.99,
    18.69,18.04,17.59,17.08,16.57,16.32,16.19,16.05,15.98,0.31,
  R=0.57,1.60,1.80,1.91,2.11,2.19,2.32,2.42,2.45,2.27,
    2.06,1.88,1.70,1.44,1.24,1.00,0.82,0.57,0.39,0.05$
NACA-W-4-2412
 $WGPLNF CHRDR=5.33,CHRDTP=3.5,SSPN=18.0,SSPNE=16.00,
  SAVSI=0.0,CHSTAT=0.0,SAVSO=3.0,CHRDBP=5.33,SSPNOP=9.625,
  TWISTA=-3.0,SSPNDD=9.625,DHDADI=1.73,DHDADO=1.73,TYPE=1.0$
NACA-H-4-0012
 $HTPLNF CHRDR=4.55,CHRDTP=2.52,SSPN=5.67,SSPNE=4.5,
  SAVSI=9.0,CHSTAT=0.0,TWISTA=0.0,DHDADI=0.0,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=2.0,
  CHRDFO=2.0,SPANFI=1.18,SPANFO=5.67,NTYPE=1.0$
NACA-V-4-0009
 $VTPLNF CHRDR=4.58,CHRDTP=2.12,SSPN=4.45,SSPNE=4.25,
  SAVSI=35.0,CHSTAT=0.25,TYPE=1.0$
DAMP
DERIV DEG
TRIM
NEXT CASE
```

B.1.5 Cessna Citation I

```
CASEID Citation I Model 500
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=10000.0,
  LOOP=2.0$
 $OPTINS SREF=278.5,CBARR=6.42,BLREF=47.08$
 $SYNTHS XCG=18.9,ZCG=3.125,XW=17.0,ZW=0.00,ALIW=1.5,XH=35.0,ZH=3.96,
  ALIH=0.0,XV=31.88,ZV=1.86,XVF=31.88,VERTUP=.TRUE.$
 $SYNTHS HINAX=36.0$
 $BODY NX=10.0,
   X =0.0,1.0,2.7,6.3,9.8,7.3,25.0,29.3,2.3,43.5,
  ZU =0.0,0.8,1.4,1.9,3.7,3.7,3.7,3.2,3.0,0.0,
  ZL =0.0,-0.8,-1.2,-1.3,-1.4,-1.4,-1.4,-0.5,0.0,2.0,
   R =0.0,1.2,1.9,2.5,2.6,2.6,5.4,4.5,1.6,0.0$
NACA-W-5-23015
 $WGPLNF CHRDR=8.9,CHRDTP=2.94,SSPN=23.54,SSPNE=20.8,
  SAVSI=6.0,CHSTAT=0.25,TWISTA=3.0,DHDADI=4.0,TYPE=1.0$
NACA-H-6-64A010
 $HTPLNF CHRDR=5.0,CHRDTP=2.5,SSPN=9.42,SSPNE=9.22,
  SAVSI=10.0,CHSTAT=0.0,TWISTA=0.0,DHDADI=9.0,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,
  CHRDFO=2.0,SPANFI=2.3,SPANFO=9.42,NTYPE=1.0$
DAMP
DERIV DEG
```

```
TRIM

NEXT CASE
```

B.1.6 Cessna Citation II 550

```
CASEID Citation II Model 550
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=8000.0,
  LOOP=2.0$
 $OPTINS SREF=320.8,CBARR=6.75,BLREF=51.7$
 $SYNTHS XCG=21.9,ZCG=3.125,XW=19.1,ZW=-0.95,ALIW=2.5,XH=39.2,ZH=3.46,
 ALIH=0.0,XV=34.76,ZV=1.56,XVF=-45.0,SCALE=1.0,VERTUP=.TRUE.$
 $SYNTHS HINAX=42.0$
 $BODY NX=8.0,BNOSE=1.0,BTAIL=1.0,BLN=8.8,BLA=19.7,ITYPE=1.0,METHOD=1.0,
  X(1)=0.0,1.0,2.7,6.0,8.8,28.5,39.4,44.8,
  R(1)=0.0,1.25,2.1,2.7,2.76,2.7,1.25,0.39,
  ZU(1)=0.0,0.86,1.3,1.9,3.63,3.37,2.33,1.73,
  ZL(1)=0.0,-0.86,-1.04,-1.56,-1.56,-1.81,0.0,1.3$
NACA-W-5-23014
 $WGPLNF CHRDR=9.42,CHRDTP=3.01,SSPN=25.85,SSPNE=23.46,
  SAVSI=1.3,CHSTAT=0.25,TWISTA=-3.0,DHDADI=3.6,DHDADO=0.0,TYPE=1.0$
NACA H 4 0010
 $HTPLNF CHRDR=4.99,CHRDTP=2.48,SSPN=9.42,SSPNE=9.21,
  SAVSI=5.32,CHSTAT=0.25,TWISTA=0.0,DHDADI=9.2,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,
  CHRDFO=2.0,SPANFI=0.0,SPANFO=9.42,NTYPE=1.0$
NACA V 4 0012
 $VTPLNF CHRDTP=3.63,SSPNE=8.85,SSPN=9.42,CHRDR=9.42,SAVSI=32.3,CHSTAT=0.25,
  TYPE=1.0$

DAMP

DERIV DEG

TRIM

NEXT CASE
```

B.1.7 Ryan Navion

```
CASEID NAVION
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=2750.0,
  LOOP=2.0$
 $OPTINS SREF=180.0,CBARR=5.67,BLREF=33.38$
 $SYNTHS XCG=8.00,ZCG=-0.47,XW=6.28,ZW=-2.12,ALIW=2.0,XH=21.64,ZH=0.78,
 ALIH=0.0,XV=23.21,ZV=0.0,XVF=19.76,ZVF=1.25,VERTUP=.TRUE.$
 $SYNTHS HINAX=23.0$
 $BODY NX=18.0,ITYPE=1.0,
  X=0.0,0.314,0.666,2.352,4.077,5.449,6.115,6.939,7.644,8.311,
   9.840,11.055,12.505,14.191,17.327,20.503,23.639,27.755,
  ZU=1.019,1.372,1.490,1.764,2.038,2.078,2.509,2.979,3.136,3.215,
   3.136,2.900,2.470,1.686,1.450,1.215,0.862,0.548,
  ZL=-1.019,-1.372,-1.490,-1.764,-2.038,-2.117,-2.156,-2.195,-2.195,
   -2.195,-2.195,-2.156,-2.117,-1.960,-1.568,-1.176,-0.862,-0.392,
  S=3.765,6.422,7.433,9.992,12.799,13.815,15.802,17.685,
   18.552,18.823,18.384,17.130,14.969,10.887,6.881,3.904,2.163,0.125,
  P=6.913,8.999,9.668,11.207,12.683,13.176,14.114,15.019,15.399,
   15.533,15.003,14.765,13.749,11.702,9.299,7.039,5.618,2.292,
  R=1.176,1.490,1.568,1.803,1.999,2.097,2.156,2.176,2.215,2.215,
   2.195,2.156,2.078,1.901,1.470,1.039,0.627,0.078$
NACA-W-4-4415
 $WGPLNF CHRDR=6.89,CHRDTP=3.90,SSPN=16.69,SSPNE=14.48,
  SAVSI=2.90,CHSTAT=0.00,TWISTA=-3.0,DHDADI=7.50,TYPE=1.0$
```

```
NACA-H-4-0012
 $HTPLNF CHRDR=4.00,CHRDTP=2.60,SSPN=6.58,SSPNE=6.19,
  SAVSI=6.0,CHSTAT=0.00,TWISTA=0.0,DHDADI=0.00,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=2.0,
  CHRDFO=1.0,SPANFI=0.0,SPANFO=6.58,NTYPE=1.0$
NACA-V-4-0012
 $VTPLNF CHRDR=4.40,CHRDTP=2.10,SSPN=4.80,SSPNE=4.39,
  SAVSI=20.0,CHSTAT=0.00,TYPE=1.0$
DAMP
DERIV DEG
TRIM
NEXT CASE
```

B.1.8 North American P51

```
CASEID P-51D
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=8000.0$
 $OPTINS SREF=235.0,CBARR=6.3,BLREF=37.5$
 $SYNTHS XCG=8.25,ZCG=-1.0,XW=8.82,ZW=-2.48,ALIW=1.0,XH=25.6,ZH=1.5,
 ALIH=2.0,XV=26.5,ZV=1.0$
 $SYNTHS HINAX=27.0$
 $BODY BNOSE=2.0,BLN=2.0,NX=20.0,
  X(1)=0.0,2.2,2.59,4.81,7.39,10.9,11.8,12.3,13.5,13.9,14.4,15.4,
   16.3,17.1,18.9,22.7,26.5,30.1,31.2,32.2,
  ZU(1)=0.11,1.06,1.11,1.53,1.69,1.95,2.64,2.64,3.01,2.96,2.75,2.75,
   2.75,2.69,2.59,2.27,1.85,1.58,1.37,1.11,
  ZL(1)=-0.32,-1.21,-1.53,-2.11,-2.43,-2.59,-2.59,-2.59,-2.59,-3.27,
   -3.43,-3.43,-3.33,-3.22,-2.91,-1.48,-0.58,0.0,0.37,0.69,
  R(1)=0.26,1.08,1.32,1.80,2.09,2.3,2.64,2.64,2.8,2.75,3.17,3.09,
   3.09,3.01,2.96,1.85,1.21,0.77,0.53,0.21$
NACA-W-6-63A216
 $WGPLNF CHRDR=8.56,CHRDTP=4.07,SSPN=17.91,SSPNE=16.11,
  SAVSI=3.54,CHSTAT=0.0,DHDADI=6.77,TYPE=1.0,TWISTA=-2.0$
NACA-H-6-63A014
 $HTPLNF CHRDR=4.23,CHRDTP=2.91,SSPN=6.87,SSPNE=6.34,
  SAVSI=5.4,CHSTAT=0.0,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,
  CHRDFO=2.0,SPANFI=0.0,SPANFO=6.87,NTYPE=1.0$
DAMP
DERIV DEG
TRIM
NEXT CASE
```

B.1.9 Northrop T38

```
CASEID T-38A
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=10500.0,
  LOOP=2.0$
 $OPTINS SREF=170.0,CBARR=7.73,BLREF=25.25$
 $SYNTHS XCG=25.000,ZCG=1.15,XW=20.1667,ZW=0.00,ALIW=0.0,
        XH=34.500,ZH=0.00,ALIH=0.0,XV=32.0,ZV=1.15$
 $SYNTHS HINAX=36.0$
 $BODY NX=20.0,
  X(1)=0.00,2.00,4.00,6.00,8.00,10.00,12.00,14.00,16.00,18.00,
   18.75,22.20,25.20,27.00,28.20,31.25,34.50,37.75,41.00,44.25,
  ZU(1)=0.00,0.65,1.05,1.50,2.25,3.20,3.90,4.15,4.20,4.15,
   4.10,3.80,3.50,3.20,3.10,2.80,2.60,2.40,2.20,1.90,
  ZL(1)=0.00,-0.65,-0.90,-1.00,-1.10,-1.15,-1.10,-1.00,-0.90,-0.85,
   -0.80,-0.70,-0.65,-0.65,-0.65,-0.60,-0.50,-0.40,-0.10,0.40,
  S(1)=0.00,1.95,4.29,7.50,11.06,15.23,17.75,18.28,18.11,17.75,
   29.40,25.20,19.51,16.56,16.50,17.00,15.50,12.60,8.86,4.50,
  P(1)=0.00,5.60,8.30,11.00,13.30,15.70,17.10,17.40,17.30,17.10,
   21.80,20.20,17.70,16.30,16.30,16.80,16.20,14.60,12.30,9.00,
  R(1)=0.00,0.75,1.10,1.50,1.65,1.75,1.78,1.78,1.78,1.78,
   3.00,2.80,2.35,2.15,2.20,2.50,2.50,2.25,1.93,1.50$
NACA-W-6-65A005
 $WGPLNF CHRDR=11.221,CHRDTP=2.244,SSPN=12.625,SSPNE=10.2,
  SAVSI=24.0,CHSTAT=0.25,TYPE=1.0$
```

```
NACA-H-6-65A004
 $HTPLNF CHRDR=6.667,CHRDTP=1.667,SSPN=7.083,SSPNE=4.8$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,
  CHRDFO=1.667,SPANFI=2.3,SPANFO=7.083,NTYPE=1.0$
NACA-V-6-65A004
 $VTPLNF CHRDR=10.3,CHRDTP=2.5,SSPN=8.0,SSPNE=6.7,
  SAVSI=25.0,CHSTAT=0.25,TYPE=1.0$
DAMP
DERIV DEG
TRIM
NEXT CASE
```

B.2 Testing

B.2.1 Learjet 35

```
CASEID Learjet
 $FLTCON NMACH=1.0,MACH=0.5,STMACH=0.8,
  NALT=1.0,ALT=10000.0,
  NALPHA=5.0,ALSCHD=-2.0,0.0,2.0,4.0,6.0,
  WT=14740.0,
  LOOP=2.0$
 $OPTINS SREF=230.0,CBARR=7.0,BLREF=34.0$
 $SYNTHS XCG=24.0,ZCG=-1.0,XW=20,ZW=1.59,ALIW=1.0,XH=41.2,ZH=-9.0,
 ALIH=0.0,VERTUP=.TRUE.$
 $SYNTHS HINAX=43.0$
 $BODY NX=4.0,
  X=0.0,5.0,35.0,42.0,
  R=0.0,2.0,2.0,0.0$
NACA-W-6-64-110
 $WGPLNF CHRDR=10.15,CHRDTP=5.14,SSPN=16.5,SSPNE=13.96,
  SAVSI=16.26,CHSTAT=0.25,TWISTA=2.0,DHDADI=4.76,TYPE=1.0$
NACA-H-6-64-010
 $HTPLNF CHRDR=5.08,CHRDTP=2.54,SSPN=8.57,SSPNE=7.93,
  SAVSI=26.57,CHSTAT=0.0,TWISTA=1.0,TYPE=1.0$
 $SYMFLP FTYPE=1.0,NDELTA=9.0,DELTA(1)=-20.0,-15.0,-10.0,-5.0,0.0,
  5.0,10.0,15.0,20.0,PHETE=0.0,PHETEP=0.0,CHRDFI=3.0,
  CHRDFO=2.0,SPANFI=0.0,SPANFO=8.57,NTYPE=1.0$
DAMP
DERIV DEG
TRIM
```

```
NEXT CASE
```