Master of Science Thesis

Incremental Model Based Actor Critic Designs for Optimal Adaptive Flight Control

Investigation and Implementation of Online Flight Control Methods

Shanza Ali Zafar

04-01-2018

Faculty of Aerospace Engineering · Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

Incremental Model Based Actor Critic Designs for Optimal Adaptive Flight Control

Investigation and Implementation of Online Flight Control Methods

Master of Science Preliminary Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Shanza Ali Zafar

04-01-2018

Faculty of Aerospace Engineering · Delft University of Technology

Delft University Of Technology

Department Of

Control and Operations

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Incremental Model Based Actor Critic Designs for Optimal Adaptive Flight Control"** by **Shanza Ali Zafar** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 04-01-2018

Supervisor: _____

Dr ir Q.P. Chu

Reader: _____

Dr ir E. van Kampen

Reader: _____

ir. Y. Zhou

Reader: _____

Dr.ir. W. (Wouter) van der Wal

Summary

The goal of this project is to investigate and implement online optimal and adaptive controllers based on Actor Critic Designs for flight control. For online learning, nonlinear system is approximated locally by linear model using incremental form. In this project, novel controllers based on incremental models are discussed which includes Incremental Model based Dual Heuristic Programming(IDHP), and Incremental Model based Action Dependent Dual Heuristic Programming(IADDHP). Also, already existing model-free Action Dependent Dual Heuristic Programming (ADDHP) controller is also investigated which uses Finite Difference Method (FDM) instead of Incremental Model.

Three experiments are carried out to evaluate the performance of aforementioned online model-free controllers for flight control. In first experiment, online system identification methods are compared that are Sliding Window- Ordinary Least Square(SW-OLS), and Recursive Least Square(RLS) to select one method to compare rest of the controllers. In second experiment, Incremental model based Dual Heuristic Programming controller (IDHP) and its Action Dependent (AD) form is compared based on success rate to identify the significance of action dependence in DHP controllers. In third experiment, IADDHP is compared with the existing *model-free* ADDHP controller in the literature which uses FDM to find the require system dynamics. Comparison is done based on success rate, performance under noise, and ability to adapt to changes in system dynamics. All these experiments are implemented for nonlinear missile model control.

Experiments have shown that the RLS for online identification is better than SW-OLS due to its quick convergence. It was also shown that the Action Dependent form of IDHP has higher success rate and thus better performance. Similarly, it also performs better than already existing ADDHP controller using FDM under normal conditions, noise, and in failures. In short, among all online model-free methods, IADDHP controller has shown best performance given same parameters.

Acknowledgements

I would like to start by thanking my family who provided me with their unwavering support throughout this journey and always encouraged me. A special thanks to my mother whose love knows no bound and for all her struggles to bring me to this point. To Shehryar, thank you for being my constant support for past two and a half years. I would also like to thank all my friends and colleagues for their support and encouragement.

I am grateful to Ye Zhou (Doctoral candidate), for laying the foundation of this novel project and properly steering me in right direction when I was stuck. I would also like to offer my gratitude to Dr. Erik-Jan van Kampen for helping me even with my the most elementary problems. Finally, last but not least, I would like to express my deepest gratitude for my supervisor Dr. QiPing Chu for his time and guidance. I thoroughly enjoyed and appreciate our long discussions during the meetings. Thank you for helping me out whenever I needed help.

Delft, The Netherlands
04-01-2018

Shanza Ali Zafar

Contents

List of Figures

List of Tables

Nomenclature

Greek Symbols

α	Angle of attack
δ_e	Elevator deflection
γ	Discount factor
Λ	Forgetting factor for RLS
λ	Derivative of cost-to-go with critic input
ρ	Gaussian noise
σ	Standard deviation

Subscripts

$_t$	at time t
$_a$	Actor
$_c$	Critic
$_{ji}$	jth input to jth hidden neuron
$_{kj}$	jth hidden neuron to kth output

Superscripts

*	Optimal
$^\intercal$	Transpose

$^{(1)}$	Input to hidden layer
$^{(2)}$	Hidden layer to output layer

Roman Symbols & Abbreviations

ACD	Actor Critic Designs
AD	Action Dependent
c_t	Local cost function at time t
d_t	Desired states at time t
DHP	Dual Heuristic Programming
DP	Dynamic Programming
e_t	Innovation error
F_{t-1}	System matrix
FDM	Finite Difference Method
G_{t-1}	Control input matrix
GPI	Generalized Policy Iteration
$IDHP$	Incremental Dual Heuristic Programming
J_t	Cost-to-go function at time t
l	Learning rate
M	Length of Sliding Window
M	Mach number
m	Number of control inputs
$MIMO$	Multiple Inputs, Multiple Outputs
n	Number of states
q	Pitch rate
RL	Reinforcement Learning
RLS	Recursive Least Square
SGD	Stochastic Gradient Descent
$SW-OLS$	Sliding Window- Ordinary Least Square
u_t	Control input
x_t	System's state at time t
y_t	Input to ADDHP $[x_t u_t]$

Part I

Scientific Paper

Online Action Dependent Dual Heuristic Dynamic Programming using Incremental Model for Flight Control

Shanza Ali Zafar

***Abstract*—This paper furthers the online *model-free* approaches in Approximate Dynamic Programming (ADP) by developing Incremental Model based Action Dependent Dual Heuristic Programming (IADDHP). In IADDHP, local system dynamics is identified online which does not require any priori knowledge about the model thus making it essentially *'model-free'*. Experiments are performed using missile model for reference tracking control and the results show that the IADDHP is capable of finding near-optimal control for the task with noise and system failure. It also outperforms the already existing *model-free* ADDHP which uses finite difference method (FDM) and has advantage over it in failure detection and adaptation. Being a *model-free* method, IADDHP can be used for reference tracking control for any system.**

***Index Terms*—Action dependent dual heuristic programming (ADDHP),Finite Difference Method (FDM),Incremental model, model-free controllers, online learning, Reinforcement Learning (RL).**

I. Introduction

OPTIMAL adaptive control is of paramount importance for all control systems especially flight control as it achieves the desired objective effectively under all conditions without unnecessary control effort. To solve optimal control problem for nonlinear systems, it is required to solve Jacobi-Hamilton-Bellman (JHB) equations which is rather complicated because it involves solving nonlinear partial difference equations [1]. Most of the classical optimal controllers for nonlinear systems are designed by linearizing the system around certain operating points. These types of controllers provide suboptimal performance at other operating points and require gain scheduling. For this reason, we turn to reinforcement learning (RL) to find adaptive optimal controllers.

Consider a nonlinear discrete system which can be written as;

$$x_{t+1} = f(x_t, u_t) \tag{1}$$

where *t* is the discrete time steps and $x_t \in \mathbb{R}^n \times 1$ and $u_t \in \mathbb{R}^m \times 1$. Associated performance index or *Cost-to-go* function at time *t* can be defined as Eq.2 which is the discounted sum of all the future rewards at 't'.

$$J_t = \sum_{k=t}^{\infty} \gamma^{k-t} c_k \tag{2}$$

γ is the discount factor with value ranging in $0 < \gamma < 1$ and it describes the current worth of future rewards. c_k is the local cost function or utility function.

We can use Dynamic Programming (DP) to find the optimal control policy compromised of optimal sequential control actions u_t^* *for* $t = 0, 1, 2....$ such that the cost-to-go function is minimized(maximized) given complete system information. Bellman principal of optimality for discrete time can be written as,

$$J_t^* = \min_{u_t} c_t + \gamma J_{t+1}^* \tag{3}$$

where $J_{t+1} = J(f(x_t, u_t))$. This recursive relation of value function allows for the control action to be optimized one step at a time by working backward in time. Optimal control action $u^*(x_t)$ is the one which minimizes the optimal cost-to-go function and can be written as Eq.4.

$$u^*(x_t) = \operatorname*{argmin}_{u_t} c_t + \gamma J_{t+1}^* \tag{4}$$

DP is a very powerful tool that deals with the problem of optimal control using sequential control actions given the perfect model of the system by using the Eq.3. This is done through exhaustive research from final point to the starting point to look for the optimal policy by rejecting the suboptimal paths at different points [2]. But, it remembers all the paths which makes it quite computationally expensive for large state space hence the *Curse of Dimensionality*. Also, the backward search makes it impossible for the DP to be applied for real time applications [3]. In addition to that, for complicated nonlinear systems it is very difficult to obtain the exact value of J^*. Another problem that we face with normal tabular methods of RL is that of *generalization* that is how can experience from a small state space be approximated to the larger state space. This is important because we would like to implement the RL controllers online for the states that has not been encountered before.

To solve these problems, instead of finding the exact value of cost-to-go function, *function approximators* can be used to find an approximate value. Another important development in the RL method is the *Actor Critic structures* in which *actor* selects the action whereas *critic* evaluates the value function. Separation of policy evaluation and policy improvement makes it simpler to implement and learn. [4]. Using the approximation power of Neural Networks (NN) and combining it with concepts of DP and actor critic structures led to the development of Approximate Dynamic Programming (ADP) or alternatively called Actor Critic Designs (ACD) in literature. Although any function approximator can be used instead of NN but NN are preferred due to their high approximation

power. Traditionally, critic NN is used to approximate cost-to-go function *(J)* and performs policy evaluation whereas actor NN approximates the control action. Critic and actor interact with each other to iteratively steer the control policy to the optimal policy and this is done by using model NN.

Paul J. Werbos has generalized the previously suggested structure for ADP in following categories; Heuristic Dynamic Programming (HDP) in which critic approximates the desired cost function (J) shown in Fig.1, Action Dependent HDP (AD-HDP) in which actor output is directly connected to the critic of HDP, Dual Heuristic Dynamic Programming (DHDP) in which critic approximates the derivative of J with respect to its states, and Generalized Dual Heuristic Dynamic Programming (GDHP) which approximate both J and its derivatives [5]. Comparison of these methods has shown that the ADHDP applied to the auto-landing problem has lower success ratio than HDP. DHP performs better than HDP as they are directly approximating the derivative instead of indirect calculation of derivative in HDP obtained by backpropogation through critic NN required for training of critic [6] [7]. GDHP does not significantly outperform the DHP in auto-landing task but its training is much more computationally expensive because second derivative term that need to be calculated at every time step [8]. Prokhorov & Wunsch furthered these categories by introducing the AD forms of DHP and GDHP as advanced actor critic designs and experimentally showed that these are better than HDP and ADHDP [3]. Prokhorov & Wunsch also suggested that although GDHP and ADGDHP outperforms other methods, for most applications DHP and ADDHP are adequate. Many of these structures has been used for different flight control applications. In most of these examples, first; Model network is trained offline for baseline controller and then the controller adapts to the changes in dynamics online [9], [10].

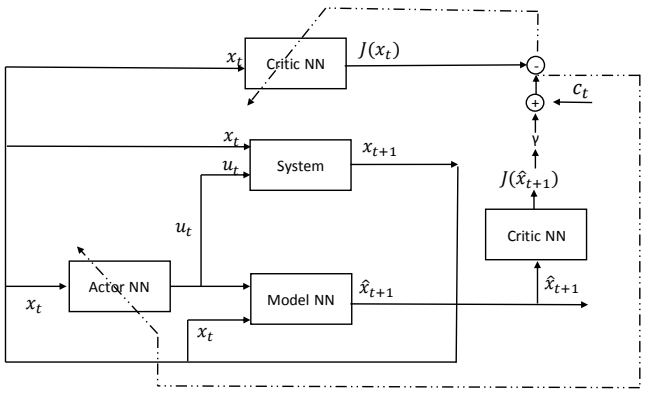

Fig. 1: HDP: An Example of ACD

As mentioned before that model NN is essential to maintain the backpropogation paths but it also requires offline training. To properly train the model NN, a large amount of data distributed over the entire state space is required to find appropriate global model estimate of the system. But in most applications, especially in flight control, obtaining this data can be quite expensive. Also it is not possible to predict beforehand which states will be encountered during the operation. Therefore, *model-free* approaches in ACD presents themselves as quite a lucrative option and encompassing the true essence of RL for control.

Earlies attempt for a *model-free* ACD is by using the AD-HDP. In this structure, model network is completely omitted as action NN can be updated by backpropogating through critic NN as shown in Fig.2. Also the requirement for model NN to train critic NN was averted by simply waiting for the next term [6] [3] but a performance degradation was observed by using this method. It is also important to note that same can

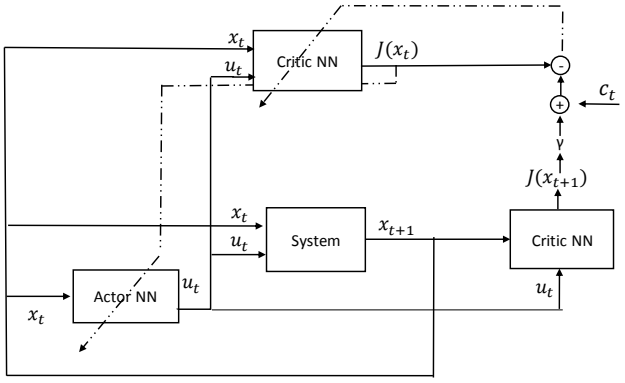

Fig. 2: Model-free ADHDP

not be done to obtain *model-free* ADDHP and ADGDHP as they don't just require one-step prediction from model NN but they also need model NN to maintain backpropogation paths. Si & Wang made very remarkable contribution in online learning algorithm by proposing a *model-free* ACD closely resembles the ADHDP structure with main distinction being completely *model-free* without compromising on accuracy by storing the previous cost-to-go value [8].This method has been used successfully for many applications and will be referred as modified ADHDP method [11] [12] [13].In flight control, ADHDP has been applied to F-16 model and comparison was done with HDP. Results showed that HDP with approximated model dynamics outperforms ADHDP under changing dynamics but ADHDP perform better with noise [14]. Russel & Si also applied the *model-free* modified ADHDP to control Apache helicopter for different realistic maneuvers and found the controller performance to be satisfactory [15]. But as the author pointed out, this study was done offline where a system learns by trial and can afford failures. Liu et al. furthered this modified ADHDP by introducing two methods to train the critic NN namely*backward in time* in which critic is trained with Q_t in Eq.**??** as critic's output by using $1/\gamma[Q_{t-1} - c_t]$ as target for training. whereas in *forward in time* approach, critic network outputs $Q(t-1)$ and is trained with $\gamma Q_t + c_t$. It is author's opinion that the *backward in time* training can lead to instabilities due to $1/\gamma$ term but it is worth noticing that the MATLAB trainlm function was used to train the critic network.

In this paper, to further improve the performance of *online model-free methods*, ADDHP is investigated. As mentioned

before, unlike ADHDP, ADDHP requires model information to maintain the backpropogations paths for critic updates. Proposed approach in this paper is to use increment based model for nonlinear system. According to this, a nonlinear model in the form of Eq.5 can be written in discrete incremental form as Eq.6, given very high sampling frequency and slow varying dynamics.

$$\dot{x}(t) = f(x(t), u(t)) \tag{5}$$

$$\Delta x_{t+1} = F_{t-1}\Delta x_t + G_{t-1}\Delta u_t \tag{6}$$

Therefore, instead of approximating the global model, local model is approximated to avoid dealing with large set of training data. This incremental model approach for control has been successfully applied to control nonlinear system.The incremental model approach for control which has been successfully applied to control nonlinear system. It's applications include Incremental Dynamic Inversion (INDI) [16] [17], and Incremental Backstepping (IBS) [18]. In ADP, an Incremental Approximate Dynamic Programming (iADP) was developed by Zhou et al. to find the near optimal control of nonlinear system [19]. In ACD, incremental HDP (IHDP) used incremental model to find the local system dynamics, instead of NN for global dynamic model, to get the next state and next state for training actor and critic. Online local system identifcation can easily be performed using Ordinary Least Square (OLS) methods or Recursive Least Square (RLS). Studies performed showed that IHDP outperforms the traditional HDP and speeds up the process of online learning for missile control [20].

In the literature, only other author who has implemented *online model-free* ADDHP is Zhen et al by making use of Finite Difference Method (FDM) [21] to find system dynamics derivatives and applied it to a ball and beam problem. This paper focuses on the development of *model-free IADDHP* and compares its results with ADDHP using FDM.

II. Online Model-Free ADDHP Structures

Model-free ADDHP designs consist of three parts: Critic for policy evaluation, Actor for policy improvement and System Dynamics Matrix as opposed to Model NN in classical model based ACD approaches as shown in Fig.3. DHP approximates the derivative of cost-to-go as critic's output which results in higher success rate and accuracy as compared to HDP. In ADDHP, there is a direct connection between actor and critic network which makes training for the actor network easier and it also improves the accuracy and convergence of the controller as the derivative of J_t with u_t is direct output of the critic. Components of *model-free* ADDHP are described as follow.

A. Critic Network

Critic in ADDHP takes $y_t = \begin{bmatrix} x_t \\ u_t \end{bmatrix}$ as its inputs and approximates the derivative of J_t with input y_t as its output, which is given by Eq.7.

$$\lambda_t = \frac{\partial J_t}{\partial \hat{y}_t} \tag{7}$$

where $x_t = \begin{bmatrix} x_1 \cdots x_n \end{bmatrix}^\intercal \in \mathbb{R}^{n\times 1}$ is system's states, $\begin{bmatrix} u_1 \cdots u_m \end{bmatrix}^\intercal \in \mathbb{R}^{m\times 1}$ is the control inputs, and $\lambda_t = \begin{bmatrix} \lambda_{t_{x_1}} \cdots \lambda_{t_{x_n}}, \lambda_{t_{u_1}} \cdots \lambda_{t_{u_m}} \end{bmatrix}^\intercal \in \mathbb{R}^{n+m\times 1}$ is critic's output.

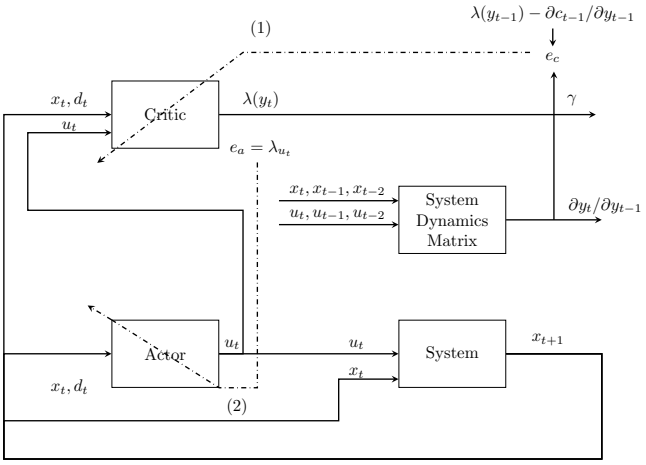

Fig. 3: Structure of Model-free Action Dependent Dual Heuristic Programming

Critic is trained by minimizing the critic error defined in Eq. 8

$$\|E_{c_t}\| = \frac{1}{2} e_{c_t}^\intercal e_{c_t} \tag{8}$$

Prediction error for training backward in time for ADDHP is stated in Eq.9.

$$e_{c_t} = \frac{\partial [J_{t-1} - \{c_{t-1} + \gamma J_t\}]}{\partial y_{t-1}} \tag{9}$$

$$e_{c_t} = \lambda_{t-1} - \{\frac{\partial c_{t-1}}{\partial y_{t-1}} + \gamma \frac{\partial y_t}{\partial y_{t-1}} \lambda_t\} \tag{10}$$

where $e_{c_t} \in \mathbb{R}^{(n+m)\times 1}$ and $e_{c_{t_k}}$, k^{th} element of e_{c_t}, corresponds to the critic error for k^{th} element of critic's input. The term $\frac{\partial y_t}{\partial y_{t-1}} \in \mathbb{R}^{(n+m)\times(n+m)}$ contains the system's dynamics information and is obtained from System Dynamic Matrix part of the *model-free* ADDHP structure. whereas, $\frac{\partial c_{t-1}}{\partial y_{t-1}}$ can be written as Eq.11

$$\frac{\partial c_{t-1}}{\partial y_{t-1}} = \begin{bmatrix} \frac{\partial c_{t-1}}{\partial x_{t-1}} \\ \frac{\partial c_{t-1}}{\partial u_{t-1}} \end{bmatrix} \tag{11}$$

c_t is the local cost function and it's definition depends upon the mission at hand. In this paper, quadratic cost function is used which is defined as Eq.12 to describe reference tracking with minimum control effort.

$$c_t(x_t, u_t) = (x_t - d_t)^\intercal Q (x_t - d_t) + u_t^\intercal R u_t \tag{12}$$

where $d_t \in \mathbb{R}^{n\times 1}$ is the desired states, $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are positive definitive matrices chosen to describe the relative importance of states and inputs in cost function. $\frac{\partial c_{t-1}}{\partial x_{t-1}}$ in Eq.11 for c_t which depends on u_t can be written as Eq.13 for time *t-1* using the chain rule.

$$\frac{\partial c_{t-1}}{\partial x_{t-1}} = \frac{\partial c_{t-1}}{\partial x_{t-1}} + \sum_{k=1}^{n} \Big[\frac{\partial c_{t-1}}{\partial u_{k_{t-1}}} \frac{\partial u_{k_{t-1}}}{\partial x_{t-1}} \Big] \tag{13}$$

For c_t described in Eq.12, $\frac{\partial c_{t-1}}{\partial x_{t-1}}$ becomes;

$$\frac{\partial c_{t-1}}{\partial x_{t-1}} = diag(Q) \circ (x_{t-1} - d_{t-1}) + diag(R) \circ (u_{t-1}^{\intercal} \frac{\partial u_{t-1}}{\partial x_{t-1}})^{\intercal} \tag{14}$$

where $\frac{\partial u_{t-1}}{\partial x_{t-1}}$ is obtained by backpropogating through actor NN structure. Similarly, $\frac{\partial c_{t-1}}{\partial u_{t-1}}$ for Eq.12 is written as;

$$\frac{\partial c_{t-1}}{\partial u_{t-1}} = diag(R) \circ u_{t-1} \tag{15}$$

where $\circ$ represents element-wise multiplication of a vector with a matrix.

1) Critic NN Structure: To approximate λ_t, one hidden layer multi-perceptron NN is used. This is show in Fig. 4 for MIMO controller. Based on the structure of critic NN, its output can be written as;

$$\lambda_{t_k} = purelin(v_{c_{t_k}}) \tag{16}$$

$$v_{c_{t_k}} = \sum_{j=1}^{N_{h_c}} w_{c_{t_{kj}}} p_{c_{t_j}} \tag{17}$$

$$p_{c_{t_j}} = tansig(q_{c_{t_j}}) \tag{18}$$

$$q_{c_{t_j}} = \sum_{i=1}^{n} w_{c_{t_{ji}}} y_{t_i} \tag{19}$$

- λ_{t_k} is the k^{th} output of neural network
- v_{t_k} is the input to k^{th} neuron.
- $w_{c_{t_{kj}}}$ is weight from j^{th} hidden neuron to k^{th} output
- p_{t_j} is the output of j^{th} hidden neuron
- q_{t_j} is the input of j^{th} hidden neuron
- $w_{c_{t_{ji}}}$ is the weight from i^{th} input to j^{th} hidden neuron
- y_i is the i^{th} input to neural network
- n is the number of states
- N_h is the number of hidden neurons

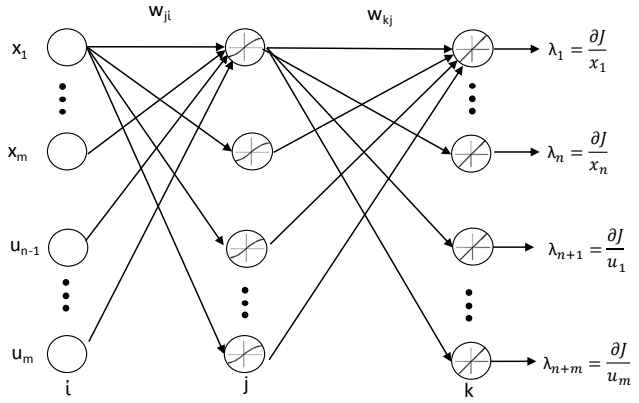

Fig. 4: Critic Neural Network for DHP Structures

where *purelin* and *tansig* transfer functions can be changed for the output neurons depending on the complexity of approximation. *Tansig* is equivalent to *tanh* mathematically but it runs faster on MATLAB with slight numerical difference [22]. These transfer functions along with their derivative function are calculated as;

$$a = purelin(b) = b \tag{20}$$

$$\frac{\partial a}{\partial b} = dpurelin(a) = 1 \tag{21}$$

$$a = tansig(b) = \frac{2}{1 + e^{(-2b)}} - 1 \tag{22}$$

$$\frac{\delta a}{\partial b} = dtansig(a) = 1 - a^2 \tag{23}$$

2) Critic NN Weight Update: In this paper, a recursive Stochastic Gradient Descent (SGD) method also know as Robbins-Monro algorithm is used to update NN weights for actor and critic due to its ability to converge to an optimal or near-optimal property [8] [4].Using Stochastic Gradient Descent method, Weight update equation can be written as

$$w_{c_{t+1}} = w_{c_t} + \Delta w_{c_t} \tag{24}$$

$$\Delta w_{c_t} = l_{c_t} \Big[- \frac{\partial E_{c_t}(x_t, u_t)}{\partial w_{c_t}} \Big] \tag{25}$$

where l_t is the learning rate. According to [23], if the learning rate satisfies the conditions in Eq.26 then for quadratic E_t, w_t will converge to global optima w^* otherwise it will converge to a local optima with time. The convergence proof of this is given in [8].

$$\lim_{t \to \infty} l_t = 0, \quad \sum_{t=0}^{\infty} l_t = \infty, \quad \sum_{t=0}^{\infty} l_t^2 < \infty \tag{26}$$

Critic NN weight update equations can be written as;

1) $\Delta w_c^{(2)}$ (Hidden to Output Layer Weights)
Using Eq.25 and structure of the critic network given by Eq.16 to Eq.19 , weight update equation from critic's hidden layer to output layer can be written as a matrix $\Delta w_c^{(2)} = [\Delta w_{c_{kj}}^{(2)}] \in \mathbb{R}^{n+m \times N_{h_c}}$

$$\Delta w_{c_{kj}}^{(2)} = -l_{c_t} \Big[\frac{\partial E_{c_t}}{\partial e_{c_{t_k}}} \Big] \Big[\frac{\partial e_{c_{t_k}}}{\partial \lambda_{t-1_k}} \Big] \Big[\frac{\partial \lambda_{t-1_k}}{\partial v_{c_{t-1_k}}} \Big] \Big[\frac{\partial v_{c_{t-1_k}}}{\partial w_{c_{t_{kj}}}} \Big] \tag{27}$$

$$\Delta w_{c_{kj}}^{(2)} = -l_{c_t} . e_{c_{t_k}} . p_{c_{t-1_j}} \tag{28}$$

Using column vectors for e_{c_t} and $p_{c_{t-1_j}}$, Eq.28 can be rewritten as Eq.29.

$$\Delta w_c^{(2)} = -l_{c_t} . e_{c_t} . p_{c_{t-1}}^{\intercal} \tag{29}$$

2) $\Delta w_c^{(1)}$ (Input to Hidden Layer Weights) Critic neural network's weight update from input layer to hidden layer

neuron can be written in a matrix $\Delta w_c^{(1)} = [\Delta w_{cji}^{(1)}] \in \mathbb{R}^{N_{h_c} \times n+m}$

$$\Delta w_{cji}^{(1)} = -l_{c_t} \sum_{k=1}^{n} \Big[\frac{\partial E_{c_t}}{\partial e_{c_{t_k}}}\Big]\Big[\frac{\partial e_{c_{t_k}}}{\partial \lambda_{t-1_k}}\Big]\Big[\frac{\partial \lambda_{t-1_k}}{\partial v_{c_{t-1_k}}}\Big] \cdot\Big[\frac{\partial v_{c_{t-1_k}}}{\partial p_{c_{t-1_j}}}\Big]\Big[\frac{\partial p_{c_{t-1_j}}}{\partial q_{c_{t-1_j}}}\Big]\Big[\frac{\partial q_{c_{t-1_j}}}{\partial w_{c_{ji}}}\Big] \tag{30}$$

$$\Delta w_{cji}^{(1)} = -l_{c_t} \sum_{k=1}^{n} e_{c_{t_k}}.w_{ckj}.dtansig(q_{c_{t-1_j}}).y_{t-1_i} \tag{31}$$

Using column vectors, entire matrix update $\Delta w_c^{(1)}$ can be written as;

$$\Delta w_c^{(1)} = -l_{c_t}\big[e_{c_t}^{\intercal}.w_c\big]^{\intercal} \circ \big[dtansig(q_{c_{t-1}}).y_{t-1}^{\intercal}\big] \tag{32}$$

B. Actor

The objective of actor is to produce the control policy u^* which can minimize (*maximize*) error between J_t and desired ultimate objective U_t.In this case, as the c_t is defined in terms of tracking error and control effort required, ideally we would like $J_t = 0$ therefore U_t is set to 0. This control policy can be written as Eq.33.

$$u^* = \underset{u_t}{\operatorname{argmin}}(J_t - U_t) = \underset{u_t}{\operatorname{argmin}}(J_t) \tag{33}$$

Actor NN weights are updated to minimize following error.

$$\|E_{a_t}\| = \frac{1}{2} e_{a_t}^{\intercal} e_{a_t} \tag{34}$$

Using Eq.35 as target for actor weight update, we can write prediction error as Eq:36.

$$\frac{\partial J_t}{\partial u_t} = 0 \tag{35}$$

$$e_{a_t} = \lambda_{u_t} \tag{36}$$

where $\lambda_{u_t} \in \mathbb{R}^{m \times 1}$ is obtained from critic i.e. $\lambda_t = \begin{bmatrix} \lambda_{x_t} \\ \lambda_{u_t} \end{bmatrix}$

1) Actor NN Structure: Actor NN approximates the optimal control policy u_t^*. For this, single hidden layer multi-perceptron NN is used similar to Fig. 4. For MIMO ADDHP controller, input to actor NN is the state vector x_t whereas its outputs is a control action vector u_t. This output is then given as input ti critic network and to simulate the real system. Equation 37 to 40 make up the actor NN.

$$u_{t_k} = tansig(v_{a_{t_k}}) \tag{37}$$

$$v_{a_{t_k}} = \sum_{j=1}^{N_{h_a}} w_{a_{t_{kj}}} p_{a_{t_j}} \tag{38}$$

$$p_{a_{t_j}} = tansig(q_{a_{t_j}}) \tag{39}$$

$$q_{a_{t_j}} = \sum_{i=1}^{m} w_{a_{t_{ji}}} x_{t_i} \tag{40}$$

2) Actor NN weight Update: Using stochastic gradient, weight update for actor can be written as Eq.41

$$\Delta w_{a_t} = l_{a_t}\big[\lambda_{u_t}\big]\frac{\partial u_t}{\partial w_{a_t}} \tag{41}$$

where $\frac{\partial u_t}{\partial w_{a_t}}$ is calculated using backpropogation through actor NN as derived below:

1) $\Delta w_a^{(2)}$ (Hidden to Output Layer Weights)
Using Eq.41, weights update equations from hidden layer to output layer for actor can be written as a matrix $\Delta w_a^{(2)} = [\Delta w_{a_{kj}}^{(2)}] \in \mathbb{R}^{m \times N_{h_c}}$.

$$\Delta w_{a_{kj}}^{(2)} = -l_{a_t}\big[\lambda_{u_{t_k}}\big].\frac{\partial u_{t_k}}{\partial v_{a_{t_k}}}\frac{\partial v_{a_{t_k}}}{\partial w_{a_{t_{k^j}}}} \tag{42}$$

$$\Delta w_{a_{kj}}^{(2)} = -l_{a_t}.\big[\lambda_{u_{t_k}}\big].dtansig(v_{a_{t_k}}).p_{a_{t_j}} \tag{43}$$

Eq.43 can be rewritten as Eq.44 using column vectors for $\frac{\partial c_t}{\partial u_t}, v_{a_t}$, and p_{a_t}).

$$\Delta w_a^{(2)} - l_{a_t}.\big[\lambda_{u_t}\big] \circ dtansig(v_{a_t})^{\intercal}.p_{a_t} \tag{44}$$

2) $\Delta w_a^{(1)}$ (Input to Hidden Layer Weights) Actor neural network weights update from input layer to hidden layer can be written in a matrix $\Delta w_a^{(1)} = [\Delta w_{a_{ji}}^{(2)}] \in \mathbb{R}^{N_{h_a} \times n}$.

$$\Delta w_{a_{ji}}^{(1)} = -l_{a_t} \sum_{k=1}^{m} \Big[\{\lambda_{u_{t_k}}\}.\frac{\partial u_{t_k}}{\partial v_{a_{t_k}}}\frac{\partial v_{a_{t_k}}}{\partial p_{a_{t_j}}}\Big].\frac{\partial p_{a_{t_j}}}{\partial q_{a_{t_j}}}. \cdot\frac{\partial q_{a_{t_j}}}{\partial w_{a_{t_{j^i}}}} \tag{45}$$

$$\Delta w_{a_{ji}}^{(1)} = -l_{a_t} \sum_{k=1}^{m} \big[\{\lambda_{u_{t_k}}\}.dtansig(v_{a_{t_k}}).w_{a_{t_{kj}}}\big]. .dtansig(q_{a_{t_j}}).x_{t_i} \tag{46}$$

Using column vectors, entire matrix update $\Delta w_a^{(1)}$ can be written as;

$$\Delta w_a^{(1)} = -l_{a_t}\big[\big[\{\lambda_{u_t}\} \circ dtansig(v_{a_{t_k}})\big]^{\intercal}.w_{a_{t_{kj}}}\big]^{\intercal} \circ dtansig(q_{a_t})^{\intercal}.x_t \tag{47}$$

C. System Dynamics

As mentioned before, the term $\frac{\partial y_t}{\partial y_{t-1}}$ in Eq.10 captures the system dynamics of the system. This can be written as;

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} \frac{\partial x_t}{\partial x_{t-1}} & \frac{\partial u_t}{\partial x_{t-1}} \\ \frac{\partial x_t}{\partial u_{t-1}} & \frac{\partial u_t}{\partial u_{t-1}} \end{bmatrix} \tag{48}$$

In classical methods, this information is obtained through the Model NN. In the *Model-free* approaches, some alternate way is required to obtain this information. In this report, two methods are investigated to obtain $\frac{\partial y_t}{\partial y_{t-1}}$. These methods are *'Finite Difference Method (FDM)'* and *'Incremental Model'*.

1) Finite Difference Method: This method is proposed by Zhen et al. in [21]. Instead of using Model NN, they suggested using the Finite Difference Method (FDM) to find the derivatives required in Eq.48. Using Taylor Series expansion, *'first-order backward difference'* derivative can be written as

$$\frac{\partial q_t}{\partial t} \simeq \frac{q(t) - q(t-\Delta t)}{\Delta t} + \mathcal{O}(\Delta t) \tag{49}$$

where $\mathcal{O}(\Delta t)$ is the truncation error associated with using *first-order* approximation. This implies that the accuracy of the approximation is proportional to the discretization step. Therefore, data should be taken at high sampling rate for better quality of approximation. Using chain rule and FDM, derivatives in Eq.48 can be written as,

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} \frac{x_t - x_{t-1}}{x_{t-1} - x_{t-2}} & \frac{u_t - u_{t-1}}{x_{t-1} - x_{t-2}} \\ \frac{x_t - x_{t-1}}{u_{t-1} - u_{t-2}} & \frac{u_t - u_{t-1}}{u_{t-1} - u_{t-2}} \end{bmatrix} \tag{50}$$

where $x_t, x_{t-1} \in \mathbb{R}^{n\times 1}$ and $u_t, u_{t-1} \in \mathbb{R}^{m\times 1}$ are column vectors. FDM method described above for online system identification is suitable for small noise-free systems with rapidly changing dynamics [24]. Therefore, proper excitation of the system is of paramount importance to avoid divergence.

2) Incremental Model: Another *model-free* approach to get $\frac{\partial y_t}{\partial y_{t-1}}$ is by using incremental model instead of Model NN. Non-linear continuous system can be defined by Eq.51 where $\dot{x}(t)$ represents the change in system dynamics and $y(t)$ is the output of system.

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \\ y(t) &= h(x(t), u(t)) \end{aligned} \tag{51}$$

Equation 51 can be linearized around a time t_o using Taylor series expansion as Eq.52 which can be written as continuous time-variant state space of incremental form as Eq.53

$$\begin{aligned} \dot{x}(t) \simeq \dot{x}(t_o) &+ \frac{\partial f(x(t), u(t))}{\partial x(t)}|_{x(t_o), u(t_o)}(x(t) - x(t_o)) \\ &+ \frac{\partial f(x(t), u(t))}{\partial u(t)}|_{x(t_o), u(t_o)}(u(t) - u(t_o)) \end{aligned} \tag{52}$$

$$\Delta\dot{x}(t) \simeq F(x(t_o), u(t_o))\Delta x(t) + G(x(t_o), u(t_o)\Delta u(t) \tag{53}$$

where $F(x(t_o), u(t_o)) = \frac{\partial f(x(t), u(t))}{\partial x(t)}|_{x(t_o), u(t_o)} \in \mathbb{R}^{n\times n}$ is the state matrix and $G(x(t_o), u(t_o)) = \frac{\partial f(x(t), u(t))}{\partial u(t)}|_{x(t_o), u(t_o)} \in \mathbb{R}^{n\times m}$ is the input matrix.

Although a system can be continuous but the data obtained from sensors is discrete. Therefore, Eq. 51 can be written in discrete form for a system with very high sampling frequency and full state observations as Eq.54.

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) \\ y_t &= h(x_t, u_t) = x_t \end{aligned} \tag{54}$$

Linearizing this equation around previous time step $t-1$ will give us Eq.55.

$$\Delta x_{t+1} \simeq F_{t-1}\Delta x_t + G_{t-1}\Delta u_t \tag{55}$$

where $\Delta x_t = x_t - x_{t-1}$, $\Delta u_t = u_t - u_{t-1}$ and $F_{t-1} \simeq \frac{\partial x_{t+1}}{\partial x_t}|_{model} \in \mathbb{R}^{n\times n}$ and $G_{t-1} \simeq \frac{\partial x_{t+1}}{\partial u_t} \in \mathbb{R}^{n\times m}$ are system matrix and input matrix at $t-1$ respectively for discrete system.As we are assuming the slowly time varying system with high sampling frequency, these matrices representing the linear system can be easily identified online by linear regression methods by taking some previous measurements into account. After analyzing Sliding Window-Ordinary Least Square($SW-OLS$) and Recursive Least Square (RLS) for system identification, RLS is selected to be used in this paper for its better performance.

Recursive Least Square

For the online identification of linear model, RLS is a very powerful method as it recursively update the estimate as new measurements become available and doesn't require saving all previous measurement. It also has better convergence than other least square methods but it is computationally more intensive [25].

For our linearized model,Eq.55 can be rewritten as Eq.56 which can be solved by RLS summarized in Eq.59 to 62 by taking $\hat{\Theta}(t) = \begin{bmatrix} F_{t-1}^\intercal \\ G_{t-1}^\intercal \end{bmatrix}$ as the parameter to be estimated and $\psi(t) = \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}$ as the regression matrix which contains information about previous measurements [26].

$$\Delta x_{t+1} = \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}^\intercal \begin{bmatrix} F_{t-1}^\intercal \\ G_{t-1}^\intercal \end{bmatrix} \tag{56}$$

$$\Delta\hat{x}(t) = \psi(t)^\intercal \hat{\Theta}(t-1) \tag{57}$$

$$e(t) = \Delta x(t) - \Delta\hat{x}(t) \tag{58}$$

$$\hat{\Theta}(t) = \hat{\Theta}(t-1) + K(t)(e(t)) \tag{59}$$

$$K(t) = Q(t)\psi(t) \tag{60}$$

$$Q(t) = \frac{P(t-1)}{\Lambda + \psi(t)^\intercal P(t-1)\psi(t)} \tag{61}$$

$$P(t) = \frac{1}{\Lambda}\big[P(t-1) - \frac{P(t-1)\psi(t)\psi(t)^\intercal P(t-1)}{\Lambda + \psi(t)P(t-1)\psi(t)^\intercal}\big] \tag{62}$$

Λ is the forgetting factor which ensures that the weight of previous values is discounted exponentially for older measurements when $\Lambda < 1$. For systems with high varying dynamics, Λ should be small but this filters small noise. For higher values of Λ(around 0.998), larger noise can be allowed but tracking the changes in parameters is slower. $P(t)$ is the covariance matrix that should be initialized as $P(0) = \kappa I$ where κ should be very large for convergence [27]. $\theta(t)$ is initialized with $F_0 = I$ and $G_0 = O$.

Backpropogation using RLS

Incremental model can be used to maintain the backpropogation paths i.e. to find the term $\frac{\partial y_t}{\partial y_{t-1}}$ in Eq.10. Using chain rule and actor NN, component of matrix can be written as;

$$\frac{\partial x_t}{\partial x_{t-1}} = \frac{\partial x_t}{\partial x_{t-1}}|_{model} + \frac{\partial x_t}{\partial u_{t-1}}\frac{\partial u_{t-1}}{\partial x_{t-1}}|_{model+actor} \tag{63}$$

$$\frac{\partial u_t}{\partial x_{t-1}} = \frac{\partial u_t}{\partial x_t}\frac{\partial x_t}{\partial x_{t-1}}|_{actor+model} \tag{64}$$

$$\frac{\partial u_t}{\partial u_{t-1}} = \frac{\partial u_t}{\partial x_t}\frac{\partial x_t}{\partial u_{t-1}}|_{actor+model} \tag{65}$$

Substituting $F_{t-1} \simeq \frac{\partial x_{t+1}}{\partial x_t}|_{model}$, and $G_{t-1} \simeq \frac{\partial x_{t+1}}{\partial u_t}|_{model}$ in above equations, $\frac{\partial y_t}{\partial y_{t-1}}$ becomes ;

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} F_{t-2} + G_{t-2}\frac{\partial u_{t-1}}{\partial x_{t-1}}^\intercal \frac{\partial u_t}{\partial x_t}F_{t-2}^\intercal \\ G_{t-2}^\intercal \quad \frac{\partial u_t}{\partial x_t}G_{t-2}^\intercal \end{bmatrix} \tag{66}$$

III. Algorithm for ADDHP

Pseudo-code for ADDHP with FDM is given in Algorithm 1. Similar to this, Algorithm 2 gives an overview of Incremental model based ADDHP i.e. IADDHP.

Algorithm 1 *Model-free* ADDHP-FDM

Initialization of variables x_0, γ, $w_c^{(1)}$,$w_c^{(2)}$, $w_a^{(1)}$,$w_a^{(2)}$, Q, and R
for $i = 1 \rightarrow N$ **do** ▷ Total simulation time=$N \times dt$
  if $i = 1$ **then**
    for $k = 1 \rightarrow 3$ **do**
      $x_{k+1} \leftarrow$ Excite the real system with u_k
    end for
    $\frac{\partial y_t}{\partial y_{t-1}}|_0 \leftarrow$ Calculate the initial system dynamics matrix using FDM
  end if
  u_t, $u_{t-1} \leftarrow$ Forward calculate actor (x_t, w_{a_t}), (x_{t-1}, w_{a_t})
  $\lambda_t, \lambda_{t-1} \leftarrow$ Forward calculate critic $(y_t$=$\begin{bmatrix} x_t \\ u_t \end{bmatrix}$,$w_{a_t})$, (y_{t-1}, w_{a_t})
  $w_{c_{t+1}} \leftarrow$ Update critic NN weights$(y_{t-1}$,λ_t, λ_{t-1}, $\frac{\partial c_{t-1}}{y_{t-1}}$, $\frac{\partial y_t}{\partial y_{t-1}})$
  $\lambda(x_t) \leftarrow$ Forward calculate critic$(\hat{y}_t, w_{c_{t+1}})$
  $w_{a_{t+1}} \leftarrow$ Update actor NN weights(x_t, λ_{u_t})
  $u_t \leftarrow$ Forward calculate actor$(x_t, w_{a_{t+1}})$
  $x_{t+1} \leftarrow$ Simulate real system $(x_t, u_t + u_{PE})$
    ▷ Persistent Excitation(PE) for exploration
  $l_t \leftarrow$ Update learning rate for actor and critic
  $\frac{\partial y_t}{\partial y_{t-1}} \leftarrow$ Update system dynamics matrix for new measurements using FDM
end for

IV. Simulation for Flight Control

In this section, numerical simulation setup for the application of aforementioned ADDHP algorithms for missile reference tracking control is discussed. Different experiments are designed to identify and investigate various parameters to find the optimal controller setting and the controllers are compared based on their performance.

Algorithm 2 *Model-free* IADDHP

Initialization of variables like $x_0, \gamma, \Lambda, P_0, \Theta_0$,, $w_c^{(1)}$,$w_c^{(2)}$, $w_a^{(1)}$,$w_a^{(2)}$, Q, and R
for $i = 1 \rightarrow N$ **do** ▷ Total simulation time=$N \times dt$
  if $i = 1$ **then** ▷ To calculate F_o, G_o
    for $k = 1 \rightarrow 2$ **do**
      $x_{k+1} \leftarrow$ Excite the real system with u_k
    end for
    $\psi(t)|_0 \leftarrow$ Calculate initial regression matrix
  end if
  $\hat{x}_i \leftarrow$ Forward calculate model estimate $(x_i, \psi(t), \Theta(t-1))$ ▷ Equation 57
  $K(t), Q(t) \leftarrow$ Calculate matrices ▷ Equation 60 and 61
  $\Theta(t) \leftarrow$ Update parameter estimate matrix $K(t), e_t, \Theta(t-1))$ ▷ Equation 59
  $P(t) \leftarrow$ Update co-variance matrix $(P(t-1), \psi(t))$ ▷ Equation 62
  u_t, $u_{t-1} \leftarrow$ Forward calculate actor (x_t, w_{a_t}), (x_{t-1}, w_{a_t})
  $\lambda_t, \lambda_{t-1} \leftarrow$ Forward calculate critic $(y_t$=$\begin{bmatrix} x_t \\ u_t \end{bmatrix}$,$w_{a_t})$, (y_{t-1}, w_{a_t})
  $w_{c_{t+1}} \leftarrow$ Update critic NN weights$(y_{t-1}$,λ_t, λ_{t-1}, $\frac{\partial c_{t-1}}{y_{t-1}}$, $\frac{\partial y_t}{\partial y_{t-1}})$
  $\lambda(x_t) \leftarrow$ Forward calculate critic$(\hat{y}_t, w_{c_{t+1}})$
  $w_{a_{t+1}} \leftarrow$ Update actor NN weights(x_t, λ_{u_t})
  $u_t \leftarrow$ Forward calculate actor$(x_t, w_{a_{t+1}})$
  $x_{t+1} \leftarrow$ Simulate real system $(x_t, u_t + u_{PE})$
    ▷ Persistent Excitation(PE) for exploration
  $l_t \leftarrow$ Update learning rate for actor and critic
  $\psi(t) \leftarrow$ Update regression matrix based on new measurements
  $\frac{\partial y_t}{\partial y_{t-1}} \leftarrow$ Update system dynamics matrix using RLS for backpropogation ▷ Equation 66
end for

A. Missile Model

To test the applicability of the *online ACD controllers* for flight control, a simple missile model is used to simulate short period dynamics of the air vehicle. This nonlinear model for pitch plane can be written as:

$$\dot{\alpha} = q + \frac{\bar{q}Sg}{m_a V_T} C_z(\alpha, q, M_a, \delta_e) \tag{67}$$

$$\dot{q} = \frac{\bar{q}Sd_l}{I_{yy}} C_m(\alpha, q, M_a, \delta_e) \tag{68}$$

where angle of attack (α) and pitch rate (q) are the states to be controlled by changing control fin deflection angle (δ_e). System is initialized with $\alpha_o = 5.7\,\text{deg}$ and $q = 0\ \frac{rad}{sec}$.

$$C_z(\alpha, q, M_a, \delta_e) = C_{z_1}(\alpha, M_a) + B_z\delta_e \tag{69}$$

$$C_m(\alpha, q, M_a, \delta_e) = C_{m_1}(\alpha, M_a) + B_m\delta_e, \tag{70}$$

$$B_z = b_1 M_a + b_2, \tag{71}$$

$$B_m = b_3 M_a + b_4, \tag{72}$$

$$C_{z_1}(\alpha, M_a) = \phi_{z_1}(\alpha) + \phi_{z_2} M_a, \tag{73}$$

$$C_{z_2}(\alpha, M_a) = \phi_{m_1}(\alpha) + \phi_{m_2} M_a, \tag{74}$$

$$\phi_{z_1}(\alpha) = h_1\alpha^3 + h_2\alpha|\alpha| + h_3\alpha, \tag{75}$$

$$\phi_{m_1}(\alpha) = h_4\alpha^3 + h_5\alpha|\alpha| + h_6\alpha, \tag{76}$$

$$\phi_{z_2} = h_7\alpha|\alpha| + h_8\alpha, \tag{77}$$

$$\phi_{m_2} = h_9\alpha|\alpha| + h_{10}\alpha, \tag{78}$$

Aerodynamic coefficient and other parameters used for simulation of missile model are given in Table I and are valid for $-10\,\text{deg} < \alpha < 10\,\text{deg}$ [28].

TABLE I: Aerodynamic parameters for Missile Model

Parameter	Value	Parameter	Value	Parameter	Value
$\bar{q}$	6132.8 $\frac{lb}{ft^2}$	S	0.44ft^2	m_a	450 lb
V_T	3109.3 $\frac{ft}{sec}$	d_l	0.75 ft	I_{yy}	182.5 $\frac{slug}{ft^2}$
M_a	2	b_1	1.6238	b_2	-6.7240
b_3	12.0393	b_4	-48.2246	h_1	-288.7
h_2	50.32	h_3	-23.89	h_4	303.1
h_5	-246.3	h_6	-37.56	h_7	-13.53
h_8	4.185	h_9	71.51	h_{10}	10.01

The purpose of this model is to simulate the real system with the sampling frequency of 100Hz. As the controllers discussed in this project are *model-free*, therefore no direct information from this model is used in the controllers.

B. Experiments and Results

Reference tracking control problem is used to analyze the performance of controllers. Sinusoidal wave of 10 deg amplitude is used as reference signal for α with 0.05 rad/sec angular frequency.

For actor and critic NN, $2 \times 3 \times 1$ and $3 \times 3 \times 3$ structure is used respectively with all the initial NN weights randomly initialized in the range of (-0.5,0.5). l_r is *annealed* slowly with time to avoid instability. This is, after 50 time steps, l_r is given as;

$$l_{a_t} = l_{a_t} = bl_o \geq 0.05 \tag{79}$$

where $b = 0.95$ and l_o represents the initial learning rate which will be varied. Different scenarios are investigated by 100 independent runs with random initial weights. If after 20 sec of 600 sec simulation, the absolute error between required and desired state is greater than 1.5 deg then the trial is considered unsuccessful. In these simulations, no internal weight update cycles are used neither does a controller learns trials. Also, for ADDHP-FDM, a limit is set on $\frac{\partial y_t}{\partial y_{t-1}}$ matrix as it tends to diverge very easily.

1) Without Noise or Failure: IADDHP and ADDHP-FDM are simulated for 100 independent runs for various l_r to identify the optimal l_r for each method as to compare these methods fairly. It can be seen in Fig.5 that ADDHP-FDM works better when the l_r is low. Whereas, IADDHP performs better at higher l_r. This can be explained by observing how the weights are being updated. As in FDM, small denominator terms can lead to a very big $\frac{\partial y_t}{\partial y_{t-1}}$ matrix which, coupled with higher l_r, results in higher values of weights and can

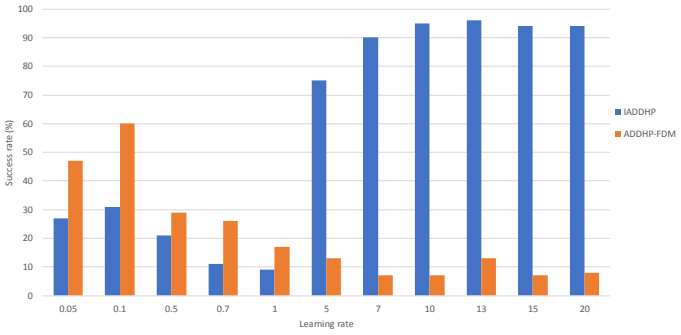

Fig. 5: Success rate of *model-free* controllers with various learning rates

cause a run to diverge very fast. Therefore small l_r are a safer option with highest success rates. Whereas in RLS for system identification, previous values are averaged which keeps $\frac{\partial y_t}{\partial y_{t-1}}$ from getting excessively big. A higher learning rate is required for faster convergence to optimal policy. For $l_r = 0.1$, ADDHP-FDM has maximum success rate of 60% whereas for IADDHP it is 93% at $l_r = 13$. It can also be seen that after $l_r = 10$, there is little variation in success rate of IADDHP.

To understand the reason behind the divergence of FDM methods, temporal progression of $\frac{\partial y_t}{\partial y_{t-1}}$ was observed for various randomly initialized unsuccessful runs. It was observed that the u_t for all these runs become saturated at some point in time which leads to ill-condition $\frac{\partial y_t}{\partial y_{t-1}}$ matrix and eventually the divergence of the system output. This is shown in Fig.6 for four randomly initialized unsuccessful runs for initial iterations that lead to unbounded output.

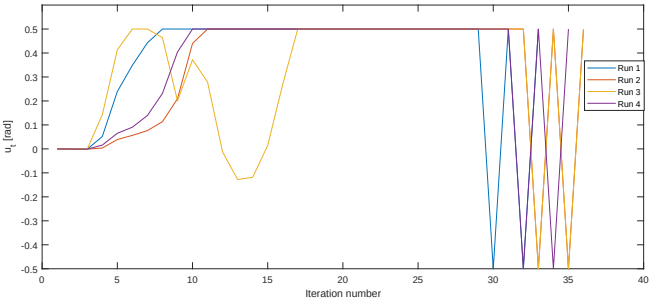

Fig. 6: Control input for unsuccessful runs

The saturation occurs because we have imposed a limit on the control action of 0.5 [rad] to avoid large control fin deflections. Because of this saturation, $\frac{\partial x_t}{\partial u_{t-1}}$ and $\frac{\partial u_t}{\partial u_{t-1}}$ term in Eq.48 goes to infinity and undefined variable respectively. To compare this with IADDHP, we have used the same unsuccessful run NN weights for actor and critic using IDHP. IADDHP follows the task without any instability whereas ADDHP-FDM fails to do so. Figure 7 compares the control policies of the unsuccessful ADDHP-FDM with successful IADDHP. It can be observed that the control policy using FDM is quite aggressive compared to the control policy using RLS.

TABLE II: Comparison for *model-free* controllers for optimal learning rate

Controller Name	l_r	$\bar{c}_t$	$\bar{u}_t[deg]$	$\bar{t}_{compt}[sec]$	$\bar{t}_{set}[sec]$
IADDHP	13	0.031	6.8	17.5	0.12
ADDHP-FDM	0.1	0.03	7.3	14.5	1.41

This could be because of the higher excitation signal required for FDM methods.

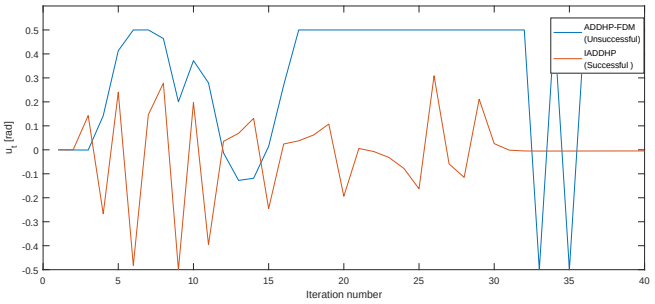

Fig. 7: Successful IADDHP vs unsuccessful ADDHP-FDM using same initial weights

Table II shows average values of different performance measure for successful runs for optimal learning rates. It can be seen that the ADDHP-FDM requires less computation time due to its simplicity in calculating the system dynamics matrix. Furthermore, It also requires higher average control effort for almost same average local cost. Figure 8 shows one of the successful response for reference tracking for ADDHP-FDM and IADDHP controllers. It can be seen that both methods can adequately follow a reference when the networks are converged.

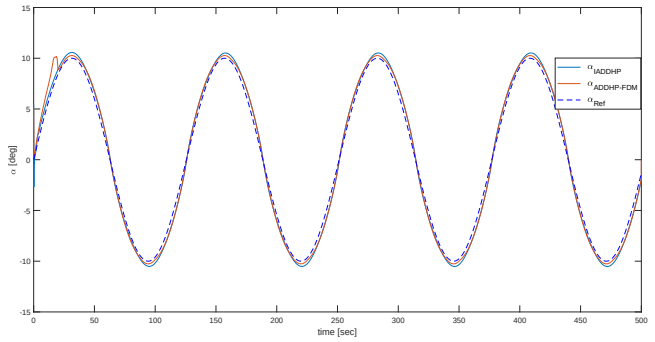

Fig. 8: α Reference tracking for *model-free* ADDHP controllers

C. With Noise

To check the performance of both controller under noise, a uniform Gaussian noise is added to sensor readings i.e. $x_t = x_t + \rho$ where ρ is the uniformly distributed noise.For IADDHP, noise with standard deviation $(\sigma) = [0.010.05]$ is added to the α in degrees whereas half of this is added to q. Table III shows the success rate for these 2 noise settings. For $\sigma = 0.01$, success rate has increased to 97% from 93% with no noise case. This increase in success rate when noise is applied is counter intuitive but this is because of the additional excitation of the system which enables the controller to explore.

TABLE III: Performance of IADDHP with noise

σ	Success rate
0.01	97%
0.05	62%

For ADDHP-FDM, σ for noise is decreased till the controller allowed the noise to pass. The result of this is summarized in Table IV. It turns out that the ADDHP-FDM controller allows noise with 40 times lower σ than IADDHP to pass through.

TABLE IV: Performance of ADDHP-FDM with noise

σ	Success rate
0.0002	53%
0.001	14%

D. With Failures

One of the properties of controllers using RL is their ability to adapt to the changing dynamics. Failures are introduced by suddenly changing the dynamics of the simulated system halfway through the simulation by changing signs of coefficients C_z and b_2 in missile model. No information about the failure is provided to controller which means that controller must be able to detect these changes and reset accordingly. This failure identification is done by *innovation* term e_t in RLS of IADDHP. Figure 9 and 10 show that around 250 secs, controller identify the failure and quickly adapts to the change .

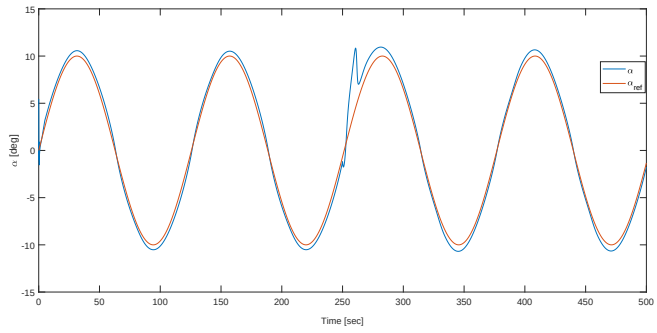

Fig. 9: Reference tracking with IADDHP controller with sudden sign change in C_z

As in ADDHP-FDM, there is no mechanism to predict the states i.e. identify if the identified model matches with the *'real system'*, therefore there is no way to identify the failure for adaptation.

V. Conclusion

The focus of this paper is to investigate incremental model based ADDHP; an online controllers which does not require

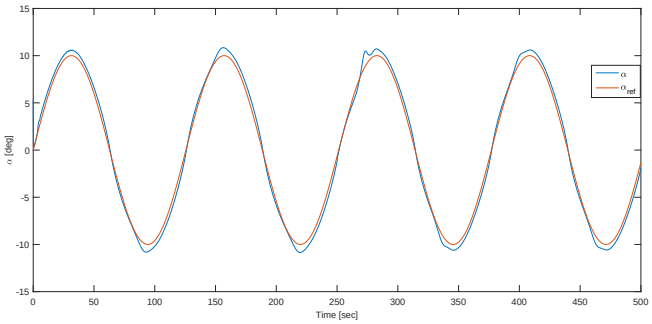

Fig. 10: Reference tracking with IADDHP controller with sudden sign change in b_2

any system information beforehand as opposed to the classical ADDHP method which model NN. Missile control is simulated to test the applicability of the controller. From the experiments, it was found that that incremental model based ADDHP can follow the reference satisfactorily with a very high success rate. It also outperforms ADDHP-FDM, an already existing *model-free* controller based on the basis of success rate, performance under different noise conditions and ability to adapt to system dynamics changes. Also, using FDM for ADDHP can make the controller susceptible to divergence for fast changing dynamics and is unable to detect failure. Although in this project, no internal training cycles are used as they were used in the original work, it is our argument that the use of internal training cycles will improve the performance of both controllers that is ADDHP-FDM and IADDHP. Therefore it is concluded that using incremental model with RLS for system identification in *online model-free* ADDHP is better solution than using FDM despite its simplicity.

REFERENCES

[1] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[2] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[3] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[5] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems Yearbook*, vol. 22, no. 12, pp. 25–38, 1977.

[6] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367–1372, 1995.

[7] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 764–773, 2002.

[8] J. Si and Y.-T. Wang, "Online learning control by association and reinforcement," *IEEE Transactions on Neural networks*, vol. 12, no. 2, pp. 264–276, 2001.

[9] S. Ferrari and R. F. Stengel, "Online adaptive critic flight control," *Journal of Guidance Control and Dynamics*, vol. 27, no. 5, pp. 777–786, 2004.

[10] S. Balakrishnan and V. Biega, "Adaptive-critic-based neural networks for aircraft optimal control," *Journal of Guidance, Control, and Dynamics*, vol. 19, no. 4, pp. 893–898, 1996.

[11] Z. Huang, J. Ma, and H. Huang, "An approximate dynamic programming method for multi-input multi-output nonlinear system," *Optimal Control Applications and Methods*, vol. 34, no. 1, pp. 80–95, 2013.

[12] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," in *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 2. IEEE, 2001, pp. 990–995.

[13] D. Liu, H. Javaherian, O. Kovalenko, and T. Huang, "Adaptive critic learning techniques for engine torque and air–fuel ratio control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 988–993, 2008.

[14] E. Van Kampen, Q. Chu, and J. Mulder, "Online adaptive critic flight control using approximated plant dynamics," in *Machine Learning and Cybernetics, 2006 International Conference on*. IEEE, 2006, pp. 256–261.

[15] R. Enns and J. Si, "Helicopter trimming and tracking control using direct neural dynamic programming," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 929–939, 2003.

[16] S. Sieberling, Q. Chu, J. Mulder *et al.*, "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction," *Journal of guidance, control, and dynamics*, vol. 33, no. 6, p. 1732, 2010.

[17] P. Simplício, M. Pavel, E. Van Kampen, and Q. Chu, "An acceleration measurements-based approach for helicopter nonlinear flight control using incremental nonlinear dynamic inversion," *Control Engineering Practice*, vol. 21, no. 8, pp. 1065–1077, 2013.

[18] P. Acquatella, E. van Kampen, Q. P. Chu *et al.*, "Incremental backstepping for robust nonlinear flight control," *Proceedings of the EuroGNC 2013*, 2013.

[19] Y. Zhou, E.-J. van Kampen, and Q. Chu, "Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback," *Journal of Guidance, Control, and Dynamics*, 2016.

[20] E.-J. v. C. Q. P. Zhou, Ye; Kampen, "Incremental model based heuristic dynamic programming for nonlinear adaptive flight control," *IMAV conference*, 2016.

[21] Z. Ni, H. He, X. Zhong, and D. V. Prokhorov, "Model-free dual heuristic dynamic programming," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 8, pp. 1834–1839, 2015.

[22] I. The MathWorks. Hyperbolic tangent sigmoid transfer function. [Online]. Available: https://nl.mathworks.com/help/nnet/ref/tansig.html

[23] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[24] M. Mansour and J. Ellis, "Comparison of methods for estimating real process derivatives in on-line optimization," *Applied Mathematical Modelling*, vol. 27, no. 4, pp. 275–291, 2003.

[25] B. Chen, Y. Zhu, J. Hu, and J. C. Principe, *System parameter identification: information criteria and algorithms*. Newnes, 2013.

[26] L. Ljung, "System identification: Theory for the user, ptr prentice hall information and system sciences series," *ed: Prentice Hall, New Jersey*, 1999.

[27] R. Isermann and M. Mnchhof, *Identification of Dynamic Systems: An Introduction with Applications*. Springer-Verlag Berlin Heidelberg, 2011.

[28] S.-H. Kim, Y.-S. Kim, and C. Song, "A robust adaptive nonlinear control approach to missile autopilot design," *Control engineering practice*, vol. 12, no. 2, pp. 149–154, 2004.

Part II

Thesis

Chapter 1

Introduction

1.1 Introduction

Optimal adaptive control is of paramount importance for flight control as it achieves the desired objective effectively under all conditions including system failures without unnecessary control effort. To solve optimal control problem for nonlinear systems, it is required to solve Jacobi-Hamilton-Bellman (JHB) equations which is rather complicated as it involves solving nonlinear partial difference equations [1]. Most of the classical optimal controllers are designed by linearizing the system around certain operating points. These types of controllers provide suboptimal performance at other operating points and require gain scheduling.

Dynamic programming (DP) offers a solution to this but its applications are limited as it suffers from *Curse of Dimensionality* and requires complete knowledge of the systems model which also includes uncertainties, noise, and system failure information beforehand [2]. Also its *backwards* approach makes it impossible to implement this to real-time control. To solve this, Reinforcement Learning (RL) can be used which tries to achieve same results as DP but without full system information. In RL, agent modifies its actions based on the reward it receives by interacting with the environment. One such structure is Actor Critic Designs or Adaptive Critic Designs (ACD), in which critic approximates the Bellman equation to iteratively steers the control policy generated by actor to optimal control policy. In most of the conventional ACDs, interaction between critic and actor is carried out by using a *Model* structure which approximates global system dynamics by offline training. Obtaining this training data could be expensive as it comes from real flight or simulator model. Therefore the focus of this thesis is to investigate *model-free* controllers using RL that does not required offline training or any prior information about the system. Although few of these controllers already exist in the literature but our objective is find even better *model-free controllers*. We can define a general goal for this research as;

Can the already existing model-free approaches be improved?

To obtain this objective, this project describes the implementation of two novel ACDs that does not need prior knowledge or global approximation of the system. These methods are; the Incremental model based Dual Heuristic Dynamic Programming (IDHP) and Incrmental Action Dependent Dual Heuristic Programming (IADDHP). These *model free* approaches have the advantage that they can adapt online to sudden changes like failures in the systems, and perform under noise by using the local incremental model for system identification for nonlinear and uncertain system.

We can now define even more specific question that we will answer during the span of this thesis.

1. *Is action dependent form of IDHP better in performance?*
2. *Is IADDHP better than already existing online model-free ADDHP method by using Finite Difference Method?*

Performance of these different controllers is evaluated by measuring its ability to track a task, success rate, average time for stabilization and average computational time.

Literature review for this thesis is given in the following two chapters. Chapter 2 offers some background information about the basic concept of reinforcement learning which are helpful in the understanding for dynamic programming and how optimal control is achieved. Chapter 3 focuses on approximate dynamic programming and describes the motivation behind these methods along with advancements in actor critic designs and defines the preface of our research topic. Incremental approach that will be used for the controllers is described in chapter 4. Chapter 5 develops the algorithm for MIMO Incremental Model based Dual Heuristic Programming (IDHP) controller. In chapter 6, two Model free Action Dependent (AD) controllers are discussed. One is the new Incremental ADDHP while other one is ADDHP with finite difference method. Chapter 7 discusses the experimental setup for simulation to investigate aforementioned controller. Results of the simulation along with the reason for those results are presented in chapter 8. At the end, conclusion based on the results is given along with the recommendation for future work in chapter 9.

Chapter 2

Background

In this chapter, a brief background of optimal control using RL is given along with the basic concepts required for the understanding of literature of Actor Critic Designs (ACD) and current research topic.

2.1 Reinforcement Learning

In reinforcement learning, an *agent* learns its actions by interacting with the *environment* without having the complete knowledge of it. At every time step *'t'*, *agent* interacts with the environment by producing the action ($u_t \in U(x)$) based on the states ($x_t \in X$) and rewards (c_t) it receives from the environment. This is shown in Fig.2.1.

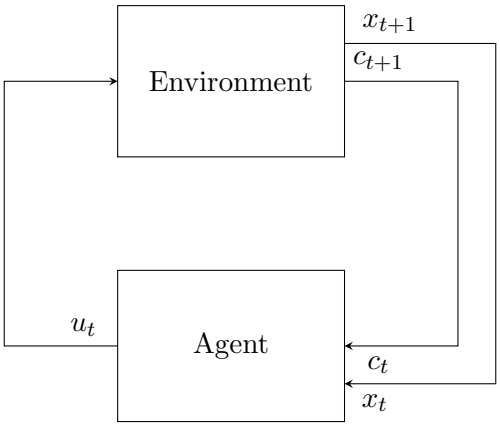

Figure 2.1: Basic Concept of Reinforcement Learning

Environment is everything which is outside of the agent whereas *reward* is the singular number associated with how well goal is being achieved at that time. The objective of the

agent is to choose u_t which maximize(or minimize) *value* function or *cost-to-go* function given by Eq.3.2 thus making it the optimal control.

$$G_t = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1} \tag{2.1}$$

G_t represents the discounted sum of all the future rewards. γ is the discount rate in the range of $0 \leq \gamma \leq 1$ and it describes how much future reward is worth at current time. If $\gamma = 0$ then the agent gives all importance to maximizing rewards at that instance. If $\gamma \to 1$, then the agent becomes more foresighted.

This basic concept of RL can be applied to the control problem for designing the adaptive controller that can update itself to find the optimal control policy. In this thesis, similar to [3], methods for solving the optimal control problem by using the concept of Value functions are considered as part of RL despite their need for complete knowledge of the system information.

2.2 Optimal Value Functions

To produce u_t, *agent* follows a certain *policy* (μ) which is the mapping of $x_t \to u_t$. This mapping describes how the action is selected given the system state at time *'t'* i.e $u_t = \mu(x_t)$. Using this we can define the *value* function for starting at state x under policy μ as Eq.2.2.

$$V^{\mu}(x) = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1}|_{x_t=x} = \sum_{k=0}^{\infty} \gamma^k c(x_{t+k}, u_{t+k}, s_{t+k+1})|_{x_t=x} \tag{2.2}$$

$V^{\mu}(x)$ is known as *state-value functions for policy* μ. It describes how good it is to be in state x when following μ. Furthermore, *action-value functions for policy* μ describes the value of starting at state x, and taking action u under policy μ subsequently. This is given as;

$$Q^{\mu}(x, u) = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1}|_{x_t=x, u_t=u} = \sum_{k=0}^{\infty} \gamma^k c(x_{t+k}, u_{t+k}, s_{t+k+1})|_{x_t=x, u_t=u} \tag{2.3}$$

The value function in RL satisfies the recursive relation which states that the value function at start state is equal to the discounted value of the expected next state and the reward obtained along the way for any μ. This recursive condition is called as *Bellman equation* and is given by Eq.2.4 for V^{μ}.

$$V^{\mu}(x) = c(x, u, x') + \gamma V^{\mu}(x') \tag{2.4}$$

where x' is the next state. The main idea of RL is to use the value functions to search for optimal policy. An optimal policy is the one that is better than all other policies and is represented by μ^*. All optimal policies have same *optimal state-value function* or *optimal action-value function* given by Eq.2.5 and 2.5 respectively [3][4].

$$V^*(x) = \max_{\mu} V^{\mu}(x) \quad \forall x \in X \tag{2.5}$$

$$Q^*(x,u) = \max_{\mu} Q^{\mu}(x,u) \quad \forall x \in X, u \in U(x) \tag{2.6}$$

Since $V^*(x)$ is the value function for a policy, it also satisfies Eq.2.4. This is called *Bellman optimality equation* and can be written as;

$$V^*(x) = \max_{u}\{c(x,u,x') + \gamma V^*(x')\} \tag{2.7}$$

$$Q^*(x,u) = c(x,u,x') + \gamma \max_{u'}\{V^*(x',a')\} \tag{2.8}$$

Optimal policy can be found easily once the value functions corresponding to the μ^* are found. For this, *Bellman optimal equations* can be used as the assignments to find the approximation for desired value functions and ultimately optimal policy.

2.3 Dynamic Programming

Dynamic Programming (DP) are the algorithms that find the optimal control policy given a perfect model of the environment. Although the applicability of classical DP methods in RL is limited due to its high computational expense for large state system and its need for complete system's information, understanding the basic idea behind the DP methods is indispensable as most of the RL methods try to attain same results as DP with limited knowledge of environment and with less computational burden [3]. DP and almost all RL methods can be understood by considering them as *Generalized Policy Iteration* (GPI) methods which encompasses the idea of alternating between finding optimal value function and optimal policy. This is shown in Fig.2.2

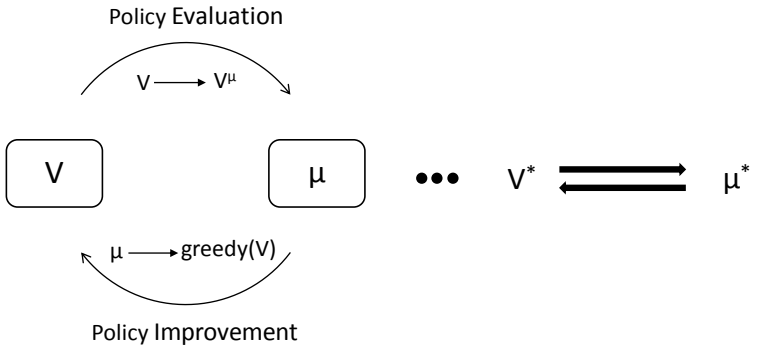

Figure 2.2: Generalized Policy Iteration

Policy Evaluation

Policy evaluation is the step to make the *state-value function* V^{μ} consistent under random policy μ. This is done iteratively by using the Bellman equation given by Eq.2.4.

Policy Improvement

Policy improvement is the step which helps find a better policy μ' given the *state-value function* for previous policy V^{μ}. A new policy is better if following condition is satisfied.

$$Q(x, \mu'(x)) \geq V^{\mu}(x) \quad \forall x \in X \tag{2.9}$$

which means that the return from the new policy for state x should be greater than previous policy, that is;

$$V^{\mu'}(x) \geq V^{\mu}(x) \quad \forall x \in X \tag{2.10}$$

The new greedy policy can be obtained by;

$$\mu'(s) = \underset{u}{\operatorname{argmax}}\{Q^{\mu}(x, u)\} \tag{2.11}$$

Policy improvement must give the better policy unless the policy is already an optimal policy [3].

Using the concept of *policy evaluation* and *policy improvements*, two methods *policy iteration* and *value iteration* can be guaranteed to find the optimal value function and policy given the complete information of the finite Markov Decision Process. In *policy iteration*, *policy evaluation* and *policy improvement* finish before other can start and alternate between each other whereas in *value iteration* completion of either of step is not required i.e. only one iteration of *policy evaluation* is completed to start *policy evaluation* and this is repeated. Although both methods achieve similar results, *value iteration* allows for the online learning of the optimal control policy with time.

Chapter 3

Approximate Dynamic Programming

In this chapter, motivation behind the development of Approximate Dynamic Programming (ADP) is discussed along with the structure of basic ADP. These methods are capable of dealing with uncertainty in the system and large state space and ideal for control of such systems.

In 1977, Paul J. Werbos [5] developed the approach for ADP which was afterwards called as Actor Critic Designs (ACD). Throughout the literature, various terms has been used interchangeably for ADP such as neuro-dynamic programming [6], neural dynamic programming [7], reinforcement learning[8][3], and adaptive critic designs. The basic idea in all these methods is similar. In this thesis, we will be using the term Approximate Dynamic Programming and Actor Critic Designs.

3.1 Motivation for ADP

As the digital implementation of the system is discrete in time therefore we consider such system for the rest of the thesis. A nonlinear discrete system can be written as;

$$x_{t+1} = f(x_t, u_t) \tag{3.1}$$

where t is the discrete time steps and $x_t \in \mathbb{R}^n \times 1$ and $u_t \in \mathbb{R}^m \times 1$. Associated performance index or *Cost-to-go* function at time t can be defined as Eq.3.2 which is the discounted sum of all the future rewards at 't'.

$$J_t = \sum_{k=t}^{\infty} \gamma^{k-t} c_k \tag{3.2}$$

γ is the discount factor with value ranging in $0 < \gamma < 1$ and its significance in the equation is described in chapter 2. c_k is the local cost function or utility function.

The objective for the DP in this thesis is to find the Optimal control policy compromised of optimal sequential control actions u_t^* *for* $t = 0, 1, 2....$ such that the cost-to-go function is minimized(maximized). Using Bellman optimality equation given in Eq.2.4, Bellman principal of optimality for discrete time can be written as,

$$J_t^* = \min_{u_t} c_t + \gamma J_{t+1}^* \tag{3.3}$$

where $J_{t+1} = J(f(x_t, u_t))$. This allows for the control action to be optimized one step at a time by working backward in time. Optimal control action $u^*(x_t)$ is the one which minimizes the optimal cost-to-go function and can be written as Eq.3.4.

$$u^*(x_t) = \underset{u_t}{\operatorname{argmin}}\, c_t + \gamma J_{t+1}^* \tag{3.4}$$

DP is a very powerful tool that deals with the problem of optimal control using sequential control actions given the perfect model of the system by using the Eq.3.3. This is done through exhaustive research from final point to the starting point to look for the optimal policy by rejecting the suboptimal paths at different points [2]. But, it remembers all the paths which makes it quite computationally expensive for large state space hence the *Curse of Dimensionality*. Also, the backward search makes it impossible for the DP to be applied for real time applications [9]. In addition to that, for complicated nonlinear systems it is very difficult to obtain the exact value of J^*. Another problem that we face with normal tabular methods of RL is that of *generalization* that is how can experience from a small state space be approximated to the larger state space. This is important because we would like to implement the RL controllers online for the states that have not been encountered before.

To solve these problems, instead of finding the exact value of cost-to-go function, existing generalization methods known as *function approximators* can be used to find an approximate value. These function approximators takes the examples of a function and try to form an approximation for entire function. Theoretically, any function approximator that is used in machine learning, curve fitting, artificial neural network (ANN or NN), or in pattern recognition can be used [3]. In this thesis, NN are used because of their high approximation power of a function along with function's derivatives as it will be described later.

Another important development in the RL method is the *Actor Critic Methods* which are the Temporal Difference(TD) methods with structures to explicitly store the actions. TD methods wait for only one time step to update $V(x_t)$. In these methods, *actor* selects the action whereas *critic* evaluates the value function. These methods are on-policy methods as critic evaluates the performance of current policy being used. Separation of policy evaluation and policy improvement makes it simpler to implement and learn. This architecture is shown in Fig.3.1.

Using the approximation power of Neural Networks (NN) and combining it with Actor Critic architectures led to the development of ADP or ACD.

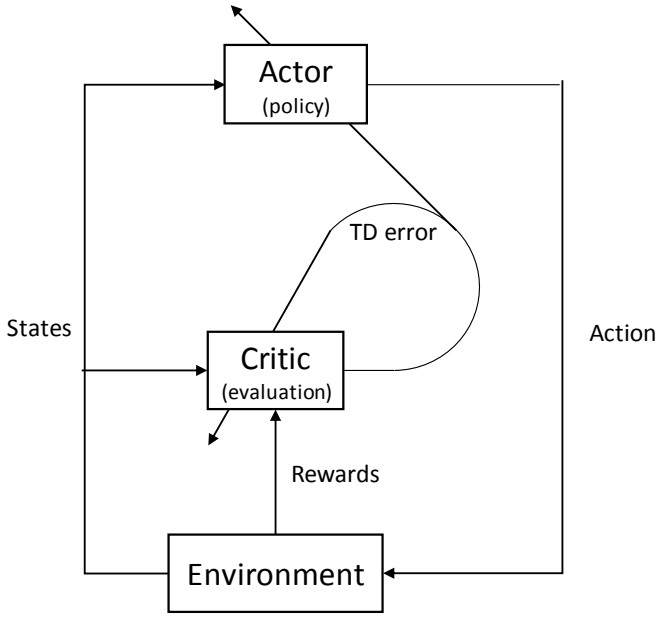

Figure 3.1: Actor Critic Architecture

3.2 Basic structure of ACD

Most of the classical ACDs have three basic components. There are; Critic, Actor, and a Model. All of these use NN to approximate their intended output. Traditionally, critic NN is used to approximate cost-to-go function *(J)* and performs policy evaluation whereas actor NN approximates the control action. Critic and actor interact with each other to iteratively steer the control policy to the optimal policy and this is done by using model NN which approximates the system dynamics and predict system next state $\hat{x_{t+1}}$. This structure is shown in Fig.3.2 and is called as Heuristic Dynamic Programming (HDP).

Model NN can be trained offline beforehand [9]. A few other methods train the model NN offline and then adapt to changes in dynamics online[10]. The need for model NN will become evident as we discuss the critic and actor NN training.

Once the model NN is trained, critic NN is updated. As mentioned earlier that the critic outputs an estimate of J given by Eq.3.2. This is done by training critic NN to minimize following error over time.

$$\|E_{c_t}\| = \sum_t \frac{1}{2} e_{c_t}^2 \tag{3.5}$$

$$e_{c_t} = J_t - \{\gamma J_{t+1} + c_t\} \tag{3.6}$$

where J_t is the output of critic for x_t and J_{t+1} is the output of critic for an estimate of next state ($\hat{x}_{t+1}$) provided by the model NN. After critic NN is updated, actor NN is trained by minimizing $\gamma J_{t+1} + c_t$ to reach optimal or sub-optimal policy depending on critic's performance. That is the backpropogation error is propagated through critic NN

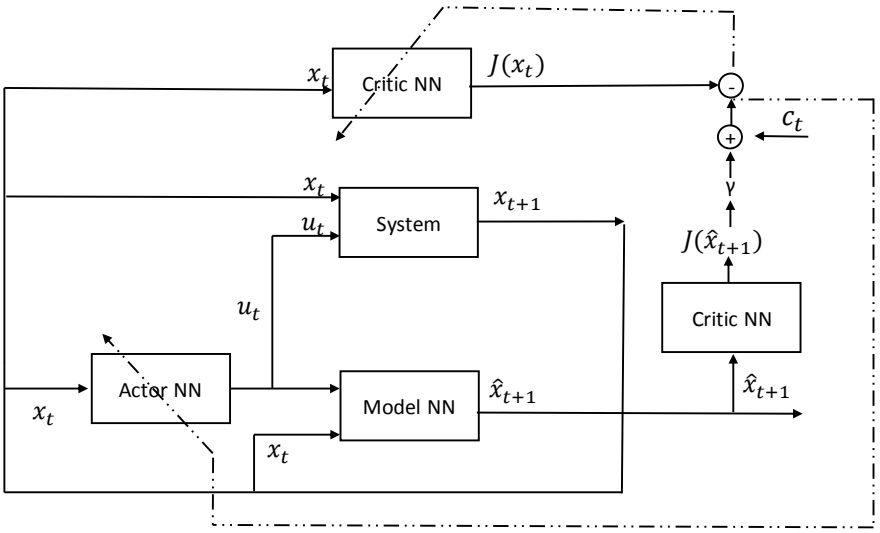

Figure 3.2: HDP: An Example of ACD

to model NN and then to action network. Therefore even if the $f(x_t, u_t)$ is completely known, model network is still required to maintain the propagation path for actor NN training.

3.3 Advancements in ACD

Paul J. Werbos has generalized the previously suggested structure for ADP in following categories; Heuristic Dynamic Programming (HDP) in which critic approximates the desired cost function (J) shown in Fig.3.2, Action Dependent HDP (ADHDP) in which actor output is directly connected to the critic of HDP, Dual Heuristic Dynamic Programming (DHDP) in which critic approximates the derivative of J with respect to its states, and Generalized Dual Heuristic Dynamic Programming (GDHP) which approximate both J and its derivatives [11]. Comparison of these methods has shown that the ADHDP applied to the auto-landing problem has lower success ratio than HDP. DHP performs better than HDP as they are directly approximating the derivative instead of indirect calculation of derivative in HDP obtained by backpropogation through critic NN required for training of critic [12] [13]. GDHP does not significantly outperform the DHP in auto-landing task but its training is much more computationally expensive because of second order derivative term that need to be calculated at every time step [14]. Prokhorov & Wunsch furthered these categories by introducing the AD forms of DHP and GDHP as advanced actor critic designs and experimentally showed that these are better than HDP and ADHDP [9]. Prokhorov & Wunsch also suggested that although GDHP and ADGDHP outperforms other methods, for most applications DHP and ADDHP are adequate. Many of these structures have been used for different flight control applications. In most of these examples, first; Model network is trained offline for baseline controller and then the controller adapts to the changes in dynamics online [10],[15].

3.3.1 Model-free ACD

As it is explained before, model NN is essential to maintain the backpropogation paths but it also requires offline training. To properly train the model NN, a large amount of data distributed over the entire state space is required to find appropriate global model estimate of the system. But in most applications, especially in flight control, obtaining this data can be quite expensive. Also it is not possible to predict beforehand which states will be encountered during the operation. Therefore, *model-free* approaches in ACD presents themselves as quite a lucrative option and encompassing the true essence of RL for control.

Earlies attempt for a *model-free* ACD is by using the ADHDP. In this structure, model network is completely omitted as action NN can be updated by backpropogating through critic NN as shown in Fig.3.3. But the requirement for model NN to train critic NN was averted by simply waiting for the next term [12][9] but a performance degradation was observed using this method.

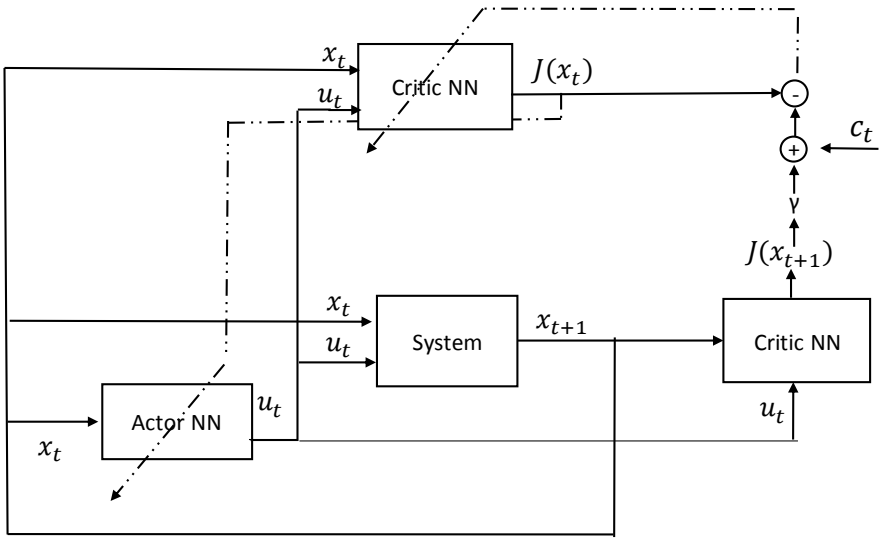

Figure 3.3: Model-free ADHDP

It is also important to note that same can not be done to obtain *model-free* ADDHP and ADGDHP as they don't just require one-step prediction from model NN but they also need model NN to maintain backpropogation paths. To understand this, lets consider the critic training error for DHP or ADDHP.

$$e_{c_t} = \frac{\partial J_t}{y_t} - \{\gamma \frac{\partial y_{t+1}}{y_t} \frac{\partial J_{t+1}}{y_{t+1}} + \frac{\partial c_t}{\partial y_t}\} \tag{3.7}$$

where $y_t = x_t$ for DHP and $y_t = [x_t; u_t]$ for ADDHP. To obtain the term $\frac{\partial y_{t+1}}{y_t}$, model NN is required even if we use the AD form and wait for the next time step.

Si & Wang made very remarkable contribution in online learning algorithm by proposing a *model-free* ACD closely resembles the ADHDP structure with main distinction being

completely *model-free* without compromising on accuracy by storing the previous cost-to-go value [14]. By doing this, training error for the critic training is given by;

$$e_{c_t} = \{\gamma Q_t + c_t\} - Q_{t-1} \tag{3.8}$$

In Eq.3.8, Q_t approximates the cost-to-go function given by;

$$Q(t) = c(t+1) + \gamma c(t+2) + \gamma^2 c(t+1) + \cdots \tag{3.9}$$

or

$$Q(t) = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} c_k \tag{3.10}$$

Comparing Eq.3.10 with Eq.3.2, we can write the following relation.

$$Q(t) = J(t+1) \tag{3.11}$$

This tells us that using Eq.3.8 to train critic NN, output Q_t is the approximation of J_{t+1} [16]. This is shown in Fig.3.4. This method has been used successfully for many appli-

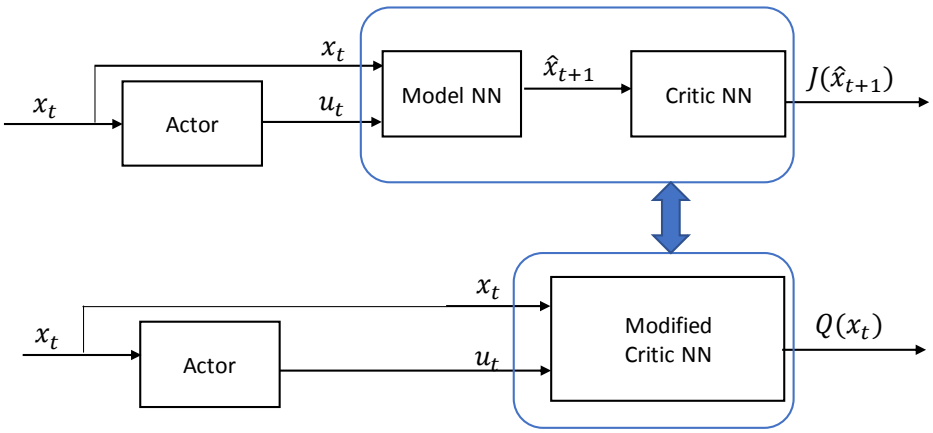

Figure 3.4: Modified ADDHP

cations and will be referred as modified ADHDP method[17][16][18]. In flight control, ADHDP has been applied to F-16 model and comparison was done with HDP. Results showed that HDP with approximated model dynamics outperforms ADHDP under changing dynamics but ADHDP perform better with noise [19]. Russel & Si also applied the *model-free* modified ADHDP to control Apache helicopter for different realistic maneuvers and found the controller performance to be satisfactory [7]. But as the author pointed out, this study was done offline where a system learns by trial and can afford failures. Liu et al. furthered this modified ADHDP by introducing two methods to train the critic NN namely*backward in time* in which critic is trained with Q_t in Eq.3.8 as critic's output by using $1/\gamma[Q_{t-1} - c_t]$ as target for training. whereas in *forward in time* approach, critic network outputs $Q(t-1)$ and is trained with $\gamma Q_t + c_t$. It is author's opinion that the *backward in time* training can lead to instabilities due to $1/\gamma$ term but it is worth noticing that the MATLAB trainlm function was used to train the critic network.

3.3.2 Current Research Topic

In the light of the literature mentioned above, we know that DHP and ADDHP has better performance than HDP and its AD form. But obtaining a *model-free* (AD)DHP is not possible due to $\frac{\partial y_{t+1}}{\partial y_{t+1}}$ term in Eq.3.7. This term is required even if we apply Eq.3.8 to get critic training error. In the literature, only other author who has used *online model-free* ADDHP is Zhen et al by making use of Finite Difference Method (FDM)[20] and applied it to a ball and beam balancing example.

To this point, all the literature mentioned above was using NN to approximate the global system model to get the next state x_{t+1} and $\frac{\partial y_{t+1}}{\partial y_{t+1}}$ term. Another take on the model part of the ACD is by using incremental based model. According to this, a nonlinear model in the form of Eq.3.12 can be written in incremental form as Eq.3.13 in discrete form, given very high sampling frequency and slow varying dynamics.

$$\dot{x}(t) = f(x(t), u(t)) \tag{3.12}$$

$$\Delta x_{t+1} = F_{t-1}\Delta x_t + G_{t-1}\Delta u_t \tag{3.13}$$

Instead of approximating the global model, local model is approximated to avoid dealing with large set of training data. This incremental model approach for control has been successfully applied to control nonlinear system. It's applications include Incremental Dynamic Inversion (INDI) [21][22], and Incremental Backstepping (IBS) [23]. In ADP, an Incremental Approximate Dynamic Programming (iADP) was developed by Zhou et al. to find the near optimal control of nonlinear system [24]. In Actor Critic Designs, incremental HDP (IHDP) used incremental model to find the local system dynamics instead of NN for global dynamics to get the next state to be used in Bellmans equations. F_{t-1} and G_{t-1} are easily identifiable using Ordinary Least Square (OLS) methods or Recursive Least Square (RLS) and can replace backpropogation through model NN terms in weight updates of Actor and Critic NN thus simplifying the procedure and making online learning feasible. Studies performed show that IHDP outperforms the traditional HDP and speeds up the process of online learning for missile control [25]. Quite recently, Zhou et al. also investigated IDHP and compared it with traditional DHP. Results are quite promising which encourages and form the basis of this project that is investigation of performance of Incremental Action Dependent Dual Heuristic Programming (IADDHP). Currently, IHDP and IDHP has been applied for control of a missile in the range of $-10\,\text{deg} \leq \alpha \leq 10\,\text{deg}$ using only elevator input.

Chapter 4

Incremental Model

In previous chapter, we have discussed the need for model NN in classical ACD to obtain estimate of J_{t+1} in Bellman optimality equation(Eq.3.6) and to update actor NN. In addition to that, it is also used to obtain $\frac{\partial y_{t+1}}{\partial y_t}$ term for (AD)DHP structures . Therefore; model NN is used in classical ACD for system identification and state prediction. But this model NN needs to be trained offline before it can be used for online adaptation which requires a lot of training data to approximate global system dynamics. To avoid that, we use $Incremental\ Model$ instead of model NN. This not only diminishes the need for offline training of the ACD but it also does not require any prior system's knowledge which makes it $'model\text{-}free'$ ACD. In this chapter, we discuss the discrete incremental model form along with two online system identification methods for the incremental model.

4.1 Continuous Incremental Model

Non-linear continuous system can be defined by Eq.4.1 where $\dot{x}(t)$ represents the change in system dynamics and $y(t)$ is the output of system.

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \\ y(t) &= h(x(t), u(t)) \end{aligned} \tag{4.1}$$

Equation 4.1 can be linearized around a time t_o using Taylor series expansion as Eq.4.2 which can be written as continues time-variant state space of incremental form as Eq.4.3

$$\dot{x}(t) \simeq \dot{x}(t_o) + \frac{\partial f(x(t), u(t))}{\partial x(t)}|_{x(t_o),u(t_o)}(x(t) - x(t_o)) + \frac{\partial f(x(t), u(t))}{\partial u(t)}|_{x(t_o),u(t_o)}(u(t) - u(t_o)) \tag{4.2}$$

$$\Delta\dot{x}(t) \simeq F(x(t_o), u(t_o))\Delta x(t) + G(x(t_o), u(t_o)\Delta u(t) \tag{4.3}$$

where $F(x(t_o), u(t_o)) = \frac{\partial f(x(t),u(t))}{\partial x(t)}|_{x(t_o),u(t_o)} \in \mathbb{R}^{n\times n}$ is the state matrix and $G(x(t_o), u(t_o)) = \frac{\partial f(x(t),u(t))}{\partial u(t)}|_{x(t_o),u(t_o)} \in \mathbb{R}^{n\times m}$ is the input matrix.

4.2 Discrete Incremental Model

Although a system can be continuous but the data obtained from sensors is discrete. Therefore, Eq. 4.1 can be written in discrete form for a system with very high sampling frequency and full state observations as Eq.4.4.

$$x_{t+1} = f(x_t, u_t) \tag{4.4}$$
$$y_t = h(x_t, u_t) = x_t$$

Linearizing this equation around previous time step $t-1$ will give us Eq.4.5.

$$\Delta x_{t+1} \simeq F_{t-1}\Delta x_t + G_{t-1}\Delta u_t \tag{4.5}$$

where $\Delta x_t = x_t - x_{t-1}$, $\Delta u_t = u_t - u_{t-1}$ and $F_{t-1} \simeq \frac{\partial x_{t+1}}{\partial x_t}|_{model} \in \mathbb{R}^{n\times n}$ and $G_{t-1} \simeq \frac{\partial x_{t+1}}{\partial u_t} \in \mathbb{R}^{n\times m}$ are system matrix and input matrix at $t-1$ respectively. As we are assuming the slowly time varying system with high sampling frequency, these matrices representing the linear system can be easily identified online by linear regression methods by taking some previous measurements. In this paper, two such methods are analyzed that are $(SW-OLS)$ and (RLS)

4.2.1 Sliding Window-Ordinary Least Square

OLS can be used to identify the matrices in Eq.4.5 by using previous measurements of Δx and Δu. In sliding window method, only window of $'M'$ previous measurements are taken into account for identification of F_{t-1} and G_{t-1} with each $t-M$ measurement forgotten at time step t. Detailed derivation of this method can be found in [25]. Equation for $'rth'$ element of Δx_{t+1} can be rewritten as Eq.4.6.

$$\Delta x_{r_{t+1}} = \begin{bmatrix}\Delta x^\intercal & \Delta u^\intercal\end{bmatrix}\begin{bmatrix}f_r^\intercal \\ g_r^\intercal\end{bmatrix} \tag{4.6}$$

where f_r and g_r are rth rows of matrix F_{t-1} and G_{t-1} respectively. Sliding window length depends on the number of unknowns in f_r and g_r i.e. $M \geq n+m$. Using a very long window for system identification will not be true representative of system at that instance because of nonlinearity of the model whereas using small window might be ill-conditioned. In this project, $M = 2(n+m)$ is sufficient. Using least squares, solution for f_r and g_r can be written as Eq. 4.7

$$\begin{bmatrix}f_r^\intercal \\ g_r^\intercal\end{bmatrix} = (A_t^\intercal A_t)^{-1}A_t^\intercal Y_{r_t} \tag{4.7}$$

where;

$$A_t = \begin{bmatrix}\Delta x_{t-1}^\intercal & \Delta u_{t-1}^\intercal \\ \vdots & \vdots \\ \Delta x_{t-M}^\intercal & \Delta u_{t-M}^\intercal\end{bmatrix}, Y_{r_t} = \begin{bmatrix}\Delta x_{r_t} \\ \vdots \\ \Delta x_{r_{t-M}}\end{bmatrix}$$

4.2.2 Recursive Least Square

For the online identification of linear model, RLS is a very powerful method as it recursively update the estimate as new measurements become available and doesn't require saving all previous measurement. It also has better convergence than other least square methods but it is computationally more intensive[26].

For our linearized model, Eq.4.5 can be rewritten as Eq.4.8 which can be solved by RLS summarized in Eq.4.10 to Eq.4.14 where $\hat{\Theta}(t) = \begin{bmatrix} F_{t-1}^{\intercal} \\ G_{t-1}^{\intercal} \end{bmatrix}$ is the parameter to be estimated and $\psi(t) = \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}$ is the regression matrix which contains information about previous measurements[27].

$$\Delta x_{t+1} = \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}^{\intercal} \begin{bmatrix} F_{t-1}^{\intercal} \\ G_{t-1}^{\intercal} \end{bmatrix} \tag{4.8}$$

$$e(t) = x(t) - \hat{x}(t) \tag{4.9}$$

$$\hat{\Theta}(t) = \hat{\Theta}(t-1) + K(t)(e(t)) \tag{4.10}$$

$$\Delta\hat{x}(t) = \psi(t)^{\intercal}\hat{\Theta}(t-1) \tag{4.11}$$

$$K(t) = Q(t)\psi(t) \tag{4.12}$$

$$Q(t) = \frac{P(t-1)}{\Lambda + \psi(t)^{\intercal}P(t-1)\psi(t)} \tag{4.13}$$

$$P(t) = \frac{1}{\Lambda}\big[P(t-1) - \frac{P(t-1)\psi(t)\psi(t)^{\intercal}P(t-1)}{\Lambda + \psi(t)P(t-1)\psi(t)^{\intercal}}\big] \tag{4.14}$$

Λ is the forgetting factor which ensures that the weight of previous values is discounted exponentially for older measurements when $\Lambda < 1$. For systems with high varying dynamics, Λ should be small but this filters small noise. For higher values of Λ(around 0.998), larger noise can be allowed but tracking the changes in parameters is slower. $P(t)$ is the covariance matrix that should be initialized as $P(0) = \kappa I$ where κ should be very large for convergence[28].

Chapter 5

Incremental Model Based Dual Heuristic Programming (IDHP)

In DHP, critic NN approximates the derivatives of cost to go function with respect to the input of critic. This direct calculation of the derivative results in more accurate weight update as the direct derivative output has better quality than indirect backpropogation through NN[9]. Hence; it provides more accuracy as compared to HDP or ADHDP but with less computational power than GDHP. Like other ACDs, IDHP also consists of 3 components. These are; Critic for policy evaluation, Actor for policy improvement, and Model to maintain backpropagation paths. Actor and Critic are made up of NN where as for model network, instead of using NN, incremental model is used which makes IDHP a *model-free* controller and simplifies the back-propagation even further. The schematics for IDHP are shown in Fig.5.1.

5.1 Critic

For IDHP, as shown in Fig.5.1, Critic's output is estimate of derivative of cost-to-go function with respect to states given by Eq.5.1

$$\lambda_t = \frac{\partial J_t}{\partial x_t} \tag{5.1}$$

where J_t is cost-to-go function defined in Eq.3.2. $x_t \in \mathbb{R}^{n\times 1}$ is input to critic network at time t.$\lambda_t \in \mathbb{R}^{n\times 1}$ is a vector where its kth component represents the derivative of cost-to-go function with respect to kth input state i.e. $\lambda_{t_k} = \frac{\partial J_t}{\partial x_{k_t}}$.

5.1.1 Critic NN Structure

To approximate λ_t, one hidden layer multi-perceptron NN is used. Weights of this network are updated to minimize the critic error given in Eq.5.6. This is show in Fig. 5.2 for MIMO controller. Based on the structure of critic NN, its output can be written as;

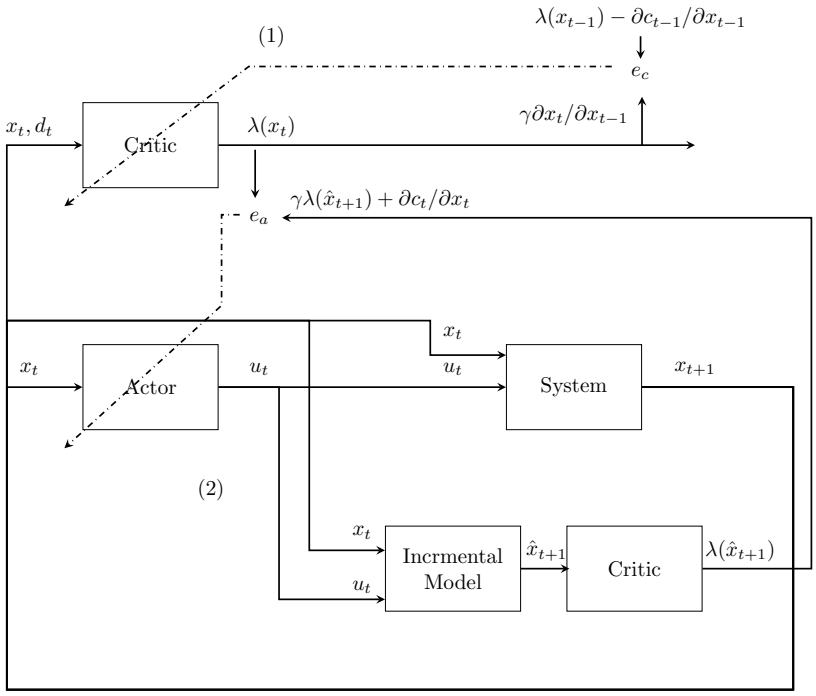

Figure 5.1: Structure of Incremental Model based Dual Heuristic Programming

$$\lambda_{t_k} = purelin(v_{c_{t_k}}) \tag{5.2}$$

$$v_{c_{t_k}} = \sum_{j=1}^{N_{h_c}} w_{c_{t_{kj}}} p_{c_{t_j}} \tag{5.3}$$

$$p_{c_{t_j}} = tansig(q_{c_{t_j}}) \tag{5.4}$$

$$q_{c_{t_j}} = \sum_{i=1}^{n} w_{c_{t_{ji}}} x_{t_i} \tag{5.5}$$

- λ_{t_k} is the kth output of neural network
- v_{t_k} is the input to kth neuron
- $w_{c_{t_{kj}}}$ is weight from jth hidden neuron to kth output
- p_{t_j} is the output of jth hidden neuron
- q_{t_j} is the input of jth hidden neuron
- $w_{c_{t_{ji}}}$ is the weight from ith input to jth hidden neuron

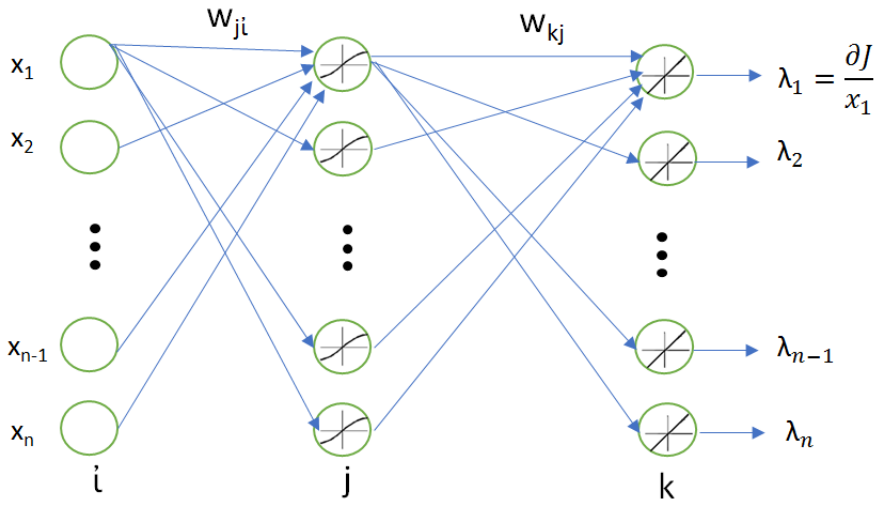

Figure 5.2: Critic Neural Network for DHP Structures

- x_i is the ith input to neural network
- n is the number of states
- N_h is the number of hidden neurons

where *purelin* and *tansig* transfer functions can be changed for the output neurons depending on the complexity of approximation. *Tansig* is equivalent to *tanh* mathematically but it runs faster on MATLAB with very small numerical difference [29]. These transfer functions along with their derivative function are calculated as;

$$a = purelin(b) = b$$

$$\frac{\partial a}{\partial b} = dpurelin(a) = 1$$

$$a = tansig(b) = \frac{2}{1 + e^{(-2b)}} - 1$$

$$\frac{\delta a}{\partial b} = dtansig(a) = 1 - a^2$$

5.1.2 Critic's Training

Critic is trained by minimizing the critic error defined in Eq. 5.6

$$\|E_{c_t}\| = \frac{1}{2} e_{c_t}^{\intercal} e_{c_t} \tag{5.6}$$

Prediction error for DHP structures can be written as Eq.5.9 for training forward in time[11].

$$e_{c_t} = \frac{\partial[J_t - \{c_t + \gamma J_{t+1}\}]}{\partial x_t} \tag{5.7}$$

$$e_{c_t} = \lambda_t - \{\frac{\partial c_t}{\partial x_t} + \gamma \frac{\partial J_{t+1}}{\partial x_t}\} \tag{5.8}$$

$$e_{c_t} = \lambda_t - \{\frac{\partial c_t}{\partial x_t} + \gamma \frac{\partial (x_{t+1})}{\partial x_t}\lambda_{t+1}\} \tag{5.9}$$

c_t is the local cost function and it's definition depends upon the mission at hand. In this project, quadratic cost function is defined as Eq.5.10 to describe reference tracking with minimum control input.

$$c_t(x_t, u_t) = (x_t - d_t)^\intercal Q(x_t - d_t) + u_t^\intercal R u_t \tag{5.10}$$

where $d_t \in \mathbb{R}^{n\times 1}$ is the desired states, $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are positive definitive matrices chosen to describe the relative importance of states and inputs in cost function.

Equation 5.12 can be used for training critic NN backward in time for DHP designs [14]. The method to get the weight update equation for both forward and backward in time is similar and both of these methods can be used to update critic weights. In this section, critic NN weight update equations using backward in time training is derived.

$$e_{c_t} = \frac{\partial[J_{t-1} - \{c_{t_1} + \gamma J_t\}]}{\partial x_{t_1}} \tag{5.11}$$

$$e_{c_t} = \lambda_{t-1} - \{\frac{\partial c_{t-1}}{\partial x_{t-1}} + \gamma \frac{(\partial x_t)}{\partial x_{t-1}}\lambda_t\} \tag{5.12}$$

e_{c_t} is a $n \times 1$ vector where $e_{c_{t_k}}$, kth element of e_{c_t}, corresponds to the critic error for kth state. λ_t and λ_{t-1} are output of critic network for two different instances for states but using weights at time 't' i.e w_{c_t}; whereas $\frac{(\partial x_t)}{\partial x_{t-1}}$ provides system's information and can be obtained from the model part of IDHP and actor. Using chain rule, this can be written as,

$$\frac{(\partial x_t)}{\partial x_{t-1}} = \frac{\partial x_t}{\partial x_{t-1}}|_{model} + \frac{\partial x_t}{\partial u_{t-1}}\frac{\partial u_{t-1}}{\partial x_{t-1}}|_{model+actor} \tag{5.13}$$

where $\frac{\partial u_{t-1}}{\partial x_{t-1}} \in \mathbb{R}^{m\times n}$ can be obtained by backpropagating through actor NN. $\frac{\partial c_{t-1}}{\partial x_{t-1}}$ in Eq.5.12 for local cost function or utility function c_{t-1} which depends on u_{t-1} can be written as Eq.5.14 using the chain rule.

$$\frac{\partial c_{t-1}}{\partial x_{t-1}} = \frac{\partial c_{t-1}}{\partial x_{t-1}} + \sum_{k=1}^{n} [\frac{\partial c_{t-1}}{\partial u_{k_{t-1}}}\frac{\partial u_{k_{t-1}}}{\partial x_{t-1}}] \tag{5.14}$$

Using the local cost function in Eq.5.10; $\frac{\partial c_{t-1}}{\partial x_{t-1}}$ becomes

$$\frac{\partial c_{t-1}}{\partial x_{t-1}} = diag(Q) \circ (x_{t-1} - d_{t-1}) + diag(R) \circ (u_{t-1}^{\intercal} \frac{\partial u_{t-1}}{\partial x_{t-1}})^{\intercal} \tag{5.15}$$

In Eq. 5.15, $\circ$ represents element-wise multiplication of a vector with a matrix.

5.1.3 Critic NN weight update

There exist many methods that can be used to update the NN weights. Each method has its own advantages and disadvantages. In this thesis, we will discuss *Robbins-Monro* algorithm but various other methods have been used for network training of ACD such as Resilient backpropogation (RPROP) and it's modified form which uses the temporal behavior of sign of error function for weights update which leads to faster convergence but it can be successfuly used for the batch learning [10] [30].

Robbins-Monro Algorithm

For the online model-free controllers discussed in this thesis, we have used a recursive Stochastic Gradient Descent (SGD) method also know as Robbins-Monro algorithm to update NN weights for actor and critic due to its ability to converge to an optimal or near-optimal property.

Using Stochastic Gradient Descent method, weight update equation can be written as;

$$w_{t+1} = w_t + \Delta w_t \tag{5.16}$$

$$\Delta w_t = l_t \big[- \frac{\partial E_t(x_t, u_t)}{\partial w_t} \big] \tag{5.17}$$

where l_t is the learning rate. According to [31], if the learning rate satisfies the conditions in Eq.5.18 then for quadratic E_t, w_t will converge to global optima w^* otherwise it will converge to a local optima with time. The convergence proof of this is given in [14].

$$\lim_{t \to \infty} l_t = 0, \quad \sum_{t=0}^{\infty} l_t = \infty, \quad \sum_{t=0}^{\infty} l_t^2 < \infty \tag{5.18}$$

Using this, critic NN weight update equations can be written as;

1. $\Delta w_c^{(2)}$ (Hidden to Output Layer Weights)
   Using Eq.5.17 and structure of the critic network given by Eq.5.2 to Eq.5.5 , weight update equation from critic's hidden layer to output layer can be written as a matrix $\Delta w_c^{(2)} = [\Delta w_{c_{kj}}^{(2)}] \in \mathbb{R}^{n \times N_{h_c}}$

$$\Delta w_{c_{kj}}^{(2)} = -l_{c_t} \big[\frac{\partial E_{c_t}}{\partial e_{c_{t_k}}} \big] \big[\frac{\partial e_{c_{t_k}}}{\partial \lambda_{t-1_k}} \big] \big[\frac{\partial \lambda_{t-1_k}}{\partial v_{c_{t-1_k}}} \big] \big[\frac{\partial v_{c_{t-1_k}}}{\partial w_{c_{t_{kj}}}} \big] \tag{5.19}$$

$$\Delta w_{c_{kj}}^{(2)} = -l_{c_t}.e_{c_{t_k}}.p_{c_{t-1_j}} \tag{5.20}$$

Using column vectors for e_{c_t} and $p_{c_{t-1_j}}$, Eq.5.20 can be rewritten as Eq.6.6.

$$\Delta w_c^{(2)} = -l_{c_t}.e_{c_t}.p_{c_{t-1}}^{\intercal} \tag{5.21}$$

2. $\Delta w_c^{(1)}$ (Input to Hidden Layer Weights) Critic neural network's weight update from input layer to hidden layer neuron can be written in a matrix $\Delta w_c^{(1)} = [\Delta w_{c_{ji}}^{(1)}] \in \mathbb{R}^{N_{h_c} \times n}$

$$\Delta w_{c_{ji}}^{(1)} = -l_{c_t} \sum_{k=1}^{n} \Big[\frac{\partial E_{c_t}}{\partial e_{c_{t_k}}}\Big]\Big[\frac{\partial e_{c_{t_k}}}{\partial \lambda_{t-1_k}}\Big]\Big[\frac{\partial \lambda_{t-1_k}}{\partial v_{c_{t-1_k}}}\Big]\Big[\frac{\partial v_{c_{t-1_k}}}{\partial p_{c_{t-1_j}}}\Big]\Big[\frac{\partial p_{c_{t-1_j}}}{\partial q_{c_{t-1_j}}}\Big]\Big[\frac{\partial q_{c_{t-1_j}}}{\partial w_{c_{t_{ji}}}}\Big] \tag{5.22}$$

$$\Delta w_{c_{ji}}^{(1)} = -l_{c_t} \sum_{k=1}^{n} e_{c_{t_k}}.w_{c_kj}.dtansig(q_{c_{t-1_j}}).x_{t-1_i} \tag{5.23}$$

Using column vectors, entire matrix update $\Delta w^{(1)}$ can be written as;

$$\Delta w_c^{(1)} = -l_{c_t}\big[e_{c_t}^{\intercal}.w_c\big]^{\intercal} \circ \big[dtansig(q_{c_{t-1}}).\hat{x}_{t-1}^{\intercal}\big] \tag{5.24}$$

5.2 Actor

In ACD, actor improves the control policy based on evaluation by critic by updating its weights. Actor takes states x_t as input and it outputs the near-optimal control action u_t. This control action is the input to the model and environment to produce estimated next state and the real next state at $t+1$.

5.2.1 Actor NN Structure

Actor NN approximates the optimal control policy u_t^*. For this, single hidden layer multi-perceptron NN is used. For MIMO IDHP controller, similar to critic NN in Fig. 5.2, input to the NN is the state vector x_t whereas its output is a control action vector $\hat{u}_t \in \mathbb{R}^{m \times 1}$. Equation 5.25 to 5.28 makes up actor NN.

$$u_{t_k} = tansig(v_{a_{t_k}}) \tag{5.25}$$

$$v_{a_{t_k}} = \sum_{j=1}^{N_{h_a}} w_{a_{t_{kj}}} p_{a_{t_j}} \tag{5.26}$$

$$p_{a_{t_j}} = tansig(q_{a_{t_j}}) \tag{5.27}$$

$$q_{a_{t_j}} = \sum_{i=1}^{m} w_{a_{t_{ji}}} x_{t_i} \tag{5.28}$$

As mentioned in section 5.1.1, transfer function for the output layer can be changed.

5.2.2 Actor NN Weight Update

The objective of actor is to produce the control policy u^* which can minimize(*maximize*) error between J_t and desired ultimate objective U_t. In this case, as the r_t is defined in terms of tracking error and control action required at t, ideally we would like $J_t = 0$ therefore U_t is set to 0. This control policy can be written as Eq.5.29

$$u^* = \underset{u_t}{\operatorname{argmin}}(J_t - U_t) = \underset{u_t}{\operatorname{argmin}}(J_t) \tag{5.29}$$

Actor neural network weights are adapted using Eq.5.30 as target. As there is no direct relation between J_t and u_t in IDHP, indirect back-propagation through critic and model is used for weight update given by Eq. 5.31. As we can see that for weight update of actor, only forward in time Bellman equation given by Eq.5.7 can be used. This backpropogation path is shown in Fig.5.1 by dotted line (2). Actor's weight update equation can be written as Eq.5.32[9].

$$\frac{\partial J_t}{\partial u_t} = 0 \tag{5.30}$$

$$\frac{\partial c_t}{\partial u_t} + \gamma \frac{\partial J_{t+1}}{\partial u_t} = 0 \tag{5.31}$$

$$\Delta w_{a_t} = l_{a_t} \big[\frac{\partial c_t}{\partial u_t} + \gamma \lambda_{t+1} \frac{\partial x_{t+1}}{\partial u_t} \big] \frac{\partial u_t}{\partial w_{a_t}} \tag{5.32}$$

Based on the Eq.5.10, $\frac{\partial c_t}{\partial u_t}$ can be written as Eq. 5.33;

$$\frac{\partial c_t}{\partial u_t} = diag(R) \circ u_t \tag{5.33}$$

1. $\Delta w_a^{(2)}$ (Hidden to Output Layer Weights)
   Using Eq.5.32, weights update equations from hidden layer to output layer for actor can be written as a matrix $\Delta w_a^{(2)} = [\Delta w_{a_{kj}}^{(2)}] \in \mathbb{R}^{m \times N_{h_c}}$.

$$\Delta w_{a_{kj}}^{(2)} = -l_{a_t} \big[\frac{\partial c_t}{\partial u_{t_k}} + \gamma \sum_{o=1}^{n} \frac{\partial J_{t+1}}{\partial x_{o_{t+1}}} \frac{\partial x_{o_{t+1}}}{u_{t_k}} \big] \frac{\partial u_{t_k}}{\partial v_{a_{t_k}}} \frac{\partial v_{a_{t_k}}}{\partial w_{a_{t_k j}}} \tag{5.34}$$

$$\Delta w_{a_{kj}}^{(2)} = -l_{a_t} . \big[\frac{\partial c_t}{\partial u_{t_k}} + \gamma \sum_{o=1}^{n} \lambda_{o_{t+1}} \frac{\partial x_{o_{t+1}}}{u_{t_k}} \big] . dtansig(v_{a_{t_k}}) . p_{a_{t_j}} \tag{5.35}$$

   Eq.5.35 can be rewritten as Eq.5.36 using column vectors $(\frac{\partial c_t}{\partial u_t}, v_{a_t}, p_{a_t})$ and a matrix $\frac{\partial x_{t+1}}{u_t}$ form.

$$\Delta w_a^{(2)} = -l_{a_t} . \big[\frac{\partial c_t}{\partial u_t} + \gamma \lambda_t \frac{\partial x_{t+1}}{u_t} \big] \circ dtansig(v_{a_t})^\intercal . p_{a_t} \tag{5.36}$$

   where $\big[\frac{\partial c_t}{\partial u_t} + \gamma \lambda \frac{\partial x_{t+1}}{u_t} \big] \in \mathbb{R}^{m \times 1}$

2. $\Delta w_a^{(1)}$ (Input to Hidden Layer Weights) Actor neural network weights update from input layer to hidden layer can be written in a matrix $\Delta W^{(1)} = [\Delta w_{aji}^{(2)}] \in \mathbb{R}^{N_{h_a} \times n}$.

$$\Delta w_{a_{ji}}^{(1)} = -l_{a_t} \sum_{k=1}^{m} \Big[\{ \frac{\partial c_t}{\partial u_{t_k}} + \gamma \sum_{o=1}^{n} \lambda_o \frac{\partial x_{o_{t+1}}}{u_{t_k}} \} . \frac{\partial u_{t_k}}{\partial v_{a_{t_k}}} \frac{\partial v_{a_{t_k}}}{\partial p_{a_{t_j}}} \Big] \frac{\partial p_{a_{t_j}}}{\partial q_{a_{t_j}}} \frac{\partial q_{a_{t_j}}}{\partial w_{a_{t_j i}}} \tag{5.37}$$

$$\Delta w_{a_{ji}}^{(1)} = -l_{a_t} \sum_{k=1}^{m} \Big[\{ \frac{\partial c_t}{\partial u_{t_k}} + \gamma \sum_{o=1}^{n} \lambda_o \frac{\partial x_{o_{t+1}}}{u_{t_k}} \} . dtansig(v_{a_{t_k}}) . w_{a_{t_k j}} \Big] dtansig(q_{a_{t_j}}) . x_{t_i} \tag{5.38}$$

Using column vectors, entire matrix update $\Delta W_a^{(1)}$ can be written as;

$$\Delta w_a^{(1)} = -l_{a_t} \big[\big[\{ \frac{\partial c_t}{\partial u_{t_k}} + \gamma \lambda^\intercal \frac{\partial x_{t+1}}{u_t} \} \circ dtansig(v_{a_{t_k}}) \big]^\intercal . w_{a_{t_k j}} \big]^\intercal \circ dtansig(q_{a_t})^\intercal . x_t \tag{5.39}$$

5.3 Model

This part of IDHP uses incremental model described in chapter 4 to obtain $\frac{\partial x_t}{\partial x_{t-1}}|_{model}$ and $\frac{\partial x_t}{\partial u_{t-1}}|_{model}$ for critic weight update in Eq.5.13 and x_{t+1} and $\frac{\partial x_{t+1}}{\partial u_t}|_{model}$ in Eq.5.32.

5.4 Algorithm for IDHP

In this section, Algorithms for IDHP methods are given. To give some idea of implementing critic weight update using *forward in time* training, pseudocode for IDHP with SW-OLS is given in Algorithm 1. Similarly pseudocode for IDHP with RLS is given in Algorithm 2 which uses *backward in time* Bellman equation for training critic and *forward in time* Bellman equation to train actor.

Algorithm 1 IDHP with SW-OLS (Forward in Time Critic Weight Update)

Initialization of variables like x_o, γ, Q, R, critic NN, and actor NN
for $i = 1 \rightarrow N$ **do** ▷ Total simulation time=$N \times dt$
  if $i = 1$ **then** ▷ To calculate F_o, G_o
    for $k = 1 \rightarrow M$ **do** ▷ M=Sliding Window Length
      $x_{k+1} \leftarrow$ Excite the system with u_k
    end for
    $\Delta x_1 = \begin{bmatrix} \Delta x_1 \cdots \Delta x_M \end{bmatrix}$ and $\Delta u_1 = \begin{bmatrix} \Delta u_1 \cdots \Delta u_M \end{bmatrix}$
  end if
  F_{t-1} and $G_{t-1} \leftarrow$ System Identification$(\Delta x_i, \Delta u_i)$ ▷ Eq.4.7
  $\hat{x}_{i+1} \leftarrow$ Forward Calculate Model Estimate(x_i, F_{t-1}, G_{t-1}) ▷ Eq.4.6
  $\lambda_t, \hat{\lambda}_{t+1} \leftarrow$ Forward calculate critic$(x_t, w_{c_t}), (\hat{x}_{t+1}, w_{c_t})$
  $u(x_t) \leftarrow$ Forward calculate actor(x_t, w_{a_t}) ▷ $\frac{\partial u_t}{\partial x_t}$ for critic NN weight update
  $w_{c_{t+1}} \leftarrow$ Update critic NN weights$(x_t, \lambda_t, \hat{\lambda}_{t+1}, \frac{\partial c_t}{x_t}, F_{t-1}, G_{t-1})$ ▷ Section 5.1.3
  $\lambda(\hat{x}_{t+1}) \leftarrow$ Forward calculate critic$(\hat{x}_{t+1}, w_{c_{t+1}})$
  $w_{a_{t+1}} \leftarrow$ Update actor NN weights$(x_t, \hat{\lambda}_{t+1}, \frac{\partial c_t}{\partial u_t}, G_{t-1})$ ▷ Section 5.2.2
  $u_t \leftarrow$ Forward calculate actor$(x_t, w_{a_{t+1}})$
  $x_{t+1} \leftarrow$ Simulate real system $(x_t, u_t + u_{PE})$ ▷ Persistent Excitation(PE) for exploration
  $\Delta x_i, \Delta u_i$ Update measurement vectors for identification by SW-OLS
  $l_t \leftarrow$ Update learning rate for actor and critic
end for

Algorithm 2 IDHP with RLS(Backward in Time Critic Weight Update

Initialization of variables like $x_o, \gamma, \Lambda, P_o, \Theta_o, Q, R$, critic NN, and actor NN
for $i = 1 \rightarrow N$ **do** ▷ Total simulation time=$N \times dt$
  if $i = 1$ **then** ▷ To calculate F_o, G_o
    for $k = 1 \rightarrow 2$ **do**
      $x_{k+1} \leftarrow$ Excite the real system with u_k
    end for
    $\psi(t) \leftarrow \Delta x_i = x_2 - x_1$ and $\Delta u_i = u_2 - u_1$
  end if
  $\hat{x}_i \leftarrow$ Forward Calculate Model Estimate$(x_i, \psi(t), \Theta(t-1))$ ▷ Eq.4.11
  $K(t), Q(t)$ Calculate gain and $Q(t)$ matrix ▷ Eq.4.12 and Eq.4.13
  $\Theta(t) \leftarrow$ Update parameter estimate matrix $K(t), \Theta(t-1), e_t$ ▷ Eq.4.10
  $P(t) \leftarrow$ Update co-variance matrix$(P(t-1), \psi(t))$ ▷ Eq.4.14
  $\lambda_{t-1}, \lambda_t \leftarrow$ Forward calculate critic$(x_{t-1}, w_{c_t}), (x_t, w_{c_t})$
  $u_{t-1} \leftarrow$ Forward calculate actor(x_{t-1}, w_{a_t}) ▷ $\frac{\partial u_{t-1}}{\partial x_{t-1}}$ for critic NN weight update
  $w_{c_{t+1}} \leftarrow$ Update critic NN weights$(x_{t-1}, \lambda_t, \lambda_{t-1}, \frac{\partial c_{t-1}}{x_{t-1}}, \Theta(t-1))$ ▷ Section 5.1.3
  $\hat{x}_{i+1} \leftarrow$ Forward Calculate Model Estimate$(x_i, \psi(t), \Theta(t))$ ▷ Eq.4.6
  $\lambda(\hat{x}_{t+1}) \leftarrow$ Forward calculate critic$(\hat{x}_{t+1}, w_{c_{t+1}})$
  $w_{a_{t+1}} \leftarrow$ Update actor NN weights$(x_t, \hat{\lambda}_{t+1}, \frac{\partial c_t}{\partial u_t}, \theta(t))$ ▷ Section 5.2.2
  $u_t \leftarrow$ Forward calculate actor$(x_t, w_{a_{t+1}})$
  $x_{t+1} \leftarrow$ Simulate real system $(x_t, u_t + u_{PE})$ ▷ Persistent Excitation(PE) for exploration
  $\psi(t) \leftarrow \Delta x_i, \Delta u_i$ Update measurement vectors for identification by RLS
  $l_t \leftarrow$ Update learning rate for actor and critic
end for

Chapter 6

Model-Free Action Dependent Dual Heuristic Dynamic Programming

In Action Dependent (AD) form of ACDs, output from the actor u_t is input to the critic along with the states x_t. This results in a direct relationship between cost-to-go function J_t and control action u_t. Because of this, in HDP methods, using AD form can result in a model-free approach with backward in time Bellman equation [14][7]. In DHP designs, using AD form can provide direct derivative of J_t with u_t as critic output. Similar to the motivation for using DHP instead of HDP i.e. direct approximation of the derivatives as an output of NN has higher quality than indirect approximation through backpropgation for multilayer NN, ADHDP is used in this project. Not only does the term $\frac{\partial J_t}{\partial u_t}$ has higher accuracy because of backpropogation through critic network but it also greatly simplifies the actor weight update equations as compared to DHP methods. As mentioned before that for critic weight update equations, system information is required (see 5.12), therefore it is not possible in DHP designs to completely omit the model network unlike ADHDP. In this chapter, two methods are discussed to obtain this information. First one is by using Finite Difference Method (ADDHP-FDM) and second one is by using incremental model (IADDHP) due to its advantages explained in chapter 4 whereas actor and critic in both methods is exactly the same and their equations are given in following sections.

6.1 Critic

Critic in ADDHP takes $y_t = \begin{bmatrix} x_t \\ u_t \end{bmatrix}$ as its inputs and approximates the derivative of J_t with input y_t as its output which is given by Eq.6.1

$$\lambda_t = \frac{\partial J_t}{\partial \hat{y}_t} \tag{6.1}$$

where $\lambda_t = \left[\lambda_{t_{x_1}} \cdots \lambda_{t_{x_n}}, \lambda_{t_{u_1}} \cdots \lambda_{t_{u_m}}\right]^\intercal \in \mathbb{R}^{n+m\times 1}$. Structure of critic NN is same as in IDHP explained in section 5.1.1. Only difference is that now the inputs to the NN also include the control action in addition to states as shown in Fig.6.1.

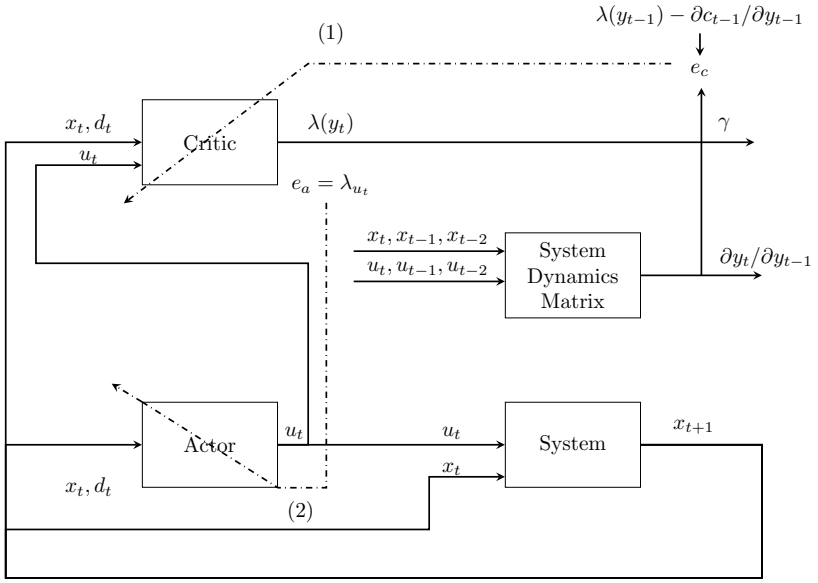

Figure 6.1: Structure of Model-free Action Dependent Dual Heuristic Programming

6.1.1 Critic NN Weight Update

For critic's training, backward in time prediction error equation is used stated in Eq.5.11 for DHP designs. For ADDHP, critic weight update equations can be formulated as following.

$$e_{c_t} = \frac{\partial[J_{t-1} - \{c_{t_1} + \gamma J_t\}]}{\partial y_{t-1}} \tag{6.2}$$

$$e_{c_t} = \lambda_{t-1} - \{\frac{\partial c_{t-1}}{\partial y_{t-1}} + \gamma \frac{\partial y_t}{\partial y_{t-1}} \lambda_t\} \tag{6.3}$$

The term $\frac{\partial y_t}{\partial y_{t-1}} \in \mathbb{R}^{(n+m)\times(n+m)}$ contains the system's dynamics information and is discussed in section 6.3 whereas, the derivative of local cost function at previous time (c_{t-1}) with y_{t-1} is given in Eq.6.4

$$\frac{\partial c_{t-1}}{\partial y_{t-1}} = \begin{bmatrix} \frac{\partial c_{t-1}}{\partial x_{t-1}} \\ \frac{\partial c_{t-1}}{\partial u_{t-1}} \end{bmatrix} \tag{6.4}$$

where values of $\frac{\partial c_t}{\partial x_t}$ and $\frac{\partial c_t}{\partial u_t}$ are given in Eq.5.15 and Eq.5.33 respectively.

During the formulation of critic NN weights update equations in section 5.1.3, a generalized case for any input was discussed therefore same equation can be used for ADDHP

with vector y_t, which is concatenation of x_t and u_t, as input instead of just x_t. Here, the layer weights are summarized.

1. $\Delta w_c^{(2)}$ (Hidden to Output Layer Weights)
   Weight update equation from critic's hidden layer to output layer can be written as a matrix $\Delta w_c^{(2)} = [\Delta w_{c_{kj}}^{(2)}] \in \mathbb{R}^{n+m \times N_{h_c}}$

$$\Delta w_{c_{kj}}^{(2)} = -l_{c_t}.e_{c_{t_k}}.p_{c_{t-1_j}} \tag{6.5}$$

   Using column vectors for e_{c_t} and $p_{c_{t-1_j}}$, Eq.6.5 becomes

$$\Delta w_c^{(2)} = -l_{c_t}.e_{c_t}.p_{c_{t-1}}^{\intercal} \tag{6.6}$$

2. $\Delta w_c^{(1)}$ (Input to Hidden Layer Weights) Critic neural network's weight update from input layer to hidden layer neuron can be written in a matrix $\Delta w_c^{(1)} = [\Delta w_{c_{ji}}^{(1)}] \in \mathbb{R}^{N_{h_c} \times n+m}$

$$\Delta w_{c_{ji}}^{(1)} = -l_{c_t} \sum_{k=1}^{n} e_{c_{t_k}}.w_{ckj}.dtansig(q_{c_{t-1_j}}).y_{t-1_i} \tag{6.7}$$

   Using column vectors, entire matrix update $\Delta w_c^{(1)}$ can be written as;

$$\Delta w_c^{(1)} = -l_{c_t}\left[e_{c_t}^{\intercal}.w_c\right]^{\intercal} \circ \left[dtansig(q_{c_{t-1}}).y_{t-1}^{\intercal}\right] \tag{6.8}$$

6.2 Actor

Similar to the actor described in section 5.2, actor NN takes states as inputs and produces the near-optimal control actions. In ADDHP methods, these control actions are input to the critic and the real system.

6.2.1 Actor NN weight Update

Actor NN structure is same as in DHP and can be described by Eq.5.25 to Eq.5.28. Unlike the DHP structure, its AD form has a direct relationship between λ_t and u_t which simplifies the actor weight update equations. Actor NN is trained to minimize following equation;

$$\|E_{a_t}\| = \frac{1}{2} e_{a_t}^{\intercal} e_{a_t} \tag{6.9}$$

Using Eq.5.30 as target for actor weight update, we can write prediction error as Eq.6.10.

$$e_{a_t} = \lambda_{u_t} \tag{6.10}$$

Using stochastic gradient, weight update for actor can be written as Eq.6.11

$$\Delta w_{a_t} = l_{a_t}\left[\lambda_{u_t}\right] \frac{\partial u_t}{\partial w_{a_t}} \tag{6.11}$$

where $\frac{\partial u_t}{\partial w_{a_t}}$ is calculated using chain rule and NN structure for $\Delta w_a^{(1)}$ and $\Delta w_a^{(2)}$ same as in section 5.2.2 because of unchanged NN structure.

1. $\Delta w_a^{(2)}$ (Hidden to Output Layer Weights)
   Using Eq.6.11, weights update equations from hidden layer to output layer for actor can be written as a matrix $\Delta w_a^{(2)} = [\Delta w_{a_{kj}}^{(2)}] \in \mathbb{R}^{m \times N_{h_c}}$.

$$\Delta w_{a_{kj}}^{(2)} = -l_{a_t} \left[\lambda_{u_{t_k}}\right] . \frac{\partial u_{t_k}}{\partial v_{a_{t_k}}} \frac{\partial v_{a_{t_k}}}{\partial w_{a_{t_k}j}} \tag{6.12}$$

$$\Delta w_{a_{kj}}^{(2)} = -l_{a_t} . \left[\lambda_{u_{t_k}}\right] . dtansig(v_{a_{t_k}}) . p_{a_{t_j}} \tag{6.13}$$

   Eq.6.13 can be rewritten as Eq.6.14 using column vectors for $\frac{\partial c_t}{\partial u_t}, v_{a_t}$, and p_{a_t}.

$$\Delta w_a^{(2)} - l_{a_t} . \left[\lambda_{u_t}\right] \circ dtansig(v_{a_t})^\intercal . p_{a_t} \tag{6.14}$$

2. $\Delta w_a^{(1)}$ (Input to Hidden Layer Weights) Actor neural network weights update from input layer to hidden layer can be written in a matrix $\Delta w_a^{(1)} = [\Delta w_{a_{ji}}^{(2)}] \in \mathbb{R}^{N_{h_a} \times n}$.

$$\Delta w_{a_{ji}}^{(1)} = -l_{a_t} \sum_{k=1}^{m} \left[\{\lambda_{u_{t_k}}\} . \frac{\partial u_{t_k}}{\partial v_{a_{t_k}}} \frac{\partial v_{a_{t_k}}}{\partial p_{a_{t_j}}}\right] . \frac{\partial p_{a_{t_j}}}{\partial q_{a_{t_j}}} . \frac{\partial q_{a_{t_j}}}{\partial w_{a_{t_j}i}} \tag{6.15}$$

$$\Delta w_{a_{ji}}^{(1)} = -l_{a_t} \sum_{k=1}^{m} \left[\{\lambda_{u_{t_k}}\} . dtansig(v_{a_{t_k}}) . w_{a_{t_k}j}\right] . dtansig(q_{a_{t_j}}) . x_{t_i} \tag{6.16}$$

   Using column vectors, entire matrix update $\Delta w_a^{(1)}$ can be written as;

$$\Delta w_a^{(1)} = -l_{a_t} \left[\left[\{\lambda_{u_t}\} \circ dtansig(v_{a_{t_k}})\right]^\intercal . w_{a_{t_kj}}\right]^\intercal \circ dtansig(q_{a_t})^\intercal . x_t \tag{6.17}$$

6.3 System's Dynamic

As mentioned before, the term $\frac{\partial y_t}{\partial y_{t-1}}$ in Eq.6.3 captures the system dynamics of the system. This can be written as;

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} \frac{\partial x_t}{\partial x_{t-1}} & \frac{\partial u_t}{\partial x_{t-1}} \\ \frac{\partial x_t}{\partial u_{t-1}} & \frac{\partial u_t}{\partial u_{t-1}} \end{bmatrix} \tag{6.18}$$

In classical methods, this information is obtained through the Model NN. In the *Model-free* approaches, alternate way is required to obtain this information. In this report, two methods are investigated to obtain $\frac{\partial y_t}{\partial y_{t-1}}$. These methods are *'Finite Difference Method (FDM)'* and *'Incremental Model'*.

6.3.1 Finite Difference Method

This method is proposed by Zhen et al. in [20]. Instead of using Model NN, they suggested using the Finite Difference Method (FDM) to find the derivatives required in Eq.6.18 to make it *model-free*. Using Taylor Series expansion, *'first-order backward difference'* derivative can be written as

$$\frac{\partial q_t}{\partial t} \simeq \frac{q(t) - q(t - \Delta t)}{\Delta t} + \mathcal{O}(\Delta t) \tag{6.19}$$

where $\mathcal{O}(\Delta t)$ is the truncation error associated with using *first-order* approximation. This implies that the accuracy of the approximation is proportional to the discretization step. Therefore, data should be taken at high sampling rate for better quality of approximation. Using chain rule and FDM (Eq.6.19), derivatives in Eq.6.18 can be written as,

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} \frac{x_t - x_{t-1}}{x_{t-1} - x_{t-2}} & \frac{u_t - u_{t-1}}{x_{t-1} - x_{t-2}} \\ \frac{x_t - x_{t-1}}{u_{t-1} - u_{t-2}} & \frac{u_t - u_{t-1}}{u_{t-1} - u_{t-2}} \end{bmatrix} \tag{6.20}$$

where $x_t, x_{t-1} \in \mathbb{R}^{n\times 1}$ and $u_t, u_{t-1} \in \mathbb{R}^{m\times 1}$ are column vectors. FDM method described above for online system identification is suitable for small noise-free systems with rapidly changing dynamics [32]. Therefore, system excitation is of paramount importance to avoid divergence.

6.3.2 Incremental Model

Another *'model-free'* approach to get $\frac{\partial y_t}{\partial y_{t-1}}$ is by using incremental model as explain in section 4.2. Nonlinear model can be linearized around a point given a high sampling rate and can be written as Eq.4.5. Using $F_{t-1} \simeq \frac{\partial x_{t+1}}{\partial x_t}|_{model}$, $G_{t-1} \simeq \frac{\partial x_{t+1}}{\partial u_t}$, and actor NN; Eq.6.18 can be written as;

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} \frac{\partial x_t}{\partial x_{t-1}}|_{model} + \frac{\partial x_t}{\partial u_{t-1}}\frac{\partial u_{t-1}}{\partial x_{t-1}}|_{model+actor} & \frac{\partial u_t}{\partial x_t}\frac{\partial x_t}{\partial x_{t-1}}|_{actor+model} \\ \frac{\partial x_t}{\partial u_{t-1}}|_{model} & \frac{\partial u_t}{\partial x_t}\frac{\partial x_t}{\partial u_{t-1}}|_{actor+model} \end{bmatrix} \tag{6.21}$$

$$\frac{\partial y_t}{\partial y_{t-1}} = \begin{bmatrix} F_{t-2} + G_{t-2}\frac{\partial u_{t-1}}{\partial x_{t-1}}^\intercal & \frac{\partial u_t}{\partial x_t}{F_{t-2}}^\intercal \\ {G_{t-2}}^\intercal & \frac{\partial u_t}{\partial x_t}{G_{t-2}}^\intercal \end{bmatrix} \tag{6.22}$$

where F_{t-2}, and G_{t-2} is identified online by using SW-OLS or RLS at $t-1$.

6.4 Algorithm for ADDHP

Pseudocode for ADDHP methods is given in Algorithm 3. These steps are followed for both methods that are IADDHP and ADDHP-FDM.

Algorithm 3 *Model-free* ADDHP

Initialization of variables like x_o, γ, critic NN, and actor NN and other variables for RLS or FDM
for $i = 1 \rightarrow N$ **do** ▷ Total simulation time=$N \times dt$
  if $i = 1$ **then**
    $\frac{\partial y_t}{\partial y_{t-1}} \leftarrow$ Calculate the initial system dynamics matrix (Using FDM or RLS)
  end if
  $u_t, u_{t-1} \leftarrow$ Forward calculate actor $(x_t, w_{a_t}), (x_{t-1}, w_{a_t})$
  $\lambda_t, \lambda_{t-1} \leftarrow$ Forward calculate critic $(y_t = [x_t u_t], w_{a_t}), (y_{t-1}, w_{a_t})$
  $w_{c_{t+1}} \leftarrow$ Update critic NN weights$(y_{t-1}, \lambda_t, \lambda_{t-1}, \frac{\partial c_{t-1}}{y_{t-1}}, \frac{\partial y_t}{\partial y_{t-1}})$ ▷ Section 6.1
  $\lambda(x_t) \leftarrow$ Forward calculate critic$(\hat{y}_t, w_{c_{t+1}})$
  $w_{a_{t+1}} \leftarrow$ Update actor NN weights(x_t, λ_{u_t}) ▷ Section 6.2
  $u_t \leftarrow$ Forward calculate actor$(x_t, w_{a_{t+1}})$
  $x_{t+1} \leftarrow$ Simulate real system $(x_t, u_t + u_{PE})$ ▷ Persistent Excitation(PE) for exploration
  $l_t \leftarrow$ Update learning rate for actor and critic
  $\frac{\partial y_t}{\partial y_{t-1}} \leftarrow$ Update system dynamics matrix (Using FDM or RLS)
end for

Chapter 7

Flight Control Simulation

In this chapter, experimental or simulation setup for the application of aforementioned ACD algorithms for missile reference tracking control is discussed. Also, different experiments are designed to identify and investigate various parameters to find the optimal controller setting. Performance criteria is established to compare the results of different algorithms.

7.1 Missile Model

To test the applicability of the *online ACD controllers* for flight control, a simple missile model is used to simulate short period dynamics of the air vehicle. This nonlinear model for pitch plane can be written as:

$$\dot{\alpha} = q + \frac{\bar{q}Sg}{m_a V_T} C_z(\alpha, q, M_a, \delta_e) \tag{7.1}$$

$$\dot{q} = \frac{\bar{q}Sd_l}{I_{yy}} C_m(\alpha, q, M_a, \delta_e) \tag{7.2}$$

where angle of attack (α) and pitch rate (q) are the states to be controlled by changing control fin deflection angle (δ_e). System is initialized with $\alpha_o = 5.7\,\text{deg}$ and $q = 0\ \frac{rad}{sec}$.

$$C_z(\alpha, q, M_a, \delta_e) = C_{z_1}(\alpha, M_a) + B_z \delta_e \tag{7.3}$$

$$C_m(\alpha, q, M_a, \delta_e) = C_{m_1}(\alpha, M_a) + B_m \delta_e, \tag{7.4}$$

$$B_z = b_1 M_a + b_2, \tag{7.5}$$

$$B_m = b_3 M_a + b_4, \tag{7.6}$$

$$C_{z_1}(\alpha, M_a) = \phi_{z_1}(\alpha) + \phi_{z_2} M_a, \tag{7.7}$$

$$C_{z_2}(\alpha, M_a) = \phi_{m_1}(\alpha) + \phi_{m_2} M_a, \tag{7.8}$$

$$\phi_{z_1}(\alpha) = h_1\alpha^3 + h_2\alpha|\alpha| + h_3\alpha, \tag{7.9}$$

$$\phi_{m_1}(\alpha) = h_4\alpha^3 + h_5\alpha|\alpha| + h_6\alpha, \tag{7.10}$$

$$\phi_{z_2} = h_7\alpha|\alpha| + h_8\alpha, \tag{7.11}$$

$$\phi_{m_2} = h_9\alpha|\alpha| + h_{10}\alpha, \tag{7.12}$$

Aerodynamic coefficient and other parameters used for simulation of missile model are given in Table 7.1 and are valid for $-10\,\text{deg} < \alpha < 10\,\text{deg}$ [33].

Table 7.1: Aerodynamic parameters for Missile Model

Parameter	Value	Parameter	Value	Parameter	Value
$\bar{q}$	6132.8 $\frac{lb}{ft^2}$	S	$0.44ft^2$	m_a	450 lb
V_T	3109.3 $\frac{ft}{sec}$	d_l	0.75 ft	I_{yy}	182.5 $\frac{slug}{ft^2}$
M_a	2	b_1	1.6238	b_2	-6.7240
b_3	12.0393	b_4	-48.2246	h_1	-288.7
h_2	50.32	h_3	-23.89	h_4	303.1
h_5	-246.3	h_6	-37.56	h_7	-13.53
h_8	4.185	h_9	71.51	h_{10}	10.01

The purpose of this model is to simulate the real system with the sampling frequency of 100Hz. As the controllers discussed in this project are *'model-free'*, therefore no direct information from this model is used in the controllers.

7.2 Experiments Setup

In this section, various experiments are formulated to answer different questions by varying different parameters. To study the effects of only desired parameters; all the controllers i.e. IDHP, IADDHP, and ADDHP-FDM have following things that are constant unless mentioned otherwise.

Reference tracking control problem is used to analyze the performance of controllers given in Equation 5.10 with $Q = \begin{bmatrix} 500 & 0 \\ 0 & 0.01 \end{bmatrix}$ and $R = \begin{bmatrix} 3 \end{bmatrix}$. Whereas, a sinusoidal wave of 10

deg amplitude is used as reference signal for α. Reference signal (d_t) can be written as;

$$d_t = \begin{bmatrix} \frac{10\pi}{180}\sin(0.05t) \\ 0 \end{bmatrix} \tag{7.13}$$

For actor and critic NN, one hidden layer with three neurons is used with all the initial NN weights randomly initialized in the range of (-0.5,0.5). Forgetting factor (γ) in Bellman equation is set to 0.7. Another important parameter in critic and actor NN weight update is the learning rate (l_r). If a constant (l_r) is used then a low (l_r) will take too long to reach the optima whereas a high (l_r) can overshoot the optimal point and can cause instability. For this project, (l_r) is *annealed* slowly with time to avoid instability. This is, after 50 time steps, l_r is given as;

$$l_{a_t} = l_{c_t} = bl_o \geq 0.05 \tag{7.14}$$

where $b = 0.95$ and l_o represents the initial learning rate which will be varied.

7.2.1 Experiment A: SW-OLS vs RLS

This experiment relates to the *Incremental Model* part of ACD. Two methods explained in section 4.2 for system identification and state prediction for incremental form of nonlinear system are implemented and compared. Following questions needs to be answered during this experiment.

1. Is the controller capable of performing given task? Can it stabilize the system?
2. Which controller has better performance in terms of reference tracking?
3. Which method is easier to implement?

Sliding window used for OLS has the length $M = 2(n + m)$ which in the missile case is 6. For RLS, $\Lambda = 0.9$, and $P = 1000$ is used. $\theta(t)$ is initialized with $F_0 = I$ and $G_0 = O$. Both of these methods are implemented on IDHP controller.

7.2.2 Experiment B: IDHP vs IADDHP

In this section, IDHP and its Action Dependent form (IADDHP) is compared. Both methods are implemented with same parameters and weight update equations for critic. Weight update equation is different for actor as mention in section 6.2. These 2 controllers are compared based on various categories. One of the most important category is the success rate. As we are using random initial weights, therefore we would like to know how often does these controllers converge to an *near-optimal* policy. Experiment is designed to answer following questions.

1. Can the system be stabilized by the controllers? Are the controllers able to produce acceptable results?
2. What is the success rate of these controller?

3. How well do they follow the given task i.e. which has minimum average local cost value?
4. What is the computational time required by the two controllers?
5. What is the average time to reach a stabilizing policy?

Local cost is a good measure of performance as it includes both the tracking error and control action required. To determine the success rate, a criteria is defined. If after 20 secs, the absolute error between required and desired state is greater than 1.5 deg then the trial is considered unsuccessful. System is simulated 100 times with different initialized weights. Each trial is independent and does not learn from previous experiences. The system is also simulated with different leaning rates.

7.2.3 Experiment C: IADDHP vs ADDHP-FDM

In this experiment, two forms of *model-free* ADDHP are compared. First one is the incremental model based ADDHP and second one is ADDHP with finite difference as described in section 6.3. To compare both controllers without any bias, both systems are excited with different persistent excitation which gives optimal performance for that controller. Also different learning rates are investigated to find best learning rate for the controller. In addition to the questions in section 7.2.2, following questions will be answered by this experiment.

1. How does a controller performs under noise?
2. Can the controller inherently detects system failure?
3. Can controller adapt to failures in system dynamics?

For these controllers, no internal training cycles are used. As using the internal training cycles will improve the results in both cases of IADDHP and ADDHP-FDM. Also, for ADDHP-FDM, a limit is set on $\frac{\partial y_t}{\partial y_{t-1}}$ matrix as it tends to diverge very easily. This limit on system dynamics changes the controller performance at different learning rates. After trying different limits, $\frac{\partial y_t}{\partial y_{t-1}} \leq 5$ is used that gave optimal performance while giving controller freedom to identify the dynamics.

To check the performance of both controller under noise, a uniform Gaussian noise is added to sensor readings i.e. $x_t = x_t + \rho$ where ρ is the uniformly distributed noise.For IADDHP, noise with standard deviation $(\sigma) = \begin{bmatrix} 0.01 & 0.05 \end{bmatrix}$ is added to the α in degrees whereas half of this is added to q. For ADDHP-FDM, σ is lowered till the controller is able to let the noise pass through it without getting unstable.

To introduce the failure in the system, at half the simulation time, system dynamics is changed by changing the sign of coefficient C_z and b_2. In IADDHP, as we are using RLS, failure can be detected with the help of innovation $(e(t))$ given by Eq.4.9. If $e(t)$ is greater than 1.5 deg then a failure is detected and NN weights for actor and critic are reset to random initial condition. In ADDHP-FDM, there is no way to detect the system failure as it is only capable of system identification and not the state prediction. All these scenarios are investigate by 100 independent runs with random initial weights.

Chapter 8

Results & Discussion

In this chapter, results of the experiments set up in chapter 7 are discussed. Main results are extracted and shown in this chapter.

8.1 Experiment A: SW-OLS vs RLS

In this experiment, IDHP controller was implemented with SW-OLS and RLS with the same parameters to see the performance of each of the system identification methods. One of the successful run is shown in Fig.8.1. As it can be seen that both SW-OLS and RLS are able to perform tracking task with little difference in the performance. But if we look at the first ten seconds of the simulation to see the adaptation behavior of the controllers (Fig.8.2), it can be seen that IDHP with SW-OLS has noisier control policy in the start. Apart from that, SW-OLS could easily become ill-conditioned or even unstable due to inverse of the matrix involved in calculation of system dynamics. RLS also provides more control to the system identified. For instance, in RLS, Λ can be tuned to allow more or less noise and innovation (e_t) which is inherent in RLS update equations can be used to detect failure in system dynamics. SW-OLS also requires more data points than RLS which increase with the number of unknowns, for initialization of the method.

For the reasons mentioned above,RLS is adopted for system identification of incremental models for rest of the experiments even though SW-OLS is easier to implement. It is also possible that the performance of SW-OLS and RLS in terms of reference tracking is comparable because of very simple nonlinear model being used. We can expect to see more difference when highly nonlinear model is used.

8.2 Experiment B: IDHP vs IADDHP

In this experiment, performance of the IDHP and IADDHP controller with RLS for incremental model system identification is discussed. It can be seen in Fig.8.3 that the

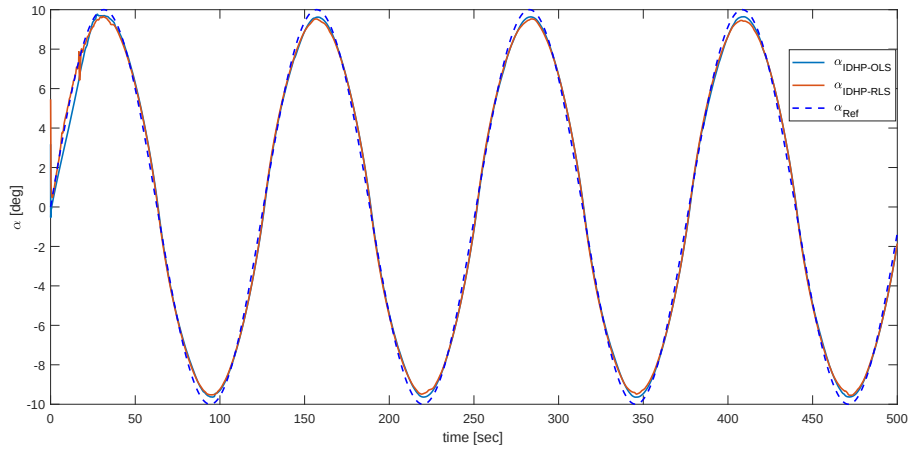

Figure 8.1: α Reference tracking using IDHP controller with SW-OLS and RLS

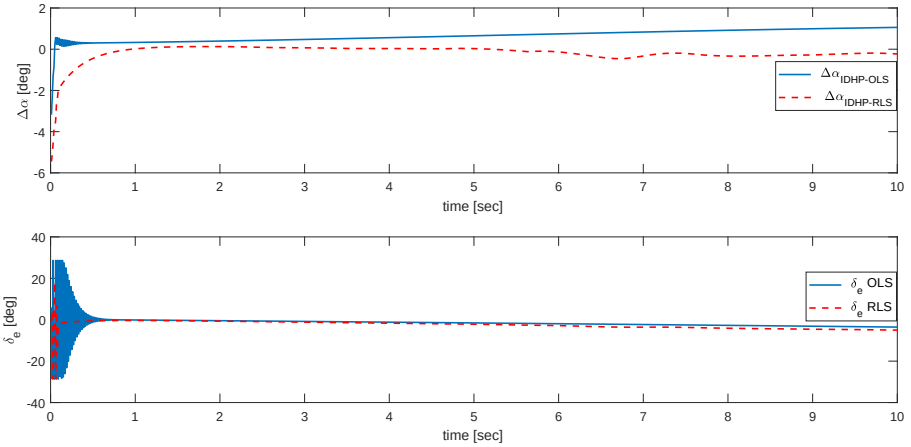

Figure 8.2: Tracking error for α state and control fin deflection(δ_e) using IDHP controller for initial 10 secs

success rate for IADDHP is higher than IDHP for higher l_r whereas for very low l_r IDHP has better success rate. But looking at the optimal l_r(maximum success rate), it can be seen that the success rate for IADDHP(93%) is higher than IDHP(58%) with $l_r = 13$ and $l_r = 10$ respectively.

Other performance criteria for optimal l_r are also given in Table.8.1. Average control effort required and average computation time for one run ($\bar{t}_{computation}$) is same for both IDHP and IADDHP. Average time to settle ($\bar{t}_{settling}$) is lower for IADDHP. These averages are computed for the successful runs only.

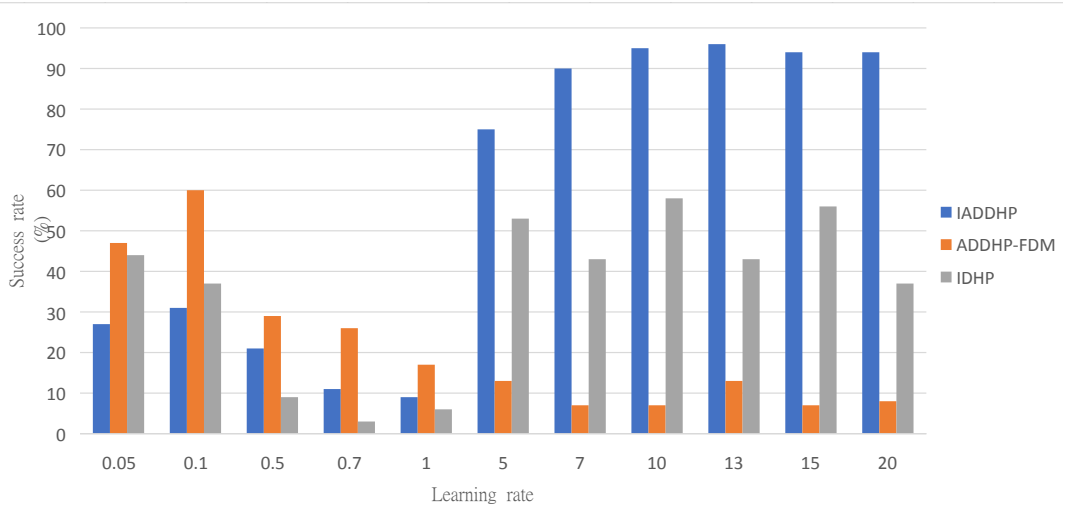

Figure 8.3: Success rate of *model-free* controllers with various learning rates

8.3 Experiment C: IADDHP vs ADDHP-FDM

This is the main experiment of this report. In this experiment, two Action Dependent controllers are compared. First one is the incremental model based ADDHP which uses RLS to find the system dynamics matrix described in section 6.3. Other method is using a very simple approach of Finite Difference Method to calculate the system dynamics matrix. Three different scenarios are investigated.

8.3.1 Without Noise or Failure

In this experiment, IADDHP and ADDHP-FDM are simulated for 100 independent for various l_r to identify the best l_r for each method as to compare these methods fairly. It can be seen in Fig.8.3 that ADDHP-FDM works better when the l_r is low. Whereas, IADDHP-FDM performs better at higher l_r. This can be explained by observing how the weights are being updated. As in FDM, even small changes can lead to a very big $\frac{\partial y_t}{\partial y_{t-1}}$ matrix. Therefore using a higher l_r can make the wights very big, very fast. Therefore small l_r are a safer option with highest success rates. Whereas in RLS for system identification, previous values are averaged. This results in convergence of $\frac{\partial y_t}{\partial y_{t-1}}$. A higher learning rate is required for quick convergence to optimal policy. For $l_r = 0.1$, ADDHP-FDM has maximum success rate of 60% whereas for IADDHP it is 93% at $l_r = 13$. It can also be seen that after $l_r = 10$, there is little variation in success rate of IADDHP.

To understand the reason behind the divergence of FDM methods, temporal progression of $\frac{\partial y_t}{\partial y_{t-1}}$ was observed for various randomly initialized unsuccessful runs. It was observed that the u_t for all these runs become saturated at some point in time which leads to ill-condition $\frac{\partial y_t}{\partial y_{t-1}}$ matrix and eventually the divergence of the system output. This is shown in Fig.8.4 for four randomly initialized unsuccessful runs for initial iterations that lead to unbounded output.

The saturation occurs because we have imposed a limit on the control action of 0.5 [rad] to avoid large control fin deflections. Because of this saturation, $\frac{\partial x_t}{\partial u_{t-1}}$ and $\frac{\partial u_t}{\partial u_{t-1}}$ term

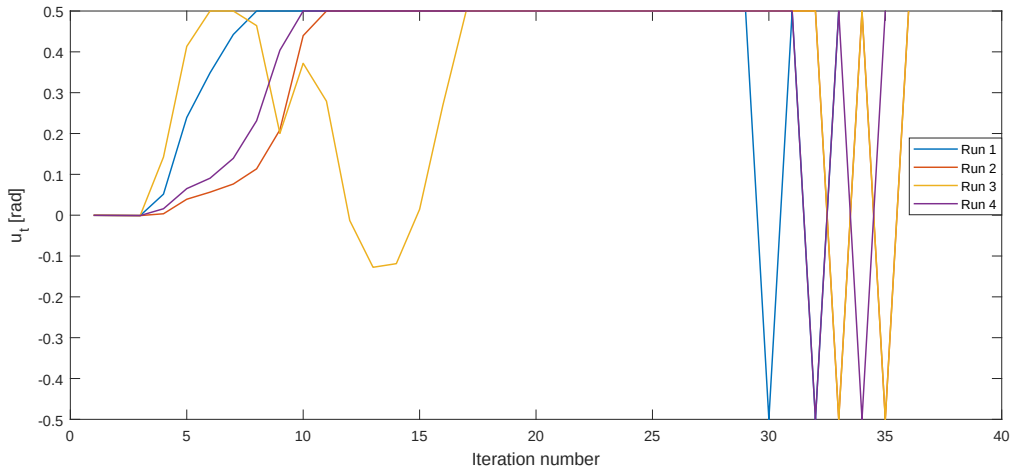

Figure 8.4: Control input for unsuccessful runs

in Eq.6.18 goes to infinity and undefined variable respectively. To compare this with IADDHP, we have used the same unsuccessful run NN weights for actor and critic using IDHP. IADDHP follows the task without any instability whereas ADDHP-FDM fails to do so. Figure 8.5 compares the control policies of the unsuccessful ADDHP-FDM with successful IADDHP. It can be observed that the control policy using FDM is quite aggressive compared to the control policy using RLS. This could be because of the higher excitation signal required for FDM methods.

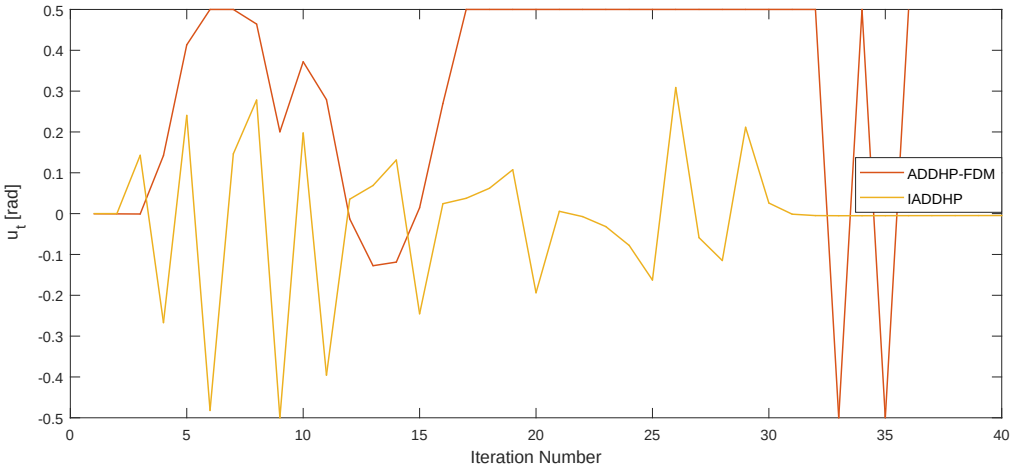

Figure 8.5: Successful IADDHP vs unsuccessful ADDHP-FDM using same initial weights

Table 8.1 shows average values of different performance measure for successful runs for optimal learning rates. It can be seen that the ADDHP-FDM requires less computation time due to its simplicity in calculating the system dynamics matrix. But its behavior is not uniform throughout the runs as it can be seen with $\bar{t}_{settling}$. It also requires higher average control effort for almost same average local cost. Figure 8.6 shows one successful response in this scenerio for ADDHP-FDM and IADDHP.

Table 8.1: Comparison for *model-free* controllers for optimal learning rate

Controller Name	l_r	$\bar{c}_t$	$\bar{u}_t[deg]$	$\bar{t}_{computation}[sec]$	$\bar{t}_{settling}[sec]$
IDHP-RLS	10	0.025	6.6	17	0.27
IADDHP	13	0.031	6.8	17.5	0.12
ADDHP-FDM	0.1	0.03	7.3	14.5	1.41

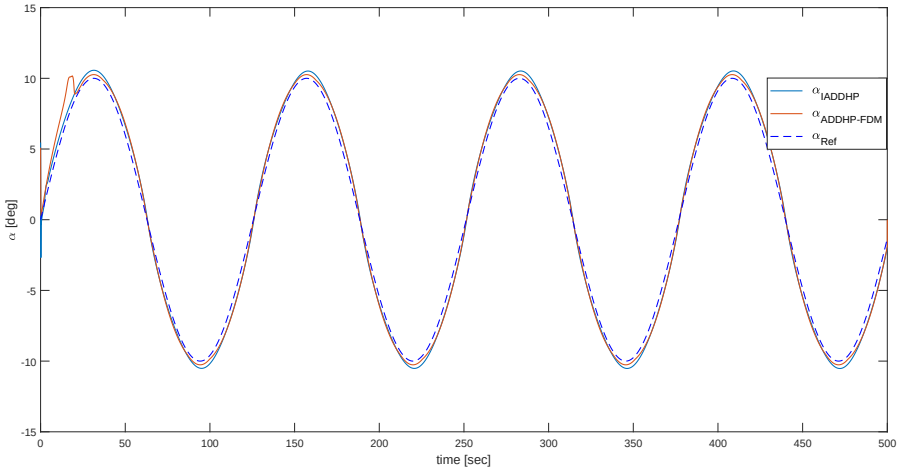

Figure 8.6: α Reference tracking for *model-free* ADDHP controllers

8.3.2 With Noise

In this section, effect of noise on AD controllers is investigated. For IADDHP with RLS, uninformly distributed Gaussian noise with standard deviation (σ) of 0.01 and 0.05 is applied to α in deg and half of this is applied to q. $l_r = 13$ corresponding to the optimal l_r in previous section is used. Table 8.2 shows the success rate for these 2 noise settings. For $\sigma = 0.01$, success rate has increased to 97% from 93% with no noise whereas for $\sigma = 0.05$, success rate has reduced to 62%. The increase in success rate when noise is applied is counter intuitive but this is because of the excitation of the system which enables the controller to explore.

Table 8.2: Performance of IADDHP with noise

σ	Success rate percentage
0.01	97%
0.05	62%

For ADDHP-FDM, σ for noise is decreased till the controller allowed the noise to pass. The result of this is summarized in Table 8.3. It turns out that the ADDHP-FDM allows noise with 40 times lower σ than IADDHP with RLS. This result is consistent with the findings in literature which states that the FDM does not perform well with noisy

measurements whereas RLS can be thought of as a noise filter therefore its performance is superior under noisy measurements.[32][34].

Table 8.3: Performance of ADDHP-FDM with noise

σ	Success rate percentage
0.0002	53%
0.001	14%

8.3.3 With Failures

One of the properties of controllers using RL is their ability to learn with the changing dynamics. In this experiment, failures are introduced by suddenly changing the dynamics of the simulated system halfway through the simulation by changing signs of coefficients C_z and b_2 explained in section 7.1. No information about the failure is given to controller which means that controller must be able to detect these changes and reset accordingly. This failure identification is done by innovation term (e_t) in RLS of IADDHP. Figure 8.7 and 8.8 show that around 250 secs,, controller identify the failure and quickly adapts to the change in signs of C_z and b_2 by returning to optimal policy.

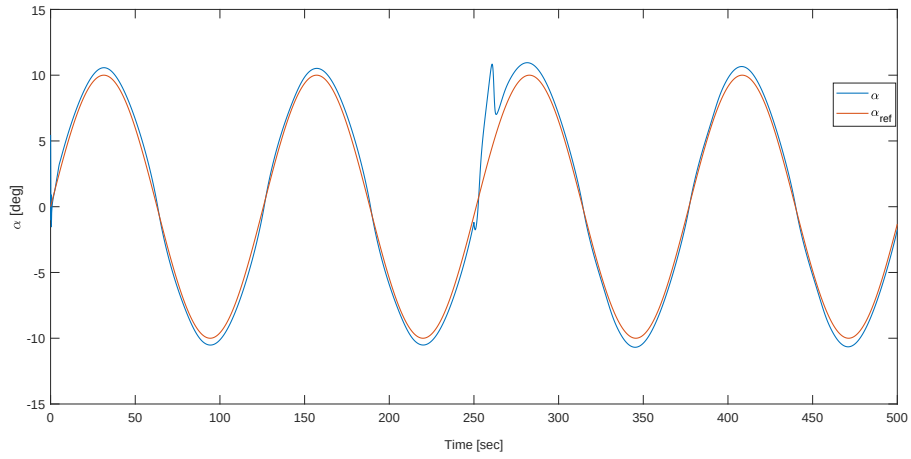

Figure 8.7: Reference tracking with IADDHP controller with sudden sign change in C_z

As in ADDHP-FDM, there is no mechanism to predict the states i.e. identify if the identified model matches with the *'real system'*, there is no way to identify the failure for adaptation.

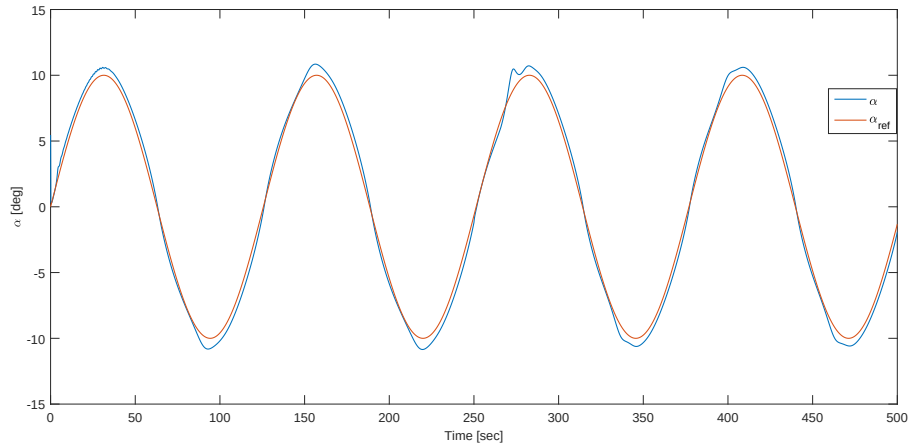

Figure 8.8: Reference tracking with IADDHP controller with sudden sign change in b_2

Chapter 9

Conclusion and Recommendations

The purpose of this report is to investigate different online controllers which does not require any system information beforehand and to learn about their applicability in various scenarios using numerical simulation. Based on the experiments carried out, in this chapter few concluding remarks are given along with the few recommendations for future work.

9.1 Conclusion

Experiment A was designed to investigate two online system identification methods. From the experiment, it is concluded that despite its complexity, RLS is better than SW-OLS for online system identification in ACD controllers because of its fast convergence which is required to reach optimal control as soon as possible specially in case of failures. It also does not suffer from ill-conditioning due to matrix inverse.

Experiment B was designed to compare *model-free* IDHP controller with its action dependent form. Initial experiments have concluded that IADDHP outperform IDHP with same parameters. IADDHP is also easier to implement than IDHP. This may be due to the fact that direct derivative is being obtained from critic NN that is being used for for actor weight update instead of indirect backpropogation. This not only results in more accurate derivative but it also ties the convergence of critic network with actor network. Therefore at this point, IADDHP can be considered ultimate *model-free* controller in terms of performance and computational complexity.

Experiment C was designed to compare already existing *model-free* ADDHP control in literature proposed by Zhen et al. using FDM. From the experiments, it was found that that incremental model based ADDHP can outperform ADDHP-FDM controller based on success rate, performance under different noise conditions and ability to adapt to system dynamics changes. Also ADDHP-FDM controller is susceptible to divergence for fast changing dynamics and unable to detect failure. Although in this project, no internal training cycles are used as they were used in the original work, it is our argument that

the use of internal training cycles will improve the performance of both controllers that is ADDHP-FDM and IADDHP. Therefore it is concluded that using RLS for system identification in *online model-free* ADDHP is better solution than using FDM despite its simplicity.

9.2 Recommendations

One of the most important thing in these online controller with high risk tasks like flight control is the assurance of 100% success rate under all conditions. Hence to improve this, during the implementation of these online controllers, there are various parameters that were identified for further investigation. Few of them are;

- Use of recurrent neural network for actor due to their dynamic temporal behavior.
- Using local adaptive learning rates algorithms like adaGrad for better and fast convergence of NN weights for online training.
- Using Radial Basis Function NN instead of current FeedForward NN due to their ability to have better local approximation.
- Using different weight update scheme like Risilient backpropogation for faster learning.

Apart from this, these newly developed methods (IDHP and IADDHP) need to be applied to complex and more nonlinear environment to check the applicability and limitations of these online controllers.

Bibliography

[1] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[2] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[4] D. Liu, Q. Wei, D. Wang, X. Yang, and H. Li, *Adaptive Dynamic Programming with Applications in Optimal Control*. Springer, 2017.

[5] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems Yearbook*, vol. 22, no. 12, pp. 25–38, 1977.

[6] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: an overview," in *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, vol. 1. IEEE, 1995, pp. 560–564.

[7] R. Enns and J. Si, "Helicopter trimming and tracking control using direct neural dynamic programming," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 929–939, 2003.

[8] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010, vol. 39.

[9] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[10] S. Ferrari and R. F. Stengel, "Online adaptive critic flight control," *Journal of Guidance Control and Dynamics*, vol. 27, no. 5, pp. 777–786, 2004.

[11] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems Yearbook*, vol. 22, no. 12, pp. 25–38, 1977.

[12] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367–1372, 1995.

[13] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 764–773, 2002.

[14] J. Si and Y.-T. Wang, "Online learning control by association and reinforcement," *IEEE Transactions on Neural networks*, vol. 12, no. 2, pp. 264–276, 2001.

[15] S. Balakrishnan and V. Biega, "Adaptive-critic-based neural networks for aircraft optimal control," *Journal of Guidance, Control, and Dynamics*, vol. 19, no. 4, pp. 893–898, 1996.

[16] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," in *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 2. IEEE, 2001, pp. 990–995.

[17] Z. Huang, J. Ma, and H. Huang, "An approximate dynamic programming method for multi-input multi-output nonlinear system," *Optimal Control Applications and Methods*, vol. 34, no. 1, pp. 80–95, 2013.

[18] D. Liu, H. Javaherian, O. Kovalenko, and T. Huang, "Adaptive critic learning techniques for engine torque and air–fuel ratio control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 988–993, 2008.

[19] E. Van Kampen, Q. Chu, and J. Mulder, "Online adaptive critic flight control using approximated plant dynamics," in *Machine Learning and Cybernetics, 2006 International Conference on*. IEEE, 2006, pp. 256–261.

[20] Z. Ni, H. He, X. Zhong, and D. V. Prokhorov, "Model-free dual heuristic dynamic programming," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 8, pp. 1834–1839, 2015.

[21] S. Sieberling, Q. Chu, J. Mulder *et al.*, "Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction," *Journal of guidance, control, and dynamics*, vol. 33, no. 6, p. 1732, 2010.

[22] P. Simplício, M. Pavel, E. Van Kampen, and Q. Chu, "An acceleration measurements-based approach for helicopter nonlinear flight control using incremental nonlinear dynamic inversion," *Control Engineering Practice*, vol. 21, no. 8, pp. 1065–1077, 2013.

[23] P. Acquatella, E. van Kampen, Q. P. Chu *et al.*, "Incremental backstepping for robust nonlinear flight control," *Proceedings of the EuroGNC 2013*, 2013.

[24] Y. Zhou, E.-J. van Kampen, and Q. Chu, "Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback," *Journal of Guidance, Control, and Dynamics*, 2016.

[25] E.-J. v. C. Q. P. Zhou, Ye; Kampen, "Incremental model based heuristic dynamic programming for nonlinear adaptive flight control," *IMAV conference*, 2016.

[26] B. Chen, Y. Zhu, J. Hu, and J. C. Principe, *System parameter identification: information criteria and algorithms*. Newnes, 2013.

[27] L. Ljung, "System identification: Theory for the user, ptr prentice hall information and system sciences series," *ed: Prentice Hall, New Jersey*, 1999.

[28] R. Isermann and M. Mnchhof, *Identification of Dynamic Systems: An Introduction with Applications*. Springer-Verlag Berlin Heidelberg, 2011.

[29] I. The MathWorks. Hyperbolic tangent sigmoid transfer function. [Online]. Available: https://nl.mathworks.com/help/nnet/ref/tansig.html

[30] R. Chandramohan, J. E. Steck, K. Rokhaz, and S. Ferrari, "Adaptive critic flight control for a general aviation aircraft: Simulations for the beech bonanza fly-by-wire testbed," Ph.D. dissertation, Wichita State University, College of Engineering, Department of Aerospace Engineering, 2007.

[31] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[32] M. Mansour and J. Ellis, "Comparison of methods for estimating real process derivatives in on-line optimization," *Applied Mathematical Modelling*, vol. 27, no. 4, pp. 275–291, 2003.

[33] S.-H. Kim, Y.-S. Kim, and C. Song, "A robust adaptive nonlinear control approach to missile autopilot design," *Control engineering practice*, vol. 12, no. 2, pp. 149–154, 2004.

[34] H. Zhang and P. Roberts, "On-line steady-state optimisation of nonlinear constrained processes with slow dynamics," *Transactions of the Institute of Measurement and Control*, vol. 12, no. 5, pp. 251–261, 1990.