Spring 2018

Adaptive dynamic programming with eligibility traces and complexity reduction of high-dimensional systems

Seaar Jawad Kadhim Al-Dabooni

ADAPTIVE DYNAMIC PROGRAMMING WITH ELIGIBILITY TRACES AND COMPLEXITY REDUCTION OF HIGH-DIMENSIONAL SYSTEMS

by

SEAAR JAWAD KADHIM AL-DABOONI

A DISSERTATION

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

2018

Approved by

Dr. Donald C. Wunsch, Advisor
Dr. Jagannathan Sarangapani
Dr. R. Joe Stanley
Dr. Maciej Zawodniok
Dr. Cihan Dagli

PUBLICATION DISSERTATION OPTION

This dissertation has been prepared using the Publication Option. All papers are formatted in the style used by the Missouri University of Science and Technology, which are listed as follows:

Paper I: Pages 6 - 55; S. Al-Dabooni, and D. Wunsch, "Model Order Reduction Based on Agglomerative Hierarchical Clustering," accepted in *IEEE Trans. Neural Netw. and Learn. Syst.*, 2017.

Paper II: Pages 56 - 87; S. Al-Dabooni, and D. Wunsch, "Heuristic dynamic programming for mobile robot path planning based on Dyna approach," IEEE/INNS, International Joint Conference on Neural Networks (IJCNN), pp. 3723 - 3730, Jul. 2016.

Paper III: Pages 88 - 113; S. Al-Dabooni, and D. Wunsch, "Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning," IEEE/INNS, International Joint Conference on Neural Networks (IJCNN), pp. 2820-2827, May 2017.

The other papers are submitted to *IEEE Trans. Neural Netw. and Learn. Syst.*, which are listed as follows: Paper IV: Pages 114 - 157; S. Al-Dabooni, and D. Wunsch, "The Boundedness Conditions for Model-Free HDP(λ)."

Paper V: Pages 158 - 208; S. Al-Dabooni, and D. Wunsch, "Online Model-Free N-Step HDP with Stability Analysis."

Paper VI: Pages 209 - 261; S. Al-Dabooni, and D. Wunsch, "An Improved N-Step Value Gradient Learning Adaptive Dynamic Programming Algorithm for Online Learning, with Convergence Proof and Case Studies."

Paper VII: Pages 262 - 325; S. Al-Dabooni, and D. Wunsch, "Convergence Analysis Proofs for Recurrent Neuro-Fuzzy Value-Gradient Learning with and without Actor."

ABSTRACT

This dissertation investigates the application of a variety of computational intelligence techniques, particularly clustering and adaptive dynamic programming (ADP) designs especially heuristic dynamic programming (HDP) and dual heuristic programming (DHP). Moreover, a one-step temporal-difference (TD(0)) and n-step TD (TD(λ)) with their gradients are utilized as learning algorithms to train and online-adapt the families of ADP. The dissertation is organized into seven papers. The first paper demonstrates the robustness of model order reduction (MOR) for simulating complex dynamical systems. Agglomerative hierarchical clustering based on performance evaluation is introduced for MOR. This method computes the reduced order denominator of the transfer function by clustering system poles in a hierarchical dendrogram. Several numerical examples of reducing techniques are taken from the literature to compare with our work. In the second paper, a HDP is combined with the Dyna algorithm for path planning. The third paper uses DHP with an eligibility trace parameter (λ) to track a reference trajectory under uncertainties for a nonholonomic mobile robot by using a first-order Sugeno fuzzy neural network structure for the critic and actor networks. In the fourth and fifth papers, a stability analysis for a model-free action-dependent HDP(λ) is demonstrated with batch- and online-implementation learning, respectively. The sixth work combines two different gradient prediction levels of critic networks. In this work, we provide a convergence proofs. The seventh paper develops a two-hybrid recurrent fuzzy neural network structures for both critic and actor networks. They use a novel n-step gradient temporal-difference (gradient of TD(λ)) of an advanced ADP algorithm called value-gradient learning (VGL(λ)), and convergence proofs are given. Furthermore, the seventh paper is the first to combine the single network adaptive critic with VGL(λ).

ACKNOWLEDGMENTS

Thanks to my God for helping me and giving me the blessings to complete this work. I would like to thank my advisor, Dr. Donald C. Wunsch, for his continuous invaluable support me. Dr. Wunsch has given me the most professional guidance throughout choosing modern research topics, paper writing, technical presentations, and all required resources for producing good works.

I would like to express my sincere gratitude to all my committee members, Dr. Jagannathan Sarangapani, Dr. R. Joe Stanley, Dr. Maciej Zawodniok and Dr. Cihan Dagli, for their valuable advice and recommendations, and for their precious time in examining this dissertation. I am very grateful to the Higher Committee for Educational Development (HCED) in Iraq for granting me a Ph.D. scholarship and financial support. I am very thankful for my research group members in the Applied Computational Intelligence Laboratory (ACIL) for their fruitful discussion. They are wonderful praters and helpers.

Big thanks to my father, my mother, my brothers, and my sisters with all their nieces, nephews, and brothers-in-low for support, encouragement, and prayers throughout my study. I would also thanks Hassanin Al-Dabooni, Dr. Suadad Al-Dabooni, Dr. Alaa Al-Kinani, Seemaa Al-Dabooni, Dr. Hussein Shakarchi, and Dr. Turki Younis for their orientation and support.

Last but not least, I would like to thank to my friends Shahin Korkis Adam, Majid Hameed, Safaa Norri, Ahmed Adnan, Alaa Norri, Yassar Al-Tob, Wessam Al-Adbi, Yousif Khanjer, Mustafa Muzael, Ethar Alkamil, Ali Albattat, and Safaa Aiad for all the supports and encouragements.

TABLE OF CONTENTS

Page

PUBLICATION DISSERTATION OPTION iii

ABSTRACT iv

ACKNOWLEDGMENTS v

LIST OF ILLUSTRATIONS xiv

LIST OF TABLES xxx

SECTION

1. INTRODUCTION 1

    1.1. SYSTEMS WITH REDUCING COMPLEXITY 1

    1.2. ADVANCED ADAPTIVE DYNAMIC PROGRAMMING 1

    1.3. RESEARCH CONTRIBUTIONS 2

        1.3.1. Model Order Reduction Based on Agglomerative Hierarchical Clustering 2

        1.3.2. Heuristic Dynamic Programming for Mobile Robot Path Planning Based on Dyna Approach 3

        1.3.3. Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning 3

        1.3.4. The Boundedness Conditions for Model-Free HDP(λ) 4

        1.3.5. Online Model-Free N-Step HDP with Stability Analysis 4

        1.3.6. An Improved N-Step Value Gradient Learning Adaptive Dynamic Programming Algorithm for Online Learning, with Convergence Proof and Case Studies 5

1.3.7. Convergence Analysis Proofs for Recurrent Neuro-Fuzzy Value-Gradient Learning with and without Actor 5

PAPER

I. MODEL ORDER REDUCTION BASED ON AGGLOMERATIVE HIERARCHICAL CLUSTERING 6

ABSTRACT 6

1. INTRODUCTION 7

2. PROBLEM STATEMENT 11

3. HIERARCHICAL CLUSTERING SYSTEM POLES ALGORITHM 11

3.1. Choice of Distance Measure 12

3.1.1 Hierarchical Cluster Poles Based on Euclidean-distance .. 12

3.1.2 Hierarchical Clustered Poles Based on MSE 13

3.2. Determining Transfer Function Coefficients 15

3.2.1 PA Method 15

3.2.2 GA Method 17

4. SIMULATION AND ANALYSIS RESULTS WITH NUMERICAL EXAMPLES 18

4.1. Comparative Studies From Published Literature 18

4.2. Case Study for Large Order System (Building Model Structure) 21

5. MODEL REDUCTION FOR A MULTIVARIABLE DYNAMIC MODEL BY USING HC-PE 26

6. CONCLUSION 41

APPENDICES

A. Stability guarantee for a new reduced model 47

B. Stability proof for a closed loop reduced system 49

C. Comparison table of HC-PE with other reduction methods, combining the Pade approximation approach and GA. The best MSE results are shown in bold. 51

ACKNOWLEDGMENT 53

BIBLIOGRAPHY 53

II. HEURISTIC DYNAMIC PROGRAMMING FOR MOBILE ROBOT PATH PLANNING BASED ON DYNA APPROACH 56

ABSTRACT 56

1. INTRODUCTION 57

2. COLLISION-FREE NAVIGATION 59

2.1. Simulation of Mobile Robot Platform 59

2.2. Description of the Fuzzy Controller for Obstacle Avoidance 60

2.2.1 Distance 61

2.2.2 Angle 61

2.2.3 Angular Speed 61

3. FUNDAMENTAL REINFORCEMENT LEARNING PARAMETERS 64

4. MOBILE ROBOT PATH PLANNING BASED ON DYNA-HDP 66

4.1. Architecture of The Dyna-HDP System 66

4.2. k-max Certainty Method 69

4.3. Knowledge Sharing for Distributed Mobile Robots 70

5. SIMULATION RESULTS AND ANALYSIS 76

5.1. Applying Dyna-HDP in A New Environment (Building Map) 83

6. CONCLUSION 85

BIBLIOGRAPHY 85

III. MOBILE ROBOT CONTROL BASED ON HYBRID NEURO-FUZZY VALUE GRADIENT REINFORCEMENT LEARNING 88

ABSTRACT 88

1. INTRODUCTION 89

2. FUNDAMENTAL PRELIMINARIES 90

2.1. Dynamic Modeling of the Mobile Robot 90

2.2. DP/RL Algorithms 93

2.3. FNN 95

3. MOBILE ROBOT CONTROL BY FNN-BASED OF VGL(λ) 96

3.1. VGL(λ) Structure Learning by Using FNN 96

3.2. Mobile Robot Control Based on FNN-VGL(λ) 98

3.2.1 Critic Network Training Algorithm 99

3.2.2 Actor Network Training Algorithm 101

4. SIMULATION RESULTS 102

4.1. First Case (Effectiveness of λ value) 102

4.2. Second Case (with/without Actor Network) 103

5. CONCLUSION 111

BIBLIOGRAPHY 111

IV. THE BOUNDEDNESS CONDITIONS FOR MODEL-FREE HDP(λ) 114

ABSTRACT 114

1. INTRODUCTION 115

2. STRUCTURE OF MODEL-FREE HDP(λ) 116

2.1. HDP(λ) Learning Views 116

2.2. The Critic Neural Network 121

2.3. The Actor Neural Network 123

3. STABILITY ANALYSIS 125

3.1. Lyapunov Approach 125

3.2. Preliminaries 126

3.3. The Dynamical System Stability Analysis 129

4. SIMULATION RESULTS 133

4.1. Case I: Nonlinear System Problem 133

4.2. Case II: Inverted Pendulum 134

4.3. Case III: 3-D Maze Problem 142

5. CONCLUSION 153

BIBLIOGRAPHY 154

V. ONLINE MODEL-FREE N-STEP HDP WITH STABILITY ANALYSIS. 158

ABSTRACT 158

1. INTRODUCTION 159

2. THE ONLINE MODEL-FREE NSHDP(λ) 162

2.1. NSHDP(λ) Architecture 162

2.2. The One-Step Critic Network (CN(0)) 166

2.3. The N-Step Critic Network (CN(λ)) 168

2.4. Actor Network (AN) 170

3. STABILITY ANALYSIS FOR NSHDP(λ) 172

3.1. Basics of The Lyapunov Approach 173

3.2. Assumptions 174

3.3. The Stability Analyses for The Dynamical System 178

4. SIMULATION STUDY 182

4.1. First Case: Nonlinear System Problem 183

4.2. Second Case: Inverted Pendulum 189

4.2.1 Description for The Inverted Pendulum Dynamic Model 189

4.2.2 Simulation Results 191

4.3. Third Case: 2-D Maze Problem 196

5. CONCLUSION 204

BIBLIOGRAPHY 204

VI. AN IMPROVED N-STEP VALUE GRADIENT LEARNING ADAPTIVE DYNAMIC PROGRAMMING ALGORITHM FOR ONLINE LEARNING, WITH CONVERGENCE PROOF AND CASE STUDIES 209

ABSTRACT 209

1. INTRODUCTION 210

2. THE ONLINE NSVGL(λ) STRUCTURE DESIGN 214

2.1. Improved Leaning of Temporal Sequences 214

2.2. Improved Exploration\Exploitation Trade-off 216

2.3. Memory Efficiency 219

2.4. Improved Actor Network Training 220

2.5. Faster Convergence Via Two-Critic Iteration 221

3. CONVERGENCE PROOF 221

3.1. One-step and *n*-step DT-HJB Equations 221

3.2. Derivation of Iteration NSVGL(λ) Algorithm 224

3.3. Convergence of Iterative NSVGL(λ) Algorithm 225

4. NEURAL NETWORK ARCHITECTURE DESIGN 234

4.1. The One-Step Critic Network $\left(\hat{G}^0\left(x_k, \hat{\omega}_c^0\right)\right)$ 234

4.2. The *n*-step Critic Network $(\hat{G}^\lambda\left(x_k, \hat{\omega}_c^\lambda\right))$ 236

4.3. Actor Network $(\hat{A}\left(x_k, \hat{\omega}_a\right)))$ 237

5. SIMULATION STUDIES 239

5.1. Case I: Nonlinear System Problem 240

5.2. Case II: Mobile Robot Dynamic Model 245

6. CONCLUSION 255

BIBLIOGRAPHY 256

VII. CONVERGENCE ANALYSIS PROOFS FOR RECURRENT NEURO-FUZZY VALUE-GRADIENT LEARNING WITH AND WITHOUT ACTOR 262

ABSTRACT 262

1. INTRODUCTION 263

2. THE ACTOR-CRITIC AND SINGLE-CRITIC OF VGL(λ) ARCHITECTURE DESIGNS 266

2.1. Adaptive Actor-Critic Approach 269

2.1.1 The n-step Critic Network 269

2.1.2 The Actor Network 271

2.2. Single Adaptive *n*-step Critic Approach 272

2.2.1 The n-step Optimal Control Equation 272

2.2.2 The n-step Critic Network 272

3. NF STRUCTURES FOR VGL(λ) AND SNVGL(λ) APPROACHES 274

3.1. Recurrent Neuro-Fuzzy (RNF) Structure 274

3.2. Takagi-Sugeno Recurrent Neuro-Fuzzy (TSRNF) Structure 277

4. CONVERGENCE ANALYSIS OF THE VGL(λ) AND SNVGL(λ) APPROACHES 282

4.1. DT-HJB Equation for $R^{\lambda}(k)$ 282

4.2. Derivation of Iteration VGL(λ) 285

4.3. Convergence of Iterative VGL(λ) Algorithm 286

5. COMPACTING THE NF-VGL(λ) AND NF-SNVGL(λ) ITERATIVE ALGORITHM WITH RNF AND TSRNF STRUCTURES 292

5.1. The Value-Iteration-based NF-VGL(λ) 293

5.1.1 The *n*-step Critic Network 293

5.1.2 The Actor Network 294

5.2. The Value-Iteration-based NF-SNVGL(λ) 295

5.2.1 The *n*-step Critic Network 295

5.2.2 The *n*-step Optimal Control 295

6. SIMULATION STUDY 296

6.1. The Nonlinear Dynamic Model of Mobile Robot 296

6.2. Simulation Results 299

7. CONCLUSION 319

BIBLIOGRAPHY 319

SECTION

2. SUMMARY AND CONCLUSIONS 326

2.1. MODEL ORDER REDUCTION BY CLUSTERING SYSTEM POLES 326

2.2. ADAPTIVE DYNAMIC PROGRAMMING WITH N-STEP PREDICTION PARAMETER 327

BIBLIOGRAPHY 329

VITA 341

LIST OF ILLUSTRATIONS

Figure Page

PAPER I

1 General diagram for using HC-PE to reduce high order models. Any system model is built by deriving physical laws (physical modeling) or by observing data (identification modeling). Some models are represented by partial differential equations (e.g. heat transfer equation), and other models are built by ordinary differential equations (e.g. robotics). Finite difference techniques are used to discretize partial differential equations to derive a numerical approximation for ordinary differential equations [3]. In this work, we focus on the models, which are built by using ordinary differential equations (G). In combination with PA or GA, HC-PE improves the evaluation for the reduced model (G_r) through selecting minimum MSEs of clusters made from poles. To find the minimum MSE, all MSE values should be calculated for each level (order). MSE is calculated between the original model and the reduced models which is represented by the blue dashed line. 10

2 Three cases that should be considered while clustering poles using either Euclidean-distance or MSE as the similarity. The first case is clustering real and complex poles separately. The second case is applying a full-state feedback approach (pole placement method) in unstable systems. The third case is retaining the poles in the imaginary axis and in the origin of the s-plane to the reduced model. 14

3 Flowchart for the HC-PE algorithm, which starts from n^{th} order and calculates the reduced model in $r^{\text{th}} = n^{\text{th}} - 1$ order, which becomes the base model (an optimal simulated original model at r^{th} order) to calculate a next reduced model $n^{\text{th}} - 1$ order and so on until reaching the 2^{nd} order. 16

4 The dendrogram for hierarchical cluster poles based on Euclidean distance for $G_{10}(s)$ combined with the PA approach. It clearly has large error values in most reduction orders. 20

5 The dendrogram for hierarchical clusters poles based on the HC-PE-PA algorithm for $G_{10}(s)$. Minimum error and stability in the levels are generated by using this algorithm. Cutting is done at level two to generate the 2^{nd} order reduced model for comparing [19] and [21]. 21

6 The Time Step response for step responses for original, [19] and [21], and HC-PE-PA. 22

7 The dendrogram for hierarchical clustering of poles based on the HC-PE-PA algorithm for $G_8(s)$. Stability and fluent minimum error tracking level is generated by using this algorithm. Level three was cut to generate a 3rd order reduced model to compare with [19]. 23

8 The time step responses for the original, [19] and HC-PE-PA. 23

9 Bode plot for step responses for the original, [19] and HC-PE-PA to show the stability. 24

10 Cluster poles of the Los Angeles building model. The blue star, red circle, and green diamond shape symbols describe the best six clusters (three clusters in positive imaginary axis part with 3 mirror them in negative imaginary axis part), which is obtained by applying the HC-PE technique. The new system poles reduced model (six cluster centers) show as two big symbols (blue stars, red circles, and green diamonds). 27

11 A comparison between random (mode 1) and semi-random (mode 2) initial chromosomes in population for 3 independent runs. Bold blue and red curves are a mean of mode 1 and mode 2, respectively, while thin blue and red curves are represented upper and lower fitness values for 3 runs. 28

12 The time step responses for the 48th order original building model and the 6th order reduced models obtained via HC-PE-PA, HC-PE-GA, Hankel and balanced truncation techniques. HC-PE has the best performance compared with the other two approaches. MSE for HC-PE-PA, HC-PE-GA, Hankel and balanced truncation are $1.758e-09$, $6.807e-10$, $3.089e-07$, and $2.827e-08$, respectively. 29

13 The Bode plot frequency responses for the 48th order original building model and the 6th order reduced models obtained via HC-PE-PA, HC-PE-GA, Hankel and balanced truncation techniques. HC-PE-PA has the best approximation compared with the others, which is clear from the lowest frequency range until 0.5Hz. 30

14 Configuration model of triple link inverted pendulum. 35

15 The step response results after applying HC-PE-PA to reduce the model of the triple link inverted pendulum to 3rd order. (a) The cart displacement. (b) The lower angle. (c) The middle angle. (d) The upper angle. The original 8th order model reduces after applying RCG and LQRCG. 37

16 The step response after applying HC-PE-PA and HC-PE-GA to reduce the model of a triple link inverted pendulum to 3rd order. (a) The cart displacement. (b) The lower angle. (c) The middle angle. (d) The upper angle. The original 8rd order model applies the LQRCG method. 38

17 The step response after applying HC-PE-PA and HC-PE-GA to reduce the model of a triple link inverted pendulum to 3rd order after changing mass parameters. (a) The cart displacement. (b) The lower angle. (c) The middle angle. (d) The upper angle. The LQRCG method is applied to make the original 8th order model stable. 39

18 Simulink model of LQRCG applied to the triple link inverted pendulum for original 8th and 3rd order reduced model after applying HC-PE for both Pade approximation and GA control by PID control with a disturbance input. 42

19 The responses after applying the disturbance signal on the cart displacement state for comparison among LQRCG, HC-PE-PA, and HC-PE-GA reduced models.(a) The cart displacement.(b) The lower angle. 43

20 The responses after applying the disturbance signal on the first state (cart displacement) to compare LQRCG, HC-PE-PA, and HC-PE-GA reduced models. (a) The middle angle state. (b) The upper angle state. GA performs slightly better for the transient response, but PA has acceptable transient and superior steady state performance. 44

21 2-D simulation for a triple linked inverted pendulum model. (a), (b) and (c) show the simulation for the HC-PE-PA reduced model starting from the 0 position to the final desired cart positions which are 4, 6 and 10, respectively.... 45

PAPER II

1 Differential wheeled mobile robot platform. It forms as 180° field of view in front by nine sensors with 22.5° separation angle to sense an object about 0.2 meter away from its body. 60

2 Distance fuzzy membership functions. These fuzzy set definitions are used for input variable which are consisted from three triangular-shaped membership functions (good, near, and far). 61

3 Angle fuzzy membership functions. These fuzzy set definitions are used for input variable which are consisted from five triangular-shaped membership functions (very negative, negative, zero, positive, and very positive). 62

4 Angular speed fuzzy membership functions. These fuzzy set definitions are used for output variables for both left and right wheels which are consisted from five triangular-shaped membership functions (backward, slow backward, stop, slow forward, and forward). 62

5 Block diagram for Dyna-HDP Path Planning. u_t^i is the action vector at time t for robot i which is consisted of three actions (turn left, turn right, and moving forward) denoting as a_t^i. s_t^i is the input states vector for robot i at time t which are represented by the nine sensor readings $S_{d5} - S_{d9}$, the relative different angle (θ_{diff}), and the relative distance (D_g). A reinforcement function (r) can get from (2) for state s_t^i and a_i. All robots share the same virtual model to maximize the value function J_t^i for all agents at the same time. The backpropagation path is shown by dashed lines for action and critic networks, and for updating the rules for k-max certainty. 66

6 A flow chart for k-max certainty selection procedure for agent i. A green dashed line border represents the exploitation approach which has been triggered by taking a greedy action among k-certainty rules (sensory-action pairs). The exploration approach is represented by a blue dashed line border which is selected action randomly among uncertainty rules (or trail). 71

7 Flowchart for implementation the multi-agents Dyna-HDP approach on maze problem. 75

8 a) The environment for testing the exploration/exploitation strategy which shows the initial start position for this agent. b) The near-optimality trajectory from the starting point to the target. c) The comparison between $\varepsilon-$greedy and k-max certainty by using Q-learning algorithm. 77

9 The first case comparison the number of steps per episodes. This compares among Dyna-HDP with other conventional algorithms, one step Q-learning, Sarsa (λ), and Dyna-Q, for one agent (no information sharing with the others). . 79

10 The second case results (the cooperation between two agents by sharing their experiences by using the Dyna-HDP approach). Every agent has own experience and tries to support the other one. 79

11 The Third case results (Cooperation among five agents by using the Dyna-HDP approach). 80

12 Five mobile robots are distributed in unknown environment surrounding by wall in meter scale with many obstacles on it. The robots are distributed in random start position and heading angle to achieve same task (reaching to target position). 80

13 The near optimality trajectories for the five mobile robots from start positions to the same target position after applying the third case which is all agents share their experiences with the same approach. 81

14 Comparison among all three cases by using Dyna-HDP for average learning. In third case, the results are stable with the improvement in the learning speed of distributed autonomous agents by sharing their experiences. 82

15 a) The trajectories for two agents to run Task 1 b) The trajectories for two agents to run Task 2 c) The trajectories for one agent to run Task 3 d) The simulation result for three tasks among Dyna-HDP. 84

16 Path for mobile robot to implement the third task with a change in the environment by adding a new obstacle. FCOA gives the mobile robot the capability to adapt with this changing. 85

PAPER III

1 Schematic diagram of FNN used in ADP. Premise (σ_i^l, m_i^l) and consequent $(c_{1j}^l, \ldots, c_{(n+1)j}^l)$ parameters are updated by using the backpropagation gradient algorithm. The weights between the premise and hidden layers are always one. 97

2 Adaptation in VGL(λ) for the mobile robot trajectory control tracking. In general, forward pathways are represented by the solid lines while pathways of backpropagation are shown by dashed lines, and the small solid black dots represent a connection. The sequential environment is used in this work by storing all states (velocity vectors) and actions (torque vectors) in forward pathways. The TD-error between $\tilde{g}_t$ and $\bar{g}_t$, as in (11), is used for updating critic network weights with applying (23) for $\bar{g}_t$. MUX box is used to feed (23) as one thick dashed line by gathering the signals. The actor weights are updated by applying (24). For robustness testing, various values of $\bar{F}$ and $\bar{\tau}_d$ are injected into this dynamic model to examine the influences on $\tau(t)$. The robot pose trajectory (x_c, y_c, θ) is obtained by using the mobile robot kinematic equation (7) with fourth order Runge-Kutta integration for the derivative of the coordination and orientation of the robot. The Runge-Kutta integration also uses to solve the dynamic model (8) within 0.01 sec for the sample time. 100

3 The actor and critic performance at $\lambda = 0$. (a) Typical system trajectories for both velocity states over time at last stable learning iteration. (b) The average of MSE for both velocity states over iterations. (c) The average torques applying to the left and right wheels over iterations. 104

4 The control performance at $\lambda = 0.9$. (a) - (c) has same descriptions for (a) - (c) as in Fig. 3. 105

5 Comparison of the average performance at $\lambda = 0$ and $\lambda = 0.9$. (a) The MSE for both velocity state errors. (b) Requirement torque values. (c) The function value, which is calculated from (1). 106

6 Critic/actor and critic/optimal-torque approaches set to follow a circular trajectory at $\lambda = 0.9$ with disturbances for NN and FNN. (a) The X-Y trajectories for the mobile robot. 108

7 Critic/actor and critic/optimal-torque approaches set to follow a circular trajectory at $\lambda = 0.9$ with disturbances for NN and FNN. (b) and the average of MSE over iterations, respectively. (c) The absolute average values for the left and right control torques over iterations. 109

8 Critic/actor and critic/optimal-torque approaches set to follow a circular trajectory at $\lambda = 0.9$ with disturbances for NN and FNN. (d) The value of the torques over time for the last learning iteration for FNN. The cost-to-go value is shown in (e).. 110

PAPER IV

1 Schematic for the adaptation of the novel model-free HDP(λ) structure design according to (8). Forward pathways are represented by solid lines, while backpropagation pathways are shown by dashed lines. The small solid black dots represent a connection path. The initial target-values ($\bar{v}(t)$) are provided by substituting $\lambda = 0$ with (8), which is identical to the traditional HDP. The TD-error between $\bar{v}(t-1)$ and $\hat{v}(t-1)$, as in (76), is used to update critic network weights (84), which is represented by red dashed line. The actor network weights are estimated by back-propagating the prediction error (blue dashed line), which is equal to the value function ($\hat{v}(t-1)$) through the critic network, and updating the actor's weights according to (28)...................... 120

2 Schematic diagram of the critic network in HDP(λ). As mentioned by Werbos in [30], this structure is more general than a traditional three-layer feedforward neural network by fully connecting all neurons. It models a variety of functional forms [34]. We set all weights that connect input nodes with output nodes to zero; therefore, it is similar to a three-layer feedforward neural network structure. $\hat{\omega}_c^{\{h\}}$ represents a hidden weights, which connect the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_c^{\{o\}}$, which connect both input and hidden layers with the output layer. $s_k(t)$ is the kth hidden node input of the critic network, and $r_k(t)$ is the corresponding output the hidden node. Here, we only apply a hyperbolic tangent threshold function ($\phi(.)$) to the hidden neurons. 122

3 Schematic diagram of the actor network in HDP(λ). We set all weights that connect input nodes with output nodes to zero as well as the connected weights between the outputs themselves. $\hat{\omega}_a^{\{h\}}$ represents hidden weights which connect the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_a^{\{o\}}$, which connect both input and hidden layers with the output layer. $p_k(t)$ is the kth hidden node input of the actor network, and $q_k(t)$ is the corresponding output of the hidden node. Here, we only apply a hyperbolic tangent threshold function ($\phi(.)$) on the hidden neurons. 124

4 Comparisons of system responses (the state vector trajectories and the action sequence) with HDP($\lambda = 0.95$), and traditional HDP($\lambda = 0$). 134

5 Cost error over iteration and 30 time step critic and actor errors. The upper figure shows the cost error over iterations, which clearly illustrates fast and stable learning, while traditional HDP shows fluctuations during learning. The critic and actor errors at the last learning iteration for HDP and HDP(λ) are shown in the middle and lower figures. 135

6 Learning weights for critic and actor networks for HDP and HDP(λ). 136

7 Configuration model of the cart-pole balancing system. 136

8 The value function and target value for the last trial without noise. The initial angle $\theta(t)$ is 0.1° 138

9 Zoom-in between 80 to 220 time steps for Fig. (8). 138

10 Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system is free of noise; initial angle $\theta(t)$ is 0.1°. 139

11 Zoom-in between 400 to 520 time steps for Fig. (10). 139

12 The value function and target value for the last trial when the system is free of noise; initial angle $\theta(t)$ is 10°. 140

13 Zoom-in between 0 to 150 time steps for Fig. (12). 141

14 Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system is free of noise; initial angle $\theta(t)$ is 10°. 141

15 Zoom-in between 0 to 100 time steps for for Fig. 14. 142

16 Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system has uniform 3% sensor noise; initial angle $\theta(t)$ is 10°. 143

17 Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system has uniform 3% sensor noise; initial angle $\theta(t)$ is 10°. 143

18 The value function and target value for the last trial when the system has uniform 5% actuator noise; initial angle $\theta(t)$ is 10°. 144

19 Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system has uniform 5% actuator noise; initial angle $\theta(t)$ is 10°.144

20 Squared critic error (E_c) for noise-free with 0.1° to initial angle, noise-free with 10° to initial angle, uniform 3% sensor noise, and uniform 5% actuator noise. .. 145

21 Diagram of 3-D maze ($5 \times 5 \times 5$) with obstacles. The dark blue cube $(0, 0, 0)$ represents the initial position. The green cube $(4, 4, 4)$ represents the target position. 12 obstacles are located in $(0, 3, 0)$, $(2, 4, 1)$, $(2, 4, 2)$, $(2, 3, 2)$, $(2, 2, 2)$, $(2, 1, 2)$, $(4, 0, 0 - 4)$ and $(3, 0, 4)$, which are represented by the red cubes. Otherwise, the agent can move in free space. Three modes allow the agent to receive reward/cost values. First, The agent will receive reward 1 when it arrives to the target cube. Second, the agent will be punished by receiving -0.001 if it hits obstacles or passes the boundary. Third, the agent will receive 0 value as a reward in a free space. At any position in the maze, the agent has to select 1 action (direction) out of six actions in order to to move one step. The six actions are forward, right, backward, left, up and down, which can be seen in the figure as u_1, u_2, u_3, u_4, u_5 and u_6, respectively. We sketch this figure by using the isometric drawing tool [42]. 150

22 Mean-squared-error (MSE) learning curves for SARSA(0), Q($\lambda = 0.95$), HDP(0) and HDP($\lambda = 0.95$) for the 3-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The HDP(0.95) approach has the fastest learning with a lower MSE compared to other approaches. 151

23 Summation of accumulative reward for every single episode of SARSA(0), Q($\lambda = 0.95$), HDP(0) and HDP($\lambda = 0.95$) approaches, which is applied in the 3-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The HDP(0.95) approach reaches the largest accumulative reward compared to other methods. Because $\epsilon-$greedy learning will reset the accumulative reward value every episode, the accumulative reward values for Q(0.95), HDP(0) and HDP(0.95) converge over episodes. 152

24 $\epsilon-$greedy learning curves for SARSA(0), Q($\lambda = 0.95$), HDP(0) and HDP($\lambda = 0.95$) approaches for 3-D maze navigation benchmark as shown in Fig. 20. These curves represent the number of steps per episode, where the agent returns back to the start cube only when it reaches the target cube. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. HDP(0.95) and HDP(0) have an almost identical number of steps over episodes, which are less than those in both Q(0.95) and SARSA(0) methods. 153

PAPER V

1 Schematic diagram for the adaptation of an online model-free n-step ADHDP (NSHDP(λ)). This design uses two critic networks: the one-step critic network (CN(0)) and the n-step critic network (CN(λ)). The CN(0) produces a one-step-return value function ($\hat{v}^0(t)$) based on the ordinary temporal-difference (TD) learning algorithm, while the CN(λ) produces the average of the n-step-return value function ($\hat{v}^\lambda(t)$) based on a TD(λ) learning algorithm [27]. The TD(λ) learned from the average of the n-step-return backups, where λ represents the proportional average weight. $\lambda-$return (R_t^λ) [16] is another name for the average of the n-serp-return. The $\hat{v}^\lambda(t)$ value is identical to R_t^λ, [29]. This design is equivalent to the one-step TD backup (λ=0). It focuses on the recent information to predict the value function via CN(0). Online learning is another advantage of this design, where it speeds up the tuning without requiring any initial backup for $\hat{v}^\lambda(t)$. Furthermore, this design is a model-free learning design that does not require prior knowledge about a mathematics dynamic model. Despite the bootstrapping eligibility trace parameters (λ and γ) give the CN(λ) the ability to determine a depth (effecting via λ) and a width (effecting via γ) from information during a sequence of events (i.e., the rewards in the backward view of TD(λ), [21]). The CN(0) provides the value function that concentrates on recent events. Therefore, the NSHDP(λ) design combines the details of the current information (real-time data) with a sequence of predicted events. This combination provides the optimal decisions [40] in the control/industry field as well as [41] in the consumer/marketing field (correlation between real time and history). The weights for CN(0) and CN(λ) are updated according to the TD(0) error (blue dashed line) and the TD(λ) error (green dashed line), respectively. The actor network (AN) that provides the action values is tuned by two paths (backpropagating errors): one through the CN(0) path ($e_a^0(t)$) and the other through the CN(λ) path ($e_a^\lambda(t)$). This strategy assists AN training to correlate and combine the fluid information from CN(λ) and CN(0). These two paths are filtered via a similar value of λ, and they combine to produce a total backpropagating actor error (red dashed line). 161

2 A schematic diagram of CN(0) in NSHDP(λ). As mentioned by Werbos in [38], this structure is more general than a traditional three-layer feed-forward neural network that is fully connected among all neurons. It models a variety of functional forms as demonstrated in [39]. All weights were set so that the connection input nodes with the output nodes were zero. $\hat{\omega}_c^{0\{h\}}$ represents hidden weights, which are connected to the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_c^{0\{o\}}$, which connect both the input and hidden layers with the output layer. $a_k(t)$ is the kth hidden node input of the critic network, and $b_k(t)$ is the corresponding output of the hidden node. A hyperbolic tangent threshold function ($\phi(.)$) is applied to the hidden neurons. ... 166

3 A schematic diagram of the average n-step learning critic network (CN(λ)) in NSHDP(λ). The $\hat{\omega}_c^{\lambda\{h\}}$ represents the hidden weights which are connected to the input layer through the hidden layer. The output weights are indicated as $\hat{\omega}_c^{\lambda\{o\}}$, which connect both the input and hidden layers with the output layer. $c_k(t)$ is the kth hidden node input of the critic network, and $d_k(t)$ is the corresponding output to the hidden node. Here, only apply a hyperbolic tangent threshold function (ϕ(.)) is applied to the hidden neurons. 168

4 A schematic diagram of the actor network (AN) in NSHDP(λ). All of the weights that connect input nodes with output nodes are set to zero, as well as the connected weights between the outputs themselves. $\hat{\omega}_a^{\{h\}}$ represents the hidden weights which are connected the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_a^{\{o\}}$, which connect both input and hidden layers with the output layer. $p_k(t)$ is the kth hidden node input of the critic network, and $q_k(t)$ is the corresponding output to the hidden node. A hyperbolic tangent threshold function (ϕ(.)) is applied in the hidden neurons. 171

5 Mean squared error comparisons over iterations among the NSHDP(λ), the GrHDP and the traditional ADHDP. NSHDP(λ) has a faster learning speed than the GrHDP and the ADHDP.. 184

6 Comparisons of system response trajectories, which are the two system states and the actions for the NSHDP(λ), GrHDP and ADHDP. 185

7 Comparisons for the value functions with their targets between the NSHDP(λ) and the GrHDP. The upper figure shows the value function and its target for the GrHDP (see [7]), which is represented by a red dashed line in the upper figure part of Fig. 8. The middle and upper figures show the value functions of CN(0) and CN(λ) of the NSHDP(λ) with their targets (the left sides of Equations (10) and (11)), respectively. The solid green line in the lower part of Fig. 8 represents the difference of the middle figure, while the dashed red line represents the difference of the lower figure. 186

8 The squared critic errors. The upper figure shows the squared errors for critic and reference networks in the GrHDP structure (see [7] Equations(4) and (5)). The squared critic errors for NSHDP(λ) for both critic networks (CN(0) and CN(λ)) are illustrated in the middle and upper figures when $\lambda = 0.95$ and $\lambda = 0$, respectively. The squared error for the reference network in the GrHDP is represented by "Er". 187

9 The upper figure shows the squared backpropagating actor error after passing a critic network in the GrHDP. The middle and lower figures show $BE_a^0(t)$ and $BE_a^\lambda(t)$ (see (29)) of the NSHDP(λ) when $\lambda = 0.95$ and $\lambda = 0$, respectively. The lower figure (NSHDP(0.95)) converges faster than the upper (GrHDP) and middle (ADHDP) figures 188

10 Learning weights for the NSHDP(λ) and the GrHDP. The upper, middle and lower figures in the first column show the learning weights during the time steps for CN(0), CN(λ) and AN in the NSHDP(λ), respectively. The upper, middle and lower figures of the second column show the learning weights during the time steps for goal representation or the reference network (RN), critic network and actor network of GrHDP. 189

11 The configuration schematic diagram of the inverted pendulum balancing system. 190

12 Simulated results of balancing the inverted pendulum for $u(t)$, $\theta(t)$, and $x(t)$ when the system is free of noise. The initial angle $\theta(t)$ is 0.1°. 192

13 Zoom-in between 1000 to 1250 time steps for Fig. 12. 192

14 The value functions and their targets without noise. The backpropagating actor errors are Be_a^0 and Be_a^{λ} through CN(0) and CN(λ), respectively. The initial angle $\theta(t)$ is 0.1°. 193

15 Zoom-in between 1000 to 1250 time steps for Fig. 14. 193

16 Simulated results of balancing the inverted pendulum to show u(t), $\theta(t)$ and $x(t)$, when the system is injected with 3% uniform noise to the actuator and sensor. The initial angle $\theta(t)$ is -12°. 194

17 The value functions and their target values with similar disturbances are in the caption of Fig. 16 (upper and lower figures for CN(0) and CN(λ), respectively). The backpropagation actor errors through CN(0) (Be_a^0) and CN(λ) (Be_a^{λ}) are shown in the lower figure. The initial angle $\theta(t)$ is -12°. 195

18 Simulated results of balancing the inverted pendulum to show u(t), $\theta(t)$ and $x(t)$, when the system is injected with 3% uniform noise to the actuator and sensor. The initial angle $\theta(t)$ is -12°. 195

19 The function values with errors with the initial angle $\theta(t)$ is 12°. The actuator and sensor disturbances are similar to what in the caption of Fig. 18 196

20 Diagram of a 2-D maze (7×7) with obstacles. The point(0,0) represents the initial position. The point (6,6) represents the target position. Eleven obstacles are located at (5,0), (6,0), (2,1), (5,2), (6,2), (2,3), (3,3), (6,3), (3,6), (0,6) and (3,6), which are represented as red squares. Otherwise, the agent can move in the free space. There are three modes where that the agent can receive reward/cost values. First, the agent will a receive reward of 1 when it arrives at the target. Second, the agent will be punished by receiving -0.001 if hits obstacles or passes the bound. Third, the agent will receive a 0 value as a reward in a free space. At any position in the maze, the agent has to select 1 action (direction) out of 4 actions in order to move one step. The 4 actions are forward, right, backward and left, which can be seen in the figure as u_1, u_2, u_3 and u_4, respectively. 199

21 MSE Learning curves for the SARSA(0), Q(λ = 0.95), ADHDP and NSHDP(λ = 0.95) methods for a 2-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The NSHDP(0.95) method learns the fastest and has the MSE compared to other methods. 202

22 A summation of the accumulative reward for every single episode for the SARSA(0), Q(λ = 0.95), ADHDP and NSHDP(λ = 0.95) methods, which are applied in a 2-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The NSHDP(0.95) method has the largest accumulative reward compared to the other methods in the exploration moving mode. Because the $\epsilon-$greedy deceasing rate learning will reset the accumulative reward value at every episode, the accumulative reward values for the Q(0.95), ADHDP and NSHDP(0.95) are convergence over episodes. 203

23 $\epsilon-$greedy learning curves for the SARSA(0), Q(λ = 0.95), ADHDP and NSHDP(λ = 0.95) methods for a 2-D maze navigation benchmark as shown in Fig. 20. These curves represent the number of steps per episode, where the agent returns to the starting point only when it reaches the target cube. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The NSHDP(0.95) and the ADHDP have a nearly similar number of steps over episodes, which are less than both the Q(0.95) and the SARSA(0). ... 203

PAPER VI

1 Schematic diagram for the adaptation of a novel online n-step value gradient learning (NSVGL(λ)). Two critic networks and one actor network are used in NSVGL(λ). A combination of two critic networks is presented to speed up the tuning for online learning without needing the initial backup value function and eligibility trace parameters. The weights for the one-step critic network ($\hat{G}^0\big(x_k, \hat{\omega}_c^0\big)$) and the n-step critic network ($\hat{G}^\lambda\big(x_k, \hat{\omega}_c^\lambda\big)$) are updated according to gradient of TD(0) error (blue dashed line) and gradient of TD(λ) error (green dashed line), respectively. The actor network ($\hat{A}\big(x_k, \hat{\omega}_a\big)$) is tuned by two paths (backpropagating errors): one through $\hat{G}^0\big(x_k, \hat{\omega}_c^0\big)$ path (e_a^0) and the other through $\big(\hat{G}^\lambda\big(x_k, \hat{\omega}_c^\lambda\big)$ path (e_a^λ). This strategy can correlate and combine the information from the two critic networks. These two paths are filtered via the same values of λ, and they are added together to produce a total backpropagating actor error (red dashed line). 213

2 General feed-forward neural network, which is used in the one-step critic network, the n-step critic network, and the actor network in NSVGL(λ). 234

3 The mean-squared-error (MSE) comparisons over iteration among NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$) and the DHP approaches. The mean values from 10 independent runs are taken for all methods. The shaded region represents 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) provides a faster learning speed than DHP. 241

4 The control input during iteration for NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. 242

5 The state trajectories for the $x1$ state during iteration for NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) improves faster than DHP. . 242

6 The state trajectories for the $x2$ state during iteration for NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) improves faster than DHP. . 243

7 The average gradient trajectories of the value functions for the first system state in NSVGL($\lambda = 0.99$. The gradient of the cost functions $\left(g_i^0(x1_k)\text{ and }g_i^\lambda(x1_k)\right)$ converge to the optimal value function. 244

8 The average gradient of the second cost function trajectories for the both critic networks in the NSVGL($\lambda = 0.99$) approaches. The convergence of the gradient of the value functions for the second system state $\left(g_i^0(x2_k)\text{ and }g_i^\lambda(x2_k)\right)$ to the optimal cost function is clearly shown starting from iteration number 200. 244

9 The average mean-squared-error for two velocity states during iterations among NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) illustrates faster learning than DHP. 248

10 The first control input (left torque) during iterations among NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$), and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. 249

11 The second control input (right torque) during iterations among NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$), and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. 250

12 Comparisons of state trajectory for the first state (linear velocity) during iterations for NSVGL(λ = 0.99), NSVGL(λ = 0.5), and DHP approaches. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) has better performance than DHP, whereas it is faster improved during iteration. 251

13 Comparisons of the state trajectories for the second state (angular velocity) for NSVGL(λ = 0.99), NSVGL(λ = 0.5), and DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) learns faster than DHP. 252

14 The average gradient of the first state of value function trajectories for both critic networks in the NSVGL(λ = 0.99). The gradient of the value functions ($g_i^0(x1_k)$ and $g_i^\lambda(x1_k)$) are converged to the optimal cost function. 253

15 The average gradient of the second state of the value function trajectories for both critic networks for NSVGL(λ = 0.99). 253

PAPER VII

1 A schematic diagram for the adaptation of actor-critic VGL(λ). The weights for the critic network ($\hat{G}\big(x(k), \hat{\omega}_c\big)$) are updated according to the gradient of the TD(λ) error (black dashed line). The actor network ($\hat{A}\big(x(k), \hat{\omega}_a\big)$) is tuned by backpropagating the actor error (e_a) through $\hat{G}\big(x(k), \hat{\omega}_c\big)$ network (red dashed line). 270

2 A schematic diagram for the adaptation of a single n-step critic network of VGL(λ)(SNVGL(λ)). The weights for the critic network ($\hat{G}\big(x(k), \hat{\omega}_c\big)$) are updated according to the gradient of the TD(λ) error (black dashed line). The general n-step optimal control equation (14) (or (73) for Affine systems with the quadratic form of a utility function) is used to generate an optimal control signal. 273

3 A structure for RNF that uses both actor and critic networks for VGL(λ) and NSVGL(λ). RNF consists of three layers. The first layer is the premise layer, which has premise parameters ($\mathfrak{m}_{\mathfrak{n}\times m}, \boldsymbol{\sigma}_{\mathfrak{n}\times m}, \boldsymbol{\theta}_{\mathfrak{n}\times m}$). The second layer is the rule layer, which has $m^{\mathfrak{n}}$ rule nodes. The third layer is the consequent/output layer, which has consequent parameters ($\boldsymbol{\omega}_{p\times m^{\mathfrak{n}}}$). The premise and consequent parameters are trained by using a backpropagation gradient algorithm. To simplify the appearance of this diagram, the number of inputs, MFs and outputs are represented by $\mathfrak{n} = 3$, $m = 2$ and $p = 2$, respectively. 278

4 A structure for TSRNF that uses VGL(λ) and NSVGL(λ). TSRNF consists of five layers. The first layer represents the premise layer, which has premise parameters ($\mathfrak{m}_{\mathfrak{n}\times m}$, $\boldsymbol{\sigma}_{\mathfrak{n}\times m}$, $\boldsymbol{\theta}_{\mathfrak{n}\times m}$). The second layer is the rule layer, which has $m^{\mathfrak{n}}$ rule nodes. The third layer is the normalization layer to normalize the output values coming from the rule layer. The fourth layer is the consequent layer, which has consequent parameters ($\boldsymbol{a}^{m^{\mathfrak{n}}}_{p\times(\mathfrak{n}+1)}$). The fifth layer is the output layer to obtain the final output signal. The premise and consequent parameters are trained by using a backpropagation gradient algorithm. The number of inputs, MFs and outputs are represented by $\mathfrak{n} = 3$, $m = 2$ and $p = 2$, respectively. 283

5 Average of MSEs for the two states for comparisons RNF-VGL($\lambda = 0$) and RNF-VGL($\lambda = 0.98$) with an impact on disturbances and frictions. Five independent runs are taken. The shaded region represents the runs, while the solid line represents the mean of runs. RNF-VGL($\lambda = 0.98$) allows for faster learning than RNF-VGL($\lambda = 0$). 301

6 Initial and final learned Gaussian membership functions (GMFs) for both input states (linear and angular velocities) to the critic network in RNF-VGL(0). 301

7 Initial and final learned GMFs for both linear and angular velocity input states to the actor network in RNF-VGL(0). 302

8 Deviation of the recurrent parameters ($\boldsymbol{\theta}$) for both input velocity states from the initialized values of the critic network of RNF-VGL(0) during all of the total training steps, which is 12000 iteration with 100 time steps each. 303

9 Deviation of the recurrent parameters ($\boldsymbol{\theta}$) for both input velocity states from the initialized values of the actor network of RNF-VGL(0) during the total training steps. 304

10 Initial and final learned GMFs for both linear and angular velocity input states to the actor network in RNF-VGL(λ). 305

11 Initial and final learned GMFs for both input states to the actor network in RNF-VGL(λ). 305

12 Deviation of θ for both input velocity states from the initialized values of the critic network of RNF-VGL(λ) during all the total training steps. 306

13 Deviation of θ for both input velocity states from the initialized values of the actor network of RNF-VGL(λ) during the total training steps. 307

14 Average of MSEs for the two states for comparisons: RNF-VGL(λ) and RNF-SNVGL(λ) with $\varrho = 0.001$. Five independent runs are taken. The shaded region represents the runs, while the solid line represents the mean of the runs. RNF-SNVGL(λ) learns faster a faster learning than RNF-VGL(λ). 309

15 The X-Y circle trajectories for RNF-VGL(λ) and RNF-SNVGL(λ) with ϱ = 0.001. The mean of five independent runs is shown. RNF-SNVGL(λ) is faster, but RNF-VGL(λ) performs better and improves with long training iterations. .. 310

16 The average of the right and left input torques for the dynamic mobile robot for RNF-VGL(λ) and RNF-SNVGL(λ) with ϱ = 0.001. The shaded region represents the runs, while the solid line represents the mean of the runs. 311

17 The average gradient of the two input velocity states of the value function trajectories for the critic network in both RNF-VGL(λ) and RNF-SNVGL(λ). ... 312

18 The average of the MSEs for comparing the two states of TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with ϱ = 0.001. Five independent runs are taken. The shaded region represents the runs, while the solid line represents the mean of the runs. TSRNF-SNVGL(λ) learns faster than TSRNF-VGL(λ). 314

19 The X-Y circle trajectories for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with ϱ = 0.001. The mean of five independent runs is shown. TSRNF-SNVGL(λ) learns faster, but TSRNF-VGL(λ) perfoms better ad improves with long training iterations. 315

20 The average of the right and left input torques to the dynamic mobile robot for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with ϱ = 0.001. The shaded region represents the runs, while the solid line represents the mean of the runs. 316

21 The average gradient of the two input velocity states of the value function trajectories for the critic network in both TSRNF-VGL(λ) and TSRNF-SNVGL(λ). 317

LIST OF TABLES

Table Page

PAPER I

1 GA operations selection approach 18

2 Poles values for unstable and stable for triple link inverted pendulum model 33

3 List of abbreviations and symbols 46

PAPER II

1 Right side controller rules matrix 63

2 Left side controller rules matrix 63

PAPER III

1 Parameters of the dynamic mobile robot 103

PAPER IV

1 Performance evaluation of HDP(λ) learning controller when balancing the inverted pendulum dynamic system. The second and third columns depict the average number of trials it took to learn to balance the pole for 1000 time steps for HDP and HDP(λ) approaches, respectively. The average is based on 100 successful runs at 1000 iterations each. * actuators are subjected to noise; $^{\#}$ sensors are subjected to noise 146

PAPER V

1 Performance evaluation of the NSHDP(λ) learning controller when balancing the inverted pendulum dynamic system. The third and fourth columns depict the average number of trials it took to learn to balance the pole for 100 time steps for the GRHDP and the NSHDP(λ), respectively. The average is taken over 100 successful runs for 5 iterations each. * actuators are subjected to noise; $^{\#}$ sensors are subjected to noise 197

PAPER VI

1 Parameters of the dynamic mobile robot 255

PAPER VII

1 The values for α_i^ℓ. α_i^ℓ depend on i and ℓ. The rows for the last column ($i = \mathfrak{n}$) increase by one with each ℓ and replicate each m. The rows for the column with $i = \mathfrak{n} - 1$ increase by one each with each $\ell = m$ and republicate each m^2 and so on. 275

2 Parameters of the dynamic mobile robot.
ϱ is various random values to test the dynamic model performance in different frictions 299

3 MSE values for RNF-VGL(λ) and RNF-SNVGL(λ) with and without θ recurrent parameters at various noise levels (different ϱ values). All-iteration (All Iter) is the average of 12000 MSE, while final-iteration (Final Iter) is the MSE at the last iteration (12000th). 313

4 MSE values for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with and without θ recurrent parameters at various noise levels (different ϱ values). All-iteration (All Iter) is the average of 12000 MSE, while final-iteration (Final Iter) is the MSE at the last iteration (12000th). 318

SECTION

1. INTRODUCTION

1.1. SYSTEMS WITH REDUCING COMPLEXITY

Because there are numerous physical systems have high order mathematical models in real world, these systems require a massive of a computational complexity to address, solve and simulate. Many algorithms are used to make these systems less complicated, while retaining the properties of the original system. These algorithms have a capability of simulating affordable prototypes with fast and reliable responses. Model order reduction (MOR) is applied in many fields, such as computational biology, mechanics, fluid dynamics, circuit design, and control systems. In this dissertation, the reduction focuses on control area, which analyzes the characteristics and features of dynamic systems to reduce their complexity while keeping their properties as possible as illustrated in the first paper in the paper section of this dissertation.

1.2. ADVANCED ADAPTIVE DYNAMIC PROGRAMMING

Adaptive dynamic programming (ADP) is a powerful tool that allows an agent to learn by interacting with its environment to obtain an optimal control policy. The ADP technique uses a heuristic method to overcome a nonlinearity behavior system that generates a difficulty to solve the Hamilton-Jacobi-Bellman equation instead of the Riccati equation. The ADP technique allows agents to select an optimal action to minimize their long-term cost value by solving the Bellman equation. A heuristic dynamic programming, a dual heuristic programming and a globalized dual heuristic programming are three fundamental categorizes for ADP technique. These categorizes consist of three approximation function networks, which are actor, critic and model networks that provide decision making, evaluation, and prediction, respectively. Because a model network, which predicts the future system state, is included within these categorize, the ADP categorizes as a model-based ADP design. If the action-dependent (AD) expression is used in the ADP, then the critic network has the state and the action inputs. A model-free ADP design has been presented for online learning, which is not required the model network. Many applications have used the ADP techniques. A temporal-difference (TD) with eligibility trace parameter is a more advanced learning algorithm than the traditional TD that combines basic TD learning with an eligibility traces technique to further accelerate learning. The ADP technique is used to train an actor network to give optimal actions based on minimizing a value function that is produced from a critic network. In this dissertation, all networks are approximated by using a multilayer perceptron neural network, and hybrid neuro-fuzzy networks. We investigate in ADP with advanced TD learning and new novel structures that make system more robust, fast and stable during training as presented in the second paper until seventh paper in paper section of this dissertation.

1.3. RESEARCH CONTRIBUTIONS

This dissertation deals with the use of reducing complexity of models and applying feature forward and backward views of eligibility trace procedures with ADP in various benchmarks tasks. In concrete, we describe each paper's contribution as follows:

1.3.1. Model Order Reduction Based on Agglomerative Hierarchical Clustering. The main contribution in this work is provided a model order Reduction (MOR) technique that gives any required order of reduced model with a minimum MSE value. Instead of neglecting some poles like traditional methods, our approach engages all properties of the original system by using agglomerative hierarchical clustering of system poles depending on a performance evaluation. Therefore, the method will be called HC-PE. HC-PE is effective for converting original high order ordinary differential equations to low order equations. HC-PE with PA or GA takes the output response(s), and it calculates the MSE between the original model and the reduced model. It uses an improved modified pole clustering center in every selected pole-cluster. The pole-clusters for the original system are selected by using a performance evaluation method as a similarity criteria in agglomerative hierarchical clustering. This gives a major advantage in minimizing error between the reduced and original models. Optimizing is achieved by the pole-clusters taking the minimum MSE among all pole-clusters on a certain level in the hierarchy dendrogram. The hierarchy starts from the bottom (n^{th} order original system), merging pairs or more pole clusters at each move up until the 2^{nd} order. HC-PE deals with denominator parameters of ordinary differential equations (transfer function) for the reduced order model while PA or GA addresses the numerator parameters. By combining these two parts, we get the best performance behavior. In other words, in addition to the optimal best minimum error, HC-PE with PA (or GA) still retains stability and robustness for the reduced model

1.3.2. Heuristic Dynamic Programming for Mobile Robot Path Planning Based on Dyna Approach. The main contribution in this work is provided a combination between direct heuristic dynamic programming (HDP) and Dyna planning (Dyna-HDP). This combination provides the fast online free-model learning comparing with other traditional reinforcement learning algorithms (one step Q-learning, SARSA, Q(λ), SARSA(λ), and Dyna-Q). Whereas, this work compares these algorithms with Dyna-HDP for control of a differential-drive wheeled mobile robot navigation problem in an unknown two-dimensional indoor environment. A Second contribution in this work is merge a fuzzy Logic Controller (FLC) with Dyna-HDP to provide a collision-free navigation path for instead of staring from initial position similar a regular reinforcement learning algorithms.

1.3.3. Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning. The main contribution in this work is used a combination of eligibility trace parameter in dual heuristic dynamic programming with a first-order Sugeno fuzzy neural network structure. This combination is used with both critic and actor networks.

This approach is used to track a reference trajectory under uncertainties by computing the optimal left and right torque values for a nonholonomic mobile robot. The impacts of unmodeled bounded disturbances with various friction values is handled with a significant enhancement of the robot's capability to absorb unstructured disturbance signals and friction effects. Because of affine dynamic model for nonholonomic mobile robot, we use a critic only to calculate a optimal control signal to reduce a computational complexity with faster responses without needs an neural network identifier for system.

1.3.4. The Boundedness Conditions for Model-Free HDP(λ). This work overcomes the drawback of using eligibility-trace storage in backward view property. Thus, simplicity and performance is the first contribution of this work. The second contribution is providing a stability proof to determine what suitable learning parameters (λ, γ and critic/actor learning rates) should be used during training. Under certain conditions, we use the Lyapunov theory to prove stability for the specific case of HDP(λ). We extend the stability of model-free learning only for the one-step ($\lambda = 0$) HDP(0) approach into HDP with a general λ parameter.

1.3.5. Online Model-Free N-Step HDP with Stability Analysis. A simple interpretation and good performance are two well-known properties attached with TD(λ) approach (eligibility trace temporal difference learning). But this approach suffers from using an additional memory variable associated with each state to store the eligibility trace parameter; therefore, a high computational complexity is adjoined with. Our previous work (previous paper) solved this problem but for batch-implementation learning at least for first epoch. The work is designed is used for online-implementation learning. Thus, our structure in this work has memory efficient since it overcomes the drawback of using eligibility-trace storage and online learning. The online learning aspect with low computational is the first contribution for this work. The second contribution is that it provided stability proofs to present what a suitable learning parameters (λ, γ and critic/actor learning rates) should be during training.

1.3.6. An Improved N-Step Value Gradient Learning Adaptive Dynamic Programming Algorithm for Online Learning, with Convergence Proof and Case Studies. The fundamental contributions of this paper are as follows: First, The theoretical foundation analysis for NSVGL(λ) architecture is presented designing how the agent receives better information about the control action than traditional DHP. Memory efficiency is provided by NSVGL(λ) via online learning in contest with online VGL(λ) that uses a matrix for eligibility trace parameters to store every signal state trajectory. Second, a theoretical convergence analysis is provided for the NSVGL(λ) structure. Gradients of the one-step and *n*-step value functions are learned. We demonstrate that both gradients are monotonically nondecreasing and converges to their optimal values. These contributions are verified by simulation in two case studies with provindig a Pseudocode of NSVGL(λ).

1.3.7. Convergence Analysis Proofs for Recurrent Neuro-Fuzzy Value-Gradient Learning with and without Actor. The main contribution in this work are: First, the theoretical foundation analysis for *n*-step adaptive actor-critic approach of VGL(λ) architecture with NF (NF-VGL(λ)) is presented that illustrate how the agent receives better information about the control action than traditional DHP. Second, the single adaptive *n*-step critic approach of VGL(λ) (SNVGL(λ)) is derived to created a pioneer architecture of SNVGL(λ). SNVGL(λ) uses NF structures (NF-SNVGL(λ)) to compare with first contribution. Third, a theoretical convergence analysis is provided for the VGL(λ) and SNVGL(λ) architectures by using iterative ADP algorithm. We demonstrate that gradient are monotonically nondecreasing and converges to optimal values. Final, these advantages of VGL(λ) and SNVGL(λ) with and without recurrent feedback parameters are verified by simulation with high-nonlinear dynamic model case study with various uncertainties.

PAPER

I. MODEL ORDER REDUCTION BASED ON AGGLOMERATIVE HIERARCHICAL CLUSTERING

S. Al-Dabooni and Donald C. Wunsch

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

ABSTRACT

This paper presents an improved method for reducing high-order dynamical system models via clustering. Agglomerative hierarchical clustering based on performance evaluation (HC-PE) is introduced for model order reduction (MOR). This method computes the reduced order denominator of the transfer function model by clustering system poles in a hierarchical dendrogram. The base layer represents an n^{th} order system, which is used to calculate each successive layer to reduce the model order until finally reaching a second order system. HC-PE uses a mean squared error (MSE) in every reduced order, which includes a modified pole placement process. The coefficients for the numerator of the reduced model are calculated by using the Pade approximation (PA) or alternatively a genetic algorithm (GA). Several numerical examples of reducing techniques are taken from the literature to compare with HC-PE. Two classes of results are shown in this work. The first sets are single-input single-output (SISO) models that range from simple models to 48th order systems. The second sets of experiments are with a multi-input multi-output (MIMO) model. We demonstrate the best performance for HC-PE through minimum MSEs compared with other methods.

Furthermore, the robustness of HC-PE combined with PA or GA is confirmed by evaluating the 3rd order reduced model for the triple link inverted pendulum model by adding a disturbance impulse signal and by changing model parameters. HC-PE with PA slightly outperforms its performance with GA, but both approaches are attractive alternatives to other published methods.

Keywords: Hierarchical clustering (HC), model order reduction (MOR), Pade approximation (PA), genetic algorithm (GA), triple link inverted pendulum, linear quadratic regulator (LQR), pole replacement.

1. INTRODUCTION

Numerous physical systems have high order mathematical models. Schilders [1] shows many examples for systems that require high complexity computations to address and simulate them. Many algorithms for MOR are used to make these systems less complicated, while retaining the properties of the original system. These algorithms are capable of simulating affordable prototypes with fast and reliable responses. MOR is applied in many fields, such as computational biology, mechanics, fluid dynamics, circuit design, and control systems. This work focuses on MOR for control, which analyzes the characteristics and features of dynamic system models to reduce their complexity while keeping their properties as possible. Sandberg *et al.* [2] showed a variety of MOR algorithms. In this work, a new method for reducing high order system models is presented. Many system model descriptions exist in the literature [3]-[5], such as state space representation and transfer function representation. The roots of the denominator of a transfer function (characteristic polynomial of a system) generate frequency values, which are called poles, while the roots of the numerator are called zeros. A zero-pole representation is a description of a system in terms of the poles and zeros. A high order original system model (G(s)), which is formed by its zero-pole representation is given as follows:

$$G(s) = \mathcal{K}\frac{(s - Z_\upsilon)(s - Z_{\upsilon-1})\ldots(s - Z_2)(s - Z_1)}{(s - p_n)(s - p_{n-1})\ldots(s - p_2)(s - p_1)}, \tag{1}$$

where $\mathcal{K}$ is the gain of the system, s is the Laplace complex variable, Z_i $(i = 1, 2, \ldots, \upsilon)$ are zeros of the system, and p_i $(i = 1, 2, \ldots, n)$ are its poles. In most transfer functions, the degree of the numerator is less than the degree of the denominator $(\upsilon < n)$, which is called strictly proper [4] . A strictly proper transfer function can easily transfer to a state space representation. In this paper, we use a hierarchical clustering technique for clustering the poles to reduce the order of any original model to an r^{th} order reduced model $(G_r(s))$. Therefore, this method takes all original system characteristics by testing all system poles. It is not like traditional methods such as model truncation (Gramians and Hankel singular values) via singular value decomposition [6] , [7] . For instance, the Hankel approach deals with state energy by balancing controllability and observability Gramians to satisfy the Lyapunov equation, and the final state space sorts states according to their energy. There are many reduction techniques in the literature for both time and frequency domains of discrete and continuous linear systems. Sinha and Pal [8] show reduced order modeling based on freely collected clustering of the zeros and poles by calculating the inverse distance measure in the time domain. Pal [9] demonstrates that the reduced order model retains stability by using the PA. Neeraj and Anirudha [10] use fuzzy C-means clustering of system poles to reduce a higher order interval discrete system using the minimum Euclidean distance as a similarity criterion. Reduction in the continuous time domain is discussed by Vishwakarma and Prasad [11] ; they address the original model with state space matrices and modify Hankel matrices consisting of time-moment and Markov parameter elements to less complex matrices through the minimal realization technique. Singh *et al.* [12] present model order reduction in discrete frequency domain through mixing Chebyshev polynomials and the pole clustering method. Beyene [13] used inverse distance measure for pole-clustering and rational-interpolation techniques in frequency and time domain simulations. The main contribution in this work is provided a MOR technique that gives any required order of reduced model with a minimum MSE value. Instead of neglecting some poles like traditional methods, our approach engages all properties of the original system by using agglomerative hierarchical clustering of system poles depending on a performance evaluation. Therefore, the method will be called HC-PE. HC-PE is effective for converting original high order ordinary differential equations to low order equations as shown in Fig. 1. HC-PE with PA or GA takes the output response(s), and it calculates the MSE between the original model and the reduced model. It uses an improved modified pole clustering center in every selected pole-cluster. The pole-clusters for the original system are selected by using a performance evaluation method as a similarity criteria in agglomerative hierarchical clustering. This gives a major advantage in minimizing error between the reduced and original models. Optimizing is achieved by the pole-clusters taking the minimum MSE among all pole-clusters on a certain level in the hierarchy dendrogram. The hierarchy starts from the bottom (n^{th} order original system), merging pairs or more pole clusters at each move up until the 2^{nd} order. HC-PE deals with denominator parameters of ordinary differential equations (transfer function) for the reduced order model, while PA or GA addresses the numerator parameters. By combining these two parts, we get the best performance behavior. In other words, in addition to the optimal best minimum error, HC-PE with PA (or GA) still retains stability and robustness for the reduced model as presented in Appendix A . We demonstrate this by comparing it with several examples of other methods. All abbreviations and symbols used in the paper are summarized in Table 3. The remaining sections are organized as follows: The problem statement is in Section 2. HC-PE, PA, and GA are described in Section 4. Simulation results with numerical examples are in Section 4. The response outputs for the reduced multivariable model of a triple link inverted pendulum controlled by proportional-integral-derivative (PID) controller tuning by GA are shown in Section 5. The conclusion is in Section 6.

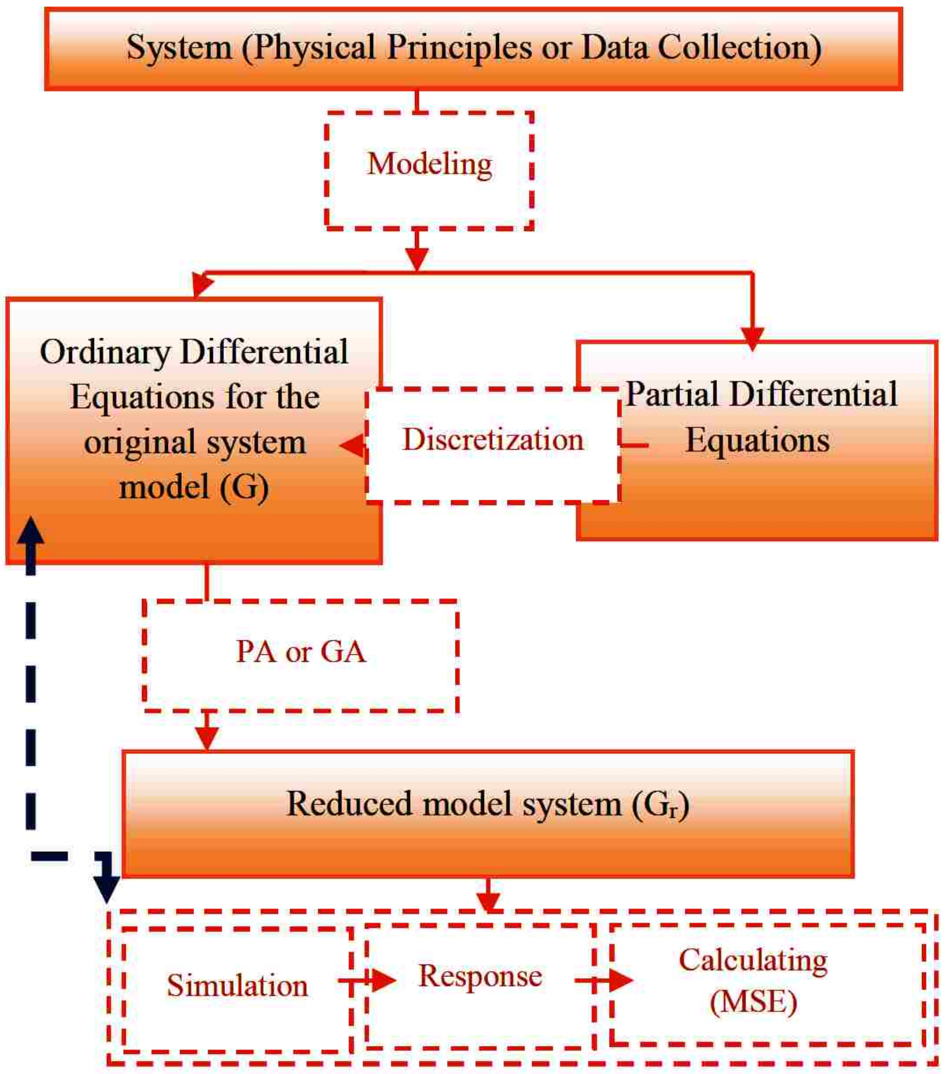

System (Physical Principles or Data Collection)
Modeling
Ordinary Differential Equations for the original system model (G)
Partial Differential Equations
Discretization
PA or GA
Reduced model system (G$_r$)
Simulation
Response
Calculating (MSE)

Figure 1. General diagram for using HC-PE to reduce high order models. Any system model is built by deriving physical laws (physical modeling) or by observing data (identification modeling). Some models are represented by partial differential equations (e.g. heat transfer equation), and other models are built by ordinary differential equations (e.g. robotics). Finite difference techniques are used to discretize partial differential equations to derive a numerical approximation for ordinary differential equations [3]. In this work, we focus on the models, which are built by using ordinary differential equations (G). In combination with PA or GA, HC-PE improves the evaluation for the reduced model (G_r) through selecting minimum MSEs of clusters made from poles. To find the minimum MSE, all MSE values should be calculated for each level (order). MSE is calculated between the original model and the reduced models which is represented by the blue dashed line.

2. PROBLEM STATEMENT

This work uses HC-PE and PA (or GA) to obtain the coefficients for a reduced order model. A continuous linear time invariant n^{th} order model's, transfer function representation is

$$G(s) = \frac{a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \ldots + a_1 s + a_0}{b_n s^n + b_{n-1}s^{n-1} + b_{n-2}s^{n-2} + \ldots + b_1 s + b_0}, \tag{2}$$

where all coefficients are known. Because of the strictly proper transfer function as in (1), we start the degree of the numerator's order from $n-1$. Then, the HC-PE approach reduces the n^{th} order model (2) to an r^{th} order of $G_r(s)$:

$$G_r(s) = \frac{c_{r-1}s^{r-1} + c_{r-2}s^{r-2} + \ldots + c_1 s + c_0}{d_r s^r + d_{r-1}s^{r-1} + d_{r-2}s^{r-2} + \ldots + d_1 s + d_0}, \tag{3}$$

where all coefficients are unknown. HC-PE is used to find the best coefficients that contain the most significant features of the high order model to retain the best time and frequency responses. HC-PE progresses by computing a next reduced model until reaching second order. Specifically, a new $G_{r-1}(s)$ is calculated by finding all unknown coefficients. The HC-PE algorithm considers a new base model (an optimal model of r^{th} order) to calculate the next reduced model of $r^{\text{th}}-1$ order and so on, until reaching the 2nd order. This procedure is significantly affected by computational complexity, which requires simple, near-optimal, and fast selection of clustered poles in stages as shown in the following sections.

3. HIERARCHICAL CLUSTERING SYSTEM POLES ALGORITHM

Hierarchical clustering algorithms organize data into levels according to a proximity matrix; the results of hierarchical clustering are often depicted by a binary tree [14]. Many techniques [15]-[17] determine similarity measures by calculating the distance between data objects. In HC-PE, the MSE of a step response between the reduced model and the original model is the similarity measurement. For comparison purposes, a traditional similarity measure (Euclidean-distance) in hierarchical clustering pole is discussed. There are three cases to be considered during clustering poles by using either the Euclidean-distance or the MSE as a similarity measure: clustering real and complex poles separately, applying a full-state feedback approach (pole placement method) in unstable systems, and retaining the poles in the imaginary axis and in the origin of the s-plane. Fig. 2 depicts these considerations. In two hierarchical pole clustering approaches, the pole index labels, which are mentioned in the binary tree diagram, are sorted in ascending order.

3.1. Choice of Distance Measure.

3.1.1. Hierarchical Cluster Poles Based on Euclidean-distance. The original system poles are taken in a general agglomerative clustering algorithm procedure as in [14], [15]. Collecting pole clustering [8], [12], are used to calculated a center for each cluster. Improvements are then chosen for theses clusters [19]. This algorithm consists of the following steps: First determine a distance between each pair of poles based on Euclidean-distance. Second, iteratively group points into a binary hierarchical tree (using single linkage clustering, see [15]). Third, cut the hierarchical tree depending on the demanded order of the reduced model. Fourth, improve the modified pole clustering based on the most dominant pole in this cluster. More descriptions for the third and fourth steps are given by

1. Follow ascending order for absolute value of the poles in the cluster (real poles $(p_1, p_2, \ldots, p_l)$ or complex conjugate poles $(p_1 \pm I_1, p_2 \pm I_2, \ldots, p_m \pm I_m)$ with $|p_1| < |p_2| < \ldots < |p_h|$, where h is l for real poles and h is m for complex poles. The imaginary parts are ordered according to real part.

2. Estimate a new cluster-pole center. If l poles are real, then the center is

$$q^c = \left(\sum_{k=1}^{l} \left(\frac{-1}{|p_k|} \right) \frac{1}{l} \right)^{-1}, \tag{4}$$

If m poles are complex then the center is

$$p^c \pm iI^c = \left(\sum_{k=1}^{m} \left(\frac{-1}{|p_k|} \right) \frac{1}{m} \right)^{-1} \pm i \left(\sum_{k=1}^{m} \left(\frac{-1}{|I_k|} \right) \frac{1}{m} \right)^{-1}, \tag{5}$$

where p^c is a cluster-center for m real parts of complex poles, and I^c is a cluster-center for m imaginary parts of complex poles.

3. Compute the improved pole cluster center as:

$$R_j = \left(\frac{1}{2} \left(\frac{-1}{|p_1|} + \frac{-1}{|R_{j-1}|} \right) \right)^{-1}, j = 1, 2, \ldots, h, \tag{6}$$

where R_j is q^c for real poles, R_j is p^c for real parts of complex poles and R_j is I^c for imaginary parts of complex poles.

3.1.2. Hierarchical Clustered Poles Based on MSE. The pole clusters for the original system in the HC-PE algorithm are selected by using a performance evaluation method. This method gives a major advantage for reducing error between the reduced and original models, which takes an optimal pole cluster for the original system in any reduced order. The optimality for selecting the pole clusters comes from taking the minimum MSE among all pole clusters in appropriate levels of the hierarchical dendrogram. The n^{th} order original model is located in the bottom of the hierarchical dendrogram. The HC-PE algorithm starts calculating the reduced model of r^{th} order ($n^{\text{th}} - 1$), which becomes the base model (an optimal simulated original model of r^{th} order) for the next level. The next reduced model $r^{\text{th}} - 1$ order is calculated according to the base model and so on until reaching the 2nd order. HC-PE uses the improved modified pole clustering center in every selected cluster. Fig. 3 depicts a flowchart for HC-PE. It starts by clustering the pair of poles. The clustered poles are in ascending order of individual absolute values whether the poles are real or complex. The center for this cluster is estimated by (4) and (5) for real and complex poles, respectively. Improving the cluster center is done by taking an average inverse distance with

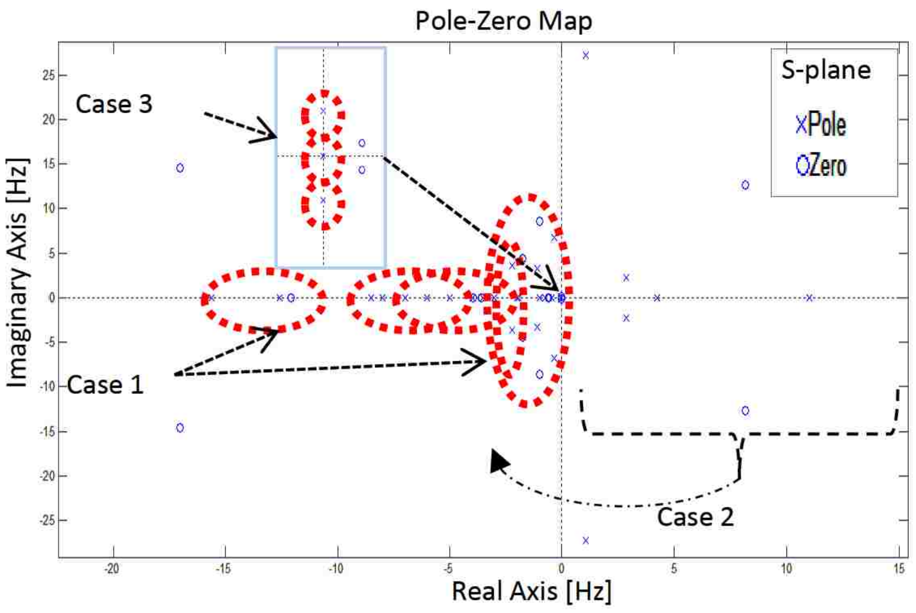


Figure 2. Three cases that should be considered while clustering poles using either Euclidean-distance or MSE as the similarity. The first case is clustering real and complex poles separately. The second case is applying a full-state feedback approach (pole placement method) in unstable systems. The third case is retaining the poles in the imaginary axis and in the origin of the s-plane to the reduced model.

the most dominant pole in this cluster. The most dominant pole is the nearest real pole or real part of the complex pole to the imaginary axis of s-plane. This procedure[1] is repeated twice if the poles are complex conjugates. In other words, the real pole center is improved directly by using (6), while the complex conjugate pole center is improved also by using (6) with a real part and imaginary part separately. When the improved center for this cluster is found (j=h+1), the MSE is calculated for this cluster and then repeated with other clusters in the same level of order in the hierarchy until all cluster options (z=n-1) are completed. The new reduced order model is selected based on the minimum MSE among all MSEs for z clusters as follows:

$$N^{z}_{cluster} = argmin_{MSE}\left(\cup^{z}_{j=1} M^{j}_{cluster}\right), \tag{7}$$

[1]The time complexity of sorting operation (e.g. merge sort method) for n poles is $O(nlog(n))$, and of calculating improved cluster center for both real and complex conjugate poles is $O(n)$.

where z is the number of cluster poles at certain levels in the binary tree, $M_{cluster}^{j}$ is a MSE for j^{th} cluster. Equation (7) is used with a single-input single-output (SISO) models. This new reduced model becomes the base model to calculate the next reduced model. Appendix A illustrates that a new improved center cluster poles have negative bounded values; therefore the new reduced model is asymptotically stable (or marginally stable if some poles are located on the imaginary axis).

3.2. Determining Transfer Function Coefficients. Whereas the reduced model has denominator polynomial coefficients obtained from hierarchical cluster poles, coefficients for the numerator are determined by using PA or GA as follows:

3.2.1. PA Method. As in D. Xue et al. [4], expand the original system $G(s)$ in a Taylor series around $s = 0$ as follows:

$$G(s) = \sum_{i=0}^{\infty} \left(T_i s^i\right), \tag{8}$$

where T_i is defined as:

$$T_i = \frac{1}{i!} \frac{d^i G(s)}{ds^i}. \tag{9}$$

This model can be equivalently obtained from state space matrices as follows:

$$G(s) = C(sI - A)^{-1} B + D, \tag{10}$$

where $A \in \mathbb{R}^{n \times n}$ is the state matrix; $B \in \mathbb{R}^{n \times \hat{n}}$ is the input matrix, $C \in \mathbb{R}^{\hat{m} \times n}$ is the output matrix, $D \in \mathbb{R}^{\hat{m} \times \hat{n}}$ the direct transition (or feedthrough) matrix, n is the number of system states (i.e., it represents an n^{th} order differential equation), $\hat{n}$ is the number of system inputs, and $\hat{m}$ is the number of system outputs. The time moment, T_i can be also calculated by using A and C matrices as follows:

$$T_i = -CA^{-(i-1)} B, i = 1, 2 \ldots, \infty. \tag{11}$$

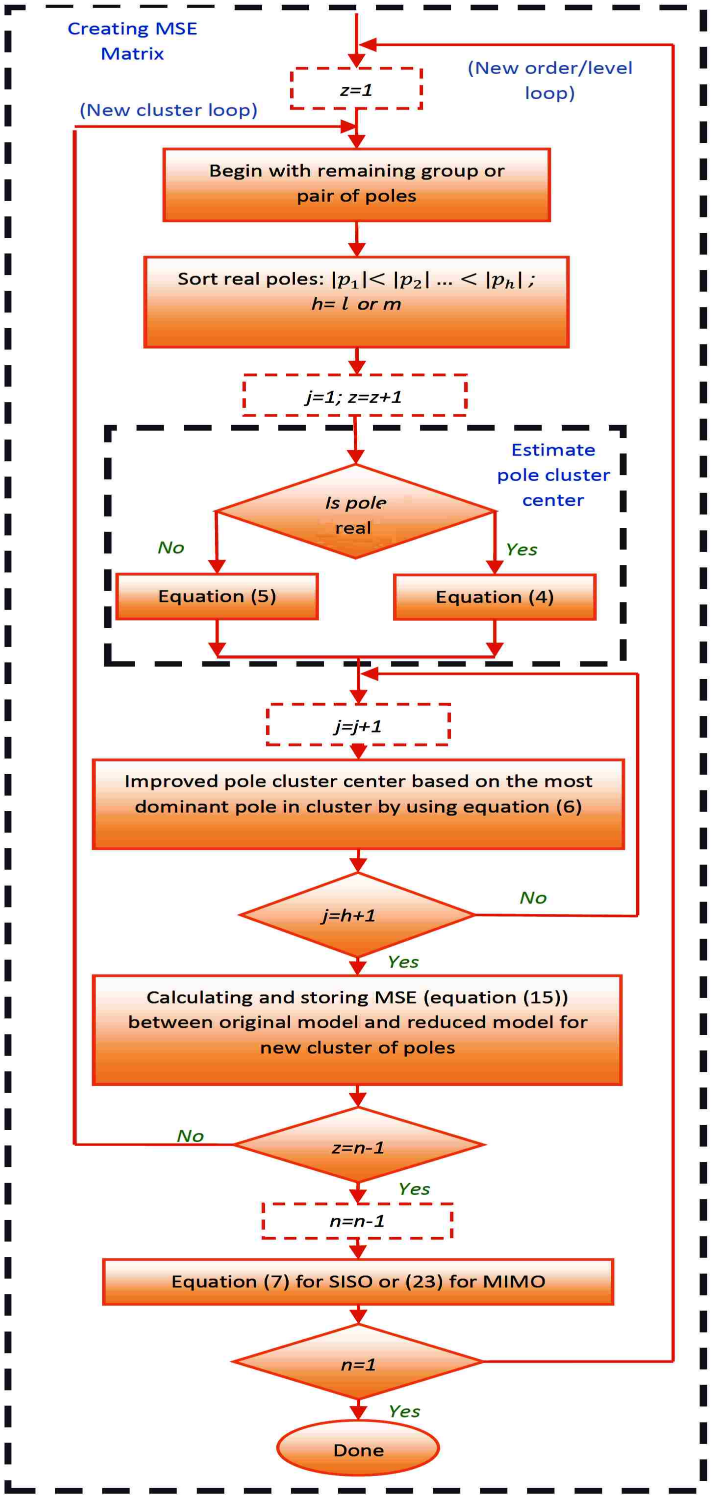

Creating MSE Matrix
(New order/level loop)
z=1
(New cluster loop)
Begin with remaining group or pair of poles
Sort real poles: $|p_1| < |p_2| \ldots < |p_h|$; $h = l$ or m
j=1; z=z+1
Estimate pole cluster center
Is pole real
No
Yes
Equation (5)
Equation (4)
j=j+1
Improved pole cluster center based on the most dominant pole in cluster by using equation (6)
j=h+1
No
Yes
Calculating and storing MSE (equation (15)) between original model and reduced model for new cluster of poles
z=n-1
No
Yes
n=n-1
Equation (7) for SISO or (23) for MIMO
n=1
Yes
Done

Figure 3. Flowchart for the HC-PE algorithm, which starts from n^{th} order and calculates the reduced model in $r^{\text{th}} = n^{\text{th}} - 1$ order, which becomes the base model (an optimal simulated original model at r^{th} order) to calculate a next reduced model $n^{\text{th}} - 1$ order and so on until reaching the 2nd order.

Singh *et al.* [19] present another expansion for $G(s)$ in a Taylor series around $s = \infty$ (the high frequencies) :

$$G(s) = \sum_{i=0}^{\infty} \left(\delta_i s^{-(i+1)}\right), \tag{12}$$

where δ is a Markov parameter. The Markov parameter is calculated by using state space matrices [4]

$$\begin{aligned} \delta_0 &= CB + D, \\ \delta_i &= CA^i B, i = 1, 2, \ldots, \infty. \end{aligned} \tag{13}$$

The coefficients of the reduced order numerator are evaluated [9]

$$\begin{aligned} c_0 = d_0 T_0;\ c_1 = d_0 T_1 + d_1 T0;\ \ldots;\ c_{\alpha-1} = d_0 T_{\alpha-1} + d_1 T_{\alpha-2} + \ldots + d_{\alpha-1} T_0; \\ c_{r-\beta} = d_r \delta_{\beta-1} + d_{r-1} \delta \beta - 2 + \ldots + d_{k-\beta+1} \delta_0;\ \ldots;\ c_{r-1} = d_r \delta_0, \end{aligned} \tag{14}$$

where α is the number of time moment parameters, and β is the number of Markov parameters. From (14), the coefficients of the numerator are known; therefore all coefficients required to represent a reduced order model are known. Kalaiselvi and Pratheep [20] expand PA method by using a Kharitonov theorem. In this paper, we use α=r (the required order of the reduced model) and β=0 for all testing models except for the building model of Section 4.B. The reason for selecting α=r and β=0 is that selections give the best performance as illustrated in [19]. Therefore, we use also these selections in a triple link inverted pendulum benchmark study case in Section 5. For the building model, a PA method was not used in the literature to reduce this model; therefore, we use a fair distribution into α and β values by selecting α=β=r/2.

3.2.2. GA Method. A GA is used to calculate the numerator polynomial coefficients for the reduced model for comparison with PA. MSE is used as a fitness function for the step response of the original and reduced model as:

$$MSE = \frac{1}{V} \sum_{i=1}^{V} \left(G(s)_i - G_r(s)_i\right), \tag{15}$$

Table 1. GA operations selection approach

GA Properties	Description
Number of Population	20 chromosomes for each generation
Selection Operator	Roulette
Crossover Operator	Stochastic uniform
Mutation Operator	Uniform (uniform random flipping)
Termination GA	no improvement for 100 generations

where V is the number of samples with 0.001 sec as a sample time during 10 sec. Table 1 illustrates the GA operators used in this paper.

4. SIMULATION AND ANALYSIS RESULTS WITH NUMERICAL EXAMPLES

4.1. Comparative Studies From Published Literature.

Several high order model examples are taken from the literature in order to compare and test the effectiveness of HC-PE. From [19] and [21], the original 10 pole model is given as

$$G_{10}(s) = \frac{N_{10}(s)}{D_{10}(s)}, \tag{16}$$

where $N_{10}(s)$= $5.407e19$, and $D_{10}(s)$ = s^{10} + $1800s^9$ +$1.37e6s^8$ + $5.76e8s^7$ + $1.45e11s^6$ + $2.27e13s^5$ + $2.14e15s^4$ + $1.15e17s^3$ + $3.13e18s^3$ + $3.24e19s$ + $5.407e19$. The dendrogram after applying hierarchical clustering of poles based on the Euclidean-distance combined with the PA approach is shown in Fig. 4. According to MSE values, the system is unstable for 6th, 7th, and 8th order reduction when cutting in these levels. The error becomes large in 3th, 4th, and 5th. Fig. 5 illustrates the dendrogram after applying the HC-PE algorithm combined with the PA approach (HC-PE-PA). Minimum error and stability in all order reduced models are generated by using the HC-PE algorithm, which helps to select any level (order of reduced model) is needed. For instance, in comparing [19] and [21], a 2nd order reduced model was found; the dendrogram at the 2nd level is cut as shown in Fig. 5. The 2nd order reduced model from [19] is given as

$$G_r(s) = \frac{-50.346s + 432.4174}{s^2 + 209.0177s + 432.4174}, \tag{17}$$

which has MSE = $2.05e-04$ for a 10 sec of step input. The 2nd order reduced model from [21] is given as:

$$G_r(s) = \frac{-28.3902s + 647.6004}{s^2 + 359.999s + 647.6019}, \tag{18}$$

which has MSE= $1.53e-04$ for a same input. The 2nd order reduced model from HC-PE-PA is:

$$G_r(s) = \frac{-2.0549s + 37.3859}{s^2 + 20.3682s + 37.3857}, \tag{19}$$

which has MSE=$7.48e-06$ for same input. Therefore, HC-PE-PA has the best performance compared with [19] and [21]. Fig. 6 shows a comparison of the step responses for the original, [19], [21], and HC-PE-PA. A second example is taken from [19]. The original another high order model with eight poles is given

$$G_8(s) = \frac{N_8(s)}{D_8(s)}, \tag{20}$$

where $N_8(s) = 18s^7 + 514s^6 + 5982s^5 + 36380s^4 + 122664s^3 + 222088s^3 + 185760s + 40320$, and $D_8(s) = s^8 + 36s^7 + 546s^6 + 4536s^5 + 22449s^4 + 67284s^3 + 118124s^3 + 109584s + 40320$. This system demonstrates another limitation of using Euclidean-distance to cluster the poles because the Euclidean-distances among all poles in (20) have the same value. In contrast, Fig. 7 shows MSE levels with stability in all orders of the reduced model that is generated by using the HC-PE-PA approach. The 3rd reduced model from [19] is given as:

$$G_r(s) = \frac{15.56s^2 + 62.64s + 18.43}{s^3 + 10.16s^2 + 27.8s + 18.43}, \tag{21}$$

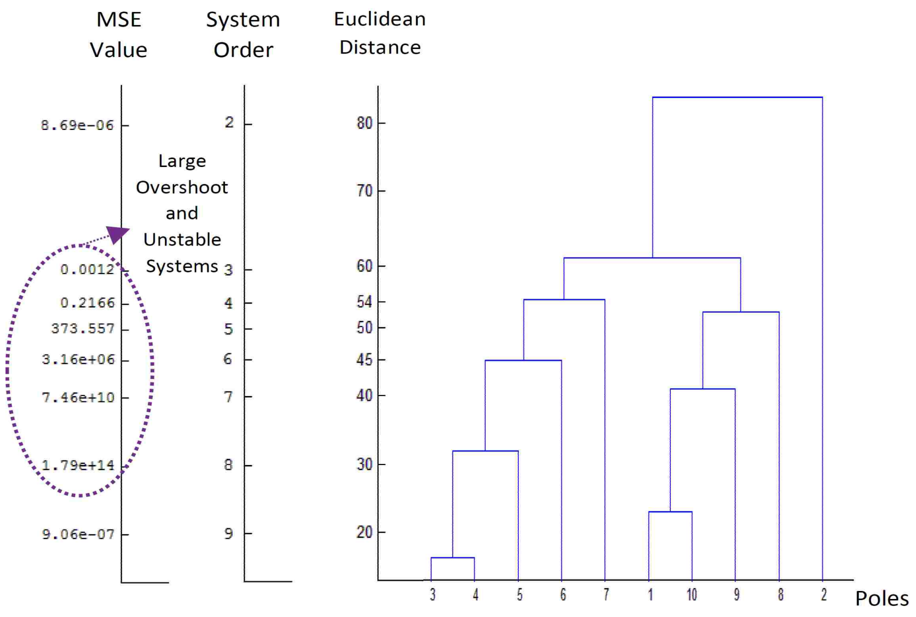


Figure 4. The dendrogram for hierarchical cluster poles based on Euclidean distance for $G_{10}(s)$ combined with the PA approach. It clearly has large error values in most reduction orders.

which has $MSE = 7.98e-04$ for 10 sec step input. The 3rd order reduced model from the HE-PE-PA algorithm is:

$$G_r(s) = \frac{17.64s^2 + 49.77s + 14.1}{s^3 + 10.01s^2 + 23.12s + 14.1}, \tag{22}$$

which has $MSE = 3.96e-06$ with same input. Therefore, HC-PE-PA has the best performance compared to [19]. Fig. 8 and Fig. 9 show a comparison to [19], and HC-PE-PA in the step responses and Bode diagram, respectively.

A comparison with other works from the literature is illustrated in Appendix C, which shows that HC-PE combined with GA (HC-PE-GA) and and PA (HC-PE-PA) has the best performance.

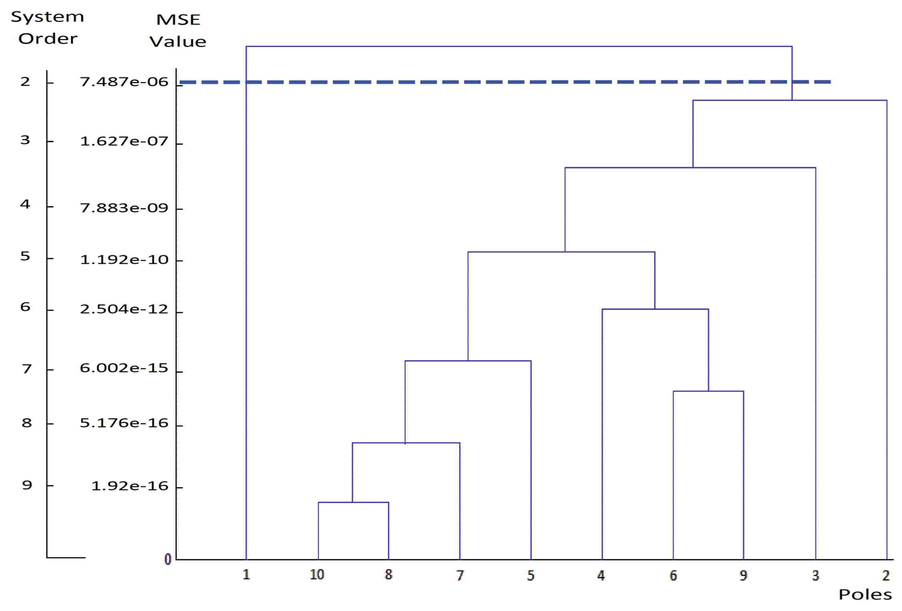


Figure 5. The dendrogram for hierarchical clusters poles based on the HC-PE-PA algorithm for $G_{10}(s)$. Minimum error and stability in the levels are generated by using this algorithm. Cutting is done at level two to generate the 2nd order reduced model for comparing [19] and [21].

4.2. Case Study for Large Order System (Building Model Structure). Another numerical example is presented to examine HC-PE performance. A building model (Los Angeles University Hospital) is reduced by using HC-PE-PA and HC-PE-GA, and the results are compared with two other familiar reduction techniques (Hankel and balanced truncation). The structure building model and reduction techniques are described by Antoulas [22], and by P. V. Dooren and Y. Chahlaoui [22]. The model of building consists of 8 floors each having 3 degrees of freedom, which are displacement in x axis, displacement in y axis, and rotation around y axis. A second-order differential equation system of 24 variables generates 48th order in its state space representation. The transfer function for this system is obtained after applying the state space representation for building model (10). The SISO building model consists of a first variable (first state) for the input of the model. The derivative of the first variable (25th state) is the output.

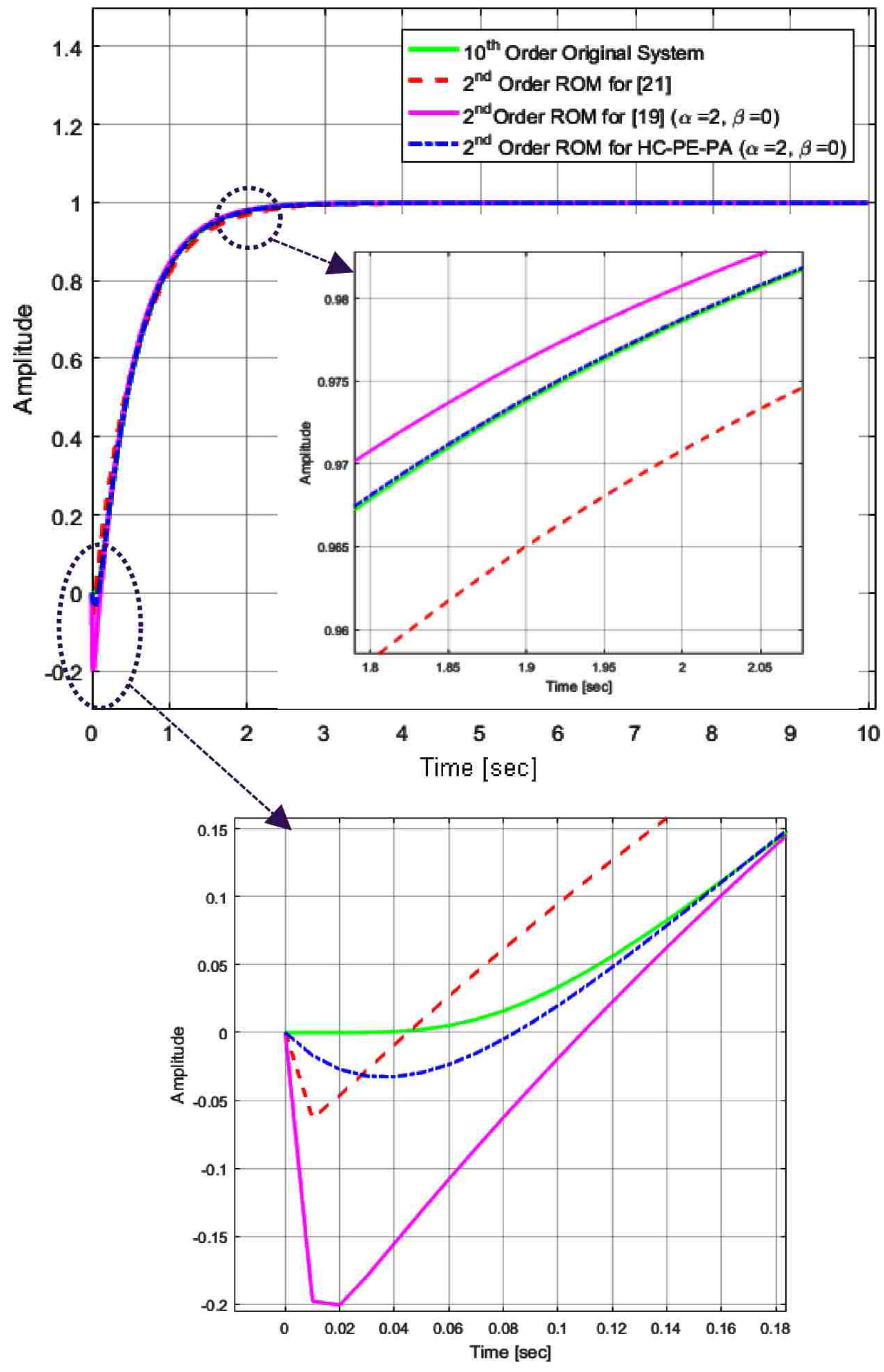


Figure 6. The Time Step response for step responses for original, [19] and [21], and HC-PE-PA.

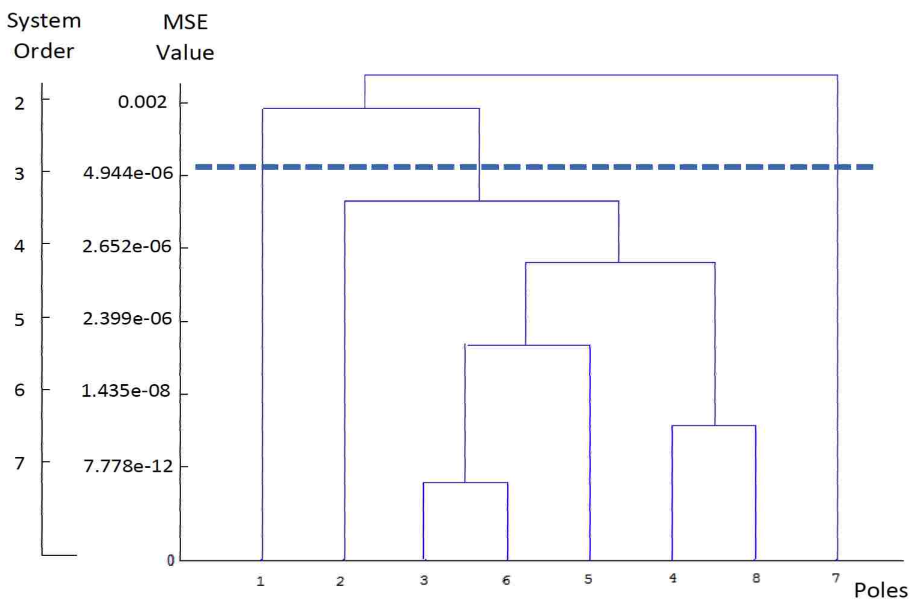


Figure 7. The dendrogram for hierarchical clustering of poles based on the HC-PE-PA algorithm for $G_8(s)$. Stability and fluent minimum error tracking level is generated by using this algorithm. Level three was cut to generate a 3rd order reduced model to compare with [19].

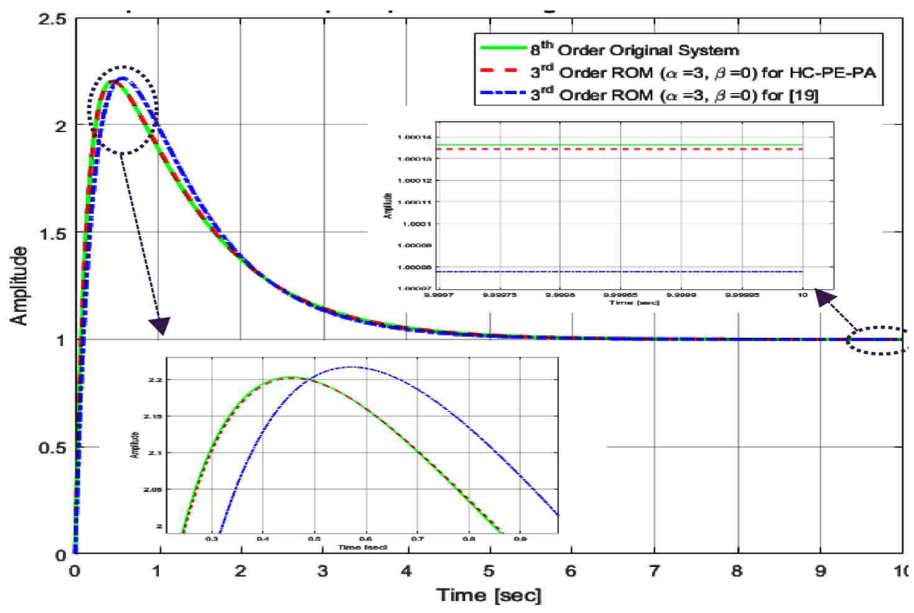


Figure 8. The time step responses for the original, [19] and HC-PE-PA.

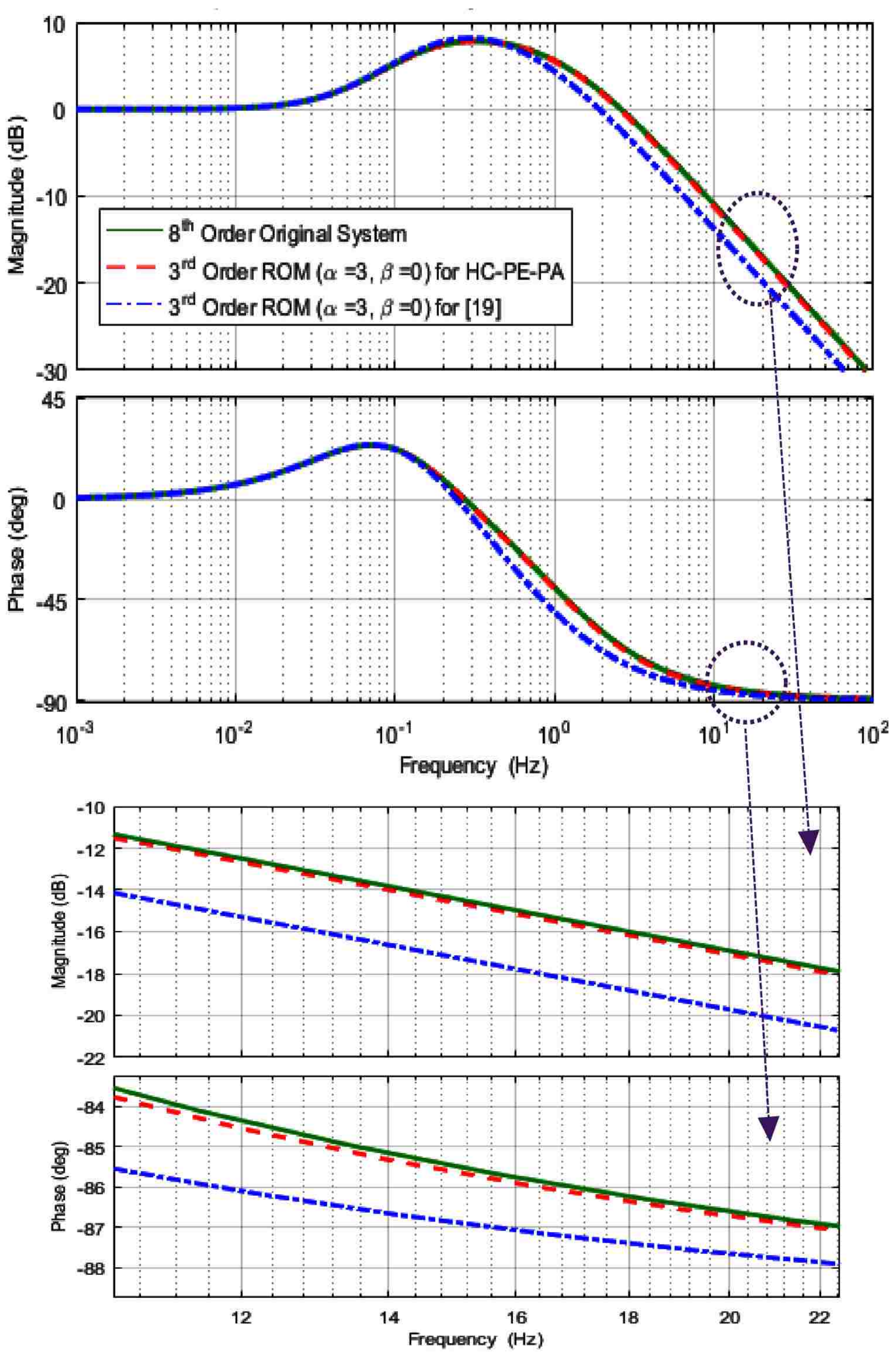


Figure 9. Bode plot for step responses for the original, [19] and HC-PE-PA to show the stability.

We reduced the building model to 6th order. Fig. 10 shows the clustered poles of the building model. After applying the HC-PE technique to get the 6th order model, there are six clusters distributed as three clusters in the positive imaginary axis with three other poles mirrored in the negative imaginary axis. The cluster centers (new system poles for the reduced model) are shown as the two big blue stars, two red circles, and two green diamonds. In this case study, we illustrate the GA's processing by showing a fitness values during generations. We select a 6th order reduced model (level number six). Each chromosome in this level has six genes (six numerator coefficients) with 20 chromosomes for population, and other operations for the GA are presented in Table 1. In this test, we terminate the GA processing if the number of generations reaches 10000, regardless of the average change in the fitness value or number of generations without improvement. We run GA with two modes. The first mode is random initial population by selecting random real values for genes in all chromosomes. The second mode is semi-random initial population by selecting random real values for genes in 19 chromosomes. Chromosome number 20, which has six values (genes) is replicated from PA coefficients. Fig. 11 demonstrates a comparison between a random and semi-random initial population for 3 independent runs. The semi-random initial population reaches the optimal fitness value around 3300 generations, while requiring 7100 generations for the random initial population. A step signal for the first variable (a motion in the first coordinate) of the building structure is applied to the 6th order reduced model to obtain the output of the building structure (derivative of the first coordinate motion). Fig. 12 shows the comparison of 20 seconds time step responses for the 48th order original building model and the 6th order reduced models, which are obtained by HC-PE-PA, HC-PE-GA, Hankel and balanced truncation techniques. Appendix C describes the MSEs and the 6th order transfer function reduced models for HC-PE-PA, HC-PE-GA, Hankel and balanced truncation. Fig. 13 shows the Bode plot for frequency response of the reduced systems of the building model. HC-PE-PA is the best approximation compared with the others. From the lowest frequency range until 0.5Hz (rad/s), both Hankel and balanced truncation techniques are far from the original model. HC-PE-GA starts close to the model at 0.02Hz. The HC-PE needs $\frac{1}{6} \cdot (n^3 - n)$ to build all $(n-1)$ of the reduced models (levels of the hierarchy dendrogram). Brute-force (exhaustive) search needs the second kind of Stirling method, which is

$$\frac{1}{r!}\sum_{j=0}^{r}\left((-1)^{r-j} \cdot \binom{j}{r} \cdot j^n\right) \tag{23}$$

clusters to create only the r^{th} reduced model as in [15]. Therefore, HC-PE is a large improvement over the Brute-force method. For instance, HC-PE-GA finds 18424 clusters in order to generate a hierarchy dendrogram for the 48th order building model, while the brute-force method needs $6.2893 \cdot 10^{44}$.

5. MODEL REDUCTION FOR A MULTIVARIABLE DYNAMIC MODEL BY USING HC-PE

HC-PE is also applied to the reduction of a linear time invariant multi-input multi-output (MIMO) system. We present two methods. The first method selects the optimal new pole-cluster like a generalized form of equation (7):

$$N^z_{cluster} = argmin_{MSE}\Big(\cup_{i=1}^{\hat{n}} \cup_{j=1}^{\hat{m}} \cup_{k=1}^{z} M^k_{cluster(i,j)}\Big), \tag{24}$$

where $\hat{n}$ is the number of model inputs, and $\hat{m}$ is the number of model outputs. The optimal clustering is obtained by comparing among z clusters for all inputs and outputs by taking k clusters in input i and output j, and doing the same procedure for input i with output $j+1$ until $\hat{m}$ and repeating with $i+1$ until $\hat{n}$. This approach calculates the MSE in a horizontal direction, while the other approach takes a vertical direction by calculating MSE for fixed k at all $\hat{n}$ inputs and $\hat{m}$ outputs and repeating with $k+1$ until z. These methods retain the important characteristics of high order models such as the steady state value (minimum MSE value) and stability, which is demonstrated by reducing the triple

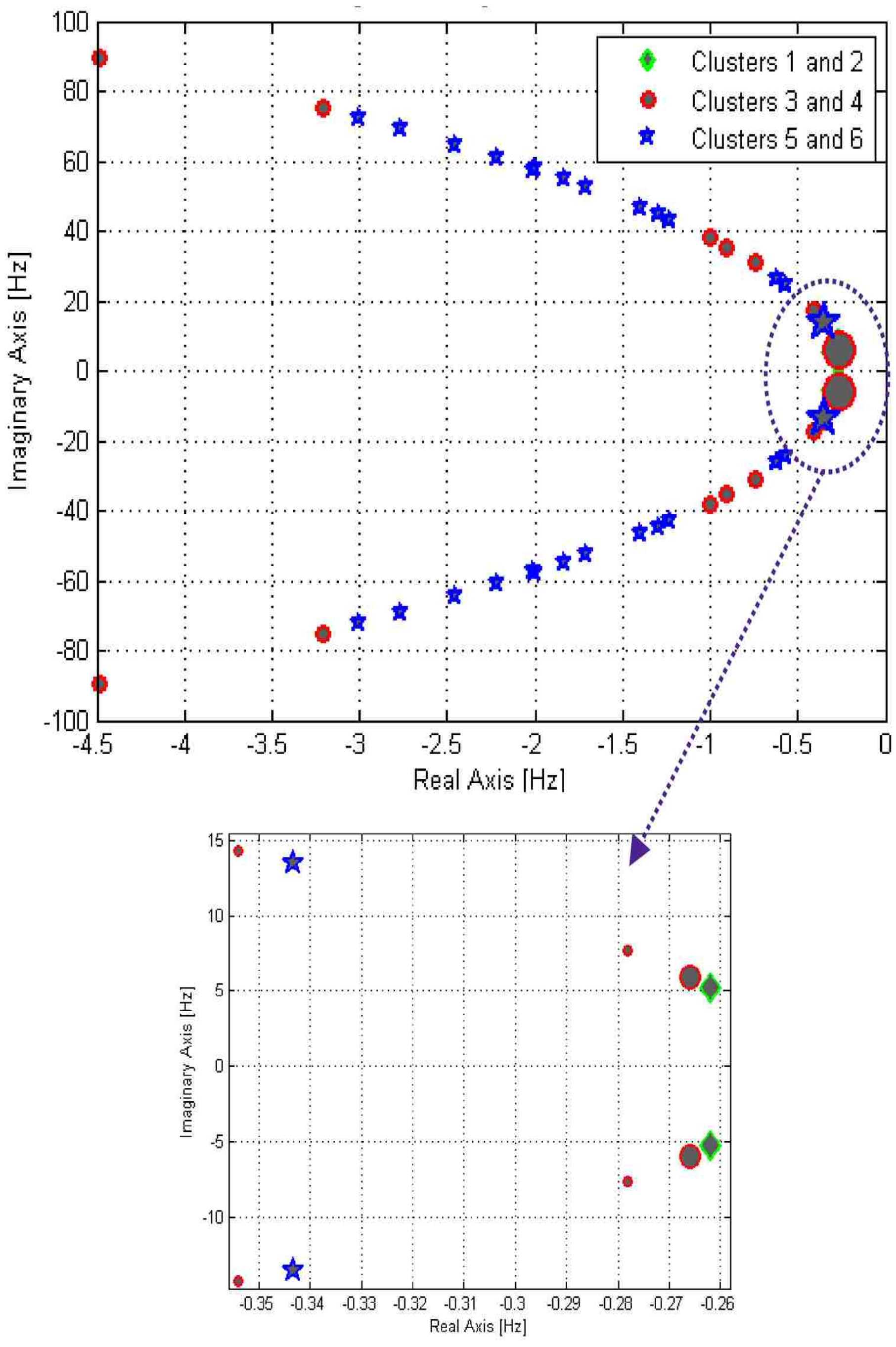


Figure 10. Cluster poles of the Los Angeles building model. The blue star, red circle, and green diamond shape symbols describe the best six clusters (three clusters in positive imaginary axis part with 3 mirror them in negative imaginary axis part), which is obtained by applying the HC-PE technique. The new system poles reduced model (six cluster centers) show as two big symbols (blue stars, red circles, and green diamonds).

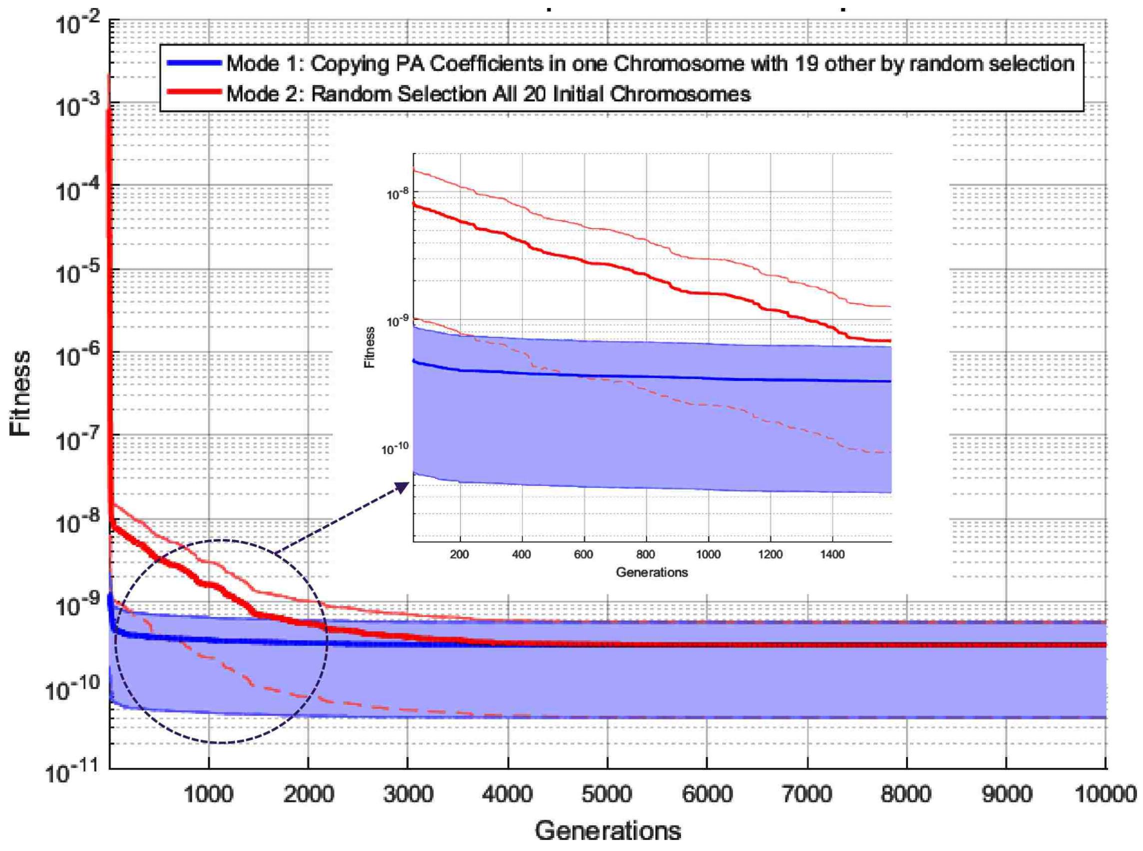


Figure 11. A comparison between random (mode 1) and semi-random (mode 2) initial chromosomes in population for 3 independent runs. Bold blue and red curves are a mean of mode 1 and mode 2, respectively, while thin blue and red curves are represented upper and lower fitness values for 3 runs.

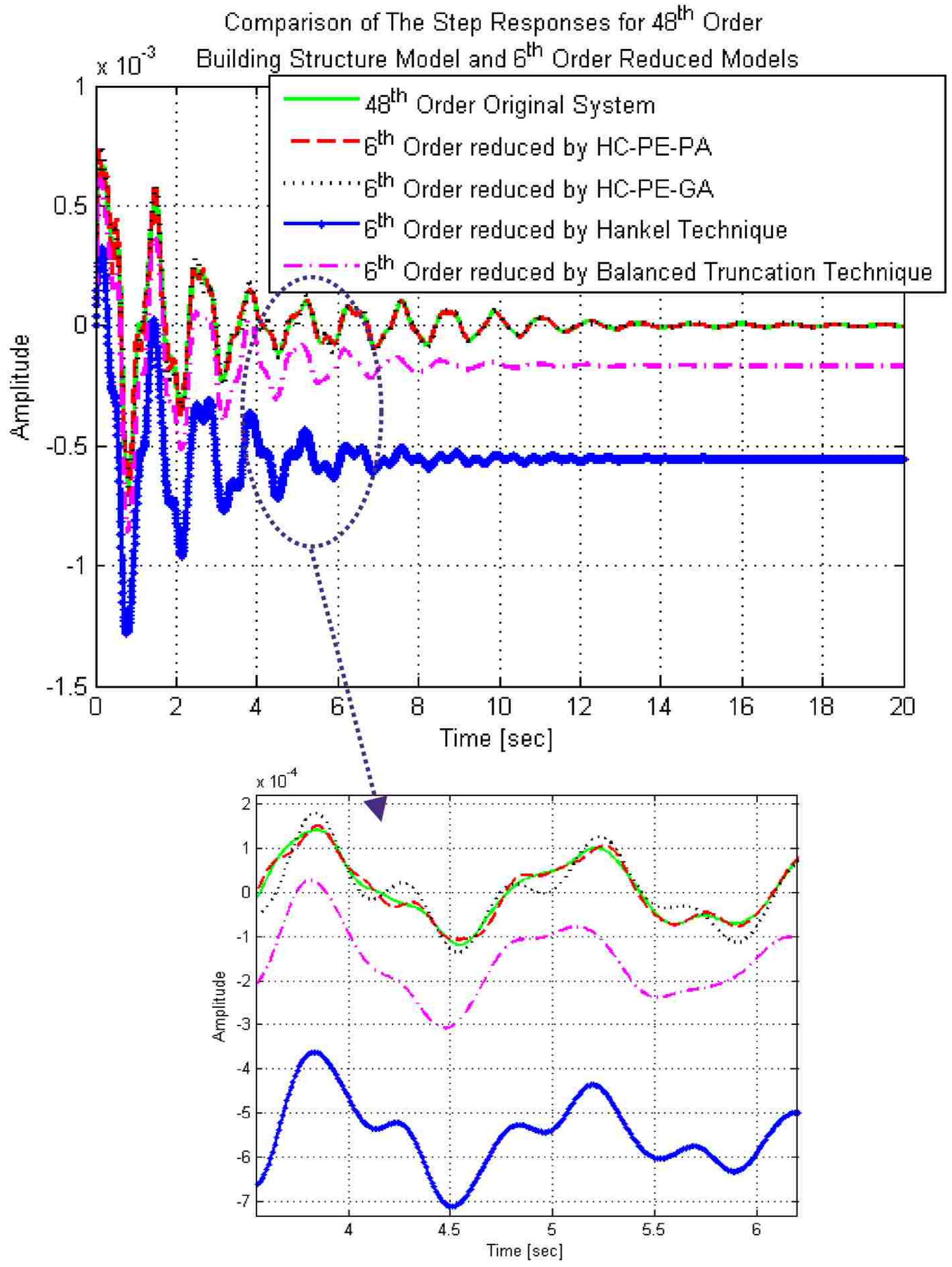


Figure 12. The time step responses for the 48th order original building model and the 6th order reduced models obtained via HC-PE-PA, HC-PE-GA, Hankel and balanced truncation techniques. HC-PE has the best performance compared with the other two approaches. MSE for HC-PE-PA, HC-PE-GA, Hankel and balanced truncation are $1.758e-09$, $6.807e-10$, $3.089e-07$, and $2.827e-08$, respectively.

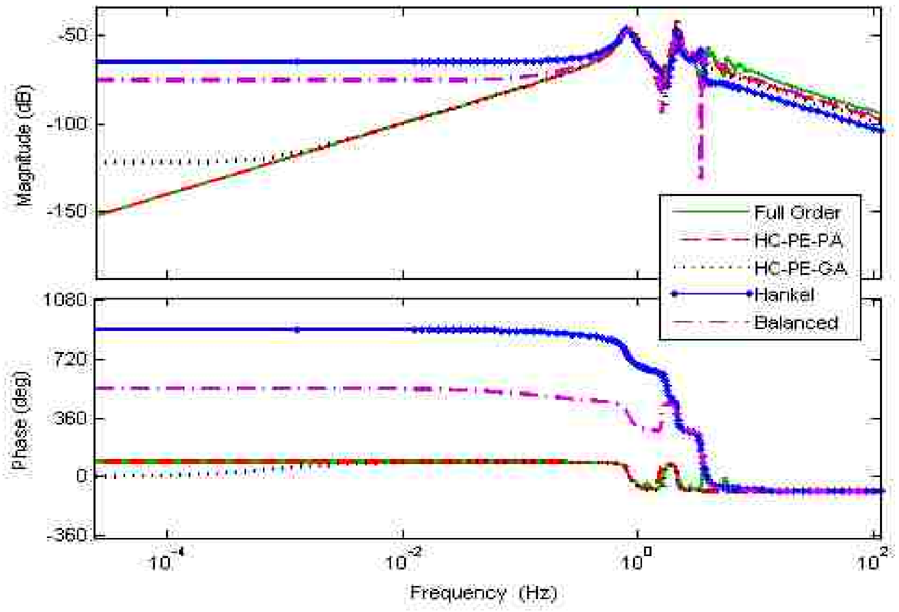


Figure 13. The Bode plot frequency responses for the 48th order original building model and the 6th order reduced models obtained via HC-PE-PA, HC-PE-GA, Hankel and balanced truncation techniques. HC-PE-PA has the best approximation compared with the others, which is clear from the lowest frequency range until 0.5Hz.

linked inverted pendulum model with one input ($\hat{n} = 1$) and four outputs ($\hat{m} = 4$) by using HC-PE-PA and HC-PE-GA. The inverted pendulum is one of the most publicized problems in control systems that can be modeled as a rocket before launch, walking robots, flexible space structures, or many others [23]. It is used as a benchmark for testing in many control algorithms as in [23]-[26]. A main point in this section is examining the effectiveness of HC-PE on the MIMO reduced model and also testing the robustness for this model after changing model parameters and after applying an external disturbance signal on it. The 8th order dynamic model for the triple link inverted pendulum is taken as the benchmark [27], [28]. This model is a multivariable, highly nonlinear, unstable system. The Lagrange method is used to derive this model, which consists of three links (lower, middle, and upper pendulums) mounted vertically on a movable cart on a straight line rail. Fig. 14 shows the schematic representation for the triple link inverted pendulum. An external action force (u) is applied displacing the cart (x) which changes the lower, middle, and upper pendulum angles (θ_1, θ_2, and θ_3) with respect to the vertical line. The state space representation for this model (single input, u, and four outputs: x, θ_1, θ_2 and θ_3) is:

$$\begin{aligned} \dot{X} &= AX + Bu, \\ Y &= CX + Du \end{aligned} \tag{25}$$

where

$$A = \begin{bmatrix} 0 & I_c \\ E^{-1}H & E^{-1}G_c \end{bmatrix}, B = \begin{bmatrix} 0 \\ E^{-1}h_0 \end{bmatrix},$$

$$E = \begin{bmatrix} a_0 & a_1 & a_2 & m_3l_3 \\ a_1 & b_1 & a_2L_1 & m_3L_1l_3 \\ a_2 & a_2L_1 & b_2 & m_3L_2l_3 \\ m_3l_3 & m_3L_1l_3 & m_3L_2l_3 & J_3 + m_3l_3^2 \end{bmatrix},$$

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & a_1g & 0 & 0 \\ 0 & 0 & a_2g & 0 \\ 0 & 0 & 0 & m_3l_3g \end{bmatrix}, h_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$G_c = \begin{bmatrix} -f_0 & 0 & 0 & 0 \\ 0 & -f_1 - f_2 & f_2 & 0 \\ 0 & f_2 & -f_2 - f_3 & f_3 \\ 0 & 0 & f_3 & -f_3 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

I_c is the 4×4 identity matrix, and $D = [0]$. Also, $a_0 = m_0 + m_1 + m_2 + m_3$, $a_1 = m_1 l_1 + m_2 L_1 + m_3 L_1$, $a_2 = m_2 l_2 + m_3 L_2$, $b_1 = J_1 + m_1 l_1^2 + m_2 L_1^2 + m_3 L_1^2$ and $b_2 = J_2 + m_2 l_2^2 + m_3 L_2^2$. All coefficients are given in [27], which are defined as: m_0 is the cart mass, m_ζ is the ζ^{th} pendulum bar mass, f_0 is the is the friction factor of cart and track L_ζ is the ζ^{th} pendulum bar length, J_ζ is the v^{th} pendulum bar rotary inertia, and f_ζ is the ζ^{th} friction factor of the ζ^{th} pendulum bar, where $\zeta = 1, 2$, or 3, and g is the constant gravity force. Because the system has four positive poles (eigenvalues), as shown in the first column of Table 2, it is unstable. In order to apply the HC-PE approach, the system should become stable by using full state feedback or the pole placement method. Pole placement is employed to place the close-loop poles to arbitrary predetermined desired locations in the s-plane by multiplying the system states with a feedback negative control gain matrix (K) to make a new input vector to the system. A final stable state model can be obtained as:

$$\dot{X} = (A - BK)X. \tag{26}$$

The pole placement approach does not apply for uncontrollable systems. The rank of the controllability matrix of the triple link inverted pendulum model $[B \quad AB \quad \ldots \quad A^7 B]$ is eight, which is equal to the rank of the state matrix; therefore, it is controllable. The second column in Table 2 shows new poles, selected randomly, and denoted randomly controlled gains (RCG). The control gains for these poles are K= [0.2160, 42.5108, -214.8757, 171.2896, -0.8791, 0.0141, -5.1804, and 7.8343].

The 3rd order reduced models after applying HC-PE-PA for the generated poles by using RCG on the stable triple link inverted pendulum for x, θ_1, θ_2, and θ_3 states are:

$$G_r^x(s) = \frac{3.778s^2 - 15.53s + 25.69}{s^3 + 5.72s^2 + 10.19s + 5.55}, \tag{27}$$

$$G_r^{\theta_1}(s) = \frac{2.618s^2 + 0.006302s - 3.267e^{-16}}{s^3 + 5.72s^2 + 10.19s + 5.55}, \tag{28}$$

Table 2. Poles values for unstable and stable for triple link inverted pendulum model

Poles for unstable system	**Random stable Poles**	**Poles for stable system (LQR))**
$1.0502 - i27.2130$	-12.5674	$-1.0525 - i27.2129$
$1.0502 + i27.2130$	-11.0345	-12.3133
11.0345	-6.0272	-11.3567
-12.5674	-4.2304	-5.5813
4.2304	-2.7213	-4.7042
-6.0272	-1.9248	$-1.3665 + i0.0567$
-1.9248	-1.0502	$-1.3665 - i0.0567$

$$G_r^{\theta_2}(s) = \frac{2.617s^2 + 0.007252s - 6.389e^{-16}}{s^3 + 5.72s^2 + 10.19s + 5.55}, \tag{29}$$

$$G_r^{\theta_3}(s) = \frac{2.617s^2 + 0.008255s - 6.8369e^{-16}}{s^3 + 5.72s^2 + 10.19s + 5.55}. \tag{30}$$

The original model is completely observable because the rank of the observability matrix $[C \quad CA \quad \ldots \quad CA^7]^T$ is eight, which is equal to rank of the state matrix, so the LQR approach can be employed to obtain the optimal pole placements. This optimality comes from finding the optimal K by balancing the control effect and system errors. LQR Control Gains (LQRCG) are derived from minimization of a quadratic performance index or cost function (J)

$$J = \int_0^\infty \left(X^T Q X + u^T R u\right) dt, \tag{31}$$

where the state-cost matrix Q is positive, semi-definite and symmetric, the index matrix R is symmetric and positive definite, and K is calculated by

$$K = R^{-1} B^T P, \tag{32}$$

where the positive definite symmetric matrix P is obtained from the solution of the matrix algebraic Riccati equation [29] as follows:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0. \tag{33}$$

The stability proof for the closed loop system in LQR is in [29]. The initial value is $R = 1$ (one input control signal), while Q is equal to $C^T C$ with multiplying 2 with $Q(1,1)$ to accelerate the cart displacement response. The third column in Table 2 shows the new optimal poles by using the LQR approach to make the system stable. The control gains for these poles are $K = [1.4142, -43.4921, 6.3479, 52.7879, 1.4597, -1.1102, 5.2459,$ and $5.0146]$. The 3rd order reduced models after applying HC-PE-PA for these poles (LQRCG) on the stable triple link inverted pendulum for for x, θ_1, θ_2, and θ_3 states are:

$$G_r^x(s) = \frac{-0.07967s^2 - 1.137s + 4.67}{s^3 + 6.974s^2 + 11.79s + 6.604}, \tag{34}$$

$$G_r^{\theta_1}(s) = \frac{0.4762s^2 + 0.001146s - 3.972e^{-16}}{s^3 + 6.974s^2 + 11.79s + 6.604}, \tag{35}$$

$$G_r^{\theta_2}(s) = \frac{0.4762s^2 + 0.001318s - 4.28e^{-16}}{s^3 + 6.974s^2 + 11.79s + 6.604}, \tag{36}$$

$$G_r^{\theta_3}(s) = \frac{0.4763s^2 + 0.001501s - 9.352e^{-16}}{s^3 + 6.974s^2 + 11.79s + 6.604}. \tag{37}$$

Fig. 15 (a), (b), (c), and (d) shows the LQRCG and RCG results for the step response after applying HC-PE-PA to reduce the final model of the triple link inverted pendulum to 3rd order for x, θ_1, θ_2, and θ_3, respectively.

Applying HC-PE-PA for optimal pole placement by using LQR not only achieves the best performance in control effect, but also gives a minimum MSE value. The MSE value calculated by (14) is 0.0051 for RCG and $7.64e - 05$ for LQRCG. For these reasons, the LQRCG reduced model is implemented with GA instead of PA with the same operation as in Table 1. The numerators for the reduced model after apply-

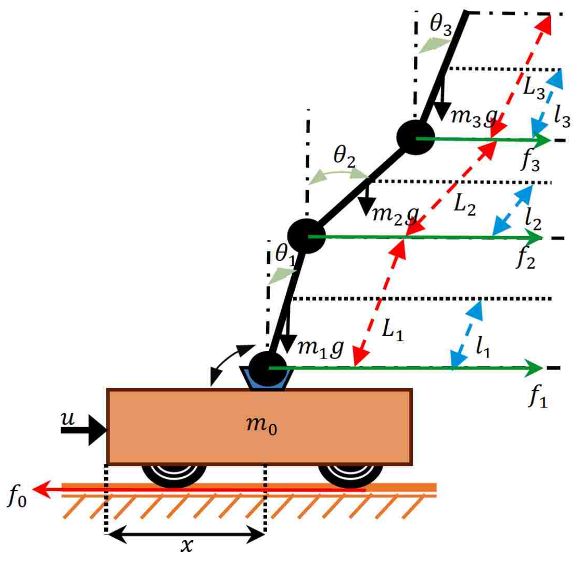


Figure 14. Configuration model of triple link inverted pendulum.

ing GA are $-0.2228s^2 - 1.1893s + 4.4673$ for x, $0.3862s^2 + 0.1195s - 0.0009$ for θ_1, $0.3702s^2 + 0.1082s - 0.0009$ for θ_2, and $0.3628s^2 + 0.1014s - 0.0008$ for θ_3. Fig. 16 (a), (b), (c), and (d) show the LQRCG results for the step response after applying HC-PE-GA to reduce the final model of the triple link inverted pendulum into 3rd order for for x, θ_1, θ_2, and θ_3, respectively. GA has the best performance when comparing with the PA approach for this desired input, where the MSE is $0.7646e - 04$ for PA and $0.6102e - 04$ for GA. We examine the performance of HC-PE when the internal parameters of the triple link inverted pendulum are changed. We change the cart mass by adding an extra 2 kg, and we change the bar masses by adding 1 kg to every bar mass. By using LQRCG, the control gains for a new model after changing parameters are $K = [-1.4142, -1234.2483, 3014.6404, -2117.8318, -29.4697, 4.2548, 93.6118, -34.8808]$. The 3rd order reduced models after applying HC-PE-PA for x, θ_1, θ_2, and θ_3, are:

$$G_r^x(s) = \frac{-0.02914s^2 + 0.1897s - 0.3561}{s^3 + 5.373s^2 + 5.379s + 0.5036}, \tag{38}$$

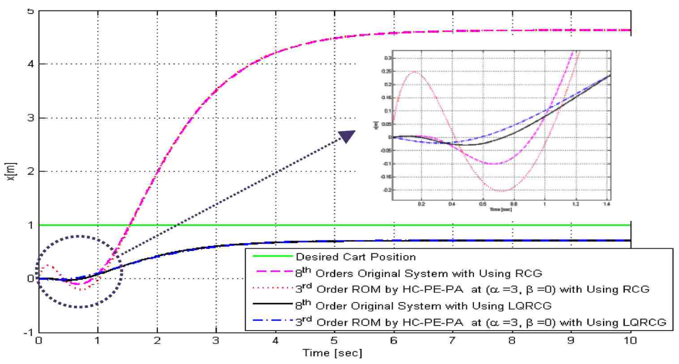
$$G_r^{\theta_1}(s) = \frac{-0.0363s^2 - 4.753e^{-17}s - 1.895e^{-18}}{s^3 + 5.373s^2 + 5.379s + 0.5036}, \quad (39)$$

$$G_r^{\theta_2}(s) = \frac{-0.0363s^2 - 2.98e^{-17}s - 1.464e^{-18}}{s^3 + 5.373s^2 + 5.379s + 0.5036}, \quad (40)$$

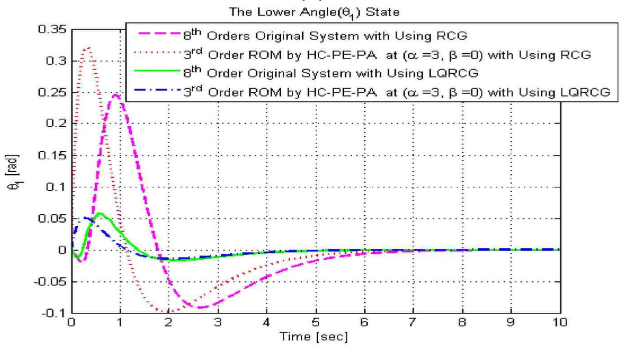
$$G_r^{\theta_3}(s) = \frac{-0.0363s^2 + 5.883e^{-17}s - 2.019e^{-18}}{s^3 + 5.373s^2 + 5.379s + 0.5036}. \quad (41)$$

The numerators for the reduced model after applying HC-PE-GA are $-0.02976s^2+0.1896s-0.3561$ for x, $-0.0291s^2-0.0012s+0.00008$ for θ_1, $-0.0316s^2-0.00107s+0.0001$ for θ_2, and $-0.02408s^2-0.0009s-0.00006$ for θ_3. Fig. 17 (a), (b), (c), and (d) show the 3rd order reduced model results for the positive step response after changing the mass parameters. Despite of large values for K variables to compensate the increasing in the model masses, a slow displacement response as shown in Fig. 17 (a). Because of these changes, low frequencies of pendulum bars and small magnitude angle values occur, as shown in Fig. 17 (b)-(c). Therefore, we increase the response time to 20 sec to display the required time to make angels reach zero. GA has the best performance compared to the PA for this time interval, where the MSE is $4.6199e-07$ for PA and $3.66812e-07$ for GA. But for long time response, the PA has better steady state performance. For instance, if the response time is 2000 sec, the MSE is $4.6233e-09$ for PA and $2.5877e-08$ for GA. This also happened when we change lengths of the pendulum bars. We add 0.3, 1.5, and 0.2 to the lengths of lower, middle, and upper pendulum bars, respectively. For 20 sec of a time response, the MSE is $1.4007e-06$ for PA and $8.1173e-07$ for GA. If for 2000 sec, the MSE is $1.4996e-08$ for PA and $5.2091e-07$ for GA.

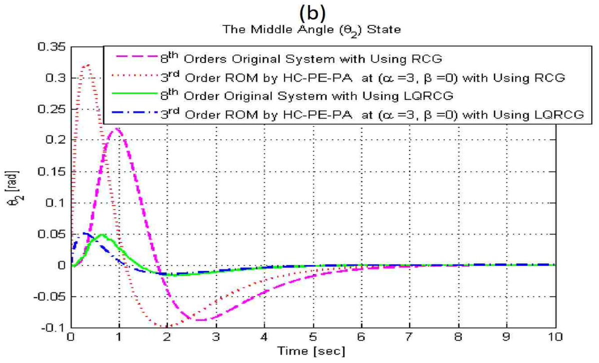
As shown in Fig. 15 (a), Fig. 16 (a), and Fig. 17 (a), the desired cart position is one, but the response does not follow the reference signal because the full-state feedback approach does not compare the output with the desired signal. To address this, we can calculate a feedforward scaling factor for certain desired input signals, but we use a PID controller to handle different situations. The optimal gains for the PID are obtained by using a GA for the step response of the desired cart position. These gains are given in the



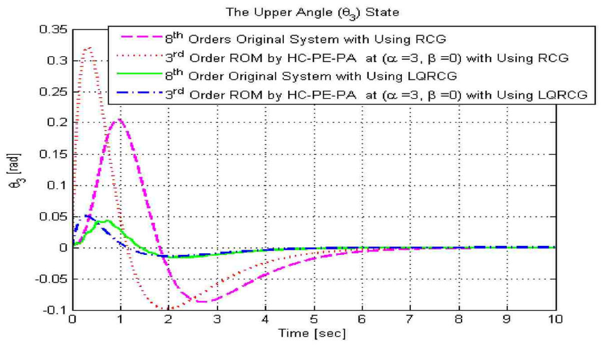
(a)



(b)



(c)



(d)

Figure 15. The step response results after applying HC-PE-PA to reduce the model of the triple link inverted pendulum to 3rd order. (a) The cart displacement. (b) The lower angle. (c) The middle angle. (d) The upper angle. The original 8th order model reduces after applying RCG and LQRCG.

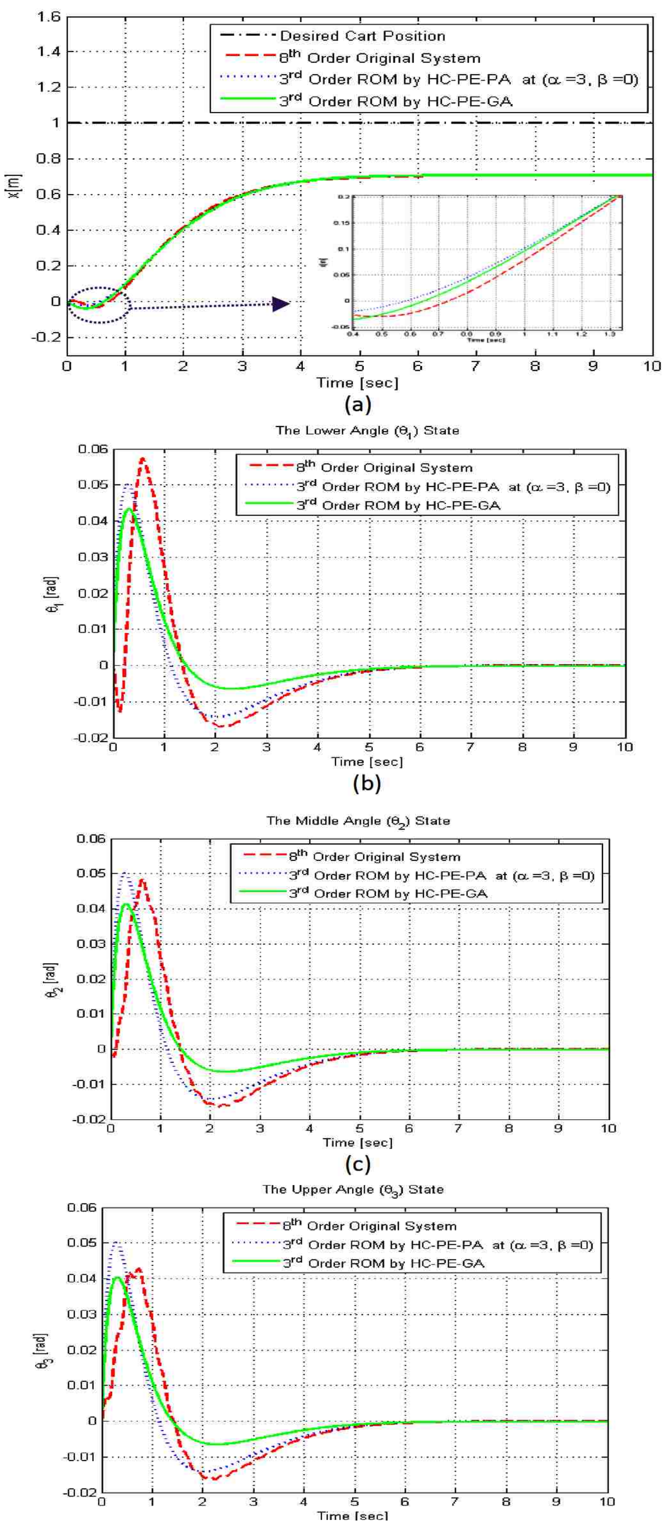
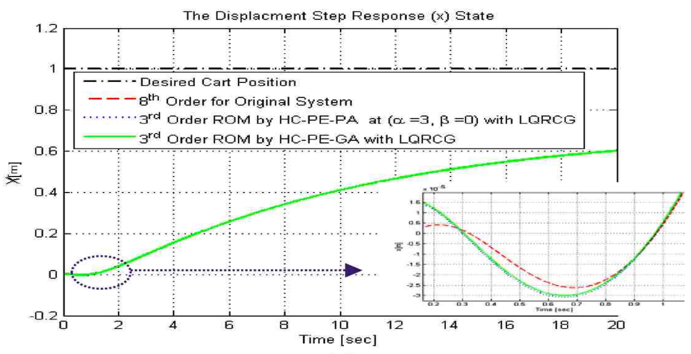
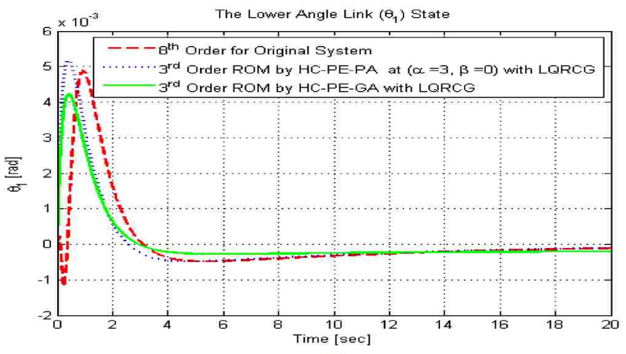


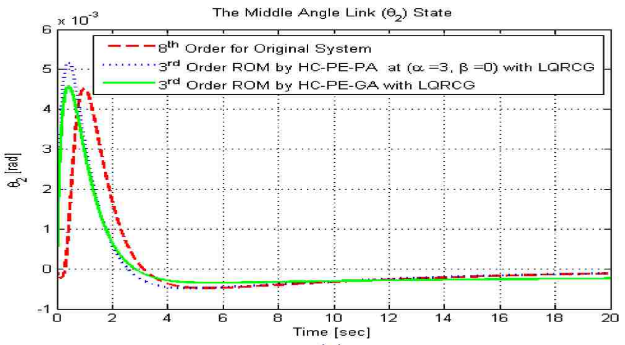
Figure 16. The step response after applying HC-PE-PA and HC-PE-GA to reduce the model of a triple link inverted pendulum to 3rd order. (a) The cart displacement. (b) The lower angle. (c) The middle angle. (d) The upper angle. The original 8rd order model applies the LQRCG method.



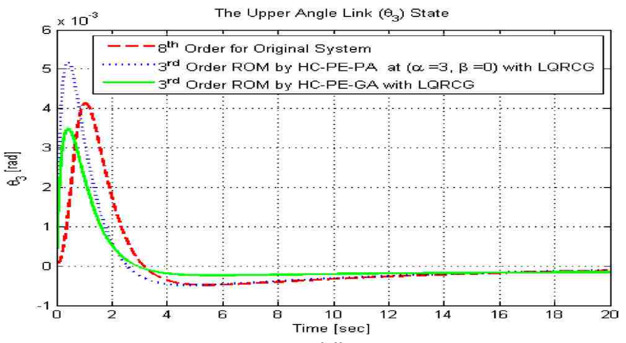
(a)



(b)



(c)



(d)

Figure 17. The step response after applying HC-PE-PA and HC-PE-GA to reduce the model of a triple link inverted pendulum to 3rd order after changing mass parameters. (a) The cart displacement. (b) The lower angle. (c) The middle angle. (d) The upper angle. The LQRCG method is applied to make the original 8th order model stable.

PID transfer function:

$$u_x(s) = K_p + K_i\frac{1}{s} + K_d\frac{N}{1 + N\frac{1}{s}}, \tag{42}$$

where K_p is proportional gain, K_i is integral gain, K_d is derivative gain, and N is the first-order derivative filter gain (for reducing noise and distortions). The fitness function is implemented to minimize the MSE between the actual and desired cart position. The final optimal values for these gains are $K_p = 2.0652$, $K_d = 1.2518$, $K_i = 1.6676$, and $N = 172.6118$. Fig. 18 illustrates a Simulink model for the original 8th order model after applying LQRCG and reduced models (PA and GA). We tune PID gains for one model (the original model) and then apply the same controller with other reduced models. This PID controller has two main advantages: first, it can adapt with various reference signals (desired cart positions); for instance, Fig. 19 (a) and (b) show a new cart displacement position ($x = 3$) and θ_1, respectively. Second, the PID controller can compensate the values for Q and R elements instead of using a trial and error method. The control input signal to the original and reduced model is bounded inside the range $[-5, 5]$. An impulse signal at the 18th sec with a one sec pulse width has been added as the disturbance input to the models (original and reduced), as shown in Fig. 18. Fig. 20 (a) and (b) show the results for θ_2 and θ_3, respectively. The best performance for a reduced order model is provided by HC-PE-GA. For the experiments, the step signals for the cart displacements of 1 and 3 were inserted. The MSE was evaluated over the subsequent 40 sec using equation (15). The MSE values for HC-PE-PA are 0.00466 and 0.0350 for 1 and 3 displacement, respectively. For HC-PE-GA, they are 0.00491 and 0.03616 for 1 and 3 displacement, respectively. However, the response for the reduced triple linked inverted pendulum dynamic system model by using HC-PE-GA is better than using HC-PE-PA. The overall average errors for x, θ_1, θ_2, and θ_3 in the 3rd order reduced model are 0.0124 for HC-PE-GA and 0.0128 for HC-PE-PA. This best performance for GA is achieved by applying the same input and tuning the reduced model numerator coefficients. The reduced model using HC-PE-PA is better than using HC-PE-GA for a different reference input, as shown in the zoomed-in portion of Fig. 20 (b). The overall average error for the reduced model for 3 displacement is 0.0888 for HC-PE-PA and 0.0913 for HC-PE-GA. This error increases in direct proportion with the time that the pendulum cart is forced into large displacements. Fig. 21 (a), (b) and (c) depict a 2-dimensional simulation for the reduced model by using HC-PE-PA technique to move a cart from the 0 position to the final desired cart positions, which are 4, 6 and 10. Optimal control gains for a closed loop reduced system model can also be applied. In other words, K_r is optimal control gain vector ($K_r \in \mathbb{R}^{r\times 1}$), which is only used to control on a r^{th} reduced model. Asymptotic stability is easily verified for the reduced model with its optimal controller, as proven in Appendix B.

6. CONCLUSION

This work demonstrates that the HC-PE algorithm has superior performance in different situations when compared to other order reduction methods. The pole clusters for the original system in the HC-PE algorithm are selected by using a performance evaluation method for similarity criteria of agglomerative hierarchical cluster analysis. This method gives a major advantage for reducing error. Optimizing the pole clusters is achieved by taking the minimum MSE among all pole clusters at an appropriate level in the hierarchical dendrogram. The HC-PE algorithm is considered a lower level in the hierarchy as the base model, which is an optimal original model at r^{th} order, and calculates a next reduced $r^{\text{th}} - 1$ order, continuing until reaching the 2nd order reduced model. This procedure makes simple, near-optimal, and fast selections of cluster poles in stages. We demonstrate the robustness of the reduced model after applying various examples in SISO and MIMO cases.

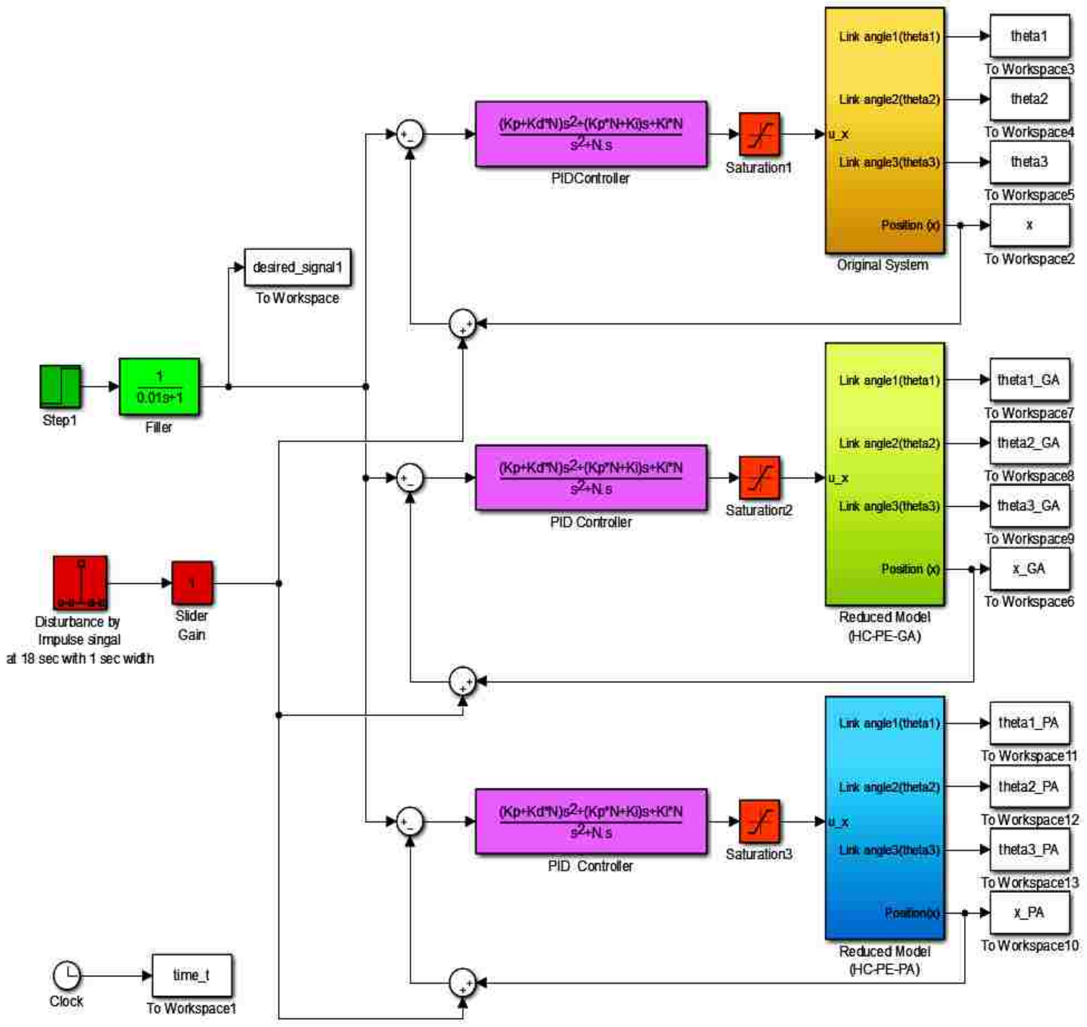
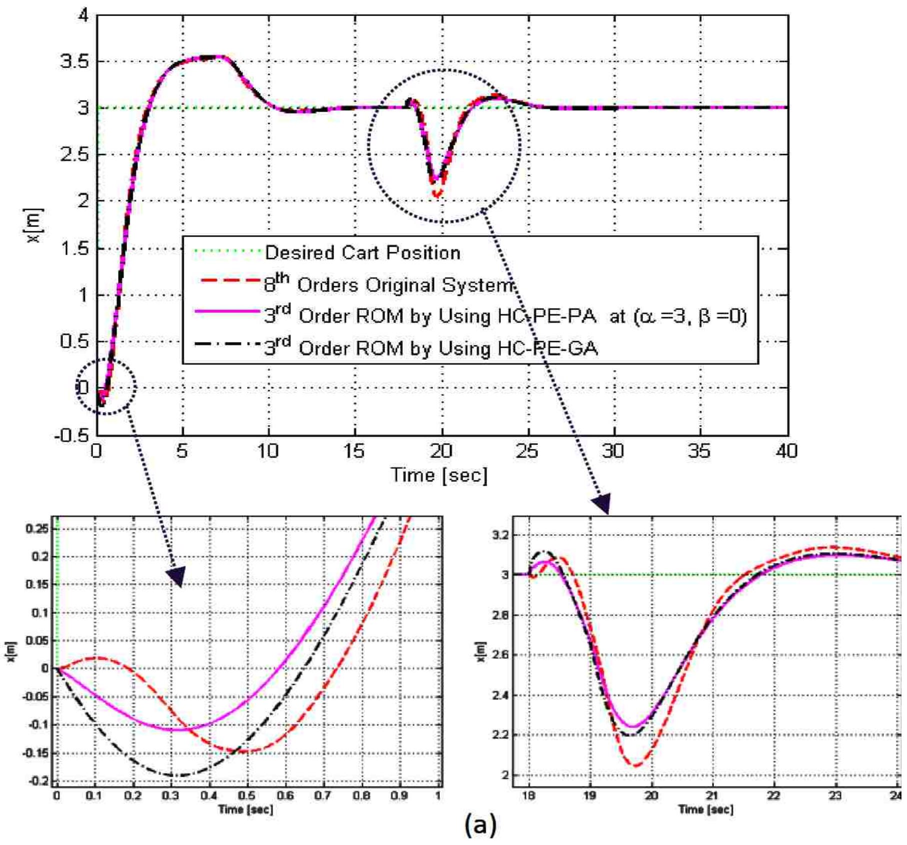


Figure 18. Simulink model of LQRCG applied to the triple link inverted pendulum for original 8th and 3rd order reduced model after applying HC-PE for both Pade approximation and GA control by PID control with a disturbance input.



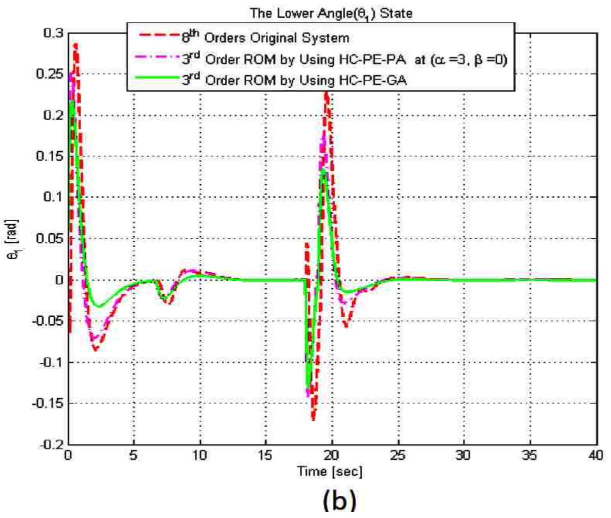


Figure 19. The responses after applying the disturbance signal on the cart displacement state for comparison among LQRCG, HC-PE-PA, and HC-PE-GA reduced models.(a) The cart displacement.(b) The lower angle.

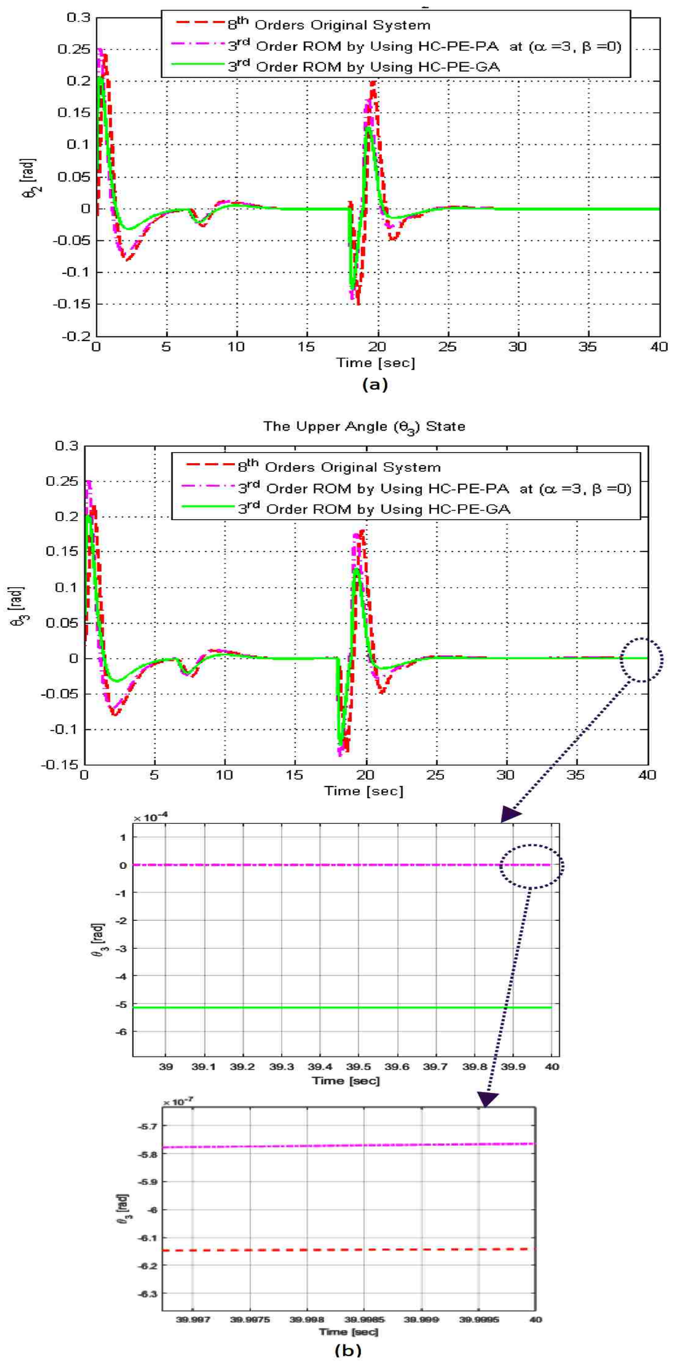


Figure 20. The responses after applying the disturbance signal on the first state (cart displacement) to compare LQRCG, HC-PE-PA, and HC-PE-GA reduced models. (a) The middle angle state. (b) The upper angle state. GA performs slightly better for the transient response, but PA has acceptable transient and superior steady state performance.

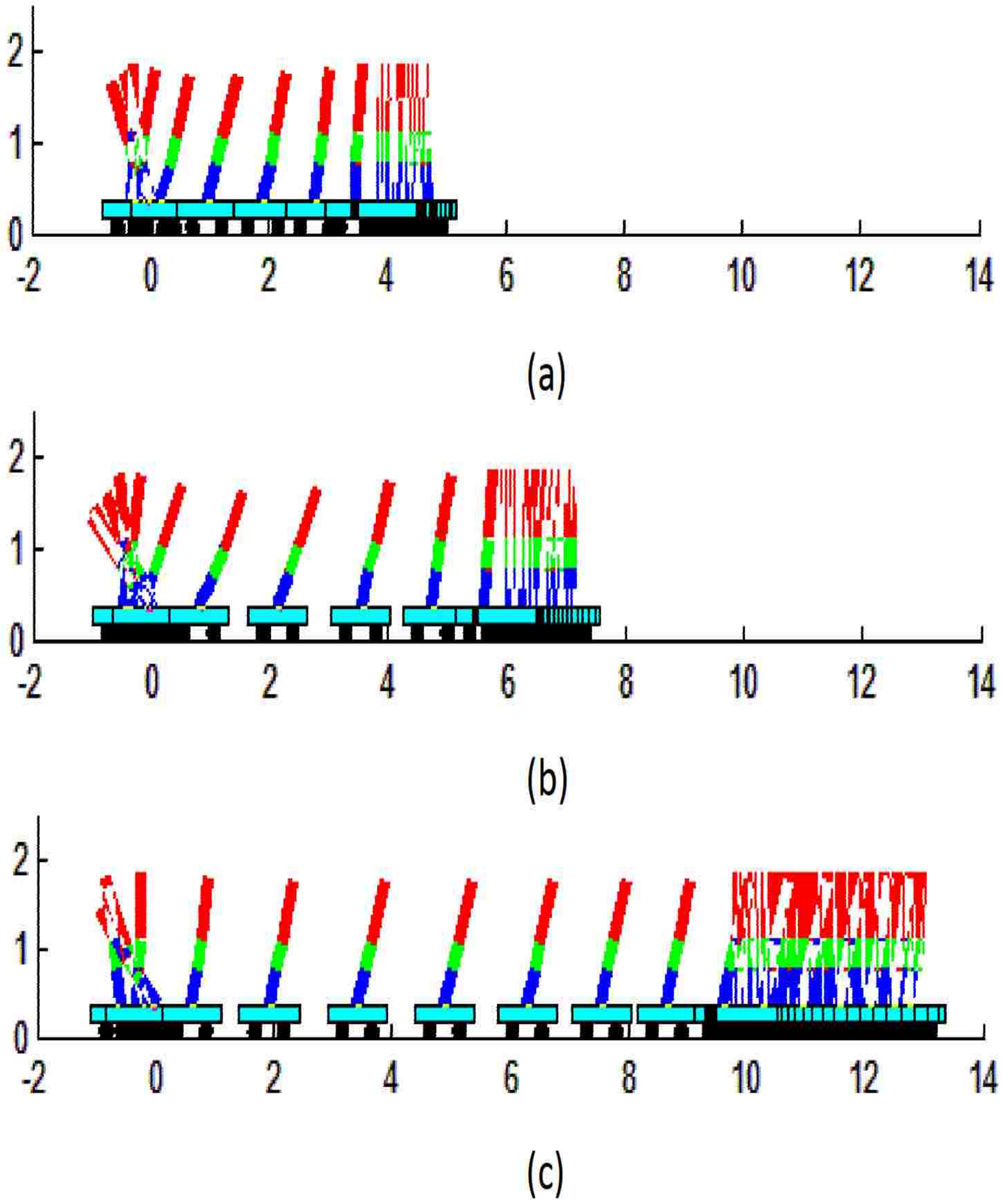


Figure 21. 2-D simulation for a triple linked inverted pendulum model. (a), (b) and (c) show the simulation for the HC-PE-PA reduced model starting from the 0 position to the final desired cart positions which are 4, 6 and 10, respectively.

Table 3. List of abbreviations and symbols

Acronyms and Symbols	**Description**
HC-PE	Agglomerative hierarchical clustering based on performance evaluation
MOR	Model order reduction
MSE	Mean squared error
PA and GA	Pade approximation and Genetic algorithm
G	The ordinary differential equations for the original model.
n	The order of an original model
G_r	The reduced model
r	The order of reduced model
u	The vector of the system's input (action force)
X	The vector of the state variables
v	The number of zeros in an original model
PID	Proportional-integral-derivative controller.
s	The Laplace complex variable
p_l	The l^{th} real pole
$p_m \pm I_m$	The m^{th} complex conjugate pole.
q^c and $p^c \pm iI^c$	The center of cluster for real and complex pole.
R_j	The j^{th} improved pole cluster center.
z	The number of clusters in same level of dendrogram
$N^z_{cluster}$	The optimal cluster among all z clusters
LQR	Linear quadratic regulator
T_i	The i^{th} time moment of the system
A	The state matrix
B	The input matrix
C	The output matrix
D	The feed through matrix
δ	Markov parameter
α	The number of time moment parameters
β	The number of Markov parameters
$N_k(s)$	The numerator polynomial of $k = n$ or r order system
$D_k(s)$	The denominator polynomial of the $k = n$ or r order system
HC-PE-PA	The HC-PE algorithm combined with the PA approach
HC-PE-GA	The HC-PE algorithm combined with the GA approach
θ_1, θ_2 and θ_3	The lower, middle, and upper pendulum angles, respectively
x	The displacement of the inverted pendulum
m_0	Cart mass
m_v	The v^{th} pendulum bar mass
L_v	The v^{th} pendulum bar length
J_v	The v^{th} pendulum bar rotary inertia
g	The constant gravity force
K	The control gain matrix
RCG	Random control gains
LQRCG	LQR control gains
J	The cost function
Q	The state-cost matrix
K_p,K_i, K_d and N	Proportional, derivative, integral and first-order derivative filter gains, respectively
$\hat{n}$	The number of model inputs
$\hat{m}$	The number of model outputs
h poles in one cluster	The number of real or complex

Appendix A.

Stability guarantee for a new reduced model

We prove that a generated pole, which is calculated by using HC-PE technique is always located in the left half of the complex plane. Therefore, the new reduced model is always stable.

Assumption A. Let the three consideration cases (Fig. 2) be satisfied, and the original model is fragmented into z clusters by using HC-PE algorithm to create a z^{th} order reduced model. Let R be the number of poles in an H cluster, where H is an arbitrary selected cluster in z clusters. Let I_H^z is a new pole, which is an improved generated center pole of H cluster.

Lemma A. Let assumption 1 hold. Then, improved center poles have a negative real parts, which are always located in the left half of the complex plane.

Lemma A Proof. After applying HC-PA to obtain a center pole of H cluster as in equation (4) and first term of equation (5), the new center pole is a negative real number denoted as C_H^z, which is given by $C_H^z = -R\left(\sum_{k=1}^{R}\left(\frac{1}{|p_k|}\right)\right)^{-1}$,where p_k is the k^{th} pole in H cluster. For improving C_H^z value, equation (6) is applied R times as shown in Fig. 3. The final improved center pole of H cluster is given by $I_H^z = -\frac{2^R|p_1||C_H^z|}{|p_1| + (2^R - 1)|C_H^z|}$, where p_1 is the most dominant pole in H cluster. I_H^z is always a negative real number, which is located in the left half of the complex plane. A similar procedure is applied on other clusters among the z clusters to produce z^{th} order reduced model. I_i^z, where $i = 1, 2, \ldots, z$, are poles for z^{th} order reduced model. Therefore, a new generated reduced model is asymptotic stable (or marginally stable if the original model has poles on the imaginary axis).

Appendix B.

Stability Proof for a closed loop reduced system

To assess stability, we show all reduced models by using HC-PE have a Hurwitz matrix for its dynamics matrix, i.e., all eigenvalues or system real poles are in the left half s-plane. Therefore, LQR can be applied on the reduced model itself, which guarantee stability as follows:

Assumption B. Let $\dot{x}_r = A_r x_r + B_r u$ be a controllable reduced system model in a state space domain by using HC-PE, where x_r is the reduced system state vector, $A_r \in \mathbb{R}^{r\times r}$ is the reduced state matrix, $B_r \in \mathbb{R}^{r\times\hat{n}}$ is the reduced input matrix, u is the input vector, r is the number of reduced system states (i.e., it represents an r^{th} reduced order differential equation),$\hat{n}$ is the number of model inputs.

Lemma B. Let assumption 1 hold. Then the reduced model system is asymptotic stable.

Lemma B Proof. The closed loop system $\dot{x}_r = (A_r - B_r K_r)x_r$, where K_r is optimal control gain for the reduced model that can be found in same way for original model [29] as in (32). A nominated Lyapunov function is $V_r = x_r^T P_r x_r$, where P_r is symmetric and positive definite. The time derivative of V_r is given as:

$$\begin{aligned}\dot{V}_r =&\dot{x}_r^T P_r x_r + x_r^T P_r \dot{x}_r^T\\ =&x_r^T[A_r - B_r K_r]^T P_r x_r + x_r^T P_r [A_r - B_r K_r] x_r\\ =&x_r^T\Big[(P_r A_r + A_r^T P_r - P_r B_r R_r^{-1} B_r^T P_r + Q_r) - Q_r - B_r R_r^{-1} B_r^T P_r\Big]x_r,\end{aligned} \tag{1}$$

where Q_r is the state-cost matrix for reduced model that is positive, semi-definite and symmetric, and R_r the index matrix for reduced model that is symmetric and positive definite. Applying (33), which is similar to a Riccati equation solution with reduced model (r order instead of n), we get

$$\dot{V}_r =x_r^T\Big[-Q_r - B_r R_r^{-1} B_r^T P_r\Big]x_r. \tag{2}$$

Thus Q_r, R_r, R_r^{-1}, and P_r are larger than zero (symmetric and positive definite), then $\dot{V}_r < 0$; therefore the closed loop is asymptotically stable.

Appendix C.

Comparison table of HC-PE with other reduction methods, combining the Pade approximation approach and GA. The best MSE results are shown in bold

Reduction Models		**Reduced Model**	**MSE**	**Reducing Models and References**
HC-PE	PA	$G_r(s)=\frac{-2.0549s+37.3859}{s^2+20.3682s+37.3857}$	7.4424e-06	10th to 2nd
	GA	$G_r(s)=\frac{-2.0052s+37.3766}{s^2+20.3682s+37.3857}$	**7.2551e-06**	Order
[21]		$G_r(s)=\frac{-50.346s+432.4174}{s^2+209.0177s+432.4174}$	2.0506e-04	Reducting
[19]		$G_r(s)=\frac{-28.3902s+647.6004}{s^2+359.999s+647.6019}$	1.5278e-04	[21] and [19]
HC-PE	PA	$G_r(s)=\frac{17.64s^2+49.77s+14.1}{s^3+10.01s^2+23.12s+14.1}$	3.9653e-06	8th to 3rd
	GA	$G_r(s)=\frac{17.7476s^2+49.6902s+14.1074}{s^3+10.01s^2+23.12s+14.1}$	**1.2062e-06**	Order Reduction,
[19]		$G_r(s)=\frac{15.56s^2+62.64s+18.43}{s^3+10.16s^2+27.8s+18.43}$	7.9863e-04	[19]
HC-PE	PA	$G_r(s)=\frac{21.3s+7.035}{s^2+9s+7.035}$	0.0021	8th to 2nd
	GA	$G_r(s)=\frac{22.0774s+7.0731}{s^2+9s+7.035}$	**0.0015**	Order
[21]		$G_r(s)=\frac{24.11429s+8}{s^2+9s+8}$	0.0048	Reductions,
[30]		$G_r(s)=\frac{7.0908s+1.9906}{s^2+3s+2}$	0.0269	[21] and
[31]		$G_r(s)=\frac{11.3909s+4.4357}{s^2+4.2122s+4.4357}$	0.0059	[30]-[32]
[32]		$G_r(s)=\frac{17.9856s+500}{s^2+13.24571s+500}$	0.1457	
HC-PE	PA	$G_r(s)=\frac{9.242s+23.379}{s^2+2.677s+2.3379}$	0.0249	4th to 2nd
	GA	$G_r(s)=\frac{9.7157s+23.01}{s^2+2.677s+2.3379}$	**0.0225**	Order Redcution,
[33]		$G_r(s)=\frac{9.236s+23.366}{s^2+2.677s+2.3366}$	0.025	[21] and [33]
HC-PE	PA	$G_r(s)=\frac{16.97s^3+58.31s^2+254.1s+136.8}{s^4+3.923s^3+20.13s^2+21.85s+6.129}$	0.0402	8th to 4th4th
	GA	$G_r(s)=\frac{15.575s^3+56.448s^2+255.017s+136.642}{s^4+3.923s^3+20.13s^2+21.85s+6.129}$	**0.0372**	Order Reduction,
[19]		$G_r(s)=\frac{19.824s^3+30.2549s^2+757.2845s+1436.5}{s^4+10.28s^3+26.56s^2+143.8s+64.37}$	0.8546	[19]
HC-PE	PA	$G_r(s)=\frac{0.0094s^5+0.0105s^4+1.157s^3+0.6293s^2+27.56s+1.04e-08}{s^6+1.742s^5+245.1s^4+267.7s^3+1.23e04s^2+6628s+1.739e05}$	1.7581e-09	48th to 6th
	GA	$G_r(s)=\frac{0.0076s^5+0.0098s^4+1.11s^3+0.6312s^2+27.87s+0.1508}{s^6+1.742s^5+245.1s^4+267.7s^3+1.23e04s^2+6628s+1.739e05}$	**6.8066e-10**	Order Reduction,
Hankel		$G_r(s)=\frac{0.0047s^5+0.0359s^4+3.217s^3+16.27s^2+329.1s-1238}{s^6+2.593s^5+681.2s^4+1120s^3+1.028e05s^2+9.17e04s+2.232e06}$	3.0895e-07	[22]
Balanced Truncation		$G_r(s)=\frac{0.0046s^5+0.0359s^4+3.217s^3+16.27s^2+329.1s-1238}{s^6+3.506s^5+783.6s^4+1611s^3+1.261e05s^2+1.141e04s+2.925e06}$	2.8268e-08	

ACKNOWLEDGMENT

The authors would like to thank P. V. Dooren and Y. Chahlaoui for sending the benchmarking data for original models. Support of the Mary K. Finley endowment and the Intelligent Systems Center is gratefully acknowledged.

BIBLIOGRAPHY

[1] W. Schilders, "Introduction to model order reduction," *Theory, Research Aspects and Applications*, vol. 13, pp. 3-32, 2008.

[2] P. Trnka, C. Sturk, H. Sandberg, V. Havlena, and J. Rehor, "Structured model order reduction of parallel models in feedback," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 3, pp. 739-752, May 2013.

[3] N. A. Gershenfeld, *The Nature of Mathematical Modeling*, Cambridge University Press, Mathematics, 1999.

[4] D. Xue, Y. chen, and D. P. Atherton, *Linear feedback control analysis and design with Matlab*, Society for Industrial and Applied Mathematics, Advances in Design and Control, Jan. 2007.

[5] K. J. Astrom and R. M. Murray. *Feedback systems: an introduction for scientists and engineers*, Princeton University Press, Apr. 2010.

[6] I. Kale, J. Gryka, G. D. Cain, and B. Beliczynski, "FIR filter order reduction: balanced model truncation and Hankel-norm optimal approximation," *IEEE Proc. Vision, Image and Signal Processing*, vol. 141, no. 3, pp. 168-174, Jun. 1994.

[7] W. Wang, G. N. Paraschos, and M. N. Vouvakis, "Fast frequency sweep of FEM models via the balanced truncation proper orthogonal decomposition," *IEEE trans. Antennas and Propagation*, vol. 59, no. 11, pp. 4142-4154, Oct. 2011.

[8] A. K. Sinha and J. Pal, "Simulation based reduced order modelling using a clustering technique," *Computers and Electrical Engineering*, vol. 16, no. 3, pp. 159-169, Jan. 1990.

[9] J. Pal, "Improved Pade approximants using stability equation method," *IEEE Proc. Electronics Letters*, vol. 19, no. 11, pp. 426-427, May 1983.

[10] N. Gupta and A. Narain, "Reduction of discrete interval systems through fuzzy-C means clustering with dominant pole retention," *Australian Control Conference (AUCC)*, pp. 348-353, Nov. 2015.

[11] C. B. Vishwakarma and R. Prasad, “Time domain model order reduction using Hankel matrix approach,” *Journal Franklin Institute*, vol. 351, no. 6, pp. 3445-3456, Jun. 2014.

[12] V. P. Singh, P. Chaubey, and D. Chandra, “Model order reduction of continuous time systems using pole clustering and chebyshev polynomials,” *IEEE Proc., Engineering and Systems Conference*, pp. 1-4, Mar. 2012.

[13] W. T. Beyene, “Pole-Clustering and Rational-Interpolation Techniques for Simplifying Distributed Systems,” *IEEE trans. Circuits and syst. Fund. Theory and App.*, vol. 46, no. 12, pp. 1468-1472, Dec. 1999.

[14] R. Xu and D. Wunsch, “Survey of clustering algorithms,” *IEEE Trans Neural Netw.*, vol. 16, no. 3, pp. 645-678, May 2005.

[15] R. Xu and D. Wunsch, *Clustering*, Wiley-IEEE Press, Oct. 2008.

[16] A. Mirzaei, and M. Rahmati, “A novel hierarchical-clustering-combination scheme based on fuzzy-similarity relations,” *IEEE Trans. Fuzzy Syst.*, vol. 18, no.1, pp. 27-39, Feb. 2010.

[17] A. Proietti, L. Liparulo, and M. Panella, “2D hierarchical fuzzy clustering using kernel-based membership functions,” *IEEE Electronics Letters*, vol. 52, no.3, pp. 193-195, Feb. 2016.

[18] L. Zheng and T. Li, “Semi-supervised Hierarchical Clustering,” *IEEE Proc. International Conference on Data Mining*, pp. 982-991, Dec. 2011.

[19] J. Singh, C. B. Vishwakarma, and K. Chattterjee, “Biased reduction method by combining improved modified pole clustering and improved Pade approximations,” *Applied Mathematical Modeling*, vol. 40, pp. 1418-1426, Jan. 2016.

[20] P. Kalaiselvi and V. G. Pratheep, “Analysis of interval system using model order reduction,” *IEEE Proc. International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1-6, Mar. 2015.

[21] G. Parmar, S. Mukher, and R. Prasad, “System reduction using factor division algorithm and eigen spectrum analysis,” *Applied Mathematical Modeling*, vol. 31, pp. 2542-2552, 2007.

[22] A. C Antoulas, D. Sorensen, and S. Gugercin, “A survey of model reduction methods for large-scale systems,” *Contemporary Mathematics*, vol. 280, pp. 193-219, Oct. 2001.

[23] Y. Chahlaoui and P. V. Dooren, “Benchmark Examples for Model Reduction of Linear Time-Invariant Dynamical Systems,” *Springer Proceedings of a Workshop Dimension Reduction of Large-Scale Systems*, vol. 45, pp. 379-392, Oct. 2003.

[24] M. Ghanavati, V. J. Majd, and M. Ghanavati, “Control of inverted pendulum system by using a new robust model predictive Control Strategy,” *IEEE Conference on Control and Communications Systems*, vol. 12, pp. 33-38, Sep. 2011.

[25] S. Jung, H. Cho, and T. C. Hsia, “Neural network control for position tracking of a two-axis inverted pendulum system: experimental studies,” *IEEE Trans Neural Netw.*, vol. 18, no. 4, pp. 1042-1048, Jul. 2007.

[26] L. B. Prasad, B. Tyagi, and H. O. Gupta, “Optimal control of nonlinear inverted pendulum dynamical system with disturbance Input using PID controller and LQR,” *IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, vol. 12, pp. 540-545, Nov. 2011.

[27] M. Juneja and S. K. Nagar, “Comparative study of model order reduction using combination of PSO with conventional reduction techniques,” *IEEE International Conference on Industrial Instrumentation and Control (ICIC)*, pp. 406-411, May 2015.

[28] S. Sehgal and S. Tiwari, “LQR control for stabilizing triple link Inverted pendulum system,” *IEEE International Conference on Power, Control and Embedded Systems*, pp. 1-5, Dec. 2012.

[29] Alok Sinha, *Linear Systems: Optimal and Robust Control*, CRC. Press, 2007.

[30] A. K. Mittal, R. Prasad, and S. P. Sharma, “Reduction of linear dynamic systems using an error minimization technique,” *Journal of Institution of Engineers (IE)*, vol. 84, pp. 201-206, Mar. 2004.

[31] S. Mukherjee, Satakshi, and R.C. Mittal, “Model order reduction using response matching technique,” *Journal Franklin Institute*, vol. 342, pp. 503-519, Aug. 2005.

[32] R. Prasad and J. Pal, “Stable reduction of linear systems by continued fractions” *Journal of Institution of Engineers (IE)*, vol. 72, pp. 113-116, Oct. 1991.

[33] I. D. Smith and T. N. Lucas, “Least-squares moment matching reduction methods” *Electronics Letters*, vol. 31, no. 11, pp. 929-930, May 1995.

II. HEURISTIC DYNAMIC PROGRAMMING FOR MOBILE ROBOT PATH PLANNING BASED ON DYNA APPROACH

S. Al-Dabooni and Donald C. Wunsch

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

ABSTRACT

This paper presents a direct heuristic dynamic programming based on Dyna planning (Dyna-HDP) for online model learning in the Markov decision process (MDP). This novel technique is composed of HDP policy learning to construct the Dyna agent for speeding up the learning time. We evaluate Dyna-HDP on a differential-drive wheeled mobile robot navigation problem in a 2D maze. The simulation is introduced to compare Dyna-HDP with other traditional reinforcement learning algorithms, namely one step Q-learning, Sarsa (λ), and Dyna-Q, under the same benchmark conditions. We demonstrate that our method has a faster near-optimal path than other algorithms with high stability. In addition, we also confirm that the Dyna-HDP method can be applied in a multi-robot path planning problem. The virtual common environment model is learned from sharing the robots' experiences which significantly reduces the learning time.

Keywords: Heuristic dynamic programming (HDP), Q-learning, Sarsa algorithm, Dyna learning, mobile robot, maze navigation, path planning.

1. INTRODUCTION

Exact path planning plays an important condition in the navigation of autonomous mobile robots. It enables robots to track an optimal collision-free path from the starting point to the target without colliding into obstacles. There are two categories for mobile robot path planning: One is a global path planning (off-line) based on a priori complete information about the environment (e.g., field testing) and the other is local path planning (on-line) based on sensory information in uncertain environments where the size, shape and location of obstacles are unknown. There are several Reinforcement Learning (RL) algorithms used to solve mobile robot path planning problems. Q-learning has been frequently used [1] - [3]. The agent is evaluated by the value function for a given state. The finite MDP, occurs when the states and actions space are finite [4]. According to Bellman's optimality principle [5], the optimal strategy is obtained by building an optimal policy for a subproblem, which can be traced backward to previous solutions and can build on the optimal policies established by other subproblems until entire process is covered. The Bellman's optimality equation can be written as [6]

$$J^*(s,a) = P^a_{ss'}[R^a_{ss'} + \gamma \max_{a \in A} J^*(s',a)] \tag{1}$$

where $J^*(s,a)$ is the value function of the current state s; $P^a_{ss'}$ is the transition probability to move to the next state s' after carrying out an action, a, which belongs to a set of all possible actions A, $R^a_{ss'}$ transition reward from s to s', and γ is the discount factor. In addition to Q-learning, temporal difference (TD) learning is used to solve Bellman's equation in the Markov decision process (MDP). More sophisticated approaches such as Dyna-Q [7] and TD (λ) [8] have been employed to improve the convergence speed for solving Bellman's principal of optimality problems. Another approach for solving Bellman's equation has been adaptive dynamic programming (ADP), which finds the optimal control policy over time. The idea is to use a function approximation structure such as neural networks to approximate the value function [9]. ADP has three fundamental approaches: Heuristic dynamic programming (HDP), dual heuristic programming, and globalized dual heuristic programming [10] - [12], which consists of three neural networks (actor, critic, and model networks) which provide decision making, evaluation, and prediction, respectively. If ADP has only actor and critic networks, then it is referred to as action dependent ADP [10], [11], [13]. Action dependent ADP with online learning [14] is mixed between adaptive critic designs and Q-learning by online updating of the value function with policy. Improving online ADP learning is done by adding dual critic networks for HDP designs [15], [16] and for DHP designs [17]. ADP can be applied in many applications. In [18], the authors prove that dual heuristic programming can control the operation of a turbo-generator more efficiently than HDP. The dual heuristic programming approach can also improve power performance in collective robotic search problem as shown in [19]. In [20], HDP controls on motion planning for wheeled mobile robot to escape from sharp corners. This work presents a new vision of HDP which is used in path planning based on Dyna algorithm; we refer on it as Dyna-HDP, which is used to obtain the value function of intelligent mobile robot navigation. Although the planning process is computationally intensive and time consuming, the total project completion should also count equal time spend in acting, model learning, and direct RL processes, which require little computation. In this work, we also apply this novel technique with multi-robots. They are employed to share their experiences by distributing the value functions among them. There are three points to consider when arranging for mobile robot navigation in an unknown environment [21]: 1) the mathematical model of the environment is generally unavailable, 2) sensory data are uncertain and imprecise due to noise, and 3) real time operation is essential; therefore, a fuzzy logic controller (FLC) is used in this work to prevent a robot from colliding against any facing obstacle (or other robots). The FLC is an attempt to reduce the training time needed. Moreover, FLC can make sure that a robot is able to move away from the trapping situations while moving towards the target. We use FLC as in [22] with changing in inputs and output function variables to adapt with our problem to deal with RL. In many

RL algorithms, an ϵ-greedy method and Boltzmann selection method are often used as action selection algorithms, but designers are required to adjust the value of ϵ (or τ in the case of a Boltzmann selection) by trial and error in order to realize a good balance between exploration and exploitation. There are dozens of researchers trying to solve this balance such as [23] - [25]. For instance in [24], the authors define two types of agents: an exploitation agent and another agent for exploration, and these agents can communicate in order to combine their experiences. The remaining sections are organized as follows: Section 2 presents a collision-free navigation. The fundamental RL algorithms are showed in Section 4. Mobile robot path planning based on Dyna-HDP is demonstrated in Section 4. The simulation results and conclusion are presented in Section 5 and 6, respectively.

2. COLLISION-FREE NAVIGATION

2.1. Simulation of Mobile Robot Platform.

Our simulated platform for a mobile robot consists of nine infrared sensors, which are mounted on the exterior of the robot's body as shown in Fig. 1. It forms an 180° field of view in front of the robot with 22.5° separation angle between every two neighboring sensors. The distance measurement sensor between the mobile robot and the obstacles is labeled as $(S_{d1} - S_{d9})$. The simulation is designed with two driving wheels independently operated, left and right wheels, with one front caster wheel to support the robot. The robot can sense a solid object about 20 centimeters away from its body. The size of the robot's body frame is 14 cm for both of width and length as shown in Fig. 1. This independent drive system not only gives the mobile robot capability to move in a straight line, and perform turns, but this system also allows the robot to have a zero turning radius. This allows mobile robot to turn directly around from its center without backward motion. A Matlab application was taken as the mobile robot platform.

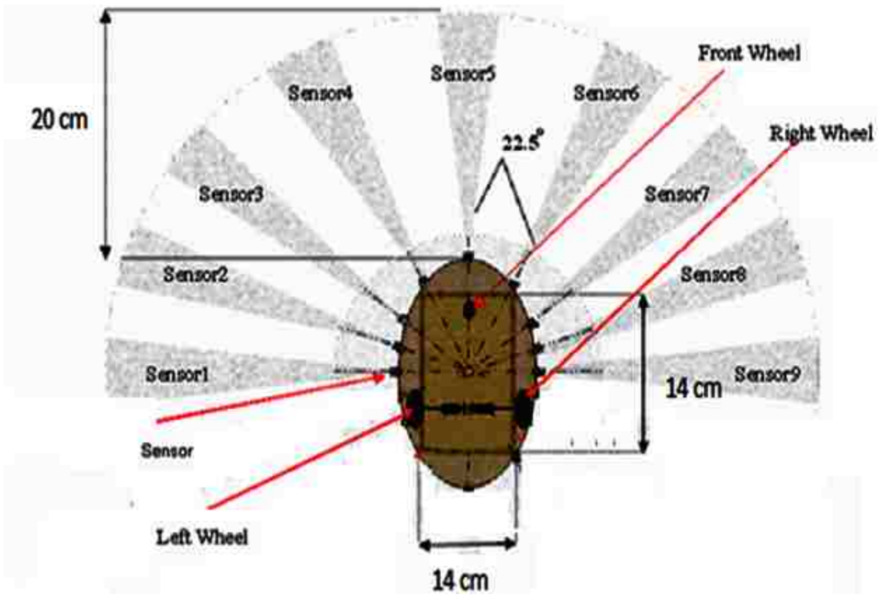


Figure 1. Differential wheeled mobile robot platform. It forms as 180° field of view in front by nine sensors with 22.5° separation angle to sense an object about 0.2 meter away from its body.

2.2. Description of the Fuzzy Controller for Obstacle Avoidance. The FLC is applied to realize a mobile robot motion in an unknown environment with obstacles. It helps to reduce training time by giving the ability to select suitable actions near to objectives. Inputs into this fuzzy controller for obstacles avoidance (FCOA) are the outputs from the sensors after a triangular calculation for left sensor group $S_{d1} - S_{d5}$, and right sensor group $S_{d5} - S_{d9}$ to obtain: The left distance which is the lowest distance between mobile robot and left obstacle, the left angle which is the angle between heading angle of the robot and left obstacle, the right distance which is the lowest distance between mobile robot and right obstacle, and the right angle which is the angle between heading angle of the robot and right obstacle. The outputs of the FCOA are right and left wheels angular speed, W_r and W_l respectively. The following membership functions for inputs (distance and angle) and outputs (angular speeds) are:

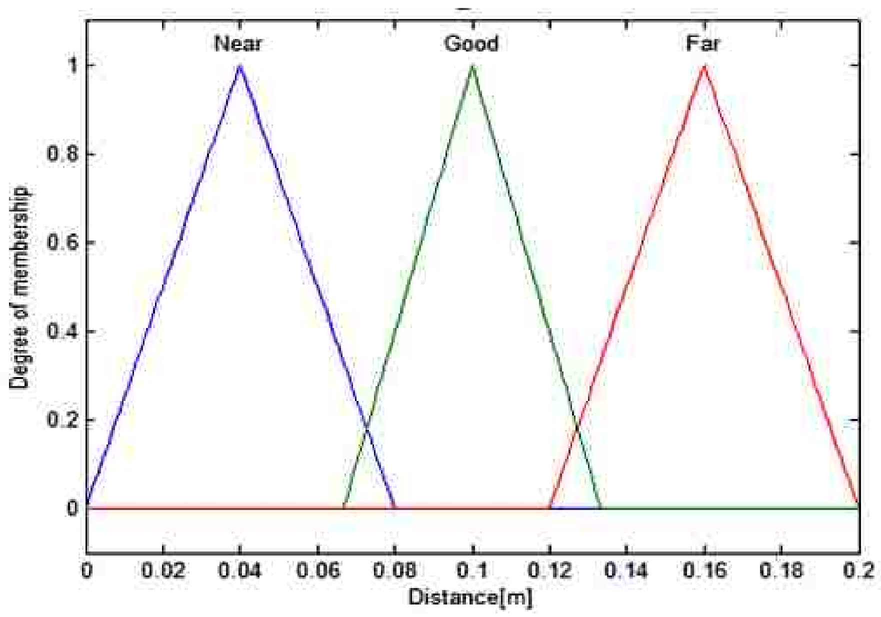


Figure 2. Distance fuzzy membership functions. These fuzzy set definitions are used for input variable which are consisted from three triangular-shaped membership functions (good, near, and far).

2.2.1. Distance. The distance membership functions represent robot conditions which are good, near, and far from the obstacle as shown in Fig. 2.

2.2.2. Angle. The angle membership functions shown in Fig. 3 represent angle conditions which are very negative, negative, zero, positive, and very positive.

2.2.3. Angular Speed. The left and right wheel angular speed membership functions shown in Fig. 4 as five triangular wheel speed conditions for both wheels which are backward, slow backward, stop, slow forward, and forward. All input variables of membership functions are used in both right and left sides. The next step illustrates the appropriate fuzzy logic rules. Two rule schemes were set up as follows: Right side control rules and left side control rules. Table I illustrates the matrix of rules for right side controller rules, while left side controller rules are illustrated in Table 1. The distance input is located on the vertical column, and the angle input is located on the horizontal row. Based on the fuzzified values of the inputs, the controller selects the appropriate angular speed conditions listed in

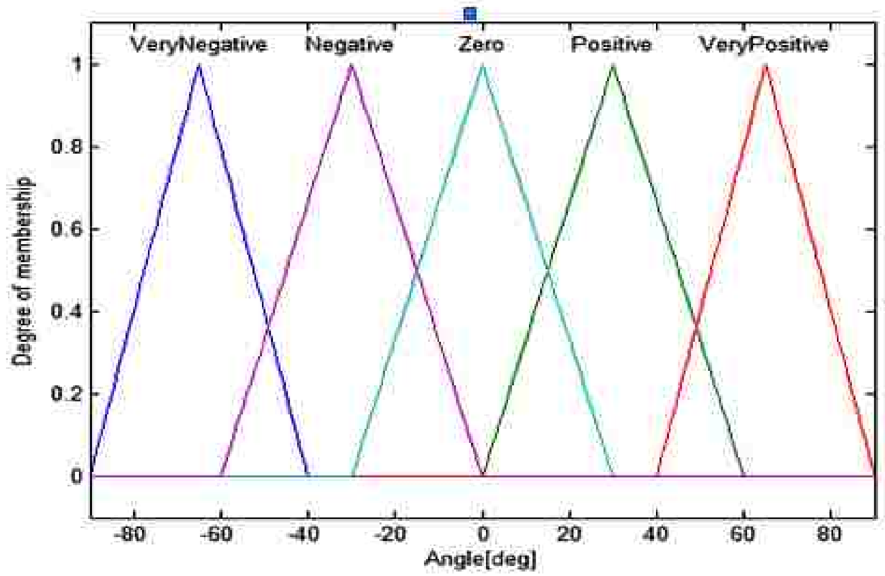


Figure 3. Angle fuzzy membership functions. These fuzzy set definitions are used for input variable which are consisted from five triangular-shaped membership functions (very negative, negative, zero, positive, and very positive).

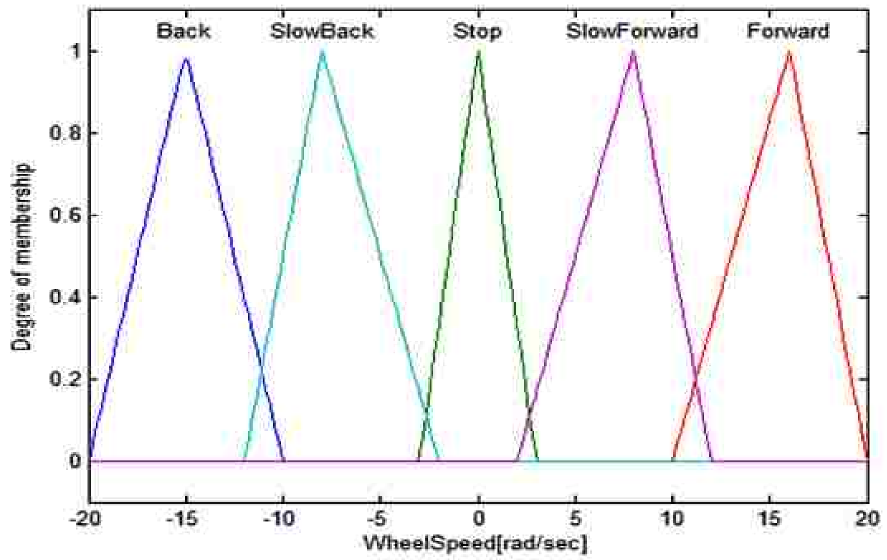


Figure 4. Angular speed fuzzy membership functions. These fuzzy set definitions are used for output variables for both left and right wheels which are consisted from five triangular-shaped membership functions (backward, slow backward, stop, slow forward, and forward).

Table 1. Right side controller rules matrix

		Angle				
		very negative	negative	zero	positive	very positive
Distance	good	RF, LSF	RF, LSF	RSF, LF	RSB, LSF	RSB, LSF
	near	RSF, LSB	RSF, LSB	RF, LF	RSB, LSF	RSB, LSF
	far	RSF, LSB	RSF, LSB	RF, LSF	RSF, LF	RSF, LF

Table 2. Left side controller rules matrix

Angular Speed		Angle				
		very negative	negative	zero	positive	very positive
Distance	good	RSF, LSB	RSF, LSB	RF, LSF	RSF, LF	RSF, LF
	near	RSF, LSB	RSF, LSB	RF, LF	RSB, LSF	RSB, LSF
	far	RF, LSF	RF, LSF	RS, LSF	RSB, LSF	RSB, LSF

the matrix (RSF a is right wheel slow forward, LSB is a left wheel slow backward, RF=right wheel forward, LSF is a left wheel slow forward, LF is a left wheel forward, RSB is a right wheel slow back, RS is a right wheel slow). The right side controller rules are used when distance from any obstacle on the right side of mobile robot is less than left side distance of obstacle and vice versa for left side controller rules.

In order to use right side control rules table, let assuming the distance measurement of an obstacle in the right side is less than in the left, and angle input variable is negative and distance input variable is near then the angular speed for the right wheel is slow forward (RSF) and the angular speed for the left wheel is slow backward (LSB) which is indicated that in a shadow box. In this work, there are two avoidance strategies. The first one is updating an environment model. The agent tries to avoid the new obstacles with increasing its experience. This strategy helps the agent to identify the environment model by using RL. Another avoidance strategy happens if the distance between two or more agents is less than the distance measurement sensors by comparing with other agents position. This strategy requires no updating experience for sensing agents while avoiding other agents (with/without obstacles avoidance).

3. FUNDAMENTAL REINFORCEMENT LEARNING PARAMETERS

An online learning technique for RL depends on trial and error through repeated interaction between an agent and environment. The agent receives the state of the environment via its sensors, and it tries to maximize the reward by selecting a correct action which depends on a reinforcement signal. The mobile robot's current position (x_c, y_c) is detected by using an odometer sensor like an encoder with no slipping or sliding assumed. A target position (x_g, y_g) is given. D_g is the distance between the current position and the target.θ_{diff} is a difference angle between the robot's heading angle (θ_g) and the goal angle (θ_r). In other words, θ_r is the angle between the robot's moving direction and a line connecting the target with the mobile robot. The goal of RL is to teach the mobile robot to align θ_g with θ_r, or in other words, to minimize θ_{diff}. The captured states in this work are represented by the sensor readings ($S_{d5} - S_{d9}$), the relative different angle θ_{diff}, and the relative distance D_g). In order to minimize the size of learning space, we quantify the previous inputs into limited variables. The sensor readings take two values, 0 if the obstacle is far away from current robot position, or 1 if it is near. The target distance variable is quantified into five values with respect to the distance between the robot and target (very far, far, middle, near, and very near). Finally, the different angle is quantified into six parts graduating from good reward into punishment as follows: $0° - \pm 15°$, $\pm 15° - \pm 30°$,$\pm 30° - \pm 45°$, $\pm 45° - \pm 60°$,$\pm 60° - \pm 75°$, and $\pm 75° - \pm 90°$. A lookup-table is used to learn the states. However, if there are too many states, an approximated function can handle this problem [3], [26]. Three actions $(a_i, i = 1, 2, 3)$ are used in this work which are: move 0.1 meter forward(a_1), turn 45° in clockwise direction (a_2), and turn 45° in counter clockwise direction (a_3). RL selects an optimal behavior to get the optimal path for mobile robot based on previous learning of the experienced series actions. The numerical reward (or punishment) returned from the environment. The aim of this work is to arrive at the goal in the shortest time with collision-free motion planning. In this work, $S_{d5} - S_{d9}$, θ_{diff}, and D_g are used to design a suitable reward or reinforcement signal from the environment. A reinforcement function (r_t) at state (s_t) after doing action (a_t) is designed as follows:

$$r_t = \begin{cases} -\sum_{i=1}^{9} C_i S_{di} + C_d D_g + C_\theta \theta_{diff} & \text{if } s_t \neq Goal \\ 2r_{t-1} & \text{if } s_t = Goal \end{cases} \tag{2}$$

where, C_i, C_d,and C_θ are small values used for balancing the weight parameters with sensor reading variables, which are the relative different angle and the relative distance, respectively. These values reflect the level of influence for each variable on mobile robot learning. Because FCOA is used in our work, the robot continues navigating in the environment without termination until reaching to target position which makes collecting the data flexible fast, and easy. The three actions generated from FCOA compare the angular speed for left W_l and right W_r wheels as follows:

$$Action = \begin{cases} a_1 & \text{if } |W_r - W_l| < \beta \\ a_2 & \text{elseif } W_r < W_l \\ a_3 & \text{otherwise} \end{cases} \tag{3}$$

where, β is the small tolerance value; the robot goes forward if the difference between left and right wheels less than β, otherwise it should be turned into left or right depending on a larger side of angular velocity of wheel than the other. We use two most popular RL algorithms (Q-learning and Sarsa) as in [4] to compare with our approach.

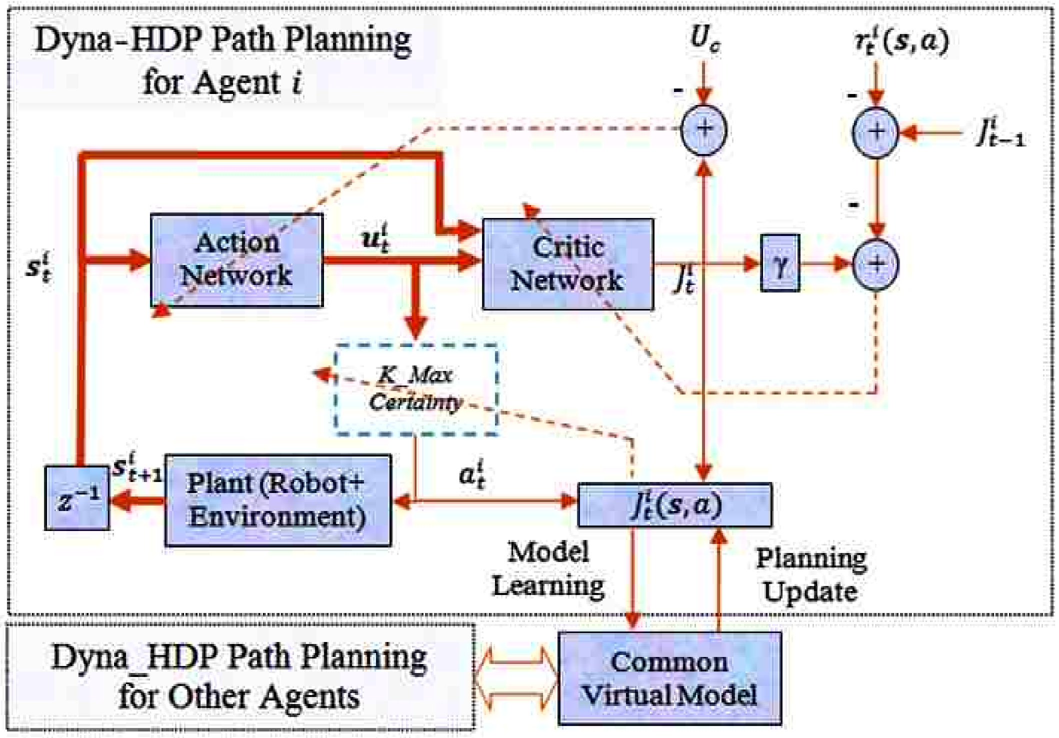


Figure 5. Block diagram for Dyna-HDP Path Planning. u_t^i is the action vector at time t for robot i which is consisted of three actions (turn left, turn right, and moving forward) denoting as a_t^i. s_t^i is the input states vector for robot i at time t which are represented by the nine sensor readings $S_{d5} - S_{d9}$, the relative different angle (θ_{diff}), and the relative distance (D_g). A reinforcement function (r) can get from (2) for state s_t^i and a_i. All robots share the same virtual model to maximize the value function J_t^i for all agents at the same time. The backpropagation path is shown by dashed lines for action and critic networks, and for updating the rules for k-max certainty.

4. MOBILE ROBOT PATH PLANNING BASED ON DYNA-HDP

The main block diagram for the featured Dyna-HDP path planning method for is illustrated in Fig. 5, which is showed the details for agent i with other blocks for multi-robot. This section will describe all aspects of this method based on the following subsections.

4.1. Architecture of The Dyna-HDP System. Dyna-Q learning algorithm is an integration of direct RL and a planning method strategy. The planning method takes model as an input to updating according to Q value learning as follows: $Q_t(s, a_i) = Q_t(s, a_i) +$

$\alpha\Big[r_t + \gamma \max_{a_i \in A, \forall i}\Big(Q_{t+1}(s, a_i)\Big) - Q(s, a_i)\Big]$, where s_t is current state, a_t^i is current action, α is learning rate, and γ is discount factor. While in Dyna-HDP method, the updating model happens by value function (J_t), which directly comes from critic neural network. The Dyna algorithm is the integration of direct RL and planning methods. The planning method takes model as an input. The model produces a prediction of the resultant next state and next reward if it is given a state and an action; therefore, it is used to mimic the environment and produce simulated experience. A real experience is generated by using direct RL method to improve the policy and the value function while improving the model as well. The transition probability in (1) can only take the value of 1 or 0 (whether directly applicable to the selected action or not); therefore, we can be rewriting the Bellman's optimality equation as:

$$J^*(s, a) = \max_{a \in A}\Big[r(s, a) + \gamma J^*(s', a)\Big] \tag{4}$$

The optimal control $\pi^*(s)$ should satisfy:

$$\pi^*(s) = arg \max_{a \in A}\Big[r(s, a) + \gamma J^*(s', a)\Big]. \tag{5}$$

HDP consists of blocks called the action network and critic network. It also uses online learning for the neural networks [14]. The action network is used to generate a control signal which is evaluated by the critic network. The multilayer perceptron structure is employed for action and critic neural networks. In this work, one hidden layer is used to construct both networks. The critic network is used to approximate the optimal cost function is defined as in (4). The temporal difference (prediction error) for the critic network is defined as

$$\delta_t = J_{t-1} - (r_t + \gamma J_t). \tag{6}$$

Therefore, the objective function to be minimized in the critic network is

$$E_t^c = \frac{1}{2}\delta_t^2. \tag{7}$$

The gradient-based adaptation for the weights update rule in the critic network can be given by

$$w_{t+1}^c = w_t^c + \triangle w_t^c. \tag{8}$$

$$\triangle w_t^c = \ell_t^c\Big[-\frac{\partial E_t^c}{\partial w_t^c}\Big]. \tag{9}$$

$$\frac{\partial E_t^c}{\partial w_t^c} = \left[\frac{\partial E_t^c}{\partial J_t}\frac{\partial J_t}{\partial w_t^c}\right] \tag{10}$$

where ℓ_t^c is the learning rate of the critic network at time t, which decreases with time until a certain small value, and (w_t^c) is the weight vector in the critic network. In order to approximate the optimal control signal as in (5), the action network adapts indirectly. The error between the desired ultimate objected (U_c) and the approximate value function (J_t) is backpropagated through critic network. The desired ultimate objective value sets to straight distance value from initial position of the mobile robot to the target, which maximizes the total reward. The error function of an action network can be defined as

$$\mu_t = J_t - U_c \tag{11}$$

Therefore, the objective function in the action network is

$$E_t^a = 0.5\mu_t^2. \tag{12}$$

The weight updating in the action network is similar to the critic network. The gradient-based adaptation for the weights update rule in the action network can be given by

$$w^a_{t+1} = w^a_t + \triangle w^a_t. \tag{13}$$

$$\triangle w^a_t = \ell^a_t \left[-\frac{\partial E^a_t}{\partial w^a_t} \right]. \tag{14}$$

$$\frac{\partial E^a_t}{\partial w^a_t} = \left[\frac{\partial E^a_t}{\partial J_t} \frac{\partial J_t}{\partial u^a_t} \frac{\partial u_t}{\partial w^a_t} \right]. \tag{15}$$

where ℓ^a_t is the learning rate of the action network at time t, which is also decreasing with time until certain small value, and w^a_t is the weight vector in the action network.

4.2. k-max Certainty Method. The main purpose of this work is to give the agent ability to increase the optimum policy efficiency within a minimum number of trials. The efficiency and minimum trials can be obtained by balancing the properties of exploration intensive strategy (non-greedy) and exploitation intensive strategy (greedy). A large body of research addresses related problems such as [23] - [25]. The exploration intensive strategy needs numerous trials for increasing the optimality because it tries to collect all experiences. The k-certainty exploration method [25] tries to identify the environment precisely with the least number of trials. The k-certainty exploration method selects all sensory-action pairs (rules) uniformly which increases the guarantee for efficient identification of the environment. The drawback for this method is not influenced by positions of rewards. This paper presents a modification which is called k-max certainty. The k-max certainty is combined the k-certainty exploration method and exploitation method through decreasing ε greedy value method. Fig. 6 depicts a flow chart for a k-max certainty method. The k-max certainty method starts with a full exploration by setting a ε greedy value with a high value and then decreases overtime by a small ratio (ε). The rule that has been selected k times is called a k-certainty rule; otherwise, a new rule becomes a part of a k-uncertainty rules. In the exploration part, if there are any k-uncertainty rules connected with the current state, an action is selected randomly among these uncertainty rules. If all rules belonged to the current state have been taken, the action would be selected to transfer to another state which own k-uncertainty rules. In other words, let say, there are four states connected with the current state, three of them have k-uncertainty rules, then a random action would be selected from these three states. These process called a k-uncertainty trail which is controlled by flag signs in each state. These flags are also used to avoid repeatedly selecting of k-certain rules that make returning case to the current state (k-certain looped). All sensing states have raised flags except k-uncertainty rules with down flags. If the current state does not have any k-uncertainty rules, the exploitation case has been triggered by taking a greedy action among k-certainty rules. Any selected action among k-certainty rules for agent i is mentioned as follows: $\pi(s^i) = arg\max_{a \in A_{cen}} J_t^i(s, a)$, where $A_{cen} = u^i = [a_1^i, a_2^i, a_3^i]$ as shown in Fig. 6. The identification environment level is directed proportionally with the k value. The k value is set to one with increasing by one if no any uncertain rule (or trail) belongs to this state, otherwise it resets to one. In case that the greedy value allows to go to the exploitation part directly, the agent selects an action as follows: $\pi(s^i) = arg\max_{a \in A} J_t^i(s, a)$, where $A = u^i$. The k-max certainty will become idle if the robot senses any objects (obstacles or robots) and the FCOA will apply two avoidance strategies as motioned in Section 2.

4.3. Knowledge Sharing for Distributed Mobile Robots. One of the most effective RL methods to integrate online planning is Dyna [27]. It uses real experience to learn a model of a certain environment. In this work, we assign an architecture for sharing information between all agents to learn the model which decreases learning time. As in Fig. 5, the common virtual model takes the states and actions from all robots to predict the next states and rewards. All robots share the same task, which means they try to reach to the specific target position. In this work, we use two worlds (actual and planning worlds) for updating value function. In order to avoid excessive time spent in the planning process which is inherently computationally intensive, the requirement time to complete this process should equal the time steps spent in: The acting, common model learning, and the direct RL

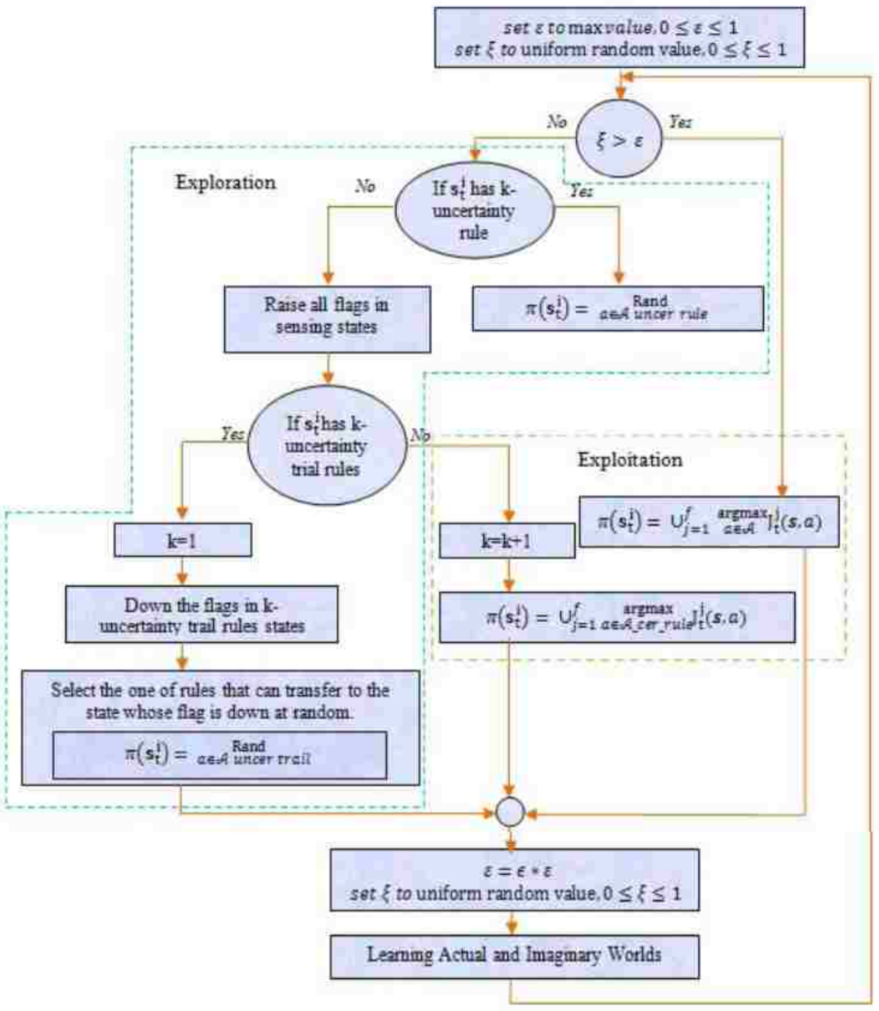

$set\ \varepsilon\ to\ \max value, 0 \le \varepsilon \le 1$
$set\ \xi\ to$ uniform random value, $0 \le \xi \le 1$
$\xi > \varepsilon$
No
Yes
Exploration
No
If s_t^i has k-uncertainty rule
Yes
Raise all flags in sensing states
$\pi(s_t^i) = \underset{a \in \mathcal{A}\ uncer\ rule}{\text{Rand}}$
If s_t^i has k-uncertainty trial rules
Yes
No
Exploitation
k=1
k=k+1
$\pi(s_t^i) = \cup_{j=1}^{f} \underset{a \in \mathcal{A}}{\text{argmax}} J_t^i(s,a)$
Down the flags in k-uncertainty trail rules states
$\pi(s_t^i) = \cup_{j=1}^{f} \underset{a \in \mathcal{A}\_cer\_rule}{\text{argmax}} J_t^i(s,a)$
Select the one of rules that can transfer to the state whose flag is down at random.
$\pi(s_t^i) = \underset{a \in \mathcal{A}\ uncer\ trail}{\text{Rand}}$
$\varepsilon = \epsilon * \varepsilon$
$set\ \xi\ to$ uniform random value, $0 \le \xi \le 1$
Learning Actual and Imaginary Worlds

Figure 6. A flow chart for k-max certainty selection procedure for agent i. A green dashed line border represents the exploitation approach which has been triggered by taking a greedy action among k-certainty rules (sensory-action pairs). The exploration approach is represented by a blue dashed line border which is selected action randomly among uncertainty rules (or trail).

process which requires little computation. In other words, the implementation time between the actual world (with common model learning) and planning world becomes equal. Both worlds have separated acting and learning approaches. In the acting approach, the mobile robot connects with other robots to decide what action should be selected. This decision is made on the basis of k-max certainty selection approach (exploitation part) if the robot does not sense any objects. After running this action in the actual world, the robot moves from its current real state to next real state. During this movement, the planning world starts to plan by selecting an action based on k-max certainty sharing information with other remaining robots in random common model state. The model for certain agent is same as the common model. During real movement for them, all agents learn and update the common model simultaneously. This learning is happened by updating the value functions as in (17). More specifically, let $M_i \in M_G$ is the mobile robot i, where $M_G = M_1, M_2, , M_f$ denotes of set of f mobile robots. In certain state, let M_i has a plan to select a best action. M_i inquires part of robots or whole group M_G to identify this state. The other robots (or perhaps only one other robot) announce the set of value functions with various actions applying in same state. M_i decides to select an action according to maximum value functions as follows:

$$\pi(s^i) = arg \max_{a \in A} \left(max \cup_{j=1}^{f} J^j(s, a)), 1 \leq j \leq f \right) \tag{16}$$

The certain state could be a real current state in the actual world or a random common model state in the planning world. In the learning approach, both worlds learn directly after taking a decision for selecting a suitable action. The learning approach for both worlds is done based on Bellman's optimal policy equation as follows:

$$J^i(s, a) = r^i(s, a) + \gamma max \cup_{j=1}^{f} \left(\max_{a \in A_{cer}} J^j(s', a) \right), \tag{17}$$

where f is the total number of mobile robots; $r^i(s,a)$ is the instant reward for robot i $(1 \leq i \leq f)$ between current state (s) and next state (s') during currying out action (a); $max \cup_{j=1}^{f} (.)$ is the maximum value among all robots; $\max_{a \in A_{cer}} J^j(s', a)$ is the maximum value function for robot j in the next state among all certainty actions (A_{cer}), which are performed in this state by various other robots. The new sensory action pair updating value function shares with all other robots; therefore, all robots eventually reach the optimal policy with a short limit time. If M_i detects any objects, it will use the FCOA with two avoidance strategies as follows: M_i updates its experiences by using (17) if the objects are only obstacles, otherwise M_i tries to only avoid these objects without any updating whereas the objects are robots with/without obstacles. In order to implement Dyna-HDP, Ns possible states is taken for a deterministic and finite MDP maze environment. Therefore, Bellman's optimality equation can be written as

$$J^*(s,a) = arg \max_{a \in A} \Big[r(s,a) + \gamma \sum_{j=1}^{N} s J^*(s', a') \Big], \tag{18}$$

Where $J^*(s,a)$ is the maximum total reward after taken an action a at state s. We introduce the entire flowchart for Dyna-HDP with multi-agents with K-Max-certainty method in Fig. 7. The implementation steps can be described as follows:

1. Start with an initial state position in the environment, which is the nine sensor readings, the relative difference angle, and the relative distance.

2. Obtain the action vector from action network, which consists of three actions (turn left, turn right, and moving forward).

3. Use the K-Max-certainty technique to select one action, balancing between exploitation and exploration as in Fig. 9. Select one optimal action maximum of the value function as (17) after comparing each agent's generated action with other agents in same state.

4. Obtain a new state from the environment after applying the optimal action and update the inputs for the critic network to obtain the new local value function.

5. Compere the value local function with other agents as (18).

6. Check if the agent moves near to any object? If yes, the fuzzy controller fuzzy controller for obstacle avoidance (FCOA) generates angular velocities for left and right wheels to avoid the object by selecting a new action (3); the action network updates as assigned punishment according to the reinforcement function (2) while updating the k-Max-certainty strategy; if no, agent moves to another step on the same trail.

7. Check if the agent reaches the target position? If yes, the procedure turns to another trial, while assigning the reward and loading the initial state in the start position; if no, the agent moves to another step with same trail.

8. Share virtual model with all agents to maximize the value function.

9. Terminate the entire learning process if the comment value function table in the virtual model remain unchanged. In other words, the value function table for all agents are same.

In other word as shown in Fig. 7, the agent i starts at its initial state in the environment. According to mobile robot sensors, the initial state for agent i (s_0^i) represents the nine sensor readings, the relative angle difference, and the relative distance at initial time. The action vector (u_t^i) consists of three actions (turn left, turn right, and moving forward). k-max certainty technique selects one action (a_t^i), conforming to the balancing between exploitation and exploration as in Fig. 5. Agent i compares its generated action with other $f - 1$ agents in same related state to take which one gives a maximum of the value function as in (17). After Agent i applies an optimal action, it obtains the new state from the plant and update the inputs for the critic network, obtains the new value function ($J_t^i(s, a)$). According to

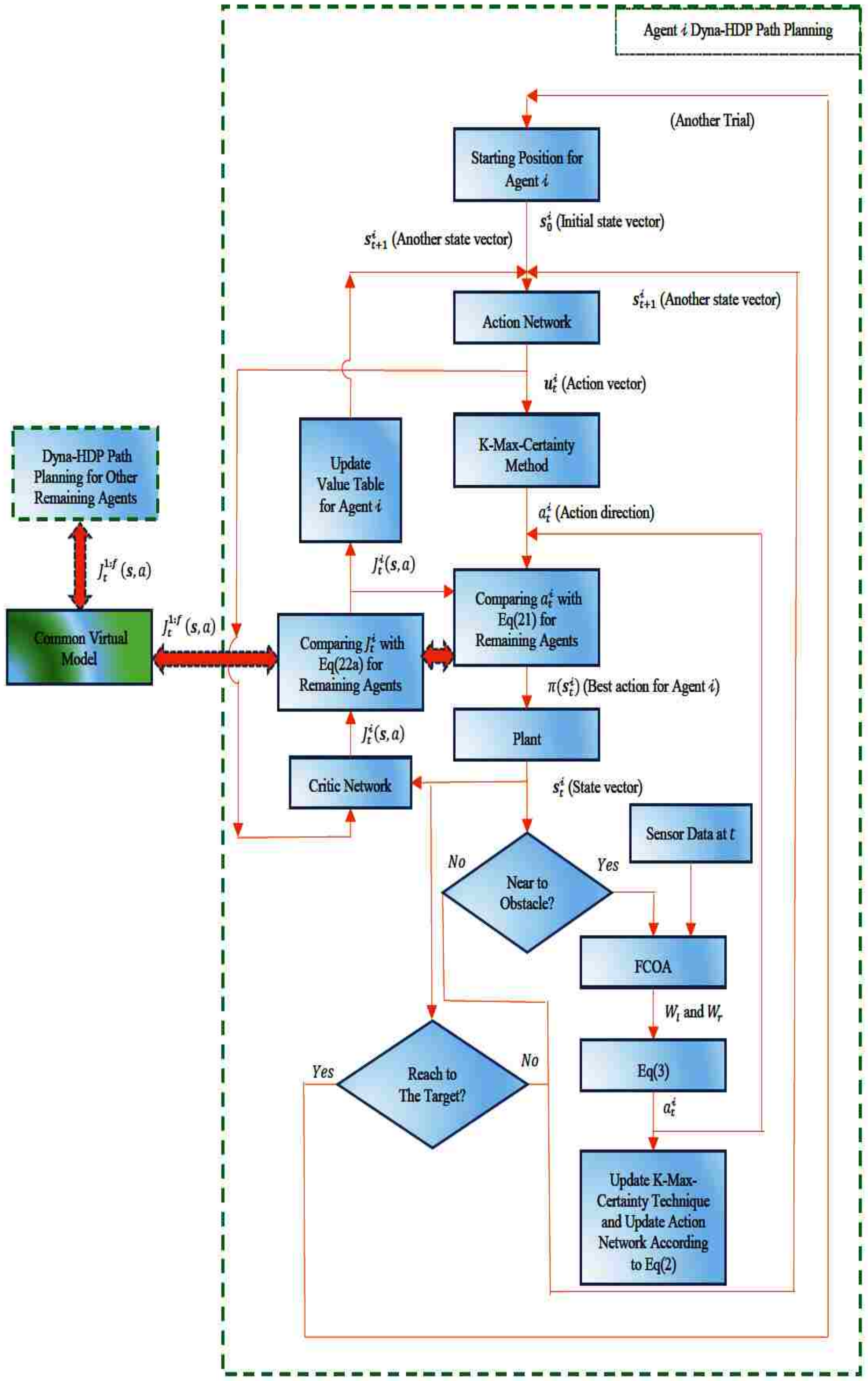
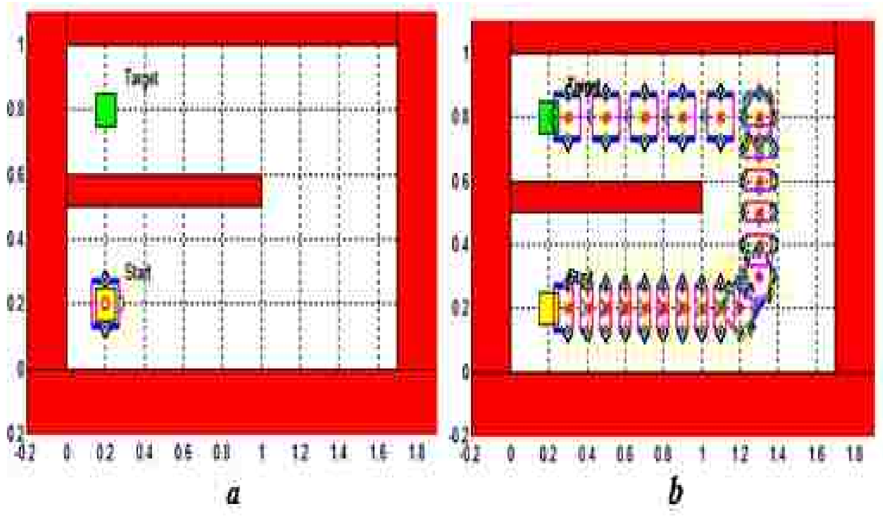

Agent i Dyna-HDP Path Planning
(Another Trial)
Starting Position for Agent i
s_0^i (Initial state vector)
s_{t+1}^i (Another state vector)
Action Network
s_{t+1}^i (Another state vector)
u_t^i (Action vector)
Update Value Table for Agent i
K-Max-Certainty Method
Dyna-HDP Path Planning for Other Remaining Agents
a_t^i (Action direction)
$J_t^{1:f}(s,a)$
$J_t^i(s,a)$
Comparing a_t^i with Eq(21) for Remaining Agents
Common Virtual Model
$J_t^{1:f}(s,a)$
Comparing J_t^i with Eq(22a) for Remaining Agents
$\pi(s_t^i)$ (Best action for Agent i)
$J_t^i(s,a)$
Plant
Critic Network
s_t^i (State vector)
Sensor Data at t
No
Yes
Near to Obstacle?
FCOA
W_l and W_r
Yes
No
Reach to The Target?
Eq(3)
a_t^i
Update K-Max-Certainty Technique and Update Action Network According to Eq(2)

Figure 7. Flowchart for implementation the multi-agents Dyna-HDP approach on maze problem.

(18), agent i updates its local value table. Checking the agent's movement, if it is near to any obstacle, the fuzzy controller FCOA generates angular velocities for left and right wheels (W_l, W_r) to avoid the object by selecting a new action (3). At the same time, the action network updates as assigned punishment according to the reinforcement function (2) while updating the k-max-certainty strategy. If the agent is far away from any obstacle, it moves to another step with same trail. When the agent reaches to the target position, the procedure turns to another trial with assigning the reward and load the initial state in the start position, otherwise, it moves to another step with same trail. All agents share the same virtual model to maximize the value function J_t^i for all agents at the same time. The process is terminated if the common value function table in the virtual model remain unchanged.

5. SIMULATION RESULTS AND ANALYSIS

In this section, we apply Dyna-HDP to learn the distribution of autonomous mobile robots on an unknown environment. Walls surround this environment, with many obstacles distributing on it. But before testing Dyna-HDP approach, we test k-max certainty approach with traditional $\varepsilon-$greedy method in a simple environment as shown in Fig. 8 a. A final optimal trajectory (collision-free path) presents in Fig. 8 b from starting point (0.2 for both x-y axis with 0° for the orientation angle) to a target position. This final optimal collision-free trajectory is obtained after 120 episodes by using tradition Q-learning algorithm. Fig. 8 c demonstrates that k-max certainty has better performance than $\varepsilon-$greedy. The x-axis in Fig. 8 c gives the statistics of the learning periods (episodes), and the y-axis means the number of steps for each episode. k-max certainty can reach to the stable running (finding the optimal and safe path) after 72 episodes while 102 episodes for $\varepsilon-$greedy. Furthermore, a number of steps over 120 episodes for k-max certainty is 3.5213e+04 for k-max certainty approach and 7.6606e+04 for $\varepsilon-$greedy (54.033% improvement). We consider three cases for testing our approach (Dyna-HDP) at the same complex environment. In the first case,



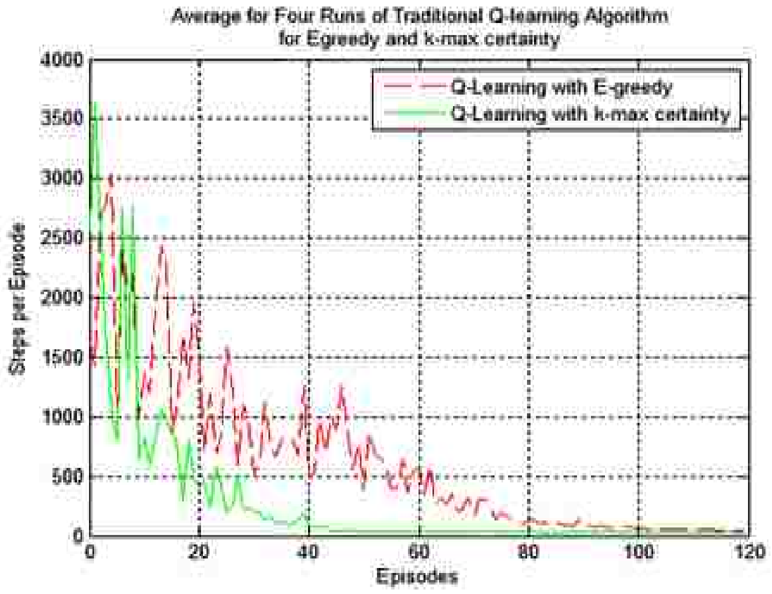


Figure 8. a) The environment for testing the exploration/exploitation strategy which shows the initial start position for this agent. b) The near-optimality trajectory from the starting point to the target. c) The comparison between $\varepsilon-$greedy and k-max certainty by using Q-learning algorithm.

we compare Dyna-HDP as discussed in Section IV with other conventional algorithms, one step Q-learning, Sarsa (λ), and Dyna-Q, for one agent (without sharing any information with other agents and all algorithms use k-max certainty approach). The initial start position for this agent is 0.2 for both x-y axis with 45°Âř for the heading angle. Fig. 9 illustrates the simulation results for this case. Dyna-HDP obtains a quicker near-optimality path than one step Q-learning, Sarsa (λ), and Dyna-Q with more stability through a small value of changing in the step number per each episode. Dyna-HDP improves about 47.667% ($3.5555e + 05$ out of $6.7940e + 05$) of number of steps compared with Dyna-Q algorithm; moreover, Dyna-Q learning algorithm needs around of 192 episodes to reach to the near-optimal path, while 145 episodes in Dyna-HDP (the near-optimality path length for this case needs 107 steps). The stability comes from a sample mean of a number of steps over a number of episodes and diversity for this sample. The sample mean for Dyna-Q learning algorithm from 50 to 150 episodes is 3.2048e+03 steps while 1.1553e+03 steps for Dyna-HDP. In a second case, two agents share their experiences by using Dyna-HDP approach as shown in Fig. 10. Every agent has its own experience and tries to support the other. This support is carried out by sharing information from high rate experience to low rate at certain state. For instance, agents M_i has more experience in certain state (or sensory information) than M_j in same state; therefore, M_i learns M_j about this state and vice versa for learner-teacher interaction with other states. In the third case, all agents share their experiences with the same approach. Fig. 11 shows the results for third cases. In these figures (second and third cases), the results are stable with the improvement in the learning speed of distributed autonomous agents by sharing their experiences. The environment, which is used in three cases are shown in Fig. 12 with two (agent 1 and agent 2) and five mobile robots for first and second case, respectively. The robots are distributed in random start positions and the heading angles (orientations) to achieve a same task (reaching a specific target area). Fig. 13 demonstrates the near-optimal trajectories for five agents from start positions to target after applying the third case. Fig. 14 shows all cases together

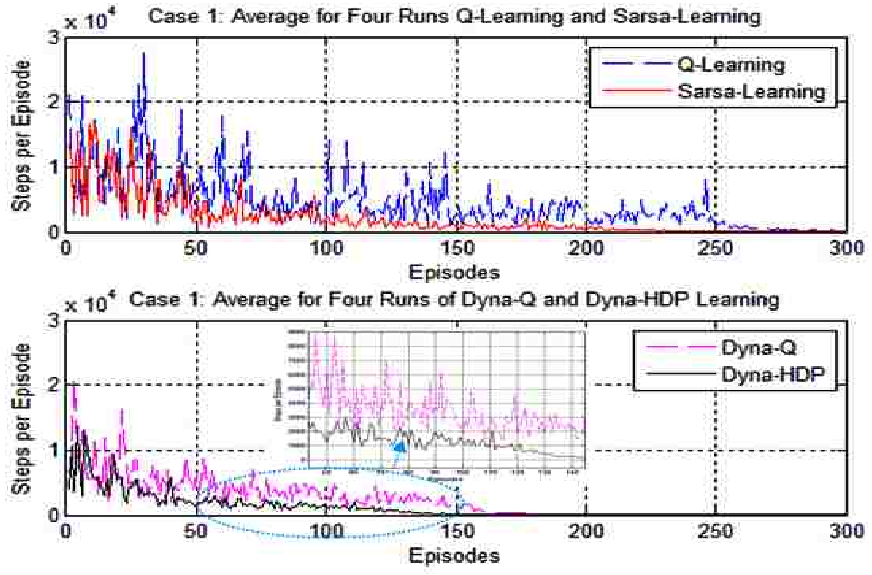


Figure 9. The first case comparison the number of steps per episodes. This compares among Dyna-HDP with other conventional algorithms, one step Q-learning, Sarsa (λ), and Dyna-Q, for one agent (no information sharing with the others).

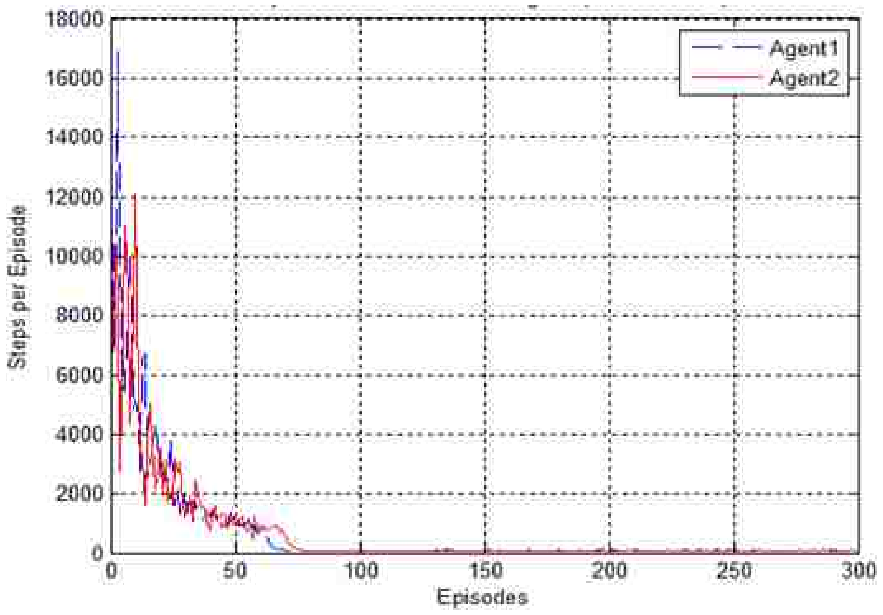


Figure 10. The second case results (the cooperation between two agents by sharing their experiences by using the Dyna-HDP approach). Every agent has own experience and tries to support the other one.

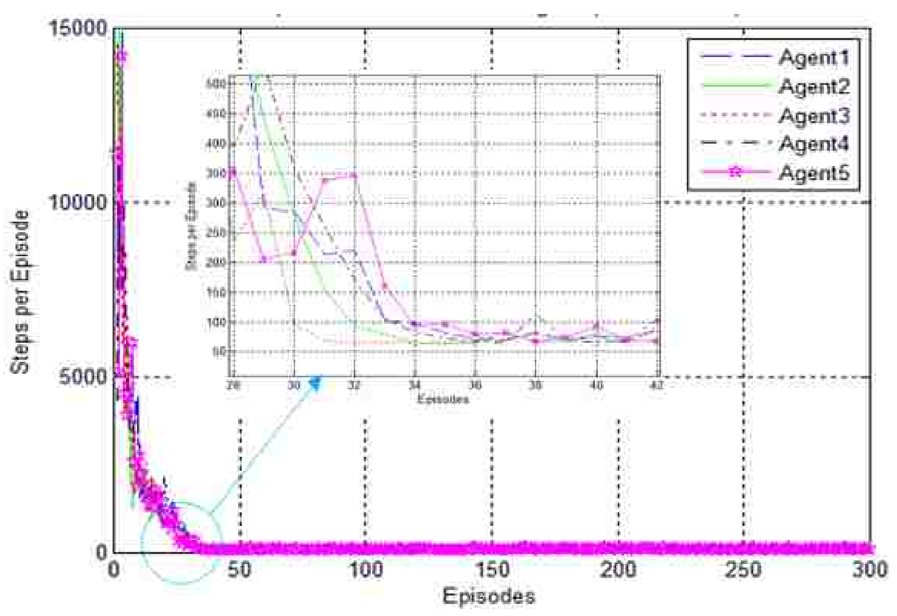


Figure 11. The Third case results (Cooperation among five agents by using the Dyna-HDP approach).

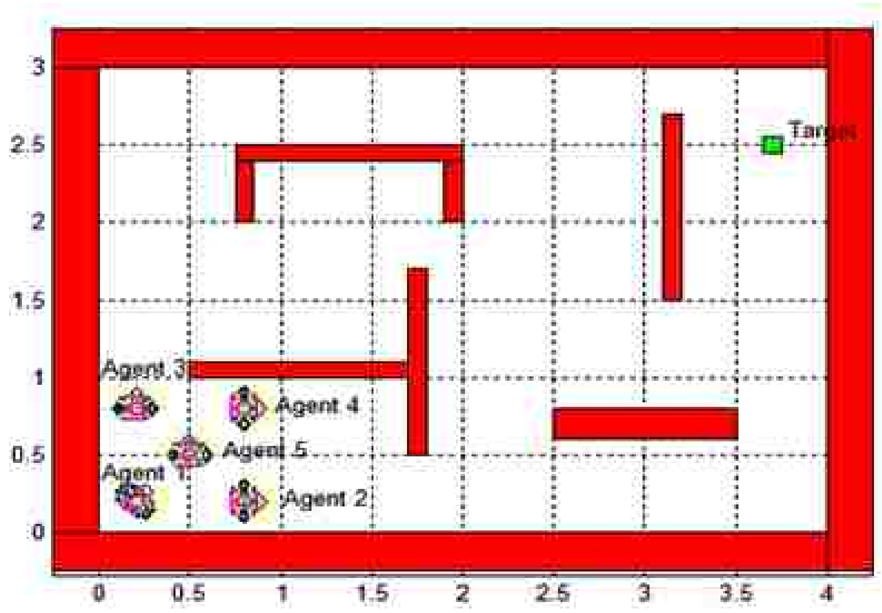


Figure 12. Five mobile robots are distributed in unknown environment surrounding by wall in meter scale with many obstacles on it. The robots are distributed in random start position and heading angle to achieve same task (reaching to target position).

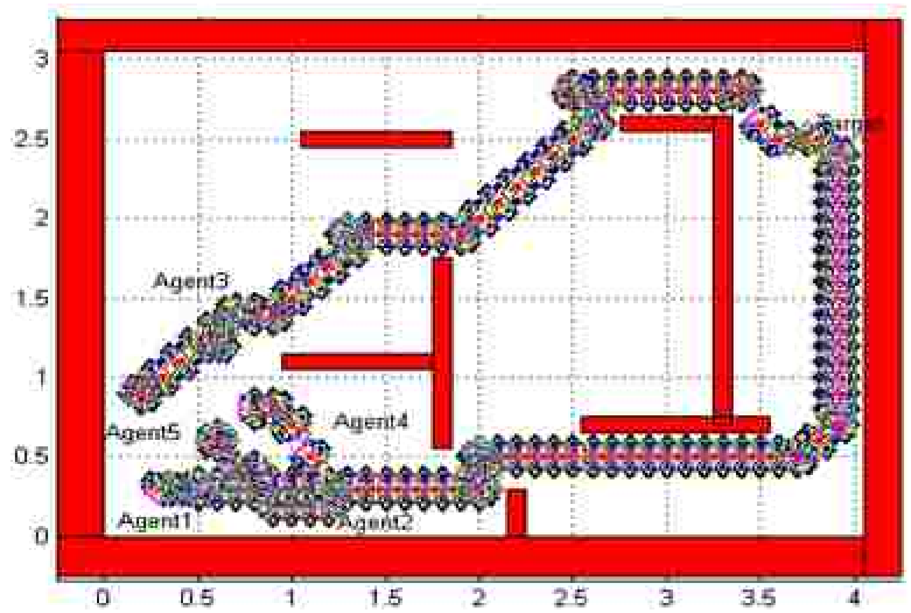


Figure 13. The near optimality trajectories for the five mobile robots from start positions to the same target position after applying the third case which is all agents share their experiences with the same approach.

to demonstrate quality improvement due to cooperation among agents. The x-axis in all three cases gives episodes, and the y-axis means the number of steps for each episode. In this simulation, we scale the inputs to the action network to be in [0, 1], and we set four independent runs (300 episodes for each run). Because each episode will be terminated when the robot reaches the target position area, the numbers of steps are not necessarily same. The final results take the average steps/episodic among all runs as shown in Fig. 14. For fair comparison, we used same parameters, which are required in traditional RL algorithms (one step Q-learning, Sarsa (λ), Dyna-Q) and Dyna-HDP. The basic parameters, which are used for this comparison are set as follows: the learning rate (α = 0.1), the discount rate (γ = 0.95), the decay-rate parameter for eligibility traces (λ = 0.9) for Sarsa (λ) learning algorithm, the staring decay-rate parameter for greedy selection (ε = 1), the ratio value for reduction ε=0.99, which is used to decrease the greedy probability value by $\epsilon = \varepsilon * \epsilon$ after each episode until reaching to minimum value (ϵ = 0.05); k=2 for k-max

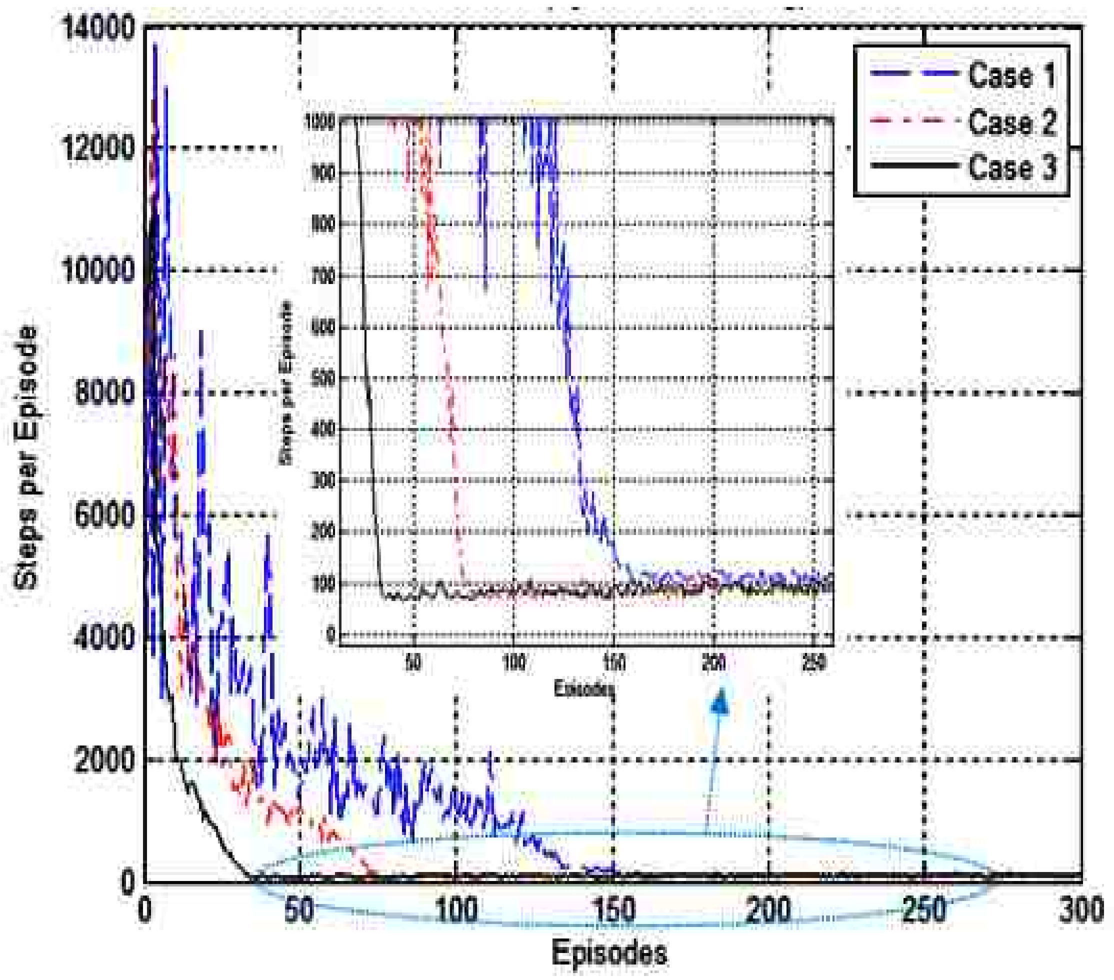


Figure 14. Comparison among all three cases by using Dyna-HDP for average learning. In third case, the results are stable with the improvement in the learning speed of distributed autonomous agents by sharing their experiences.

certainly selection action criteria, and let N=5 (N is the number of planning steps in the planning world which is the same the transition time between any two states in the actual world). For HDP parameters, the initial learning parameters are set as: ℓ_t^c=0.005 for critic network and ℓ_t^a=0.003 for action network. Both learning rates are decreased via dividing by 3 for every 30 episodes. The stopping criteria for the action and critic networks are 20, 30 respectively; the training for either network will be terminated if the error drops under $1e4$ or if the number of iterations meets the stopping threshold. The number of neurons in the hidden layer is n_c=12 for critic network and n_a=10 for action network.

5.1. Applying Dyna-HDP in A New Environment (Building Map). In the previous environment testing, every agent has own experience and also tries to learn dependently to achieve a single task. We implemented the Dyna-HDP learning method with another environment as shown in Fig. 15 with same previous procedures. The new environment has three tasks (targets). Two agents, M_1 and M_2, share to complete the first task to fine the near-optimal trajectory as shown in Fig. 15 a while M_3 and M_4 share to implement the second task as shown in Fig. 15 b, and the last simple task gives to single agent (M_5) as shown in Fig. 15 c. Fig. 15 d illustrates the simulation results for cooperative agents to run own specific task. Because of the FCOA, the agent can avoid any new obstacles encountered. We run the mobile robot in the same last task with an extra obstacle. We put a new obstacle in the agent's way in order to check the ability of this controller to adapt. Fig. 16 demonstrates the capability of the mobile robot to complete the task successfully. The mobile robot moves toward the goal point as main task; if the mobile robot senses an obstacle by 9 sensors, it will immediately use the FCOA to avoid this collision.

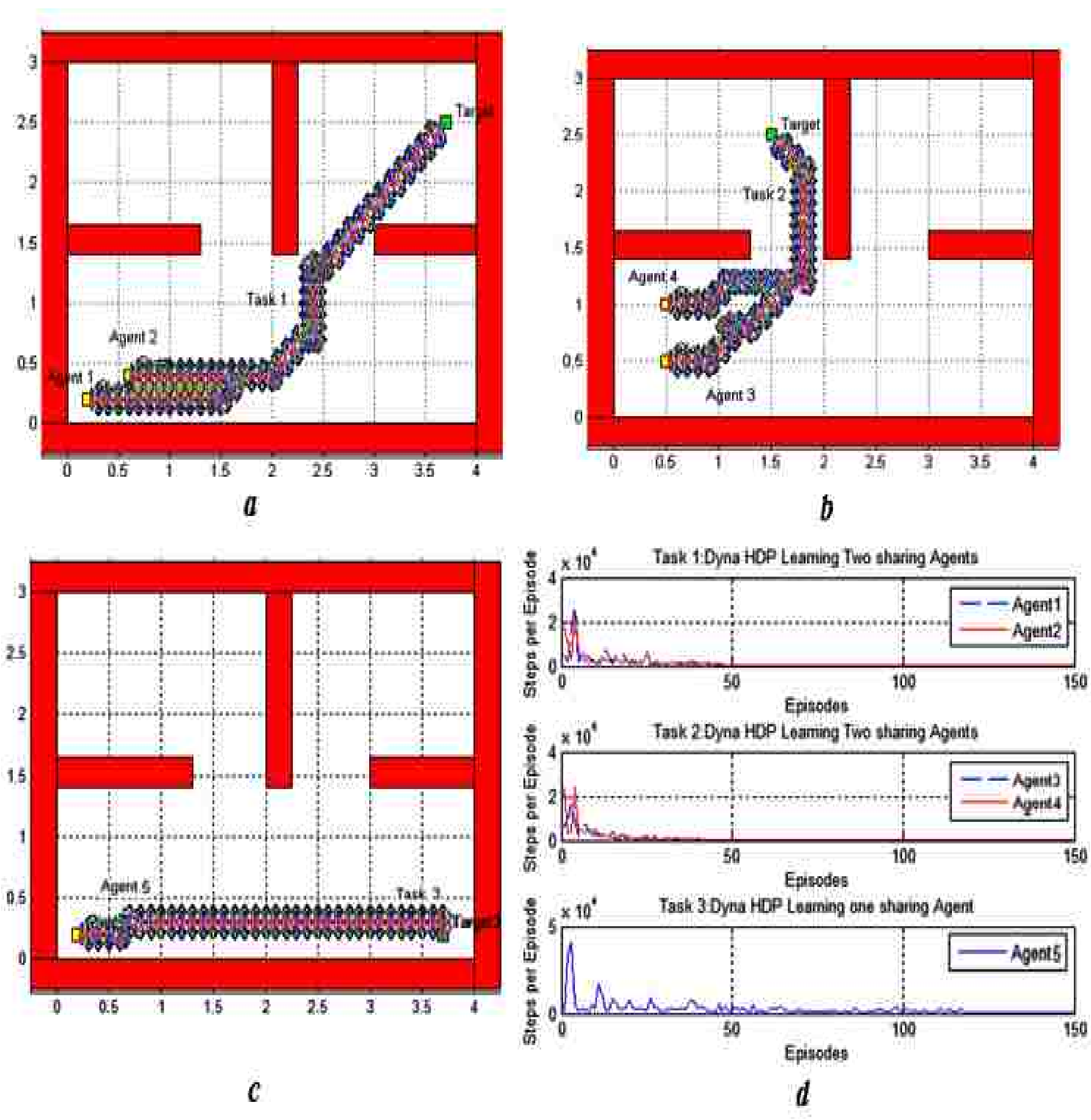


Figure 15. a) The trajectories for two agents to run Task 1 b) The trajectories for two agents to run Task 2 c) The trajectories for one agent to run Task 3 d) The simulation result for three tasks among Dyna-HDP.

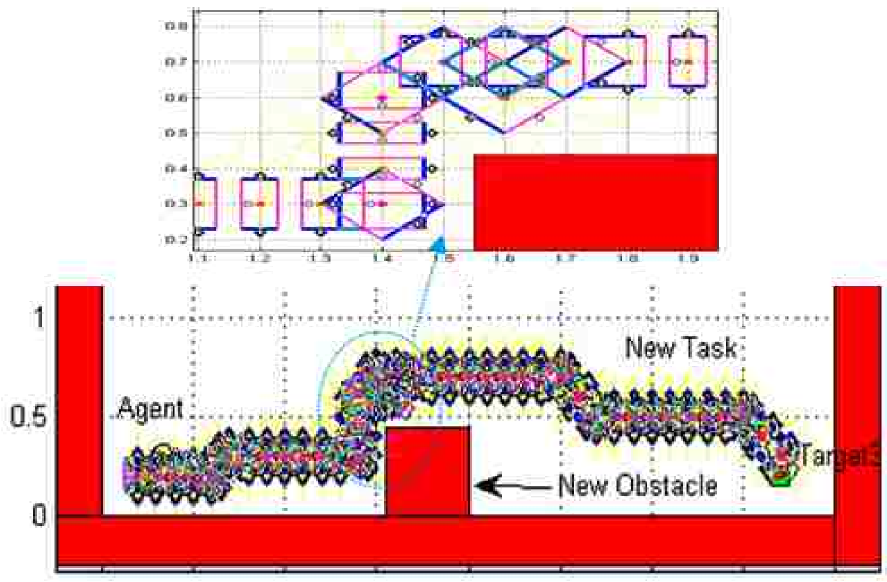


Figure 16. Path for mobile robot to implement the third task with a change in the environment by adding a new obstacle. FCOA gives the mobile robot the capability to adapt with this changing.

6. CONCLUSION

This paper has presented a novel technique, Dyna-HDP, using ADP for the online planning and learning algorithm in combination with the Dyna method. The performance of Dyna-HDP was excellent during navigation of a mobile robot compared to one step Q-learning, Sarsa (λ), and Dyna-Q. This paper has also demonstrated that Dyna-HDP in multi-robots cooperative navigation has a significant advantage to enhance the efficiency of the virtual common environment model. The simulation has confirmed how this group of cooperative robots decreases their individual navigation time.

BIBLIOGRAPHY

[1] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol.43, no.5, pp.1141 - 1153, Sept. 2013.

[2] H. Xiao, L. Liao, and F. Zhou, “Mobile robot path planning based on Q-ANN,” *Proceedings of the IEEE International Conference on Automation and Logistics,* pp. 2650 - 2654, Aug. 2007.

[3] G. Yang, E. Chen, and C. An, “Mobile robot navigation using neural Q-learning,” *Proceedings of the IEEE International Conference on Machine Learning and Cybernetics,* vol.1, pp. 48 - 52, Aug. 2004.

[4] R. S. Sutton, and A. Barto, *Reinforcement Learning: An Introduction, Cambridge,* U.K.: Cambridge Univ. Press, 1998.

[5] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[6] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control.* NewYork, NY, USA: Wiley, 2012.

[7] R. S. Sutton, “Integrated architectures for learning, planning, and reactingbased on approximating dynamic programming,” *Proc. 7th Int. Conf. Mach. Learn,* pp. 216 - 224, 1990.

[8] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Mach. Learn*, vol. 3, no. 1, pp. 9 - 44, 1988.

[9] D. Liu, and H. Zhang, “A Neural Dynamic Programming approach For Learning Control Of Failure Avoidance Problems,” *International Journal Of Intelligent Control And Systems,* vol. 10, No. 1, 21 - 32, Mar. 2005.

[10] D. Prokhorov, and D. C. Wunsch, “Adaptive critic designs,” *IEEE Transactions on Neural Networks,* vol. 8, pp. 997 - 1007, Sept. 1997.

[11] J. Si, A. G. Barto, and W. B. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming,* New York, NY, USA: Wiley, 2004.

[12] P. J. Werbos, “Approximate dynamic programming for real-time control and neural modeling,” *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches (Chapter 13)*, Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[13] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, “Adaptive critic designs: A case study for neurocontrol,” *Neural Networks*, vol. 8, pp. 1367 - 1372, 1995.

[14] J. Si, and Y. Wang, “Online learning control by association and reinforcement,” I*EEE Trans. Neural Netw.,* vol. 12, no. 2, pp. 264 - 276, Mar. 2001.

[15] H. He, Z. Ni, and J. Fu, “A three-network architecture for on-line learning and optimization based on adaptive dynamic programming,” *Neurocomputing*, vol. 78, no. 1, pp. 3 - 13, Feb. 2012.

[16] X. Fanga, D. Zhenga, H. He, and Z. Nib, “Data-driven heuristic dynamic programming with virtual reality”, vol. 166, pp. 244 - 255, Oct. 2015.

[17] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, “GrDHP: A General Utility Function Representation for Dual Heuristic Dynamic Programming,” *IEEE transactions on neural networks and learning systems,* vol. 26, no. 3, pp. 614 - 627, mar. 2015.

[18] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Dual heuristic programming excitation neurocontrol for generators in a multimachine power system,” *IEEE Trans. Ind. Appl.,* vol. 39, no. 2, pp. 382 - 394, Mar. 2003.

[19] N. Zhang, and D. C. Wunsch, “A Comparison of Dual Heuristic Programming (DHP) and Neural Network Based Stochastic Optimization Approach on Collective Robotic Search Problem,” *Proceedings Neural Networks of the IEEE*, vol.1, pp. 248 - 253, Jul. 2003.

[20] C. Lian, and X. Xu, “Motion Planning of Wheeled Mobile Robots Based on Heuristic Dynamic Programming,” *IEEE Proceeding of the 11th World Congress on Intelligent Control and Automation Shenyang*, pp. 576 - 580, Jul. 2014.

[21] X. Yang, M. Moallem, and R. V. Patel, “A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation,” *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 35, no. 6, pp. 1214 - 1224, Dec. 2005.

[22] S. Pawlikowski, “Development of a Fuzzy Logic Speed and Steering Control System For an Autonomous Vehicle,” *Master thesis, University of Cincinnati, Department of Mechanical Engineering*, Jan. 1999.

[23] O. Caelen, and G. Bontempi, “Improving the exploration strategy in bandit algorithms,” *International Conference on Learning and Intelligent Optimization,* pp 56 - 68, 2008.

[24] T. Tateyama, S. Kawata, and Y. Shimomura, “Parallel Reinforcement Learning Systems Using Exploration Agents and Dyna-Q Algorithm,” *SICE Annual Conference*, pp. 2774 - 2778, Sep. 2007.

[25] K. Miyazaki, M. Yamamura, and S. Kobayashi, “K certainty exploration method: an action selector to identify the environment in reinforcement learning,” *Artificial Intelligence*, vol. 91, no.l, pp. 155 - 171, l997.

[26] Y. Zhang, and M. Feng, “Application of reinforcement learning based on artificial neural network to robot soccer,” *Journal of Harbin Institute of Technology*, vol.36 ,no.7, pp. 859 - 861, Jul. 2004.

[27] K. Ito, and Y. Imoto, “A study of reinforcement learning with knowledge sharing for distributed autonomous system,” *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 3, pp. 1120 - 1125, Jul. 2003.

III. MOBILE ROBOT CONTROL BASED ON HYBRID NEURO-FUZZY VALUE GRADIENT REINFORCEMENT LEARNING

S. Al-Dabooni and Donald C. Wunsch

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

ABSTRACT

This paper uses value gradient learning (VGL) to track a reference trajectory under uncertainties, by computing the optimal left and right torque values for a nonholonomic mobile robot. VGL is a high-performance algorithm in adaptive dynamic programming (ADP). Here, it is used as a critic function after fitting a first-order Sugeno fuzzy neural network (FNN) structure to critic and actor networks. Moreover, this work handles the impacts of unmodeled bounded disturbances with various friction values. The simulation is introduced to compare two approaches. The first uses an actor network that confirms the ability of the mobile robot dynamic model to follow a desired trajectory. This approach demonstrates a significant enhancement of the robot's capability to absorb unstructured disturbance signals and friction effects. The second type of results use a critic-optimal-control approach, calculating the optimal control signal for the affine dynamic model of the robot. This completely removes the actor network to exploit reduced computational complexity with faster responses. The simulation is introduced to compare both cases.

Keywords: Nonholonomic dynamic mobile robot, fuzzy neural network, Adaptive Dynamic Programming(ADP).

1. INTRODUCTION

A nonholonomic mobile robot kinematic model is one where the state depends on the taken path. It is one of the most well-known benchmarks in the literature. This paper considers a two-wheeled dynamic, nonholonomic model. There are three classes of mobile robot navigation [1]: tracking a reference trajectory, following a path, and point stabilization. In this work, the tracking a reference trajectory is considered. To simplify the nonholonomic tracking problem, the magnitude of disturbances is taken as a small value, or, in many studies, it is ignored. In this article, we track a reference trajectory with various values of disturbance to emulate a real robot. Fuzzy logic handles these uncertainties at various levels [2]. This work uses a FNN design for the actor and critic networks. In ADP, the optimal control problems are solved, which allows agents to select an optimal action to minimize a long-term cost value through solving Bellman's equation [3] and [4]:

$$J(x_t) = \langle \sum_{k=t}^{\infty} \gamma^{k-t} U(x_k, u(x_k)) \rangle, \tag{1}$$

where $\langle . \rangle$ is the expectation symbol, $J(x_t)$ is an value function (cost-to-go value) for a state vector ($x \in \mathbb{R}^n$) at time step t, $\gamma \in [0, 1]$ is a constant discount factor, $U(x_k, u_k)$ is an instantaneous utility cost function at time step k for x after applying a certain action vector $u \in \mathbb{R}^m$, and F is a final time (independent actions). Equation (1) can be rewritten as

$$J(x_t) = \langle U(x_k, u(x_k)) + \gamma J(x_{t+1}) \rangle, \tag{2}$$

where $J^{x_{t+1}}$ is an optimal value function for the next state (x_{t+1}), which can be obtained according to the environment model function ($f(x_t, u_t)$). Reinforcement learning (RL) and ADP are used to train the actor network to give the optimal actions based on minimizing the cost-to-go value that is produced from the critic network. The actor function approximator ($A(x_t, w_a)$) is denoted for the actor network with ω_a parameters. This function produces

u_t after feeding it by the input system state at time t (x_t), while $\tilde{v}(x_t, \omega_c)$ is the function approximator for the critic network with ω_c parameters, which produces a cost-to-go value for x_t. After full training of these networks, the optimal action values are obtained from the actor network as:

$$u_t^* = arg \min_{u \in \mathbb{R}^m} \langle U(x_k, u(x_k)) + \gamma \tilde{v}(x_{t+1}, \omega_c) \rangle, \tag{3}$$

ADP has three fundamental approaches: heuristic dynamic programming (HDP), dual heuristic programming (DHP), and globalized dual heuristic programming [5] and [6], which consists of three approximation function networks (actor, critic, and model networks) that provide decision making, evaluation, and prediction, respectively. If the approach has only actor and critic networks, it is referred to as action-dependent (AD), resulting in the terminology ADHDP for HDP and ADDHP for DHP. In [7] and [8], the authors implemented an online learning value function for ADHDP and ADDHP, respectively. Improving online ADP learning is done by adding dual critic networks for HDP and DHP [9] - [11]. Many applications have used ADP techniques seen in [12] - [15]. Fairbank and Alonso [16], inspired by TD(λ) approaches [17], extended DHP by including a bootstrapping parameter (λ). They called it a value gradient learning (VGL) with λ for eligibility traces. In this work, we use VGL(λ) with FNN to control a nonholonomic mobile robot to follow any given reference trajectory. The remaining sections are organized as follows: Section 2 presents fundamental preliminaries for dynamic mobile robots, ADP, and FNN. The mobile robot control by FNN-based VGL(λ) is presented in Section 2. The simulation results and conclusion are presented in Section 4 and Section 5, respectively.

2. FUNDAMENTAL PRELIMINARIES

2.1. Dynamic Modeling of the Mobile Robot. A differential-drive mobile robot contains two independently driven wheels mounted on the left and right of its chassis at the same axis, and a castor wheel (free rotating wheel) at the front for balancing the mobile robot. An inertial Cartesian frame represents the position of the mobile robot, while $q = [x_c, y_c, \theta]^T$ is the set of coordinates for the center of mass of the robot and the robot orientation with respect to the Cartesian frame. The two independent driving wheels are provided with the necessary torque for generating a left angular velocity (w_L) and a right angular velocity (w_R), which in turn generate a linear velocity (v_1) and angular velocity (v_2) for the mobile robot as follows:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.5\bar{r} & -0.5\bar{r} \\ \dfrac{\bar{r}}{2b} & -\dfrac{\bar{r}}{2b} \end{bmatrix} \begin{bmatrix} w_R \\ w_L \end{bmatrix}, \tag{4}$$

where $\bar{r}$ is wheel radius and b is half of the robot width. The different forces for the mobile robot mechanical motion are considered in the literature for the dynamic model but not the kinematic model. The kinematic model is considered only for the motion. As stated in [18] - [20], the dynamic model of the mobile robot has $\bar{n}$ dimensional configuration space, which subjects to r constraints as described by

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)u + A^T(q)\Psi, \tag{5}$$

with $A(q)\dot{q} = 0$ as a constrained kinematic wheel, where $q \in \mathbb{R}^{\bar{n}}$ is coordinate vector, $M(q) \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is a is a symmetric positive definite inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is the centripetal and Coriolis matrix, $F(\dot{q}) \in \mathbb{R}^{\bar{n}}$ is a surface friction force vector, $G(q) \in \mathbb{R}^{\bar{n}}$ is a gravity vector, $\tau_d \in \mathbb{R}^{\bar{n}}$ is a bounded unknown disturbance, $B(q) \in \mathbb{R}^{\bar{n}\times\mathrm{n}}$ is a input transformation matrix, $u \in \mathbb{R}^{\mathrm{n}}$ is the input torque vector, $A(q) \in \mathbb{R}^{r\times\bar{n}}$ is the full rank matrix associated with constraints, and $\Psi \in \mathbb{R}^r$ the Lagrange multiplier (constraint forces) vector. In this case study, There are two control inputs, which are a left torque (τ_L) and a right torque (τ_R). Since the system does not change in vertical position and has a constant value for potential energy, $G(q)$ is set to zero. The viscous and coulomb frictions are commonly used to represent the effecting of the fraction model by the following description [21]

$$f(w) = f_v w + f_c \Gamma(w), \tag{6}$$

where w is the angular relative velocity between the contact bodies, $f_v > 0$ is the viscous parameter, and $f_c > 0$ is the coulomb parameter, and $\Gamma(.)$ is a sigmoid function. In order to reduce the dynamic model as in (5) from $\bar{n}$ to $m = \bar{n} - r$ with riding the Lagrange multiplier out, equation (5) is pre-multiplied by spanning the linear independent null space of the $A(q)\dot{q}$ matrix, which is denoted as Jacobian matrix of $S_c(q) \in \mathbb{R}^{\bar{n}\times m}$. In this case, a kinematic equation is given as follows:

$$\dot{q} = S_c(q)v, \tag{7}$$

where

$$\dot{q} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix}, \quad S_c(q) = \begin{bmatrix} cos(\theta) & -dsin(\theta) \\ sin(\theta) & dcos(\theta) \\ 0 & 1 \end{bmatrix}, \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},$$

and d is a center of gravity. The final elegant affine dynamic model is obtained by substituting the kinematic equation (77), its derivative ($\ddot{q} = \dot{S}_c(q)v + S_c(q)\dot{v}$), and (75) into (76) to obtain

$$\dot{v} = -\bar{M}(q)^{-1}\Big(\bar{V}(q,\dot{q})v + \bar{F}(\dot{q}) + \bar{\tau}_d\Big) + \bar{M}(q)^{-1}\bar{\tau}, \tag{8}$$

where $\bar{M}(q)$ is an invertible matrix, which is defined

$$\bar{M}(q) = \begin{bmatrix} m_T + 2\frac{I^b_{YY}}{\bar{r}^2} & 0 \\ 0 & m_T d^2 + I_T + 2I^b_{YY}\frac{b^2}{\bar{r}^2} - 4m_w d^2 \end{bmatrix},$$

$$S_c(q) = \begin{bmatrix} 0 & -dv_2(m_T - 2m_w) \\ dv_2(m_T - 2m_w) & 0 \end{bmatrix}, \quad \bar{\tau}_d = \begin{bmatrix} \bar{\tau}_R \\ \bar{\tau}_L \end{bmatrix},$$

$$\bar{F}(\dot{q}) = \frac{1}{\bar{r}} \begin{bmatrix} f_v(w_R + w_L) + f_c(\Delta(w_R) + \Delta(w_L)) \\ bf_v(w_R - w_L) + bf_c(\Delta(w_R) - \Delta(w_L)) \end{bmatrix}, \quad \text{and}$$

$$\bar{\tau} = \bar{B}\tau = \begin{bmatrix} 0.5\bar{r} & -0.5\bar{r} \\ \dfrac{\bar{r}}{2b} & -\dfrac{\bar{r}}{2b} \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix} (\Delta(.) \text{ is a sigmoid function}).$$

All mobile robot dynamic mode parameters will be defined in the simulation results section in Table 1. The unknown disturbance $\bar{\tau}_d$ is bounded as $\|\bar{\tau}_d\|$. The variables of $\bar{\tau}_L$ and $\bar{\tau}_R$ are unknown disturbances impacting on the left and the right wheels, respectively.

2.2. DP/RL Algorithms. This section describes a simple view for the development stages for ADP/RL approaches. Throughout this work, all defined vectors are columns, and all time step subscribed on variables is summarized by subscripting them on the first function variable; for instance: $\tilde{v}_t \equiv \tilde{v}(x_t, w_c), A_t \equiv A(x_t, w_a), f_t \equiv f(x_t, u_t), U_t \equiv U(x_t, u_t) \equiv U(x_t, A_t(x, \omega))$, and $\tilde{g}_{t+1} \equiv \tilde{g}(f(x_t, u_t), \omega_c)$. Backpropagation through time (BPTT) by Werbos [23] is the first stage used in adaptive/optimal control. For any given trajectory, this approach efficiently finds an optimal controller by combining the BPTT with gradient descent parameters. The update for this approach is done as a partial derivative for the value function as in (2) with respect to the approximation parameters for the control function. Another stage in ADP/RL algorithms uses the critic function to determine the optimal policy [5], [6], [23],[24]. The HDP algorithm is connected by the critic network to the action network through the model function. Because the TD algorithm by Sutton [17] has less complexity and more accurate than conventional prediction learning methods, we use it to update the critic weights. The minimizing TD-error between the approximate-value function ($\tilde{v}_t(x, \omega_c)$) and target-value ($\bar{v}_t$) is used for updating citric weights. The weights for the actor network are updated for α learning rate as follows:

$$\omega_a = \omega_a - \alpha \frac{\partial \tilde{v}_t}{\partial \omega_a}, \tag{9}$$

where $\frac{\partial \tilde{v}_t}{\partial \omega_a}$ is given as follows:

$$\frac{\partial \tilde{v}_t}{\partial \omega_a} = \frac{\partial A_t}{\partial \omega_a}\Big(\frac{\partial U_t}{\partial u} + \gamma \frac{\partial f_t}{\partial u}\frac{\partial \tilde{v}_{t+1}}{\partial x}\Big) + \gamma \frac{\partial \tilde{v}_{t+1}}{\partial \omega_a}. \tag{10}$$

Equation (10) is used also to update the actor network for the DHP algorithm, which is equivalent to equation (10) from [5], unlike HDP with single output. The way to update the action weights in HDP (or ADHDP) is by fixing the model and critic weights and then applying $\frac{\partial \tilde{v}_t}{\partial \tilde{v}_t}$ (i.e., a constant) as the backpropagated error signal. There are two different ways to implement the critic network for DHP scheme [16]: the scalar-critic function, which the outputs from critic network represent as the partial derivatives of the value function with respect to the vector of system state ($\frac{\partial \tilde{v}_t}{\partial x}$), and the vector-critic function, which the outputs represents by $\tilde{g}$ (identical to $\frac{\partial \tilde{v}_t}{\partial x}$) where $\tilde{g}$ is the approximate-value gradient function. The critic network weights in DHP are updated by minimizing the TD-error between $\tilde{g}$ and target-gradient value ($\bar{g}$) as follows:

$$\omega_c = \omega_c - \beta \sum_{t=1}^{F} \Big(\frac{\partial \tilde{g}_t}{\partial \omega_c}(\bar{g}_t - \tilde{g}_t)\Big), \tag{11}$$

where $\bar{g}_t$ is given as

$$\bar{g}_t = \Big(\frac{\partial U_t}{\partial x} + \gamma \tilde{g}_{t+1}\frac{\partial f_t}{\partial x}\Big) + \frac{\partial A_t}{\partial x}\Big(\frac{\partial U_t}{\partial u} + \gamma \tilde{g}_{t+1}\frac{\partial f_t}{\partial u}\Big). \tag{12}$$

This TD-error is equivalent to equations (6) and (7) from [5]; the Jacobian matrix $\frac{\partial \tilde{g}_t}{\partial \omega_c}$ can use either the vector-critic function, which is called as a DHP *style critic*, or a scalar-critic function ($\frac{\partial^2 \tilde{v}_t}{\partial x \partial \omega_c}$), which is called a *GDHP style critic*, where GDHP comes from globalized DHP. The GDHP uses the function value and its derivatives by combining both HDP and DHP. The ADP methods discussed thus far rely on differentiated models of acceptable accuracy. This accurate model function that interacts with the learner might not be available a *priori*. Si and Wang [7] do not use any model function, preferring real-time learning during interacting with the environment. Their approach is roughly like ADHDP, but with a model-free. Instead of using a model, they store the previous value function and combine with the current. ADHDP uses two inputs for the critic network $[x_t, u_t]^T$, unlike model-based, which use only the x_t. VGL(λ) is a further step in ADP/RL fields introduced by Fairbank and Alonso [16]. VGL(λ) extends DHP by including a bootstrapping parameter (λ). It is similar to Sutton [17] by extending TD(0) to TD(λ), but with the DHP approach (VGL(0) is equivalent to traditional DHP). In this work, the FNN is applied in the VGL(λ) approach to control mobile robot motion in noisy ambiance.

2.3. FNN. Since the FNN approach combines the advantages for fuzzy and pure neural methods, it has become a popular research topic [25]-[27]. The FNN is composed of five layers (Fuzzification / premise, firing strength, normalization, defuzzification / consequent, and output) with nodes for each one as in [27] and [28]. The corresponding l^{th} if-then rule from the fuzzy logic system with multi-outputs can be written as follows:
Rule $l : IF \quad x_1 = A_1^l \quad AND \ldots AND \quad x_n^l \quad THEN \quad f_j^l = \sum_{i=1}^{n} \left(c_{ij}^l x_i + c_{(n+1)j}^l \right)$, where A_i^l represents the fuzzy linguistic variable for l^{th} rule, $i = 1, 2 \ldots n$ (n is number of inputs to the system, which is equaled to the number of mobile robot dynamic states); and $f_j^l \in V(j) \subset \mathbb{R}^m$ is the j^{th} output from the system and $V(j)$ is the output universe of discourse. The consequent parameters are $c_{1j}^l, \ldots, c_{(n+1)j}^l \in \mathbb{R}$ for l^{th} rule, $j = 1, 2 \ldots m$ (m is number of outputs from the system, which is equaled to the number of input torques to the mobile robot model). In this work, we used backpropagation gradient to update both premise and consequent parameters. Further detail about the structure FNN for VGL(λ) will be discussed in the next section.

3. MOBILE ROBOT CONTROL BY FNN-BASED OF VGL(λ)

In this section, a novel simple first-order Sugeno FNN structure is designed to adapt with the critic and actor networks for controlling the robot trajectory.

3.1. VGL(λ) Structure Learning by Using FNN. Fig. 1 shows a new structure for FNN by converting the single output structure of FNN [27] into m outputs with only three layers (premise, hidden, and consequent layers). Instead of connecting weights as a conventional neural network, the premise (σ_i^l, m_i^l) parameters for the Gaussian membership function and consequent $(c_{1j}^l, \ldots, c_{(n+1)j}^l)$ parameter coefficients for a first order linear function are used. The Gaussian membership function corresponding to the linguistic label of the input variable is given as follows:

$$\mu_{A_i^l}(x_i) = e^{-\frac{1}{2}\left(\frac{x_i - m_i^l}{\sigma_i^l}\right)^2}, \tag{13}$$

where the premise parameters $(\sigma_i^l \in U_i,\ m_i^l > 0)$ are the mean and variance parameters at l^{th} rule in i^th input fuzzy set $(\mu_{A_i^l}(x_i))$, where $U_i \subset \mathbb{R}^n$ is the input universes of discourse. The approximation function for forward pass can be expressed as follows:

$$\begin{aligned} p^l &= \sum_{i=1}^{n} \left(\frac{x_i - m_i^l}{\sigma_i^l}\right)^2, \\ s^l &= S(p^l) = e^{(\frac{p^l}{2})}, \end{aligned} \tag{14}$$

$$q^l = \sum_{l=1}^{\Re} s^l h^l = H(s^l, p^l) = \frac{s^l}{q^l} \tag{15}$$

$$y_j = \sum_{l=1}^{\Re} f_j^l h^l, \tag{16}$$

where $i = 1, 2 \ldots n$, $l = 1, 2 \ldots \Re$ ($\Re$ is the number of rules), and $j = 1, 2 ... m$. The partial

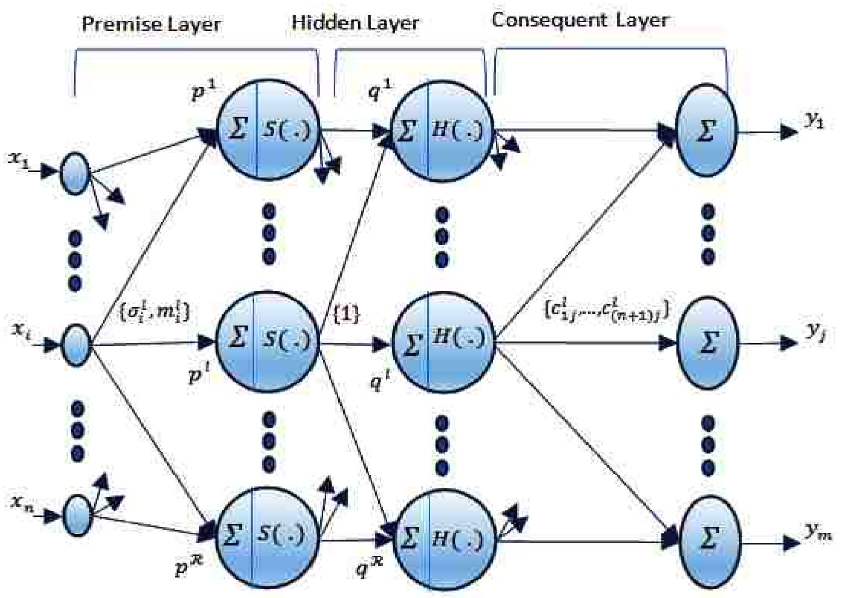


Figure 1. Schematic diagram of FNN used in ADP. Premise (σ_i^l, m_i^l) and consequent $(c_{1j}^l, \ldots, c_{(n+1)j}^l)$ parameters are updated by using the backpropagation gradient algorithm. The weights between the premise and hidden layers are always one.

derivatives of the total output (y) with respect to c_{1j}^l and $c_{(n+1)j}^l$ for l rule are given as follows:

$$\frac{\partial y}{\partial c_{1j}^l} = h^l x_i, \qquad \frac{\partial y}{\partial c_{(n+1)j}^l} = h^i, \tag{17}$$

while partial derivatives of the y with respect to σ_i^l and m_i^l for l rule are given as follows:

$$\frac{\partial y}{\partial m_i^l} = \sum_{j=1}^{m} f_j^l \frac{(1-h^l)}{q^l} s^l \frac{(x_i - m_i^l)}{(\sigma_i^l)^2}, \tag{18}$$

$$\frac{\partial y}{\partial sigma_i^l} = \sum_{j=1}^{m} f_j^l \frac{(1-h^l)}{q^l} s^l \frac{(x_i - m_i^l)^2}{(\sigma_i^l)^3}, \tag{19}$$

where y is the sum of squared m errors. Palit *et al.* [29] presented the details about the backpropagation output signals and chain rules for multi-input multi-output neuro-fuzzy network. A partial derivative of the y with respect to the inputs x_i (the i^{th} state) for l rule is required for implementing the VGL(λ) approach, which is given as follows:

$$\frac{\partial y_j}{\partial x_i} = h^l c^l_{1j} - \sum_{j=1}^{m} f^l_j \frac{(1-h^l)}{q^l} s^l \frac{(x_i - m^l_i)}{(\sigma^l_i)^2}, \tag{20}$$

where $\frac{\partial y_j}{\partial x_i}$ is yielded from the FNN structure by the propagated back output signal to the consequence fuzzy part (first term of (20)) and to the premise fuzzy part (second term of (20)).

3.2. Mobile Robot Control Based on FNN-VGL(λ). The structure of FNN shown in Fig. 1 is similar to the multi-inputs multi-output conventional neural network; therefore, we assume the premise parameters for the Gaussian membership function as the first layer weights, while the last layer weights are the consequent parameter coefficients for first order linear equations. The weights between the premise and consequent layers are always one. The FNN is applied for both critic and actor networks, which can be calculated for the forward pass as in (14) - (16). The total weights for critic and actor networks are denoted as ω_c and ω_a, respectively. Finding an optimal actor network makes the mobile robot follows a continuous differentiated desired trajectory with minimum torque values. This situation subjects the influences of unstructured disturbance signals ($\bar{\tau}_d$) and surface frictions ($\bar{F}$). As in (7) and (8), the mobile robot has two states: v_1 and v_2. These two states are denoted as column vector v_t at t. These states are entered into the actor and critic networks, as shown in Fig. 2. The two outputs of the actor are a right torque (τ_R) and left torque (τ_L), which are denoted as column vector τ_t at t. The two approximate-value gradient functions at t from the actor network are equal to $\frac{\partial \tilde{v}_t}{\partial v_1}$ for v_1 state and $\frac{\partial \tilde{v}_t}{\partial v_2}$ for v_2 state. Since the DHP style critic is used, $\frac{\partial \tilde{v}_t}{\partial v_1}$ and $\frac{\partial \tilde{v}_t}{\partial v_2}$ are denoted as $\tilde{g}_{v_1}$ and $\tilde{g}_{v_2}$, respectively. In other words, the input state for both critic and actor networks is $x = [x_1, x_2]^T$, which is equivalent to $v_t = [v_1(t), v_2(t)]^T$, while the output values for the FNN ($y = [y_1, y_2]^T$) are $\tilde{g}_t = [\tilde{g}_{v_1})(t), \tilde{g}_{v_2}(t)]^T$ for the critic network and $\tau_t = [\tau_R(t), \tau_L(t)]^T$ for the actor network. We define a quadratic utility cost function as follows:

$$U_t(v, \tau) = (v_t - v_d)^T Q(v_t - v_d) + \tau_t^T R\tau_t, \tag{21}$$

where $Q = Q^T \geq 0 \in \mathbb{R}^{n\times n}$, $R = R^T > 0 \in \mathbb{R}^{m\times m}$ and $v_d = [v_{d1}, v_{d2}]^T$, which has a constant value for all finite horizon time. v_{d1} is a desired linear velocity, and v_{d2} is a desired angular velocity for $t = 0$ to $F - 1$ where the cost function at the final step ($t = F$) is only equal to the first term of (21). The main objective of this cost function is minimizing the error between actual and desired velocities and minimizing the torque energy signals near to zero values. The diagonal matrix elements in the weighing matrices (Q) and (R) determine the value of minimizing the trajectory tracking and the energy, respectively.

3.2.1. Critic Network Training Algorithm. A new definition for $\bar{v}_t$, which is identical to λ-Return as in [28], is presented as follows:

$$\bar{v}_t(v) = U_t + \gamma\Big(\lambda\bar{v}_{t+1}(v) + (1-\lambda)\tilde{v}_{t+1}(v)\Big). \tag{22}$$

Equation (22) is related to Q-learning(λ) and Sarsa(λ) that plays an important role in the learning algorithm of RL. Fairbank and Alonso [16] have derived and adapted the VGL(λ) algorithm from (22), which has improved efficiency of learning control problems. The value learning as in (22) for HDP(λ) should supplement a stochastic exploration (a randomization of trajectory starting points, or policy) for learning in a deterministic environment to reach into a local optimization. In VGL (or DHP) methods, a single trajectory needs to obtain a local optimization without requiring any exploration approach. The sequential environment is used in this work by storing all vector states and vector actions in the forward pathway. The backward pathway is used to update the critic network from the final storing state until the first one. Equation (11) is used to update the critic network weights for the VGL(λ) by minimizing the TD-error between $\tilde{g}_t(v, \omega_c)$ and $\bar{g}_t$, as shown in Fig. 2, whereas $\bar{g}_t$ is defined

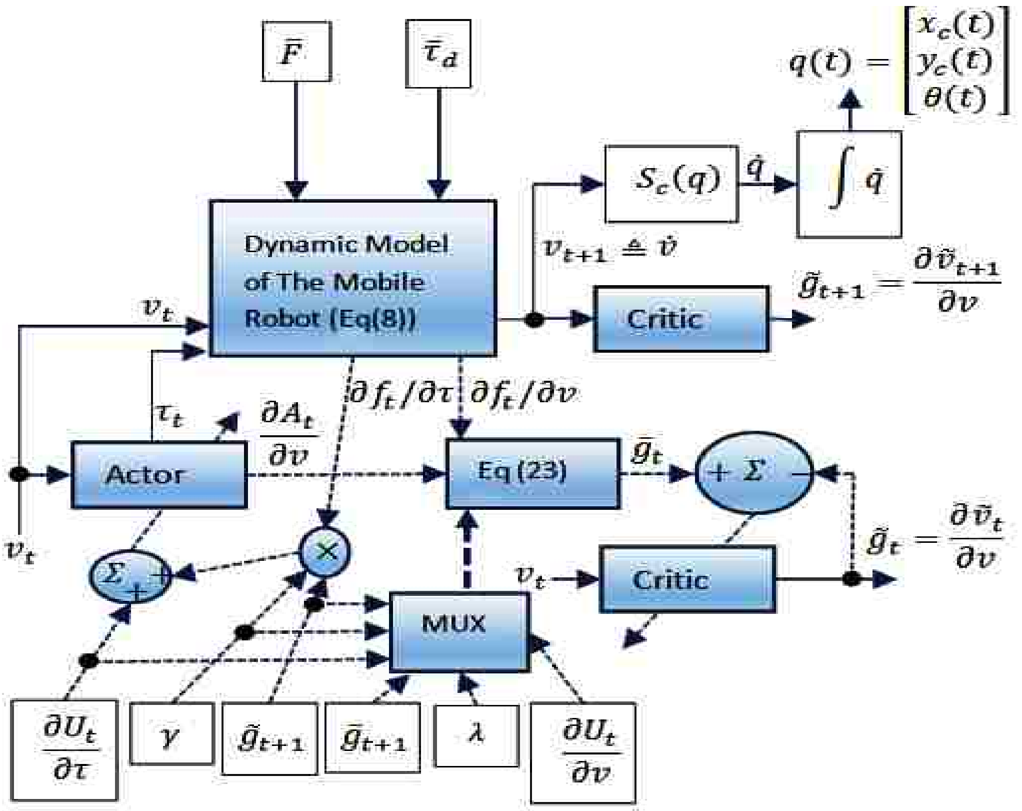

$\bar{F}$
$\bar{\tau}_d$
$q(t) = \begin{bmatrix} x_c(t) \\ y_c(t) \\ \theta(t) \end{bmatrix}$
$S_c(q)$
$\dot{q}$
$\int \dot{q}$
Dynamic Model of The Mobile Robot (Eq(8))
$v_{t+1} \triangleq \dot{v}$
$\tilde{g}_{t+1} = \frac{\partial \tilde{v}_{t+1}}{\partial v}$
Critic
v_t
τ_t
$\frac{\partial A_t}{\partial v}$
$\partial f_t / \partial \tau$
$\partial f_t / \partial v$
Actor
Eq (23)
$\bar{g}_t$
$+ \Sigma -$
$\tilde{g}_t = \frac{\partial \tilde{v}_t}{\partial v}$
v_t
Σ
MUX
v_t
Critic
$\frac{\partial U_t}{\partial \tau}$
γ
$\tilde{g}_{t+1}$
$\bar{g}_{t+1}$
λ
$\frac{\partial U_t}{\partial v}$

Figure 2. Adaptation in VGL(λ) for the mobile robot trajectory control tracking. In general, forward pathways are represented by the solid lines while pathways of backpropagation are shown by dashed lines, and the small solid black dots represent a connection. The sequential environment is used in this work by storing all states (velocity vectors) and actions (torque vectors) in forward pathways. The TD-error between $\tilde{g}_t$ and $\bar{g}_t$, as in (11), is used for updating critic network weights with applying (23) for $\bar{g}_t$. MUX box is used to feed (23) as one thick dashed line by gathering the signals. The actor weights are updated by applying (24). For robustness testing, various values of $\bar{F}$ and $\bar{\tau}_d$ are injected into this dynamic model to examine the influences on $\tau(t)$. The robot pose trajectory (x_c, y_c, θ) is obtained by using the mobile robot kinematic equation (7) with fourth order Runge-Kutta integration for the derivative of the coordination and orientation of the robot. The Runge-Kutta integration also uses to solve the dynamic model (8) within 0.01 sec for the sample time.

as follows:

$$\bar{g}_t = \gamma\Big(\frac{\partial f_t}{\partial v} + \frac{\partial f_t}{\partial \tau}\frac{\partial A_t}{\partial v}\Big)K + \Big(\frac{\partial U_t}{\partial v} + \frac{\partial U_t}{\partial \tau}\frac{\partial A_t}{\partial v}\Big), \tag{23}$$

where $K = \lambda\bar{g}_{t+1} + (1-\lambda)\tilde{g}_{t+1}$. All the partial derivative terms in (11) and (23) exist. The partial derivative terms related to FNN are $\frac{\partial A_t}{\partial v}$ and $\frac{\partial \tilde{g}_t}{\partial \omega_c}$. The term $\frac{\partial A_t}{\partial v}$ is implemented by using (20) to replace the FNNs' output vector (y) with τ and the FNNs' input vector (x) with v. The term $\frac{\partial \tilde{g}_t}{\partial \omega_c}$ is implemented by using (17)-(19) with taking into account that the critics' weights are premise and consequent parameters, and replacing the FNNs' output vector (y) with the value-gradient functions.

3.2.2. Actor Network Training Algorithm. As shown in Fig. 2, the actors' weights are updated by applying $\tilde{g}_t(v, \omega_c)$ instead of $\tilde{v}_t(v, \omega_c)$ in (9), and the recursion expanding of $\Delta(\omega_a) = \frac{\partial \tilde{g}_t}{\partial \omega_a}$ is defined as follows

$$\Delta(\omega_a) = \sum_{t\geq 0} \gamma^t \frac{\partial A_t}{\partial \omega_a}\Big(\frac{\partial U_t}{\partial \tau} + \gamma\tilde{g}_{t+1}\frac{\partial f_t}{\partial \tau}\Big). \tag{24}$$

The terms in (24), which are related to FNN, are $\frac{\partial A_t}{\partial \omega_a}$ and $\tilde{g}_{t+1}$. The term $\frac{\partial A_t}{\partial \omega_a}$ is implemented by using (17)-(19) with taking into account that the actors' weights are premise and consequent parameters and replacing the FNN's output vector (y) by τ, while the $\tilde{g}_{t+1}$ term is the outputs from FNN at $t+1$. Because the form for this dynamic system is an affine system with the quadratic utility cost function, the actor network can be omitted. As described by Dierks *et al.* [30], the optimal torque signal at time step $t4$ (τ_t^*), which is equivalent to the right side of (3), can be derived. To calculate the optimal torque in the VGL(λ) approach, we subject the gradient of HamiltonâĂŞJacobiâĂŞBellman to zero as follows:

$$\frac{\partial \bar{v}_t}{\partial \tau_t} = \min_{\tau_t^*}\Big(\frac{\partial v_t^T Q v_t + \tau_t^T R \tau_t}{\partial \tau_t} + \frac{\gamma\partial\lambda\big(\bar{v}_{t+1}^* + (1-\lambda)\tilde{v}_{t+1}^*\big)}{\partial \tau_t}\Big) = 0, \tag{25}$$

$$\tau_t^* = -\frac{\gamma}{2}R^{-1}\Big(\frac{\partial v_{t+1}}{\partial \tau_t}\Big)^T \frac{\partial\big(\bar{v}_{t+1}^* + (1-\lambda)\tilde{v}_{t+1}^*\big)}{\partial v_{t+1}}, \tag{26}$$

$$\tau_t^* = -\frac{\gamma}{2} R^{-1} \left(\bar{M}(q)^{-1} \bar{B} \right)^T \left(\bar{g}_{t+1}^* + (1-\lambda) \tilde{g}_{t+1}^* \right), \tag{27}$$

where $\tilde{g}_{t+1}^*$ and $\bar{g}_{t+1}^*$ are the optimal target- and approximator-value gradient, respectively. The actor network, in this case, is always fully trained to generate the optimal torque vector. Case study number two in the simulation results section will demonstrate that this method has the faster response comparing with the actor/critic method

4. SIMULATION RESULTS

Two case studies are presented in this section to demonstrate the performance of the mobile robot behavior with VGL(λ) technique. In both cases, we build a simulation for the mobile robot dynamic model to represent (8). Table 1 presents the parameters that are used to implement the model. The common parameters used in both two cases are: The adaptive learning rate for critic and actor networks starting at $\beta = 10^(- 6)$ and $\alpha = 10^(- 4)$, respectively, and these values decrease after each 10th iteration according to $\beta(or\alpha) = \beta(or\alpha) * 0.999$ when they exceed 800 iterations. The number of maximum iterations is set to 10000. The sample time, which is used the fourth order Runge-Kutta integration to solve the robot dynamic model, is set to 0.01 sec. The discount factor (γ) is set to 0.95. The initial weights for premise and consequent parameters are set within the range of [-5 5] with $\Re = 240$. The average of three successful trails is recorded. The average expression is represented the average for all generated data in one iteration for the error, approximate or target-value gradient, and control torque singles; for instance, the error or the mean square error (MSE) for v_1 at certain iterations is $(\sum_{t=1}^{T} 0.5(v_d 1 - v_1(t))^2)/T$, where T is the total number of sampling time, which is 100 for 1 sec.

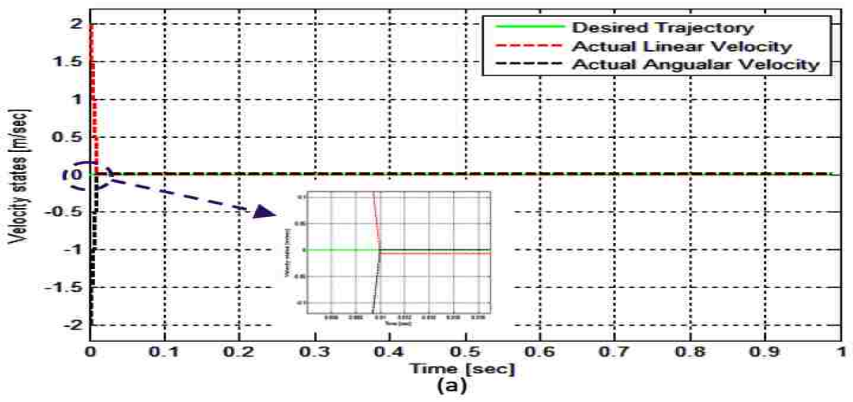
4.1. First Case (Effectiveness of λ value). This case shows the comparison for state trajectories between $\lambda = 0$ and $\lambda = 0.9$. The initial system state values are assigned to $v(0) = [2 - 2]^T$, and the desired velocity vector is $v_d = [00]^T$ during 1 sec. The weight matrices for reinforcement instantaneous signal (Q and R) are set to the identity matrices. In

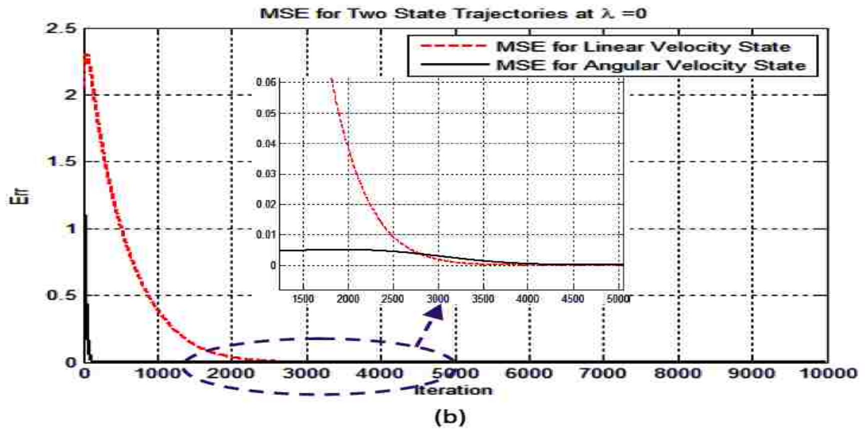
Table 1. Parameters of the dynamic mobile robot

Symbol	Description	Value
m_T	The mass of chassis	$10kg$
m_w	The mass of each wheel	$2kg$
$\bar{r}$	The wheel radius	$0.05m$
b	The half of the robot width	$0.1m$
d	The center of gravity offset form rear axle	$0.1m$
I_{YY}^{b}	The wheel moment of inertial	$1kg.m^2$
I_T	The platform total moment of inertia	$5kg.m^2$
f_v	The viscous friction coefficient	$0.001N.m.s$
f_c	The Coulomb friction coefficient	$0.001N.m.s$

this case, there is not any impact for the friction coefficients ($f_v = f_c = 0$) and unstructured disturbances ($\bar{\tau}_d = [00]^T$). Fig. 3 (a)-(c) show the control performance at $\lambda = 0$. The system trajectories for both velocity states over time are demonstrated in Fig. 3 (a). Fig. 3 (b) presents the average of MSE for both velocities over iterations. Fig. 3 (c) shows the average torques for the left and right wheels over iterations, which are bounded to $|3 \times 10^{-5}|$. Fig. 4 (a)-(c) show the performance at $\lambda = 0.9$ for trajectory, the average of MSE, torque signal values which are bounded to $|0.5 \times 10^{-5}|$, respectively. Fig. 3 and Fig. 3 demonstrate the effectiveness of bootstrapping parameter. For instance, the MSE for the mean of the two states in the final iteration for $\lambda = 0$ is 1.269e-04, while it is 2.765e-08 for $\lambda = 0.9$. Moreover, Fig. 3 and Fig. 4 demonstrate that a high value of λ makes the system more stable and fast learning. Fig. 5 (a)-(c) show the comparison performance between $\lambda = 0$ and $\lambda = 0.9$ in depth for the average of MSE, torques, and the value function as in (1), respectively. The controller with $\lambda = 0.9$ reaches to the near-optimal stable value function ($J = 7.2$) around 450 iterations while the controller with $\lambda = 0$ requires 2200 iterations.

4.2. Second Case (with/without Actor Network). The optimal torque vector is used as in (27) instead of the actor network. The critic / optimal-torque approach is replaced by a full training actor network. Because actor and critic networks (critic/actor approach) are





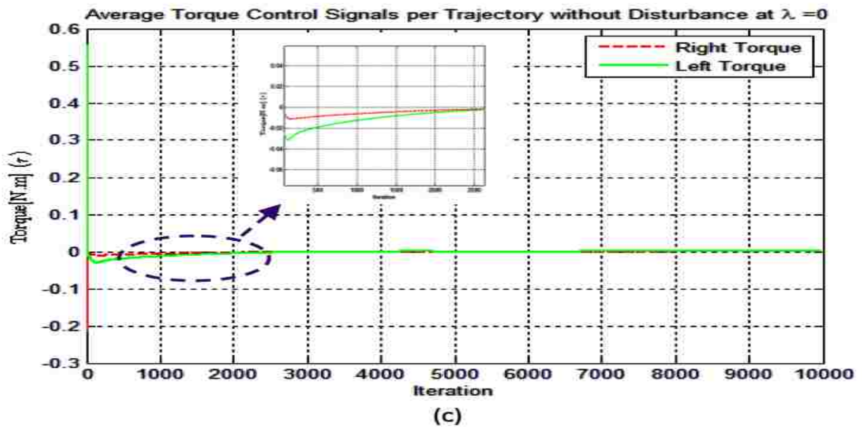


Figure 3. The actor and critic performance at $\lambda = 0$. (a) Typical system trajectories for both velocity states over time at last stable learning iteration. (b) The average of MSE for both velocity states over iterations. (c) The average torques applying to the left and right wheels over iterations.

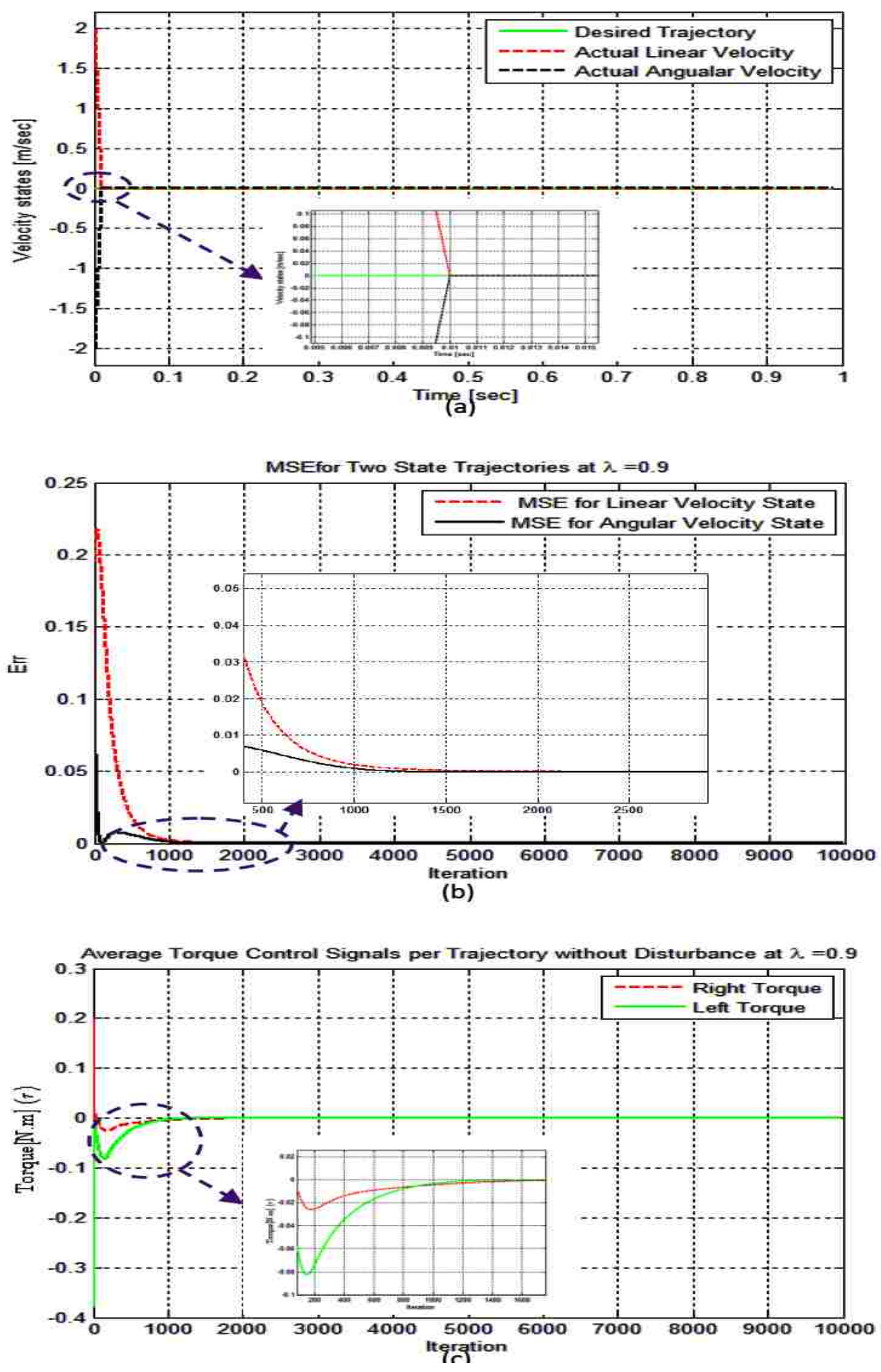


Figure 4. The control performance at $\lambda = 0.9$. (a) - (c) has same descriptions for (a) - (c) as in Fig. 3.

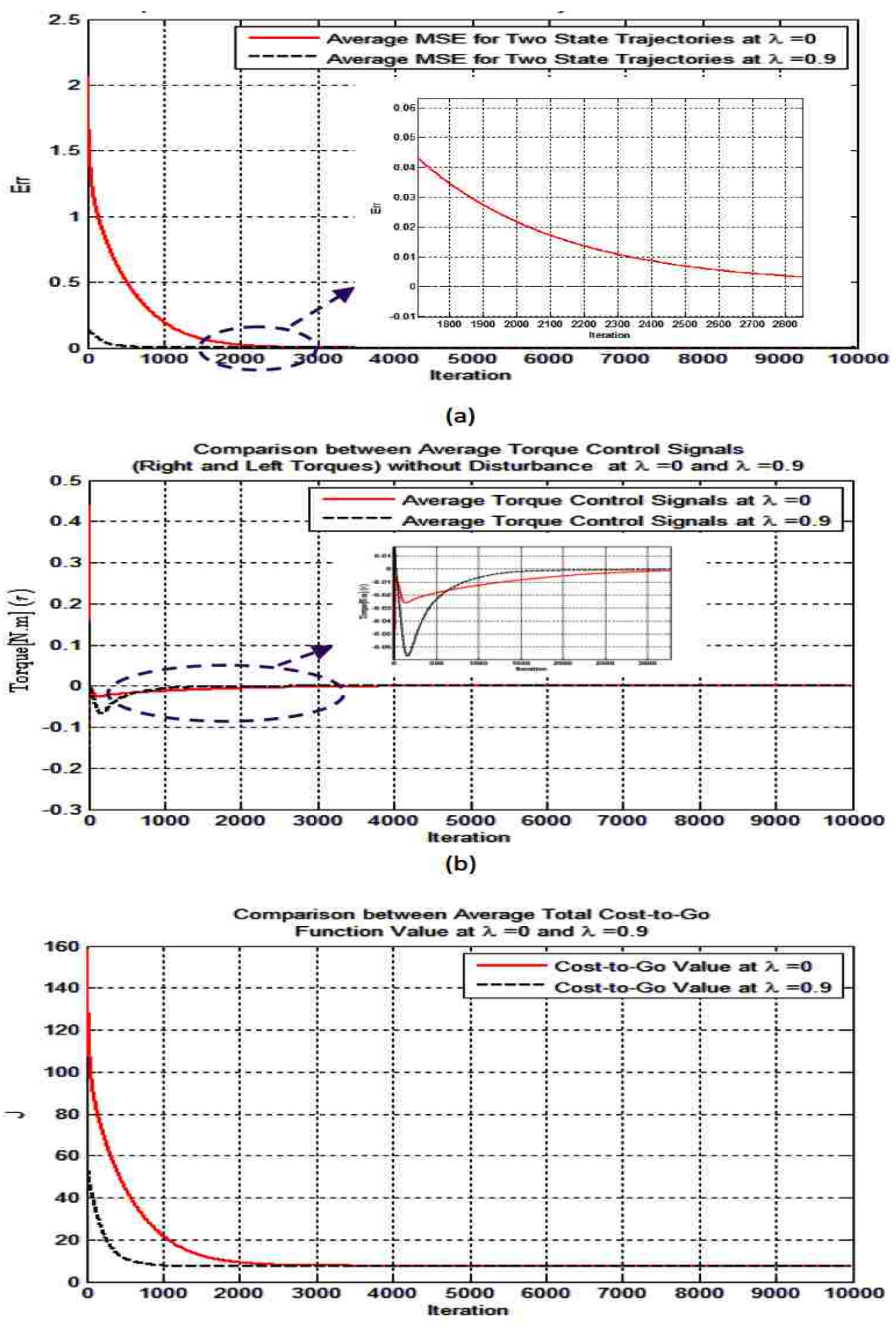


Figure 5. Comparison of the average performance at $\lambda = 0$ and $\lambda = 0.9$. (a) The MSE for both velocity state errors. (b) Requirement torque values. (c) The function value, which is calculated from (1).

simultaneously trained, interference between these networks may occur. We demonstrate that the critic/optimal-torque approach is faster than the critic/actor approach to reach into stable behavior, but it is absorbs less noises. Without the actor network, the difficulty of unpredictable intervention can be skipped. This study case shows full circle trajectories at $\lambda = 0.9$ with disturbances. We use three layers multilayer perceptron neural network (NN) with 85 neurons for hidden layer for comparing with FNN. The other requirement learning parameters for NN can be found in [15]. Both FNN and NN have almost same behavior with noise-free environment ($f_v = f_c = 0$ and $\bar{\tau}_d = [00]^T$) for both critic/actor and critic/optimal-torque approaches. While NN becomes gradually worse response with increasing the amplitude of the noise signals. Fig. 6 (a), Fig. 7 (b)-(c) and Fig. 8 (d)-(e) demonstrate the performance for critic/actor and critic/optimal-torque approaches to follow circle desired trajectory with a large impact of disturbances. The friction coefficients set to $f_v = 0.2$ and $f_c = 0.32$, and the unstructured disturbance is bounded to $\|\bar{\tau}_d\| \leq 113.13$ via setting with ±80 bounded random value for both $\bar{\tau}_L$ and $\bar{\tau}_R$. In this case study, the desired velocity vector is $v_d = [3.5 \quad 6.5]^T$. Fig. 6 (a) X-Y trajectories for the mobile robot for NN and FNN, where the optimal torque for the FNN approach is the best performance. Fig. 7 (b) shows the average of MSE over iterations, which is 0.09 ± 0.02 for the FNN actor/critic approach and $2.6 \pm 0.0001 \times 10^{-3}$ for the FNN critic / optimal-torque approach, while for the NN networks, the MSE's are 0.4 ± 0.3 and 0.08 ± 0.09 an, respectively. Fig. 8 (c) shows the absolute average values for the left and right control torques over iterations. In the stable region of iterations, the torques are bounded 1.33 ± 0.02 for the FNN critic/actor approach and 1.45 ± 0.1 for the FNN critic/optimal-torque approach. While for the NN networks, the absolute average torques are 3.1 ± 0.13 for critic / optimal-torque approach and 2.4 ± 0.17 for critic/actor approach. Fig. 8 (d) shows the FNN networks for right and left torques over time in upper and lower figures, respectively, at the stable learning iteration for both critic / actor and critic / optimal-torque approaches. Fig. 8 (e) shows the value function for the FNN (1), which declines until iteration number around 300, where

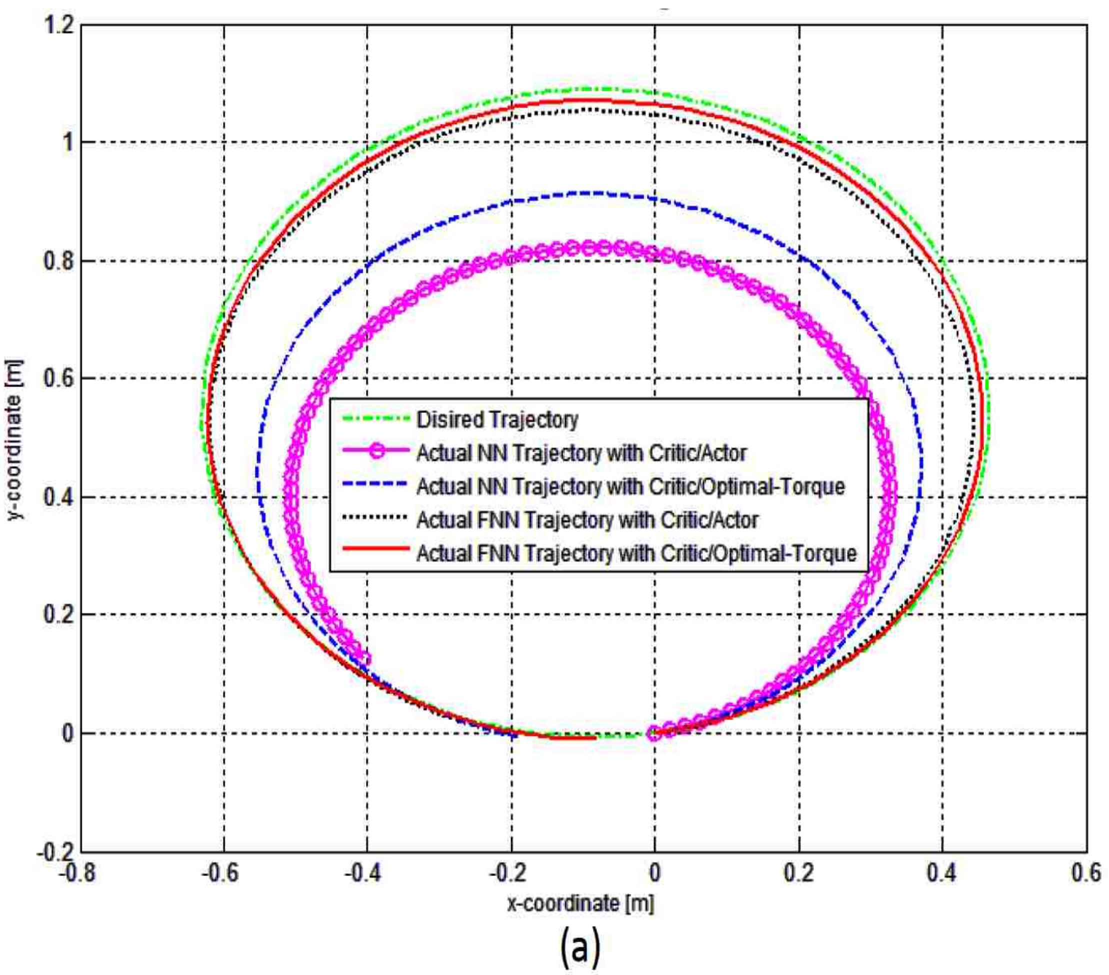
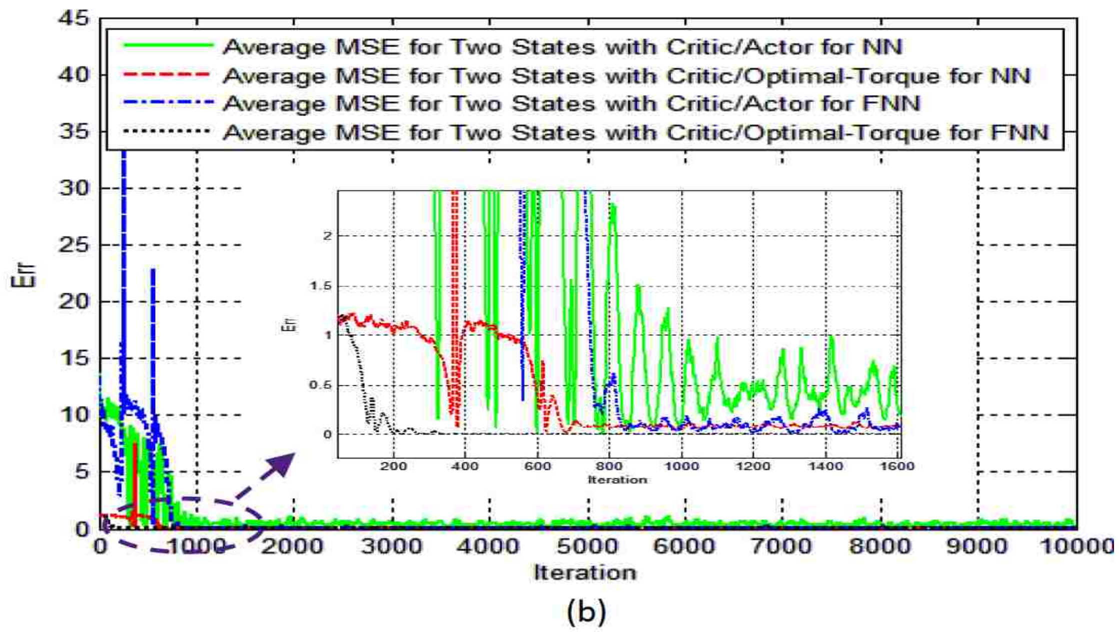


Figure 6. Critic/actor and critic/optimal-torque approaches set to follow a circular trajectory at $\lambda = 0.9$ with disturbances for NN and FNN. (a) The X-Y trajectories for the mobile robot.

it becomes stable and equal to 50 ± 3 for the critic / optimal-torque approach, while it is 50 ± 15 near iteration 800 for the critic/actor approach. While for the NN networks, the stable value function are 90 ± 19 near iteration 600, and 90 ± 24 near iteration 850, for critic / optimal-torque and critic/actor approaches, respectively. As shown in Fig. 7 (b) and Fig. 8 (e) for FNN approach, critic/optimal-torque is faster and has better performance than the critic/actor technique because it reaches into the stability of the optimal cost function value with minimum MSE at iteration number 400, while the critic / actor technique needs 1200 to become stable. However, the critic / actor technique has better performance with



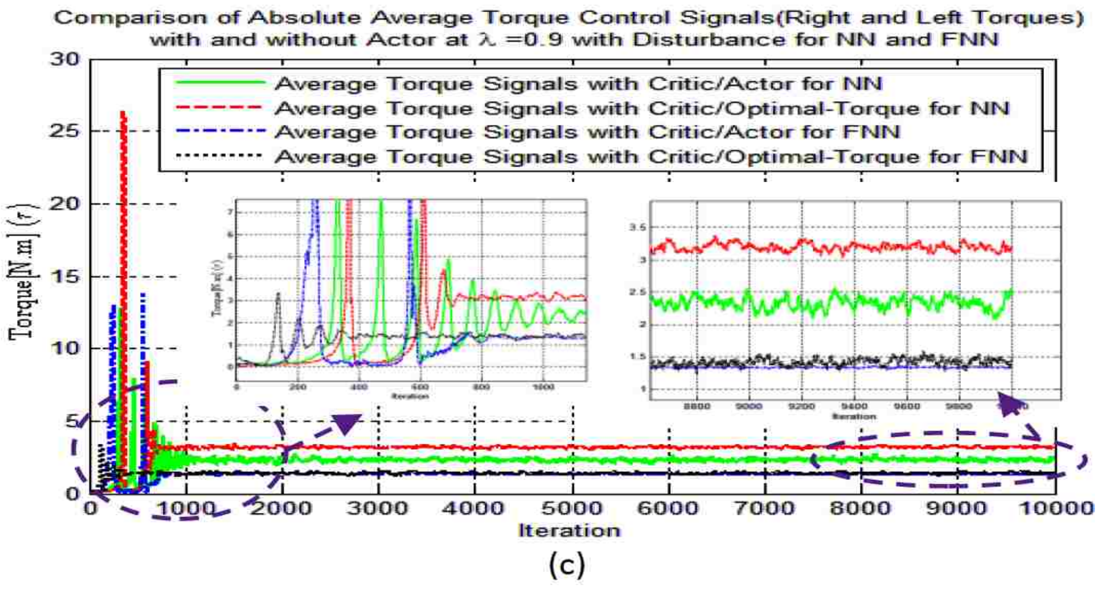
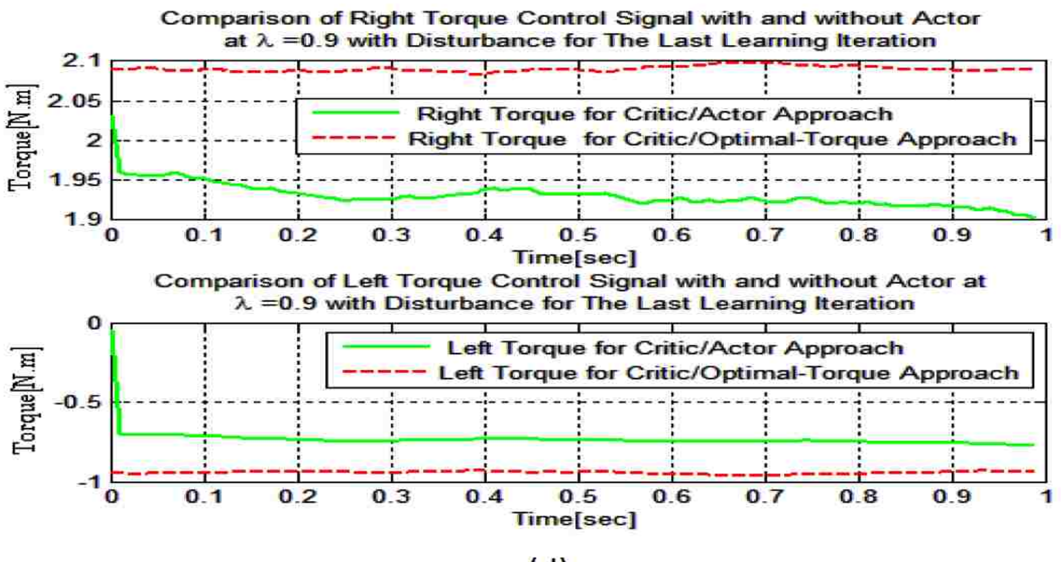
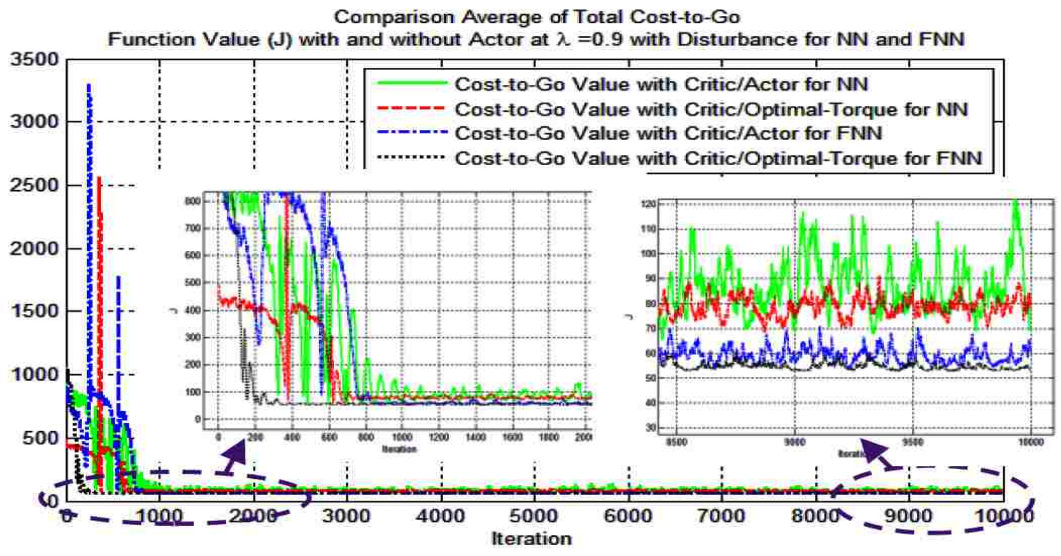


Figure 7. Critic/actor and critic/optimal-torque approaches set to follow a circular trajectory at $\lambda = 0.9$ with disturbances for NN and FNN. (b) and the average of MSE over iterations, respectively. (c) The absolute average values for the left and right control torques over iterations.



(d)



(e)

Figure 8. Critic/actor and critic/optimal-torque approaches set to follow a circular trajectory at $\lambda = 0.9$ with disturbances for NN and FNN. (d) The value of the torques over time for the last learning iteration for FNN. The cost-to-go value is shown in (e).

a noisy environment than the critic/optimal-torque. As shown in Fig. 7 (c) and Fig. 8 (d) for FNN approach, critic/actor technique can absorb most unstructured disturbance signals and friction effects to produce the relatively small amount of torque.

5. CONCLUSION

Tracking a reference trajectory for a mobile robot by using VGL(λ) has been presented. In this work, a new structure of the FNN is used to fit the VGL algorithm with both critic and actor networks. The effectiveness of the λ value, and following trajectories with disturbance impacts with optimal torques calculation are two case studies. The simulation of the mobile robot can deal with the impacts of unmodeled bounded disturbances with various friction parameter values. It successfully reaches the designated reference trajectory. The critic / optimal-torque technique is faster and performs better than critic / actor technique because it reaches into the stable value (the optimal cost function value and the minimum MSE), but the critic / actor technique has better performance with a noisy environment. The critic/actor technique can address the effects of most unstructured disturbance / friction signals.

BIBLIOGRAPHY

[1] C. Samson, "Velocity and torque feedback control of a nonholonomic cart," *Advanced Robot Control*, pp. 125 - 151, Jan. 1991.

[2] S. Mitra, and Y. Hayashi, "Neuro-fuzzy rule generation: survey in soft computing framework," *IEEE Trans. Neural Networks*, vol. 11, no. 3, pp. 748 - 768, Aug. 2002.

[3] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[4] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. NewYork, NY, USA: Wiley, 2012.

[5] D. Prokhorov, and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Networks*, vol. 8, pp. 997 âĂŞ 1007, Sep. 1997.

[6] P. J. Werbos, “Approximate dynamic programming for real-time control and neural modeling,” *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches (Chapter 13)*, Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[7] J. Si, and Y.-T. Wang, “Online learning control by association and reinforcement,” *IEEE Trans. Neural Networks*, vol. 12, no. 2, pp. 264 - 276, Mar. 2001.

[8] Z. Ni, H. He, X. Zhong, and D. Prokhorov, “Model-Free Dual Heuristic Dynamic Programming,” *IEEE Trans. Neural Networks*, vol. 26, no. 8, pp. 1834 - 1839, Aug. 2015.

[9] H. He, Z. Ni, and J. Fu, “A three-network architecture for on-line learning and optimization based on adaptive dynamic programming,” *Neurocomputing*, vol. 78, no. 1, pp. 3 âĂŞ 13, Fib. 2012.

[10] X. Fanga, D. Zhenga, H. He, and Z. Nib, “Data-driven heuristic dynamic programming with virtual reality,” *Neurocomputing*, vol. 166, pp. 244 - 255, Oct. 2015.

[11] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, “GrDHP: A General Utility Function Representation for Dual Heuristic Dynamic Programming,” *IEEE Trans. Neural Networks*, vol. 26, no. 3, pp 614 - 626, mar. 2015.

[12] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Dual heuristic programming excitation neurocontrol for generators in a multimachine power system,” *IEEE Trans. Ind. Appl.*, vol. 39, no. 2, pp. 382 - 394, Mar. 2003.

[13] N. Zhang, and D. C. Wunsch, “A Comparison of Dual Heuristic Programming (DHP) and Neural Network Based Stochastic Optimization Approach on Collective Robotic Search Problem,” *IEEE Trans. Neural Networks*, vol.1, pp. 248 - 253, Jul. 2003.

[14] C. Lian, and X. Xu, “Motion Planning of Wheeled Mobile Robots Based on Heuristic Dynamic Programming,” *IEEE Proceeding on Intelligent Control and Automation Shenyang*, pp 576 - 580, July 2014.

[15] S. Al-Dabooni, and D. Wunsch, “Heuristic Dynamic Programming for Mobile Robot Path Planning Based on Dyna Approach,” *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 3723 - 3730, Jul. 2016.

[16] M. Fairbank, and E. Alonso, “Value-Gradient Learning,” *IEEE International Joint Conference on Neural Networks (IJCNN)*), pp. 1 - 8, Jun. 2012.

[17] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, pp. 9 - 44, Aug. 1988.

[18] R. Fierro, and F. L. Lewis, “Control of a Nonholonomic Mobile Robot Using Neural Networks,” *IEEE Trans. Neural Networks*, vol. 9, no. 4, pp. 589 - 600, Jul. 1998.

[19] W. S. Lin, L. H. Chang, and P. C. Yang, “Adaptive critic anti-slip control of wheeled autonomous robot,” *IET Control Theory and Applications*, vol. 1 , no. 1, pp. 51 - 57, Jan. 2007.

[20] T. Dierks, and S. Jagannathan, “Neural Network Output Feedback Control of Robot Formations,” *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 40, no.2, pp. 383 - 399, Apr. 2010.

[21] R. Kelly, J. Llamas, and R. Campa, “A measurement procedure for viscous and coulomb friction,” *IEEE Trans. on Instrumentation and Measurement*, vol.49, no. 4, pp. 857 - 861, Aug. 2000.

[22] P. J. Werbos, “Backpropagation through time: What it does and how to do it,” *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550 - 1560, 1990

[23] J. Si, A. G. Barto, and W. B. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming.* New York, NY, USA: Wiley, 2004.

[24] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, “Adaptive critic designs: A case study for neurocontrol,” *Neural Networks*, vol. 8, pp. 1367 - 1372, Dec. 1995.

[25] Y. Liu, Y. Lin, S. Wu, C. Chuang, and C. Lin, “Brain Dynamics in Predicting Driving Fatigue Using a Recurrent Self-Evolving Fuzzy Neural Network,” *IEEE Trans. Neural Network,* vol. 27, no. 2, pp. 347 - 360, Feb. 2016.

[26] X. Luo, Y. Lv, R. Li, and Y. Chen, “Web Service QoS Prediction Based on Adaptive Dynamic Programming Using Fuzzy Neural Networks for Cloud Services,” *IEEE Access, Special Section on Challenges for Smart Worlds,* vol. 3, pp. 2260 - 2269, Nov. 2015.

[27] F. Lin, P. Chou, C. Chen, and Y. Lin, “DSP-Based Cross-Coupled Synchronous Control for Dual Linear Motors via Intelligent Complementary Sliding Mode Control,” *IEEE Trans.Industrial Electronics,* vol. 59, no. 2, pp. 1061 - 1073, Feb. 2012.

[28] R. S. Sutton, and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[29] A. K. Palit, G. Doeding, W. Anheier, and D. Popovic, “Backpropagation based training algorithm for Takagi-Sugeno type MIMO neuro-fuzzy network to forecast electrical load time series,” *Proc. of Int. Conf. on FUZZ-IEEE*, vol. 1, pp. 86 - 91, May 2002.

[30] T. Dierks, and S. Jagannathan, “Online Optimal Control of Affine Nonlinear Discrete-Time Systems With Unknown Internal Dynamics by Using Time-Based Policy Update,” *IEEE Trans. Neural Network*, vol. 23 , no. 7 , pp. 1118 - 1129, June 2012.

IV. THE BOUNDEDNESS CONDITIONS FOR MODEL-FREE HDP(λ)

S. Al-Dabooni and Donald C. Wunsch
Department of Electrical & Computer Engineering
Missouri University of Science and Technology
Rolla, Missouri 65409–0050
Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

ABSTRACT

This paper provides stability analysis for a model-free action-dependent heuristic dynamic programming (HDP) approach with an eligibility trace long-term prediction parameter (λ). HDP(λ) learns from more than one future reward. In this work, we prove its uniformly ultimately bounded (UUB) property under certain conditions. Previous works present a UUB proof for traditional HDP (HDP($\lambda = 0$)), but we extend the proof with the λ parameter. By using Lyapunov stability, we demonstrate the boundedness of the estimated error for the critic and actor neural networks as well as learning rate parameters. Three case studies demonstrate the effectiveness of HDP(λ). The trajectories of the internal reinforcement signal nonlinear system are considered as the first case. We compare the results with the performance of HDP. The second case study is a single link inverted pendulum. We investigate the performance of the inverted pendulum by comparing HDP(λ) with regular HDP, with different levels of noise. The third case study is a 3-D maze navigation benchmark, which is compared with SARSA, Q(λ), HDP and HDP(λ). All these simulation results illustrate that HDP(λ) has a competitive performance.

Keywords: Approximate dynamic programming (ADP), model-free, action dependent (AD), heuristic dynamic programming (HDP), λ-return, Lyapunov stability, uniformly ultimately bounded (UUB).

1. INTRODUCTION

Adaptive dynamic programming (ADP) allows agents to select an optimal action sequence by minimizing their long-term cost:

$$J(x(t)) = \langle \sum_{k=t}^{F} \gamma^{k-t} U(x(k), u(k)) \rangle, \tag{1}$$

where $\langle . \rangle$ is the expectation, $J(x(t))$ is a value function (cost-to-go value) for a state vector ($x \in \mathbb{R}^m$) at time step t, γ is a constant discount factor, and $U(x(k), u(k))$, denoted as $U(k)$ for short, is an instantaneous utility function at time step k for x after applying an action vector $u \in \mathbb{R}^{\mathrm{n}}$. F is the final time. Thereby solving Bellman's equation [1], we can find the predicted value fuction. As reviewed in [2], [3], ADP trains an actor network to give optimal actions based on minimizing the cost-to-go value that is produced from a critic network. Both networks are approximated by using a multilayer perceptron. $A(x(t), w_a)$ is the actor network with weights, w_a that produces $u(t)$. The $\hat{v}(x(k), u(k), w_c)$ is the function approximator for the critic network with weights, w_c, which produces $J(x(t))$. ADP has three fundamental categories: heuristic dynamic programming (HDP), dual heuristic programming (DHP), and globalized DHP as shown in [2], [4]. Each category uses three neural networks (actor, critic, and model) that provide decision making, evaluation, and prediction, respectively. The action dependent (AD) method is another version of ADP with no model that takes both states and actions as inputs to the critic. Throughout this work, we use a model-free AD method with HDP. We remove the "AD" abbreviation for simplicity and because model-free approaches are always AD. Our approach does not require *a priori* information about the environment during learning. Online learning happens through interacting with the environment. In [5], Si and Wang implemented an online learning value function for HDP, while Ni *et al.* [6] also implemented a model-free method with DHP. Online ADP learning can be improved by adding another network to support the critic network, which is shown in [7], [8] for HDP and [9] for DHP. Many applications have used ADP. In [10], DHP controlled a turbo-generator more efficiently than HDP. Collective robotic search problems were solved by using DHP [11]. Lian and Xu [12] applied HDP to allow a mobile robot to escape from sharp corners. Al-Dabooni and Wunsch [13] applied HDP with a Dyna algorithm [3] to obtain an optimal path by cooperating multi-robot navigation. Inspired by the temporal-difference (TD) approach with an eligibility trace long-term prediction parameter (λ)in [14], Fairbank and Alonso [15] (see also [16]) introduced a new ADP algorithm that extends DHP by including a λ parameter. They called it value-gradient learning. We used value-gradient learning to track a reference trajectory under uncertainties, by computing the optimal left and right torque values for a nonholonomic mobile robot [17]. Simple interpretation and good performance are two well-known properties attached with TD(λ) as presented in [3], [13]-[24]. But these works use an extra variable associated with each state, which increases computational complexity. This paper overcomes the drawback of using eligibility-trace storage, improving simplicity and performance. This paper also provides a stability proof to determine what suitable learning parameters (λ, γ and critic/actor learning rates) should be used during training. The general stability of ADP is an open problem [25]. Under certain conditions, we use Lyapunov theory to prove stability for the general case of HDP(λ). Prior contributions [25], [26] proved stability of model-free learning only for the one-step ($\lambda = 0$) HDP(0) case. We explain the HDP(λ) structure in Section 2. Section 3 presents a boundedness stability analysis; Section 6 presents simulation results, and Section 7 is the Conclusion.

2. STRUCTURE OF MODEL-FREE HDP(λ)

2.1. HDP(λ) Learning Views.

The general nonlinear system model is represented as:

$$x(t+1) = f(x(t), u(t)), \tag{2}$$

where x is the m-dimensional system state vector and u is the n-dimensional control vector. HDP solves the recursive form of (1) [27]:

$$J^*(t) = \min_{u(t)}\{U(t+1) + \gamma J^*(t+1)\}, \tag{3}$$

where J^* denotes the optimal value function of (1), and the instantaneous cost ($U(t) \in [0, 1]$) is bounded. Equation (1) is solved by using a one-step TD, or equivalently HDP(0). As in [3], the Bellman equation for a one-step-return $R_t^{(1)}$, two-step-return $R_t^{(2)}$, and n-step-return $R_t^{(n)}$ are given as follows:

$$R_t^{(1)} = U(t) + \gamma J(t+1), \tag{4}$$

$$R_t^{(2)} = U(t) + \gamma U(t+1) + \gamma^2 J(t+2), \tag{5}$$

$$R_t^{(n)} = U(t) + \gamma U(t+1) + \ldots + \gamma^{n-1} U(t+n-1) + \gamma^n J(t+n) = \sum_{k=0}^{n} \gamma^k U(t+k), \tag{6}$$

where $R_t^{(i)}$ is an i-step-return value, which is a summation of the instantaneous costs from t to i. An average of the n-step-return is a technique to achieve fair cost distribution. For instance, the average calculation of a 4-step-return can be done via a half of 2-step-return and half of a 4-step-return such that $R_t^{Av(2,4)} = \omega_2 R_t^{(2)} + \omega_4 R_t^{(4)}$, where $\omega_2 = 0.5$, and $\omega_4 = 0.5$. The proportional weights (ω_i), $i = 1, 2 \ldots$, are positive and sum to 1. The λ parameter represents the proportional weights. Therefore, we refer to λ as a long-term prediction parameter. For instance, $\omega_1 = (1 - \lambda)$ to average a one-step-return ($\lambda = 0$), which is equivalent to (4) ($R_t^{(1)} = R_t^{\lambda=0}$); to average a two-step-return, $R_t^{Av(1,2)} = \omega_1 R_t^{(1)} + \omega_2 R_t^{(2)}$, where $\omega_1 = (1 - \lambda)$ and $\omega_2 = \lambda$, and so on. In Section 3 Theorem 2, we prove that λ should be within $0 \le \lambda < 1$. This is in contrast to previous literature, which have different rules for λ value (included bounded from 0 to 1). The λ-return (R_t^{λ}) is another name for the average of an n-step-return, which is defined in general as:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)}. \tag{7}$$

For simplicity, we denote the n-step expression instead of the average of n-step-return. The previous approach is called the *forward view* of a learning algorithm. Because each step uses the knowledge of what will happen many steps later, the *forward view* is not directly implementable. The *backward view* provides an extra variable associated with each state, which is called an eligibility trace ($e_t(x)$). An accumulating trace method by Sutton [14] is a first technique that updates an $e_t(x)$ variable for every step as follows (i.e., Q(λ)-learning algorithm):

$$e_t(t) = \begin{cases} \gamma\lambda e_{t-1}(x) & if \quad x \neq x(t) \\ \gamma\lambda e_{t-1}(x) + 1 & if \quad x = x(t), \end{cases} \tag{8}$$

where λ here is used as a trace-decay parameter. Doya [28] derives continuous time eligibility traces in more detail. Because the traces increase with repeated visits to a state in accumulating traces, a truncating method is suggested by Sutton [24] to overcome the accumulative behavior. HDP(λ) does not use any eligibility trace parameters by re-deriving the λ-return as in (7), which is used to train the parameters for the approximate value function. Fairbank [29] denotes the λ-return at time t (R_t^{λ}) as a target-value ($\bar{v}(t)$), which is defined in (8). We rederive R_t^{λ} associated with the γ value to prove that R_t^{λ} is equivalent to $\bar{v}(t)$ as follows: By substituting (4) into (7) and assuming the $\hat{v}(t) = J(t)$, we obtain:

$$\begin{aligned} R_t^{\lambda} &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left(\left[\sum_{k=0}^{n-1}\gamma^k U(t+k)\right] + \gamma^n\hat{v}(t+n)\right). \\ &= (1-\lambda)\left(\left[\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{k=0}^{n-1}\gamma^k U(t+k)\right] + \sum_{n=1}^{\infty}\lambda^{n-1}\gamma^n\hat{v}(t+n)\right). \end{aligned} \tag{9}$$

Expanding (6) yields

$$\begin{aligned}R_t^\lambda =&(1-\lambda)\Big[\lambda^0\big(\gamma^0 U(t)\big) + \lambda^1\big(\gamma^0 U(t) + \gamma^1 U(t+1)\big) + \lambda^2\big(\gamma^0 U(t) + \gamma^1 U(t+1) + \\ &\gamma^2 U(t+2)\big) + \ldots + \lambda^\infty \sum_{k=0}^{\infty}\big(\gamma^k U(t+k)\big)\Big] + (1-\lambda)\sum_{n=1}^{\infty}\Big[\lambda^{n-1}\gamma^n \hat{v}(t+n)\Big].\end{aligned} \tag{10}$$

Further arranging for (10) results in

$$\begin{aligned}R_t^\lambda =&(1-\lambda)\Big[\gamma^0 U(t)\big(\lambda^0 + \lambda^1 + \ldots + \lambda^\infty\big) + \gamma^1 U(t+1)\big(\lambda^1 + \ldots + \lambda^\infty\big) + \\ &\ldots + \gamma^\infty U(t+\infty)\big(\lambda^\infty\big)\Big] + (1-\lambda)\sum_{n=1}^{\infty}\Big[\lambda^{n-1}\gamma^n \hat{v}(t+n)\Big].\end{aligned} \tag{11}$$

Simplifying the first term of (11), we obtain:

$$R_t^\lambda = \sum_{n=0}^{\infty}\Big[\gamma^n U(t+n)\big(\sum_{k=n}^{\infty}\lambda^k - \lambda\sum_{k=n}^{\infty}\lambda^k\big)\Big] + (1-\lambda)\sum_{n=1}^{\infty}\Big[\lambda^{n-1}\gamma^n \hat{v}(t+n)\Big] \tag{12}$$

With more rearranging and extraction (12), we obtain:

$$\begin{aligned}R_t^\lambda =& \sum_{n=0}^{\infty}\gamma^n U(t+n)\Big[\big(\lambda^n + \lambda^{n+1} + \ldots + \lambda^\infty\big) - \big(\lambda^{n+1} + \lambda^{n+2} + \ldots + \lambda^\infty\big)\Big] + \\ &(1-\lambda)\sum_{n=0}^{\infty}\Big[\lambda^n\gamma^{n+1}\hat{v}(t+n+1)\Big],\end{aligned} \tag{13}$$

and therefore,

$$\begin{aligned}R_t^\lambda =& \sum_{n=0}^{\infty}\Big[\lambda^n\gamma^n\Big(U(t+n) + (1-\lambda)\gamma\hat{v}(t+n+1)\Big)\Big], \\ =&U(t) + \lambda\gamma\Bigg(U(t+1) + \lambda\gamma\Bigg(U(t+2) + \lambda\gamma\Bigg(U(t+3) + \ldots + \\ &\lambda\gamma\big(U(\infty) + (1-\lambda)\gamma J(\infty)\big) + \ldots + (1-\lambda)\gamma J(t+4)\Bigg) + (1-\lambda)\gamma \\ &J(t+3)\Bigg) + (1-\lambda)\gamma J(t+2)\Bigg) + (1-\lambda)\gamma J(t+1),\end{aligned} \tag{14}$$

where $U(\infty) = \bar{v}(\infty)$, which is the instantaneous cost at the infinite horizon terminal state. The final target-value according to (7) is

$$\bar{v}(t) = R_t^{\lambda} = U(t) + \lambda\gamma\bar{v}(t+1) + (1-\lambda)\gamma\hat{v}(t+1). \tag{15}$$

Using (8), we construct the HDP(λ) schematic, which consists of two components, the critic and actor, as shown in Fig. 1. The initial target ($\bar{v}(t)$) for the first trial is provided by substituting $\lambda = 0$ in (8), which is identical to the HDP(0) approach.

Many function approximators are used to implement ADP [10], [17], [31], [32]. For instance, a radial basis function neural network, [32], [33], is one option, but we keep first-layer weights constant during training for both critic and actor networks; therefore, a feedforward neural network is appropriate. Because of the elegant and extendable structure of a fully connected neural network [30], we use it in the actor and critic networks.

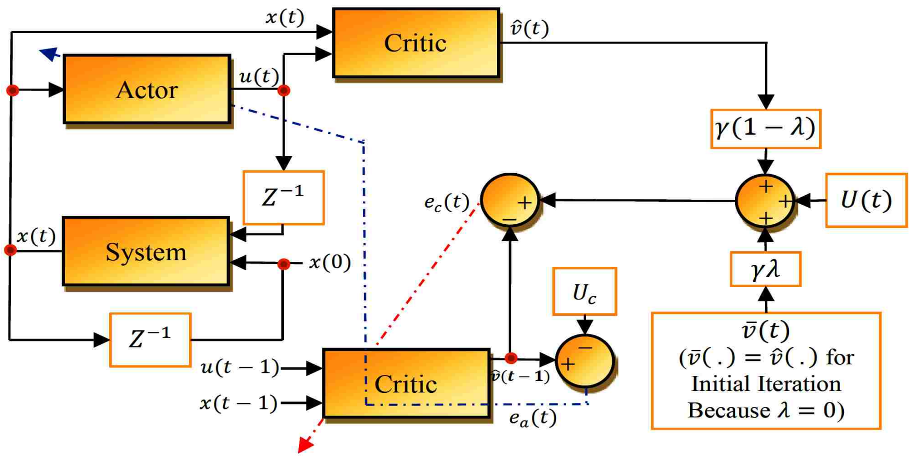


Figure 1. Schematic for the adaptation of the novel model-free HDP(λ) structure design according to (8). Forward pathways are represented by solid lines, while backpropagation pathways are shown by dashed lines. The small solid black dots represent a connection path. The initial target-values ($\bar{v}(t)$) are provided by substituting $\lambda = 0$ with (8), which is identical to the traditional HDP. The TD-error between $\bar{v}(t-1)$ and $\hat{v}(t-1)$, as in (76), is used to update critic network weights (84), which is represented by red dashed line. The actor network weights are estimated by back-propagating the prediction error (blue dashed line), which is equal to the value function ($\hat{v}(t-1)$) through the critic network, and updating the actor's weights according to (28).

2.2. The Critic Neural Network. The structure of the critic network is shown in Fig. 2. A fully connected three-layer feed-forward neural network is used in this work. The output is $\hat{v}(t)$ that learns to approximate $J(t)$ as in (1). The inputs for the critic network are the action values $(u_1(t), u_1(t), \ldots, u_{\mathfrak{n}}(t))$ and the system states $(x_1(t), x_2(t), \ldots, x_m(t))$. h_c is the number of hidden neurons, m is the number of system states, and $\mathfrak{n}$ is the number of control action values.

The hidden weights are indicated as $\hat{\omega}_c^{\{h\}}$, which can be represented in a $(m+\mathfrak{n}) \times h_c$ dimension matrix. The output weights are $\hat{\omega}_c^{\{o\}}$, which can be represented in a $h_c \times 1$ dimension matrix.

The activation function for the hidden nodes is the hyperbolic tangent $\phi(x) = (1 - e^{-x})/(1 + e^{-x})$. The forward propagating output signal according to Fig. 1 and Fig. 2 is

$$s_{(k)}(t) = \sum_{i=1}^{m} \hat{\omega}^{\{h\}}_{c_{(k,i)}} x_{(i)}(t) + \sum_{j=1}^{\mathfrak{n}} \hat{\omega}^{\{h\}}_{c_{(k,j+m)}} u_{(j)}(t), \quad k = 1, 2, \ldots, h_c, \tag{16}$$

$$r_{(k)}(t) = \phi(s_{(k)}(t)), k = 1, 2, \ldots, h_c, \tag{17}$$

$$\hat{v}(t) = \sum_{k=1}^{h_c} \hat{\omega}^{\{o\}}_{c_{(k)}} r_{(k)}(t), \tag{18}$$

where $s_{(k)}(t)$ is the kth hidden node input of the critic network, and $r_{(k)}(t)$ is the corresponding output of the hidden node. The weights for hidden ($\omega_c^{\{h\}}$) and output ($\omega_c^{\{o\}}$) layers can be implemented by back-propagating the prediction error of the critic network, which is

$$e_c(t) = U(t) + \lambda\gamma\bar{v}(t) + (1 - \lambda)\gamma\hat{v}(t) - \hat{v}(t - 1). \tag{19}$$

The objective function for the critic network is to minimize $E_c(t) = 0.5\big(e_c(t)\big)^2$ by updating the value for the weights according to the gradient descent algorithm:

$$\hat{\omega}_c(t + 1) = \hat{\omega}_c(t) + \triangle\hat{\omega}_c(t) = \hat{\omega}_c(t) - \ell_c \frac{\partial E_c(t)}{\partial \hat{\omega}_c(t)}, \tag{20}$$

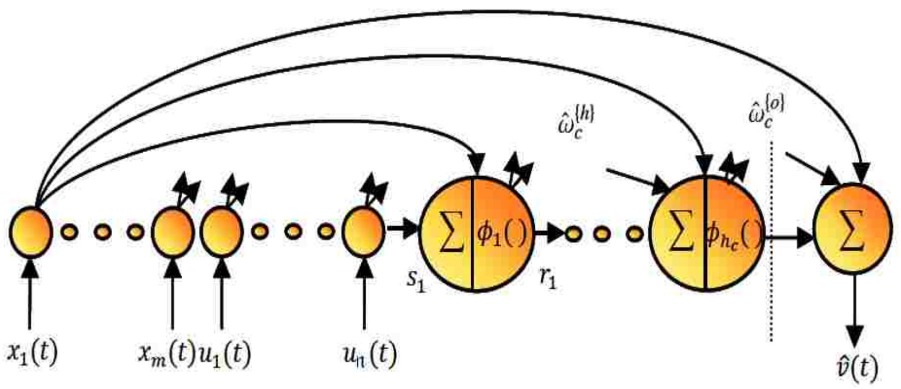


Figure 2. Schematic diagram of the critic network in HDP(λ). As mentioned by Werbos in [30], this structure is more general than a traditional three-layer feedforward neural network by fully connecting all neurons. It models a variety of functional forms [34]. We set all weights that connect input nodes with output nodes to zero; therefore, it is similar to a three-layer feedforward neural network structure. $\hat{\omega}_c^{\{h\}}$ represents a hidden weights, which connect the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_c^{\{o\}}$, which connect both input and hidden layers with the output layer. $s_k(t)$ is the kth hidden node input of the critic network, and $r_k(t)$ is the corresponding output the hidden node. Here, we only apply a hyperbolic tangent threshold function ($\phi(.)$) to the hidden neurons.

and the chain propagation path can be represented as

$$\frac{\partial E_c(t)}{\partial \hat{\omega}_c(t)} = \frac{\partial E_c(t)}{\partial \hat{v}(t)} \cdot \frac{\partial \hat{v}(t)}{\partial \hat{\omega}_c(t)}. \tag{21}$$

Then, the adaptation of the output weights is

$$\begin{aligned}\hat{\omega}_c(t+1) =& \hat{\omega}_c(t) - \ell_c \gamma (1-\lambda) \phi_c(t) [\gamma \lambda \bar{v}(t) + \gamma(1-\lambda) \hat{\omega}_c^T(t) \phi_c(t) + U(t) - \\ & \hat{\omega}_c^T(t-1) \phi_c(t-1)]^T,\end{aligned} \tag{22}$$

where $\phi_c = [r_1, r_2, \ldots, r_{h_c}]^T$, $\hat{\omega}_c = \hat{\omega}_c^{\{o\}}$, and $\hat{v}(t) = \hat{\omega}_c^T(t)\phi_c(t)$. Following [2], we chose the hidden critic weights ($\hat{\omega}_c^{\{h\}}$) initially at random and kept them constant while updating the output critic weights ($\hat{\omega}_c^{\{o\}} \equiv \hat{\omega}_c$).

2.3. The Actor Neural Network. The actor network is used to generate a near-optimal policy. Fig. 4 illustrates the structure of the actor network, which is similar to critic network structure but with multi-output. The system state is the input to this network $(x_1(t), x_2(t), \ldots, x_m(t))$, while the outputs are the actions $(u_1(t), u_2(t), \ldots, u_{\text{n}}(t))$. h_a is the number of hidden neurons, m is the number of system states, and n is the number of control action values. The hidden weights are indicated as $\hat{\omega}_a^{\{h\}}$, which can be represented in a $m \times h_a$ dimension matrix. The output weights are indicated as $\hat{\omega}_a^{\{o\}}$, which can be represented in a $h_a \times \text{n}$ dimension matrix. The activation function for the hidden nodes is the hyperbolic tangent. The activation function for the hidden nodes is the same as the critic network. The forward propagating output signal according to Fig. 1 and Fig. 4 can be expressed as follows:

$$p_{(k)}(t) = \sum_{i=1}^{m} \hat{w}_{a_{(k,i)}}^{\{h\}} x_{(i)}(t), \quad k = 1, 2, \ldots, h_a, \tag{23}$$

$$q_{(k)}(t) = \phi(p_{(k)}(t)), \quad k = 1, 2, \ldots, h_a, \tag{24}$$

$$\hat{u}_{(j)}(t) = \sum_{k=1}^{h_a} \hat{w}_{a_{(j,k)}}^{\{o\}} q_{(k)}(t), \quad j = 1, 2, \ldots, \text{n}, \tag{25}$$

where $p_{(k)}(t)$ is the kth hidden node input of the actor network, and $q_{(k)}(t)$ is the corresponding output of the hidden node.

The actor network weights are adapted by back-propagating the prediction error of the actor network ($e_a(t) = \hat{v}(t) - U_c$), where U_c is the desired ultimate cost-to-go objective value. As in [5], U_c is set to "0," corresponding to "success." The objective function for this network is to minimize $E_a(t) = 0.5\big(e_a(t)\big)^2$ by updating the weights as follows:

$$\begin{aligned} \hat{\omega}_a(t+1) =& \hat{\omega}_a(t) + \triangle\hat{\omega}_a(t) = \hat{\omega}_a(t) - \ell_a \frac{\partial E_a(t)}{\partial \hat{\omega}_a(t)} \\ =& \hat{\omega}_a(t) - \frac{1}{2}\ell_a \Big(\frac{\partial E_a(t)}{\partial \hat{v}(t)} \cdot \frac{\partial \hat{v}(t)}{\partial u(t)} \cdot \frac{\partial u(t)}{\partial \hat{\omega}_a(t)} \Big), \end{aligned} \tag{26}$$

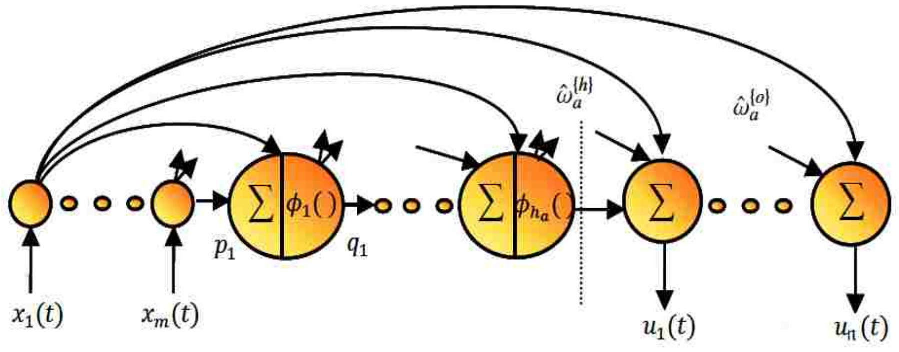


Figure 3. Schematic diagram of the actor network in HDP(λ). We set all weights that connect input nodes with output nodes to zero as well as the connected weights between the outputs themselves. $\hat{\omega}_a^{\{h\}}$ represents hidden weights which connect the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_a^{\{o\}}$, which connect both input and hidden layers with the output layer. $p_k(t)$ is the kth hidden node input of the actor network, and $q_k(t)$ is the corresponding output of the hidden node. Here, we only apply a hyperbolic tangent threshold function ($\phi(.)$) on the hidden neurons.

where

$$\frac{\partial \hat{v}(t)}{\partial u_{(k)}(t)} = \sum_{i=1}^{h_c} \frac{\partial \hat{v}(t)}{\partial s_{(i)}(t)} \cdot \frac{\partial s_{(i)}(t)}{\partial r_{(i)}(t)} \cdot \frac{\partial r_{(i)}(t)}{\partial u_{(k)}(t)}, \tag{27}$$

where $k = 1, 2, \ldots \text{n}$. The final adaptation of the action network's weights between the hidden and output layers is

$$\hat{\omega}_a(t+1) = \hat{\omega}_a(t) - \ell_a \phi_a(t)[\hat{\omega}_c^T(t) C_{BP}(t)][\hat{\omega}_c^T(t)\phi_c]^T, \tag{28}$$

where $\phi_a = [q_1, q_2, \ldots, q_{h_a}]^T$, $\hat{\omega}_a = \hat{\omega}_a^{\{o\}}$, $u(t) = \hat{\omega}_a^T(t)\phi_a(t)$, $C_{BP}(t)$ is a matrix of $h_c \times \text{n}$, and the elements for this matrix are $C_{BP_{(k,j)}}(t) = 0.5\left(1 - \phi_{c_{(k)}}^2(t)\right)\hat{\omega}_{c_{(k,m+j)}}^{\{h\}}(t)$ where $k = 1, 2, \ldots h_c$ and $j = 1, 2, \ldots \text{n}$. The actor's hidden weights ($\hat{\omega}_a^{\{h\}}$) are kept constant while its updating output weights ($\hat{\omega}_a^{\{o\}} \equiv \hat{\omega}_a$) [2].

3. STABILITY ANALYSIS

This section discusses stability for the critic and actor networks by employing Lyapunov functions.

3.1. Lyapunov Approach. Let ω_c^* and ω_a^* denote the optimal weights for the critic and actor networks as follows: $\omega_c^* = argmin_{\hat{\omega}_c} \|U(t) + \gamma(\lambda\bar{v}(t) + (1-\lambda)\hat{v}(t)) - \hat{v}(t-1)\|$ and $\omega_a^* = argmin_{\hat{\omega}_a} \|\hat{v}(t)\|$. The weight estimation error for both critic and actor networks is:

$$\tilde{\omega}(t) = \hat{\omega}(t) - \omega^*. \tag{29}$$

A more general discrete time dynamic system for weight update rules in (84) and (28) for critic and actor networks define a dynamic system of estimation errors for a general nonlinear function ($g(.)$) as:

$$\tilde{\omega}(t+1) = \tilde{\omega}(t) - g(\hat{\omega}(t), \hat{\omega}(t-1), \phi(t), \phi(t-1)). \tag{30}$$

Therefore, the stability properties of the system in (39) express the asymptotic behavior of the estimation error of the weights ($\tilde{\omega}(t)$).

Definition 1. A discrete time dynamic system (39) solution is UUB $\varepsilon > 0$, if and only if, for all $\delta > 0$ and $t_0 > 0$, there exists a positive number $N = N(\delta, \varepsilon)$ independent of t_0, such that $\|\tilde{\omega}(t)\| \leq \varepsilon$ for all $t \geq N + t_0$ when $\|\tilde{\omega}(t)\| \leq \delta$.

Theorem 1 (A property for UUB). The discrete time dynamic system (39) has a Lyapunov function $L(\tilde{\omega}(t), t)$ such that for all $\tilde{\omega}(t_0)$ in a compact set K, $L(\tilde{\omega}(t), t)$ is positive definite and the first difference, $\triangle L(\tilde{\omega}(t), t) < 0$ for $\|\tilde{\omega}(t)\| > \varepsilon$, for $\varepsilon > 0$, such that $\varepsilon-$ neighborhood of $\tilde{\omega}(t)$ is contained in K. Thus, the dynamic system is UUB and the norm of the state is bounded within a neighborhood of ε.

According to Theorem 1, an appropriate function L is selected to determine the UUB property for (39) subject to Assumptions C and A below.

3.2. Preliminaries. In this subsection, two lemmas will be presented, which are used to prove the main theorem.

Assumptions C (for the critic). Let $\tilde{\omega}_c^*(t)$ be the optimal weights for the critic network:

$$\begin{aligned}\omega_c^* =& argmin_{\hat{\omega}_c} \|U(t) + \gamma(\lambda\bar{v}(t) + (1-\lambda)\hat{v}(t)) - \hat{v}(t-1)\| \\ =& argmin_{\hat{\omega}_c} \|\gamma\lambda\bar{v}(t) + \gamma(1-\lambda)\hat{\omega}^T(t)\phi_c(t) + U(t) - \hat{\omega}^T(t-1)\phi_c(t-1)\|,\end{aligned} \tag{31}$$

assuming $\tilde{\omega}_c^*(t)$ is bounded by some positive constant, i.e., $\|\omega_c^*\| \leq \omega_c^{max}$.

Lemma C. Let assumption C hold. Then the first difference of $L_c(t) = \frac{1}{\ell_c} tr(\tilde{\omega}_c^T(t)\tilde{\omega}_c(t))$, where tr is the trace of a matrix, is expressed as follows:

$$\begin{aligned}\triangle L_c(t) \leq& -\beta^2\|\xi_c(t)\|^2 - \beta^2(1-\ell_c\beta^2\|\phi_c(t)\|^2) \times \|\xi_c(t) + \omega_c^{*^T}\phi_c(t) + \beta^{-1}\gamma\lambda\bar{v}(t) \\ &+ \beta^{-1}U(t) - \beta^{-1}\tilde{\omega}_c^T(t-1)\phi_c(t-1)\|^2 + 2\|\beta\omega_c^{*^T}\phi_c(t) + \gamma\lambda\bar{v}(t) + U(t) - \\ &\frac{1}{2}\hat{\omega}_c^T(t-1)\phi_c(t-1) - \frac{1}{2}\omega_c^{*^T}\phi_c(t-1)\|^2 + \frac{1}{2}\|\xi_c(t-1)\|^2,\end{aligned} \tag{32}$$

where $\xi_c(t) = \tilde{\omega}_c(t)\phi_c(t)$ is the approximation error of the critic network output and $\beta = \gamma(1-\lambda)$.

Lemma C Proof. The first discrete difference of the nominated Lyapunov function is given as follows:

$$\triangle L_c(t) = \frac{1}{\ell_c} tr(\tilde{\omega}_c^T(t+1)\tilde{\omega}_c(t+1) - \tilde{\omega}_c^T(t)\tilde{\omega}_c(t)). \tag{33}$$

By the updating rule of (38) and $\tilde{\omega}_c(t+1)$ as in (84), we get:

$$\begin{aligned}\tilde{\omega}_c(t+1) =& -\ell_c\beta\phi_c(t)\Big[\gamma\lambda\bar{v}(t) + \beta\omega_c^{*^T}(t)\phi_c(t) + U(t) - \hat{\omega}^T(t-1)\phi_c(t-1)\Big]^T + \\ &\Big[I - \ell_c\beta^2\phi_c(t)\phi_c^T(t)\Big]\tilde{\omega}_c(t).\end{aligned} \tag{34}$$

Assuming $P = \Big[I - \ell_c\beta^2\phi_c(t)\phi_c^T(t)\Big]$ and $Q = \Big[\gamma\lambda\bar{v}(t) + \beta\omega_c^{*^T}(t)\phi_c(t) + U(t) - \hat{\omega}^T(t-1)\phi_c(t-1)\Big]$, then by substituting P, Q, and (43) into (42), we get:

$$\triangle L_c(t) = \frac{1}{\ell_c}\Bigg(tr\Big(\tilde{\omega}_c^T(t)P^T P\tilde{\omega}_c(t)\Big) - tr\Big(\ell_c\beta\tilde{\omega}_c^T(t)P^T\phi_c(t)Q^T\Big) - tr\Big(\ell_c\beta Q\phi_c^T(t)P\tilde{\omega}_c(t)\Big) + tr\Big(\ell_c^2\beta^2 Q\phi_c^T(t)\phi_c(t)Q^T\Big)\Bigg), \tag{35}$$

$$\begin{aligned}\triangle L_c(t) = -\beta^2\|\xi_c(t)\|^2 - \beta^2\Big(1 - \ell_c\beta^2\|\phi_c(t)\|^2\Big)\|\xi_c(t)\|^2 \\ - 2\beta tr\Bigg(\Big(1 - \ell_c\beta^2\|\phi_c(t)\|^2\Big)\xi_c(t)Q^T\Bigg) + \ell_c\beta^2\|\phi_c(t)\|^2 \\ \|\beta\omega_c^{*^T}\phi_c(t) + \gamma\lambda\bar{v}(t) + U(t) - \tilde{\omega}_c^T(t-1)\phi_c(t-1)\|^2.\end{aligned} \tag{36}$$

By applying the Cauchy-Schwarz inequality, we get:

$$\begin{aligned}\triangle L_c(t) \le -\beta^2\|\xi_c(t)\|^2 - \beta^2\Big(1 - \ell_c\beta^2\|\phi_c(t)\|^2\Big) \times \|\xi_c(t) + \omega_c^{*^T}\phi_c(t) + \\ \gamma\lambda\beta^{-1}\bar{v}(t) + \beta^{-1}U(t) - \beta^{-1}\tilde{\omega}_c^T(t-1)\phi_c(t-1)\|^2 + \\ \|\beta\omega_c^{*^T}\phi_c(t) + \gamma\lambda\bar{v}(t) + U(t) - \frac{1}{2}\hat{\omega}_c^T(t-1)\phi_c(t-1) \\ - \frac{1}{2}\omega_c^{*^T}\phi_c(t-1) + \xi_c(t-1)\|^2.\end{aligned} \tag{37}$$

With further arrangement, we get (41).

Assumptions A (for the actor). Let $\tilde{\omega}_a^*(t)$ be the optimal weights for the actor network bounded by a positive constant, i.e., $\|\omega_a^*\| \le \omega_a^{max}$. Let $\xi_a(t) = [\hat{\omega}_a(t) - \omega_a^*]^T\phi_a(t) = \tilde{\omega}_a^T(t)\phi_a(t)$ be the approximation error of the actor network output.

Lemma A. Let Assumption A hold. Then the first difference of $L_a(t) = \frac{1}{\alpha\ell_a}tr(\tilde{\omega}_a^T(t)\tilde{\omega}_a(t))$ is expressed as follows:

$$\triangle L_a(t) \leq \frac{1}{\alpha}\Bigg(- \Big(1 - \ell_a(t)\|\phi_a(t)\|^2\|\hat{\omega}_a^T(t)C_{BP}(t)\|^2\Big) \times \|\hat{\omega}_c^T(t)\phi_c(t)\|^2 + \|\xi_a(t)(\hat{\omega}_a^T(t)C_{BP}(t))\|^2 + 4\|\xi_c(t)\|^2 + 4\|\omega_c^{*^T}\phi_c(t)\|^2\Bigg), \tag{38}$$

where α is a weighting factor, which will be defined in (57).

Lemma A Proof. The asymptotic behavior of the estimation error of the actor weights ($\tilde{\omega}_a(t)$) is analyzed by studying the stability of (39). The first discrete difference of the nominated Lyaponov function is given as follows:

$$\triangle L_a(t) = \frac{1}{\alpha\ell_a} tr\Bigg(\tilde{\omega}_c^T(t+1)\tilde{\omega}_a(t+1) - \tilde{\omega}_a^T(t)\tilde{\omega}_a(t)\Bigg), \tag{39}$$

By updating the rule of (38) and $\tilde{\omega}_a(t+1)$ as in (28), we get:

$$\tilde{\omega}_a(t+1) = \tilde{\omega}_a(t) - \ell_a\phi_a(t)\hat{\omega}_c^T(t)C_{BP}(t)\Big[\hat{\omega}_c^T(t)\phi_c(t)\Big]^T. \tag{40}$$

By substituting (54) into (53), we get:

$$\triangle L_a(t) = \frac{1}{\alpha\ell_a} tr\Bigg(- 2\ell_a\xi_a(t)\hat{\omega}_c^T(t)C_{BP}(t)\Big[\hat{\omega}_c^T(t)\phi_c(t)\Big]^T + \ell_a^2\|\phi_a(t)\|^2|\hat{\omega}_c^T(t)C_{BP}(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\Bigg), \tag{41}$$

$$\triangle L_a(t) = \frac{1}{\alpha}\Bigg(\|\hat{\omega}_c^T(t)\phi_c(t) - \xi_a(t)\hat{\omega}_c^T(t)C_{BP}(t)\|^2 - \|\xi_a(t)\hat{\omega}_a^T(t)C_{BP}(t)\|^2 - \|\hat{\omega}_c^T(t)C_{BP}(t)\|^2 + \ell_a\|\phi_a(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\Bigg). \tag{42}$$

By applying the Cauchy-Schwarz inequality, we obtain:

$$\triangle L_a(t) \leq \frac{1}{\alpha}\Bigg(4\|\xi_c(t)\|^2 + 4\|\omega_c^{*^T}\phi_c(t)\|^2 + \|\xi_a(t)\hat{\omega}_c^T(t)C_{BP}(t)\|^2 - \\ \|\hat{\omega}_c^T(t)\phi_c(t)\|^2 + \ell_a\|\phi_a(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\Bigg). \tag{43}$$

With a simple arrangement, we get (52).

3.3. The Dynamical System Stability Analysis. The Lyapunov candidate function is analyzed in this subsection to prove the bound of the system estimation error.

Theorem 2 (UUB for critic and actor network). Let assumptions C and assumptions A hold with a bounded reinforcement signal. Let gradient descent be used to update the weight of both citric and actor networks as (84) and (28), respectively. The errors between the optimal weights for both networks (ω_c^*, ω_a^*) and their estimates ($\hat{\omega}_c(t)$, $\hat{\omega}_a(t)$) are UUB, if the following conditions are achieved:

$$\begin{aligned} \ell_c < \frac{1}{\gamma^2(1-\lambda)^2\|\phi_c(t)\|^2}, \qquad \ell_a < \frac{1}{\|\phi_a(t)\|^2}, \\ 0 < \gamma \leq 1, \quad 0 \leq \lambda < 1, \quad \alpha > \frac{4}{\gamma^2(1-\lambda)^2} - \frac{1}{2} \end{aligned}. \tag{44}$$

Theorem 2 Proof. The definition of a candidate of the Lyapunov function is given as follows:

$$L(t) = L_c(t) + L_a(t) + L_p(t), \tag{45}$$

where $L_p(t) = \frac{1}{2}\|\xi_c(t-1)\|^2$, and the first difference for it is given as $\triangle L_p(t) = \frac{1}{2}(\|\xi_c(t)\|^2 - \|\xi_c(t-1)\|^2)$. The first difference of the Lyapunov function (60) is given as follows:

$$\begin{aligned}\triangle L(t) \leq &-\beta^2\|\xi_c(t)\|^2 - \beta^2\Big(1 - \ell_c\beta^2\|\phi_c(t)\|^2\Big) \times \|\xi_c(t) + \omega_c^{*^T}\phi_c(t) + \beta^{-1}\gamma\lambda\bar{v}(t) + \\ &\beta^{-1}U(t) - \beta^{-1}\tilde{\omega}_c^T(t-1)\phi_c(t-1)\|^2 + 2\|\beta\omega_c^{*^T}\phi_c(t) + \gamma\lambda\bar{v}(t) + U(t) - \\ &\frac{1}{2}\hat{\omega}_c^T(t-1)\phi_c(t-1) - \frac{1}{2}\omega_c^{*^T}\phi_c(t-1)\|^2 + \frac{1}{2}\|\xi_c(t-1)\|^2 + \frac{1}{\alpha}\Big(- \|\hat{\omega}_c^T(t) \\ &\phi_c(t)\|^2 + \ell_a(t)\|\phi_a(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2 + \|\xi_a(t) \\ &\hat{\omega}_c^T(t)C_{BP}(t)\|^2 + 4\|\xi_c(t)\|^2 + 4\|\omega_c^{*^T}\phi_c(t)\|^2\Big) + \frac{1}{2}\|\xi_c(t)\|^2 - \frac{1}{2}\|\xi_c(t-1)\|^2.\end{aligned} \tag{46}$$

To simplify, we add and subtract the extra term $(\pm\frac{1}{\alpha}\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2)$ into (63) while extracting β. The first difference of $L(t)$ can be rewritten as follows:

$$\begin{aligned}\triangle L(t) \leq &-\Big(\gamma^2(1-\lambda)^2 - \frac{4}{\alpha} - \frac{1}{2}\Big) \times \|\xi_c\|^2 - \gamma^2(1-\lambda)^2 \times \Big(1 - \ell_c\gamma^2(1-\lambda)^2\|\phi_c(t)\|^2\Big) \\ &\times \|\xi_c(t) + \omega_c^{*^T}\phi_c(t) + \frac{1}{\gamma(1-\lambda)} \times \Big(\lambda\bar{v}(t) + U(t) - \tilde{\omega}_c^T(t-1)\phi_c(t-1)\Big)\|^2 \\ &- \frac{1}{\alpha}\Big(\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2 - \ell_a\|\phi_a(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t) \\ &C_{BP}(t)\|^2\Big) + 2\|\gamma(1-\lambda)\omega_c^{*^T}\phi_c(t) + \gamma\lambda\bar{v}(t) + U(t) - \frac{1}{2}\hat{\omega}_c^T(t-1)\phi_c(t-1) - \\ &\frac{1}{2}\omega_c^{*^T}\phi_c(t-1)\|^2 + \frac{1}{\alpha}\|\xi_a(t)\hat{\omega}_c^T(t)C_{BP}(t)\|^2 + \frac{4}{\alpha}\|\omega_c^{*^T}\phi_c(t)\|^2 - \frac{1}{\alpha}\|\hat{\omega}_c^T(t) \\ &\phi_c(t)\|^2 + \frac{1}{\alpha}\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2.\end{aligned} \tag{47}$$

To guarantee that the second term in (47) is negative, the learning rate for the critic network (ℓ_c) has to be $(1 - \ell_c\gamma^2(1-\lambda)^2\|\phi_c(t)\|^2) > 0$; therefore, the critic learning rate should obey the condition:

$$\ell_c < \frac{1}{\gamma^2(1-\lambda)^2\|\phi_c(t)\|^2}, \tag{48}$$

where $0 < \gamma \leq 1, \quad 0 \leq \lambda < 1$.

To guarantee the third term for (47) to be negative, the learning rate for the actor network (ℓ_a) has to be $(\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2 - \ell_a\|\phi_a(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2) > 0$; therefore, the actor learning rate should obey the condition:

$$\ell_a < \frac{1}{\|\phi_a(t)\|^2}. \tag{49}$$

To guarantee that the first term in (47) is negative, the discount factor is chosen with $0 < \gamma \leq 1$, and the weighting factor (α) is selected to satisfy:

$$\alpha > \frac{4}{\gamma^2(1-\lambda)^2} - \frac{1}{2}. \tag{50}$$

$\triangle L(t)$ can be rewritten as follows:

$$\begin{aligned}
\triangle L(t) \leq & -\Big(\gamma^2(1-\lambda)^2 - \frac{4}{\alpha} - \frac{1}{2}\Big)\|\xi_c(t)\|^2 - \gamma^2(1-\lambda)^2\Big(1 - \ell_c\gamma^2(1-\lambda)^2\|\phi_c(t)\|^2\Big) \\
& \times \|\xi_c(t) + \omega_c^{*^T}\phi_c(t) + \frac{1}{\gamma(1-\lambda)}\Big(\lambda\bar{v}(t) + U(t) - \tilde{\omega}_c^T(t-1)\phi_c(t-1)\Big)\|^2 \\
& - \frac{1}{\alpha}\Big(\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2 - \ell_a\|\phi_a(t)\|^2\|\hat{\omega}_c^T(t)\phi_c(t)\|^2 \\
& \|\hat{\omega}_c^T(t)C_{BP}(t)\|^2\Big) - \frac{1}{\alpha}\|\hat{\omega}_c^T(t)\phi_c(t)\|^2 + R^2,
\end{aligned} \tag{51}$$

where R^2 is defined as the positive terms in (47), which is given as follows:

$$\begin{aligned}
R^2 = & \, 2\|\gamma(1-\lambda)\omega_c^{*^T}\phi_c(t) + \gamma\lambda\bar{v}(t) + U(t) - \frac{1}{2}\hat{\omega}_c^T(t-1)\phi_c(t-1) - \frac{1}{2}\omega_c^{*^T}\phi_c(t-1)\|^2 \\
& + \frac{1}{\alpha}\|\xi_a(t)\hat{\omega}_c^T(t)C_{BP}(t)\|^2 + \frac{4}{\alpha}\|\omega_c^{*^T}\phi_c(t)\|^2 + \frac{1}{\alpha}\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2.
\end{aligned} \tag{52}$$

Simplifying, R^2 can be rewritten as:

$$\begin{aligned}R^2 \leq 4\Big(&\gamma^2(1-\lambda)^2\|\omega_c^{*^T}\phi_c(t)\|^2 + \gamma^2\lambda^2\bar{v}^2(t) + U^2(t) + \frac{1}{4}\|\hat{\omega}_c^T(t-1)\phi_c(t-1)\|^2 + \\ &\frac{1}{4}\|\omega_c^{*^T}\phi_c(t-1)\|^2\Big) + \frac{1}{\alpha}\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2\Big(\|\hat{\omega}_a^T(t)\phi_a(t)\|^2 + \|\omega_a^{*^T}\phi_c a(t)\|^2\Big) + \\ &\frac{4}{\alpha}\|\omega_c^{*^T}\phi_c(t)\|^2 + \frac{1}{\alpha}\|\hat{\omega}_c^T(t)\phi_c(t)\|^2\|\hat{\omega}_c^T(t)C_{BP}(t)\|^2,\end{aligned} \tag{53}$$

by substituting the upper bounds for ω_c (ω_c^* and $\hat{\omega}_c$), ω_a (ω_a^* and $\hat{\omega}_a$), ϕ_c, ϕ_a, C_{BP}, and $U(t)$ to ω_{cm}, ω_{am}, ϕ_{cm}, ϕ_{am}, C_m, and U_m, respectively. Because all critic weights are upper-bounded by ω_{cm}, the target-value ($\bar{v}(t)$) is also bounded according to (8), which is denoted as ($\bar{v}_{cm}$); therefore R^2 can be rewritten as follows:

$$\begin{aligned}R^2 \leq \Big(&4\gamma^2(1-\lambda)^2 + 2 + \frac{4}{\alpha}\Big)\omega_{cm}^2\phi_{cm}^2 + \gamma^2\lambda^2\bar{v}_{cm}^2 + U_m^2 + \frac{2}{\alpha}\omega_{cm}^2C_m^2\phi_{am}^2 + \frac{1}{\alpha}\omega_{cm}^4 \\ &C_m^2\phi_{cm}^2 + \frac{4}{\alpha}\omega_{am}^2\phi_{am}^2 + \frac{2}{\alpha}\omega_{cm}^2C_m^2\omega_{am}^2\phi_{am}^2 = R_m^2.\end{aligned} \tag{54}$$

If (57) holds, then any

$$\|\xi_c(t)\| > \frac{R_m}{\sqrt{\gamma^2(1-\lambda)^2 - \frac{1}{2} - \frac{4}{\alpha}}}, \tag{55}$$

where $0 < \gamma \leq 1, \quad 0 \leq \lambda < 1$ as mentioned in (48), and α in (50), makes $\triangle L(t) \leq 0$, meaning that the errors between the optimal weights for both networks (ω_c^*, ω_a^*) and their estimates ($\tilde{\omega}_c(t)$, $\tilde{\omega}_a(t)$) are UUB.

Corollary 1: From (57), the actor learning rate condition for HDP(λ) is similar to HDP, because it is independent of λ and γ. In contrast, the critic learning rate for HDP(λ) depends on λ and γ, which is different from HDP. Thus, to ensure that the critic and actor networks are stable during learning, the λ value should not be 1, and γ should not be 0. As in [26], where $\|\phi_c\|^2 \leq h_c$ and $\|\phi_a\|^2 \leq h_a$, the learning rates for citric and actor networks can be set as

$$\ell_c < \frac{1}{\gamma^2(1-\lambda)^2 h_c} \quad \text{and} \quad \ell_a < \frac{1}{h_a}. \tag{56}$$

4. SIMULATION RESULTS

Three case studies are taken to verify the effectiveness of HDP(λ). The trajectories of the internal reinforcement signal 2-D nonlinear system are considered as the first case. We compare the results with the performance of traditional HDP. The second case study is a single link inverted pendulum. We investigate the performance of the inverted pendulum by comparing the HDP(λ) approach with the HDP approach in different noise exposure effects. The third case study is a 3-D maze navigation benchmark and comparing SARSA(0)[1] and Q(λ) [3] with HDP and HDP(λ).

4.1. Case I: Nonlinear System Problem.

Consider the following nonlinear system derived from [35]:

$$\begin{aligned} x_1(t+1) &= -sin\big(0.5x_2(t)\big) \\ x_2(t+1) &= -cos\big(1.4x_2(t)\big)sin\big(0.9x_1(t)\big) + u(t), \end{aligned} \tag{57}$$

where $x(t) = [x_1(t) \quad x_2(t)]^T \in \mathbb{R}^2$ is the state vector ($m = 2$), and $u(t) \in \mathbb{R}^1$ is the control action ($\text{n} = 1$). The external instantaneous cost function is $U(t) = x^T(t)x(t) + u^T(t)u(t)$. The discount factor ($\gamma$) is 0.9, and the eligibility trace long-term prediction parameter (λ) is 0.95. The number of hidden nodes (h_c) is 7 for the critic network. The number of hidden nodes in the actor network (h_a) is 5; and the initial learning parameters are set as $\ell_c = \ell_a = 0.01$ for both the critic and actor networks. The training for either network will be terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold for the internal cycle (30 iterations for the critic network and 40 iterations for actor network). The initial weights for all networks are the same for fair comparison, which are randomly chosen within $[-0.3, 0.3]$. As mentioned before, we fix the hidden weights ($\hat{w}_c^{\{h\}}, \hat{w}_a^{\{h\}}$) and train the output weights ($\hat{w}_c^{\{o\}}, \hat{w}_a^{\{o\}}$). We compare HDP($\lambda$) and the traditional HDP approaches with similar learning parameters. We choose the initial critic/actor weights in

[1]The SARSA (State-Action-Reward-State-Action) algorithm is used to find the series state-action pairs by using one-step learning; therefore, we write it as SARSA(0).

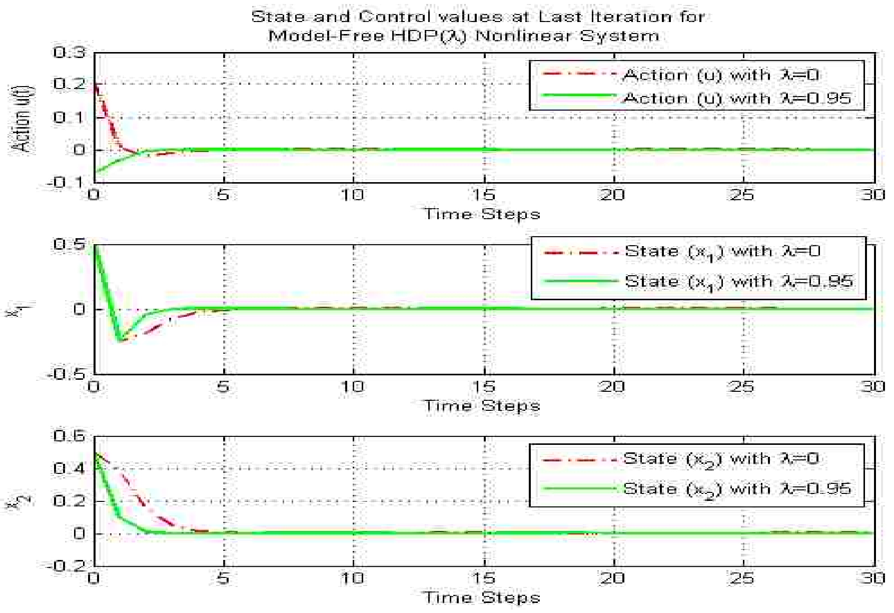


Figure 4. Comparisons of system responses (the state vector trajectories and the action sequence) with HDP(λ = 0.95), and traditional HDP(λ = 0).

the HDP(λ) approach similar to the initial critic/actor weights for the HDP approach. Fig. 6 illustrates the state trajectories and the control action sequence for 30 time steps. The results are taken from the last training iteration (iteration number 2000). We set the initial state to $x(0) = [0.5 \quad 0.5]^T$. HDP(λ) can drive the states to converge faster than traditional HDP. Compared to HDP, the improvement according to a mean-square error technique is 29.54%. The upper part of Fig. 8 shows the cost error over iterations, which clearly illustrates faster learning. HDP(λ) has smooth learning compared to the fluctuations appearing with HDP. The critic and actor errors at the last learning iteration for the HDP and HDP(λ) approaches are shown at the middle and lower figures in Fig. 8. In the last iteration, Fig. 10 illustrates the corresponding weight trajectories for both HDP and HDP(λ) approaches, which shows that the weights reached their optimal fixed values.

4.2. Case II: Inverted Pendulum. In this section, a cart-pole system has been implemented as an unstable nonlinear system. The controller (actor network) is self-learning because of no prior knowledge about the environment. The controller balances a

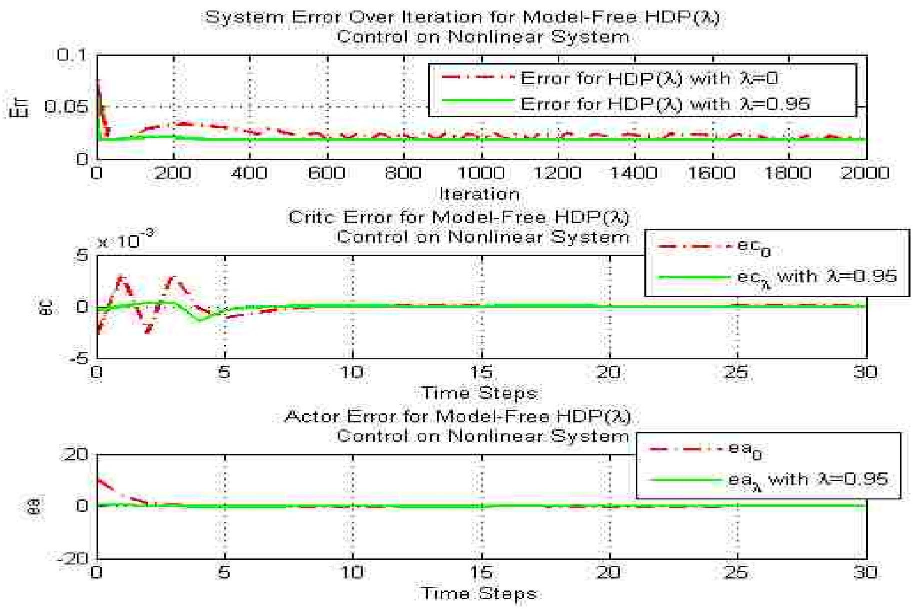


Figure 5. Cost error over iteration and 30 time step critic and actor errors. The upper figure shows the cost error over iterations, which clearly illustrates fast and stable learning, while traditional HDP shows fluctuations during learning. The critic and actor errors at the last learning iteration for HDP and HDP(λ) are shown in the middle and lower figures.

pole mounted on a cart by moving the cart to the left or right. The actor network learns via a reinforcement signal, which is either "-1" or "0" corresponding to a fallen or balanced pole, respectively. As in [5], the cart pole system model as shown in Fig. 11 is given by:

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\Big(-F - ml\dot{\theta}^2\sin\theta + \mu_c\sigma(\dot{x})\Big) - \dfrac{\mu_p\dot{\theta}}{ml}}{l\Big(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c + m}\Big)}, \tag{58}$$

$$\ddot{x} = \frac{F + ml\Big(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\Big) - \mu_c\sigma(\dot{x})}{m_c + m}, \tag{59}$$

where $g = 9.8[m/s^2]$, the acceleration due to gravity; $m_c = 1.0[kg]$, the mass of the cart; $m = 0.1[kg]$, the mass of the pole; $l = 0.5[m]$ the half-pole length; $\mu_c = 0.0005$, the coefficient of friction of the cart on the track; $\mu_p = 0.000002$, the coefficient of friction of the pole on the cart; $F = \pm 10[N]$ the force applied to the cart's center of mass; and $\sigma(.)$

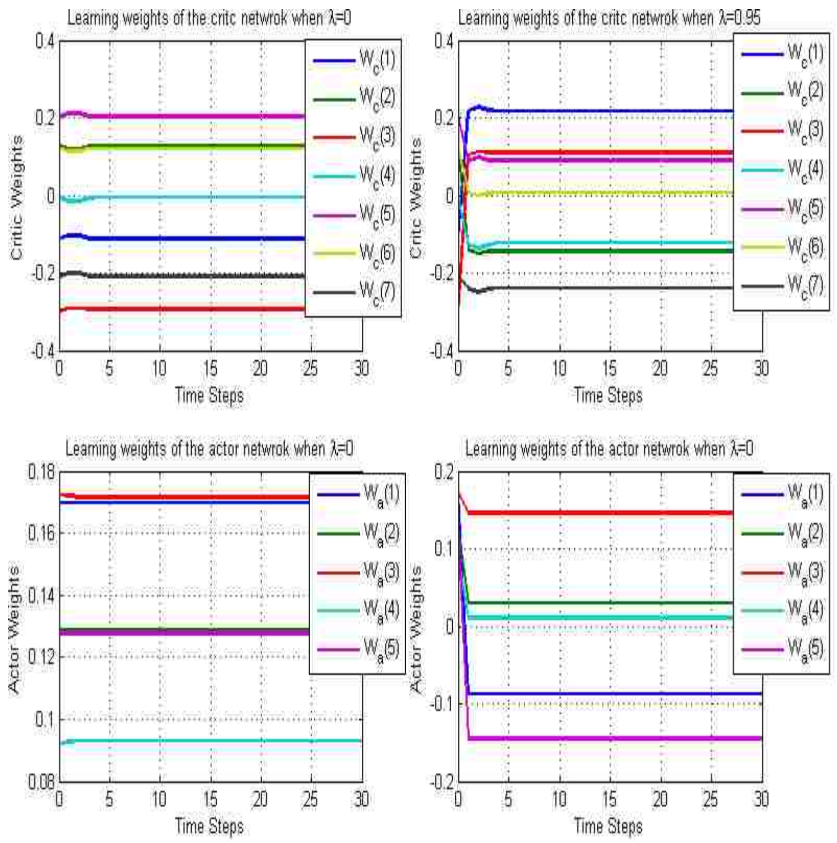


Figure 6. Learning weights for critic and actor networks for HDP and HDP(λ).

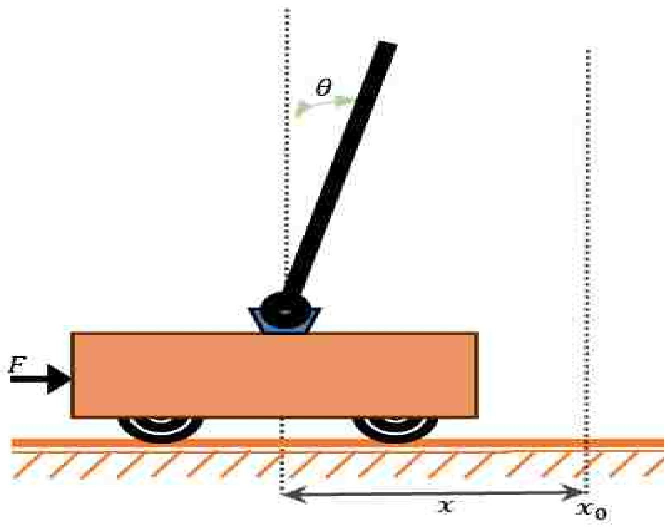


Figure 7. Configuration model of the cart-pole balancing system.

is a sigmoid function. The fourth-order Runge-Kutta method is used to solve nonlinear differential equations (73) and (74) with 0.02 s for the sample step. The pole-cart model has four states: $\theta(t)$ is the angle of the pole with respect to the vertical axis, $x(t)$ is the position of the cart, $\dot{\theta}(t)$ is the angular velocity of the pole, and $\dot{x}(t)$ is the linear velocity for the cart. In our simulation, a run has 60 consecutive trials. The run is considered successful if the last trial has balanced the pole. The first trial starts with $\lambda = 0$, while the remaining trials are set with $\lambda = 0.95$. Each successful trial has 1000 time steps to complete the balancing task. The pole has fallen if it is beyond the range of $[-12^\circ, 12^\circ]$, and also if the cart moves outside the range $[-2.4, 2.4]$ meter from the initial position. In spite of the binary force F applied to the cart, the control signal ($u(t)$) provided to the critic network is continuous. To stabilize this system, assign the critic and actor neural network parameters to match the conditions of theorem 2. The discount factor (γ) is 0.95; the number of hidden nodes (h_c) is 20 for the critic network; the number of hidden nodes in the actor network (h_a) is 24; and the initial learning parameters are set as ($\ell_c = 0.001$) for the critic network and ($\ell_a = 0.003$) for the action network. Both learning rates are decreased via dividing by 3 every 30 time steps. The stopping criteria for the action and critic networks are 120 and 100, respectively; the training for either network will be terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold. We apply HDP(λ) without any noise impact with small deviation in the initial pole-angle. In this test, we set the initial states at $\theta = 0.1^\circ$, $x = 0$ $[m]$, $\dot{\theta} = 0.5$ $[deg/s]$, and $\dot{x} = 0$ $[m/s]$. Fig. 8 shows the value function and the target-value for the last trial without noise. The output of the critic network in HDP(λ) follows the target-value (8) as shown in the enlarged Fig. 9. From top to bottom, Fig. 10 illustrates the forces applied to the center of the cart, the cart position, and the angle trajectory for the cart pole. In the noise-free system, the angle oscillates within limits $\pm 0.3495^\circ$ as shown in Fig. 11. For more challenging and realistic behavior, we have started with an initial deviation pole-angle, and we add random noise to the angle state measurement and action network output sensor noise and actuator noise, respectively.

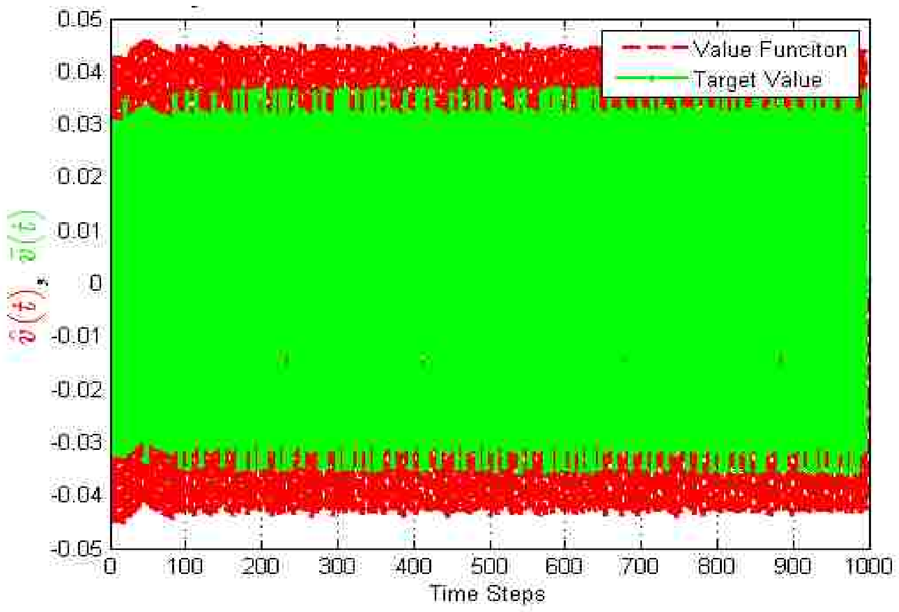


Figure 8. The value function and target value for the last trial without noise. The initial angle $\theta(t)$ is 0.1°.

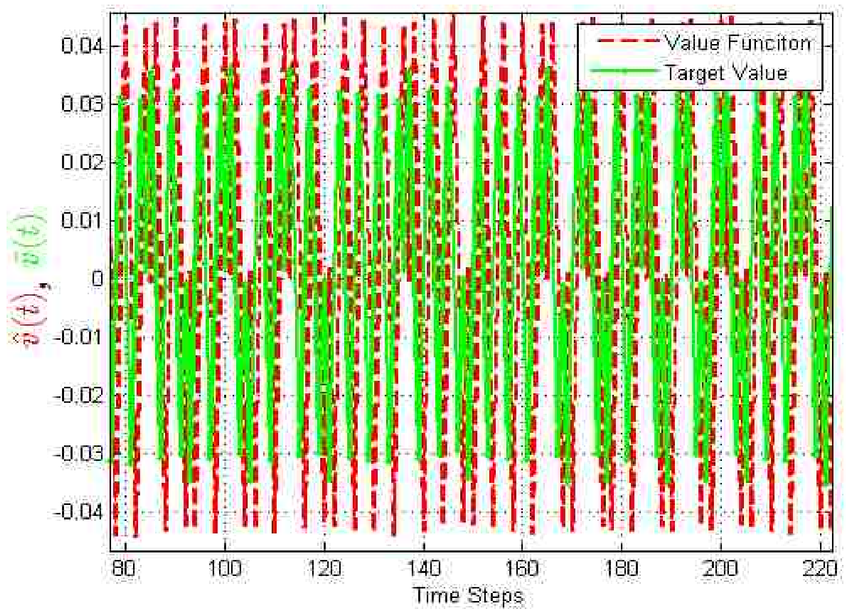


Figure 9. Zoom-in between 80 to 220 time steps for Fig. (8).

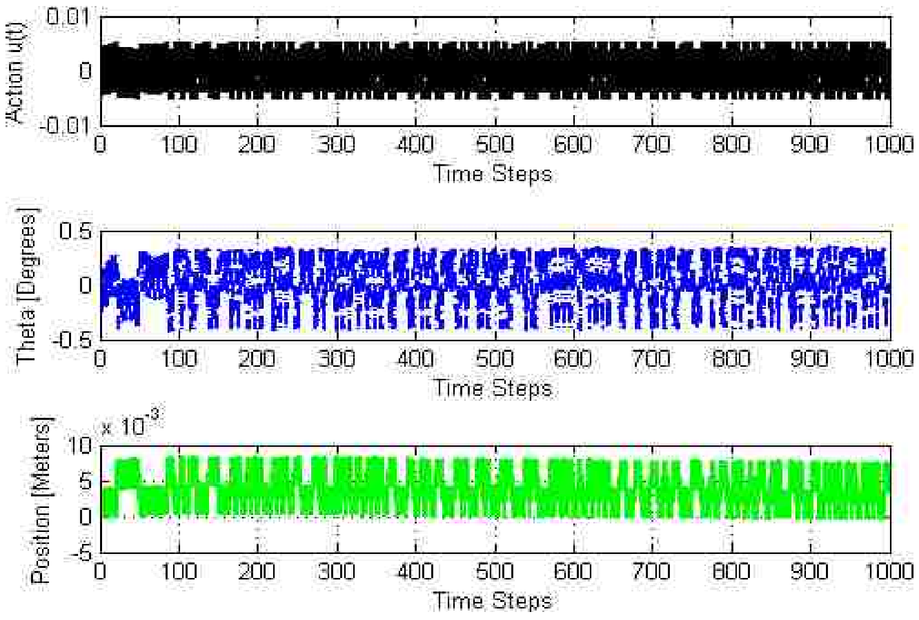


Figure 10. Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system is free of noise; initial angle $\theta(t)$ is 0.1°.

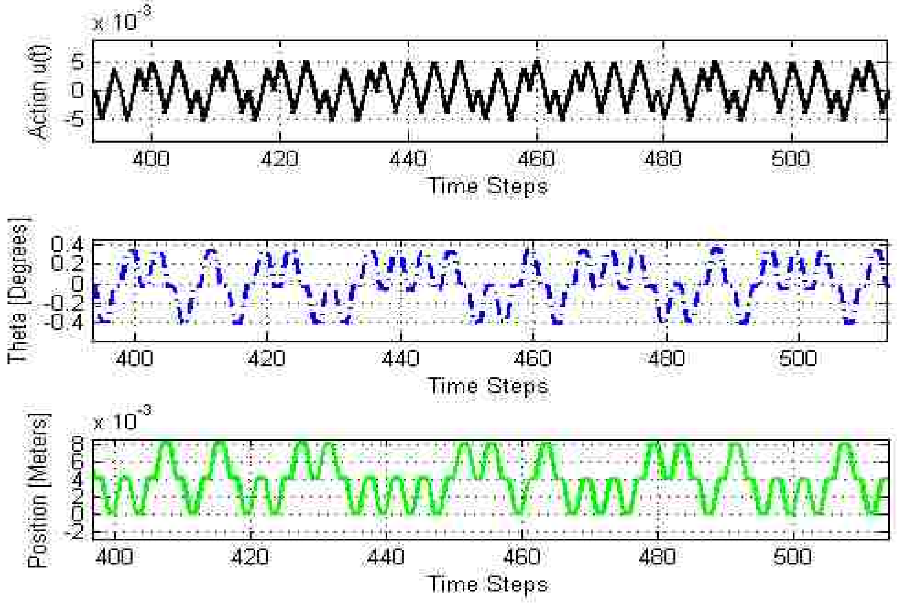


Figure 11. Zoom-in between 400 to 520 time steps for Fig. (10).

Specifically, the sensor and actuator noises are $\theta(t) = \theta(t) + \eta$ and $u(t) = u(t) + \eta$, where η is a uniformly distributed random variable. We assume that the initial angle has a disturbance of $\theta = 10°$ with respect to the vertical axis, with no change in the initial values. Fig. 12 show the value function and the target-value. The output of the critic network follows the target-value as shown in Fig. 13, and it settles down after 450 iterations. The corresponding system responses with forcing signal are shown in Fig. 14. Fig. 15 demonstrates how the actor network in HDP(λ) overcomes this large initial angle disruption during 15 iterations, converging in 250 time steps.

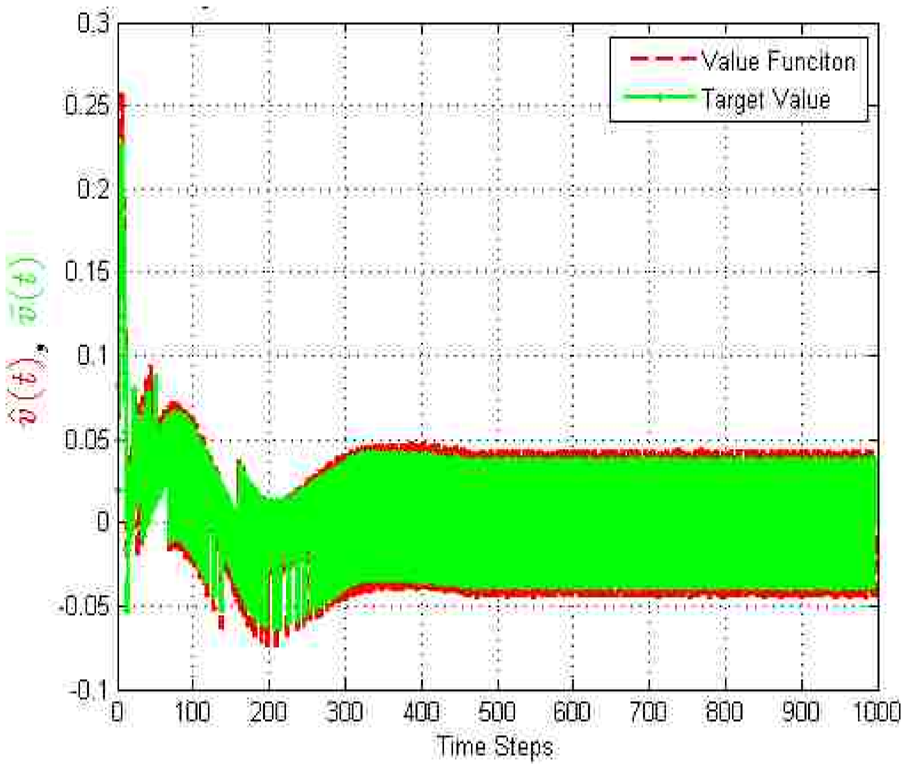


Figure 12. The value function and target value for the last trial when the system is free of noise; initial angle $\theta(t)$ is 10°.

We examine the HDP(λ) structure with another large disruption by adding 3% uniform random sensor noise during 1000 time steps with $\theta = 10°$ as the initial angle derivation. Figs. 16 and 17 demonstrate how the system also successfully passes this challenge by showing the cost values and system responses, respectively. $\pm 0.9536°$ is the

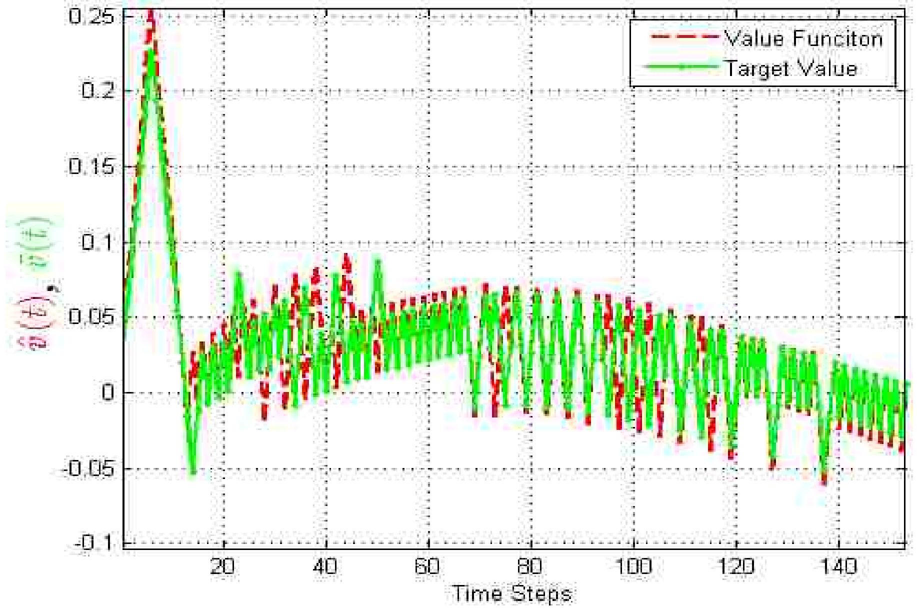


Figure 13. Zoom-in between 0 to 150 time steps for Fig. (12).

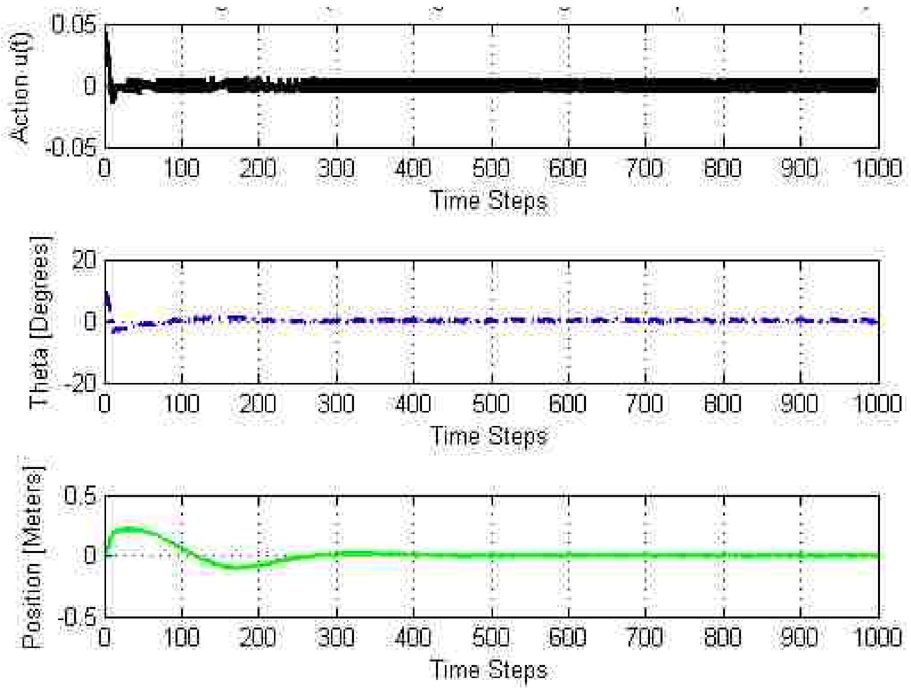


Figure 14. Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system is free of noise; initial angle $\theta(t)$ is 10°.

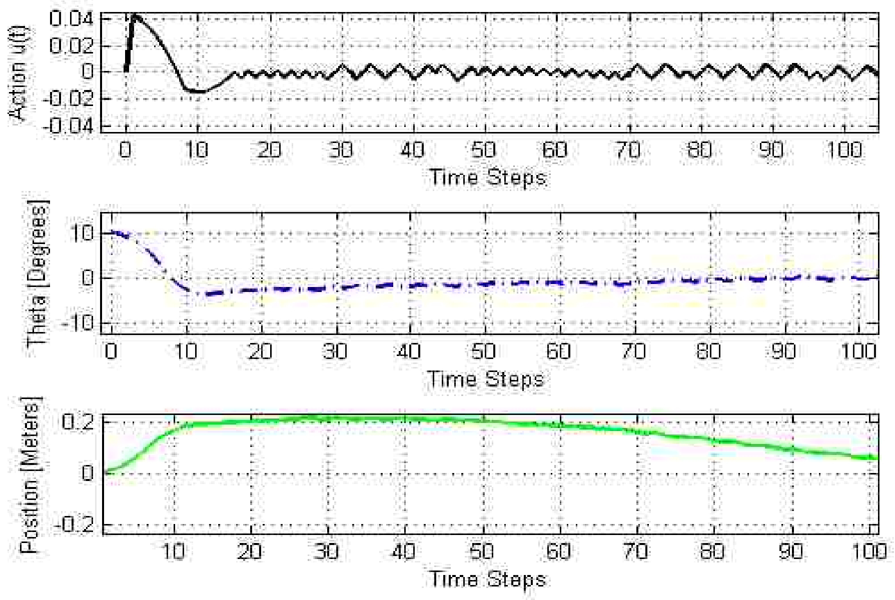


Figure 15. Zoom-in between 0 to 100 time steps for for Fig. 14.

limited oscillating angle after step number 280. For more a challenging test, we set a 5% uniformly random actuator noise to occur during 1000 time steps with $\theta = 10°$ initial angle derivation. The system successfully passes this test as shown in Figs. 18 and 19 for the cost values and system responses, respectively. The limited oscillating angle is bounded by $\pm 4.0215°$ after step number 200. For more detailed viewing of actual and target value functions, Fig. 20 illustrates a squared critic error (E_c) for all previous testing scenarios. Table I summarizes the simulation results through 100 averaged runs for 1000 time steps and 100 iterations. HDP(λ) is 13.7% better than traditional HDP, reducing the average number of iterations at various noise levels.

4.3. Case III: 3-D Maze Problem. Maze navigation has been proposed as an ADP benchmark [37]-[40], but most of them have been 2-D. In this case, we compare HDP(λ) in the 3-D maze navigation benchmark with various alorithms: SARSA(0), Q(λ) and HDP. Sutton [3] presents an explanation about SARSA(0) and Q(λ). In this case study, the data that agent uses to learn is: 1) current state vector $x(t) = [x_1, x_2, x_3]^T$, where x_1, x_2

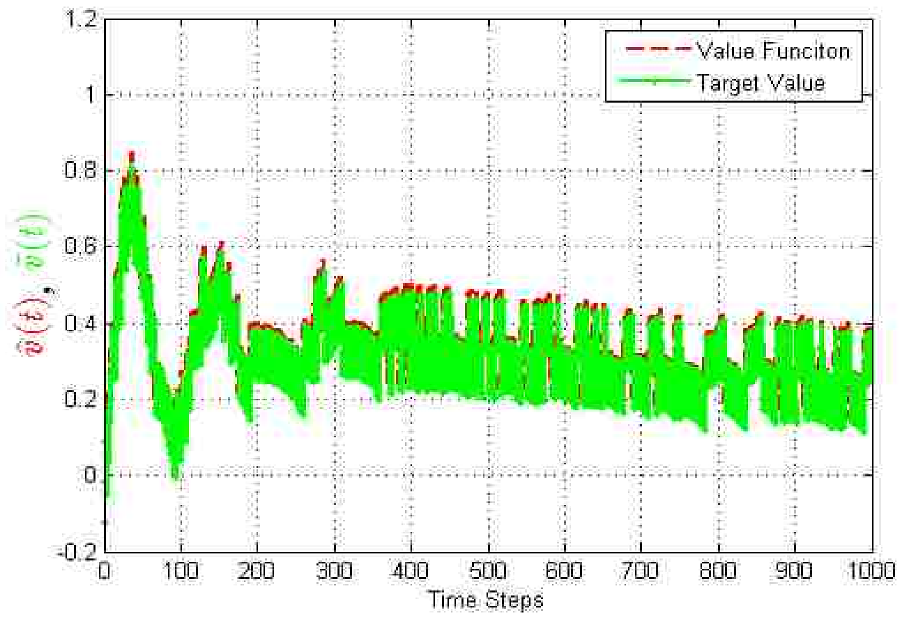


Figure 16. Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system has uniform 3% sensor noise; initial angle $\theta(t)$ is 10°.

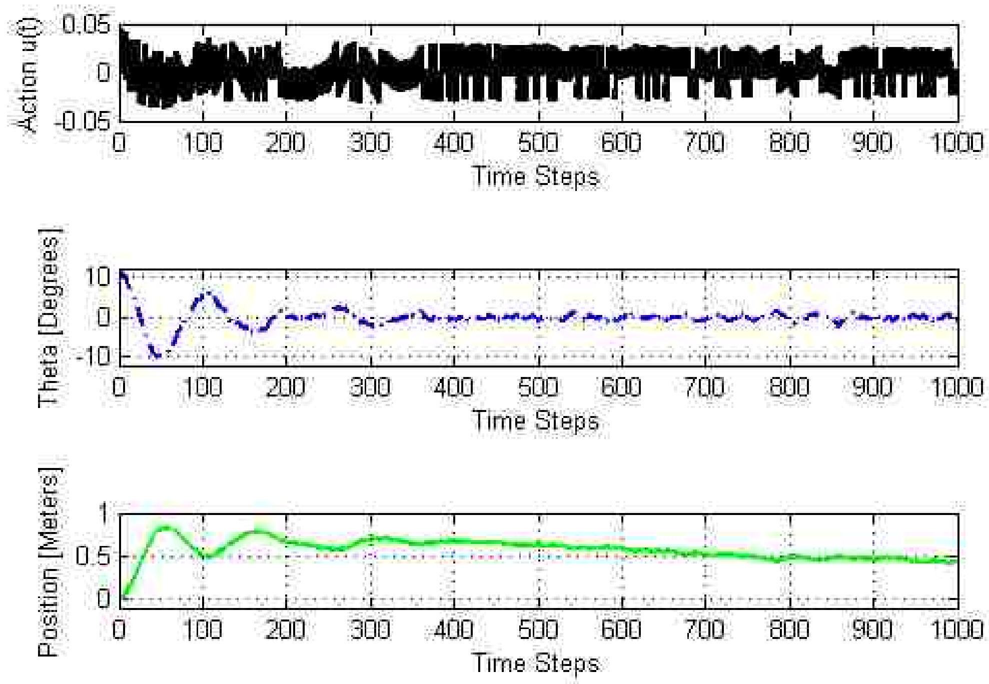


Figure 17. Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system has uniform 3% sensor noise; initial angle $\theta(t)$ is 10°.

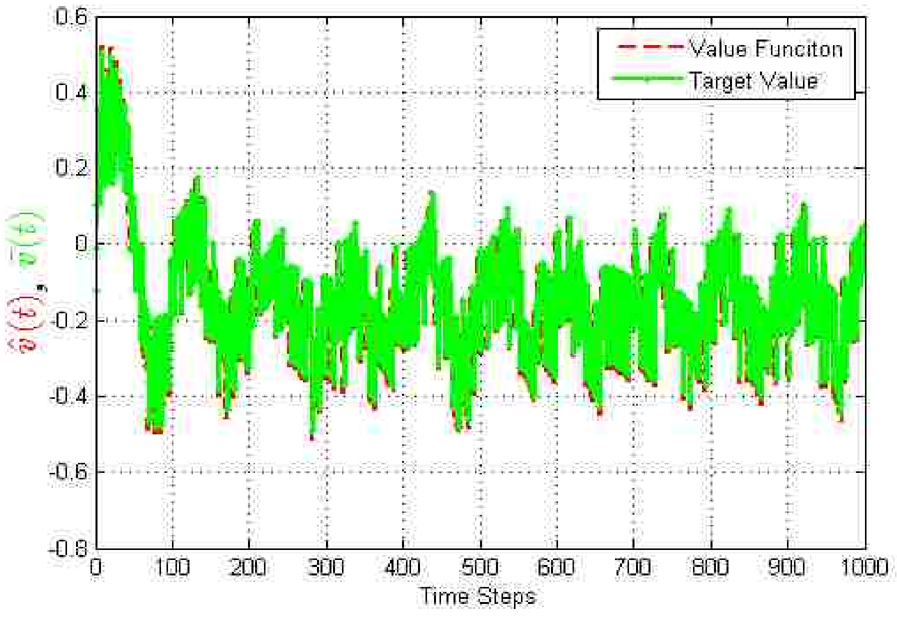


Figure 18. The value function and target value for the last trial when the system has uniform 5% actuator noise; initial angle $\theta(t)$ is 10°.

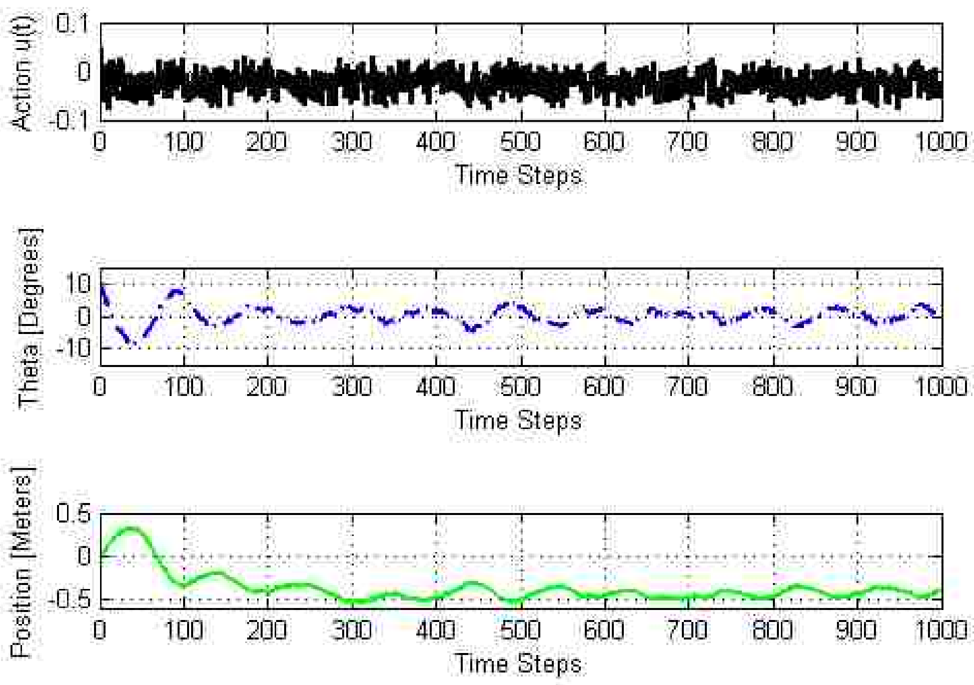


Figure 19. Simulated results of balancing the inverted pendulum for control signal, $\theta(t)$, and $x(t)$ when the system has uniform 5% actuator noise; initial angle $\theta(t)$ is 10°.

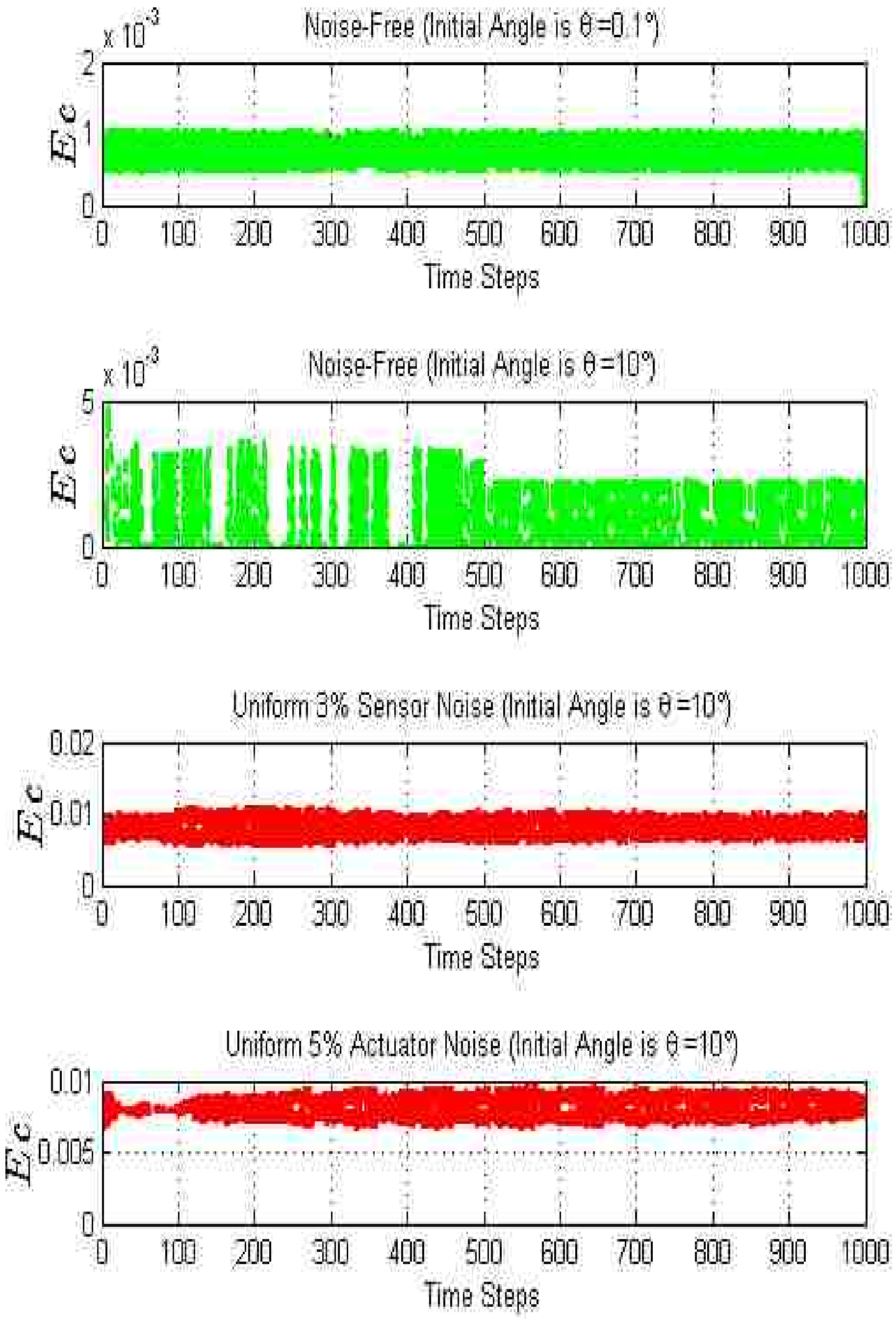


Figure 20. Squared critic error (E_c) for noise-free with 0.1° to initial angle, noise-free with 10° to initial angle, uniform 3% sensor noise, and uniform 5% actuator noise.

Table 1. Performance evaluation of HDP(λ) learning controller when balancing the inverted pendulum dynamic system. The second and third columns depict the average number of trials it took to learn to balance the pole for 1000 time steps for HDP and HDP(λ) approaches, respectively. The average is based on 100 successful runs at 1000 iterations each. * actuators are subjected to noise; $^\#$ sensors are subjected to noise

Noise Type	**HDP(λ)**	**HDP**
Uniform* 5%	13.85	**13.36**
Uniform* 10%	**13.93**	15.63
Uniform$^\#$ 5%	**13.77**	14.1
Uniform$^\#$ 10%	**57.38**	63.86
Uniform$^{*\#}$ 5%	**19.35**	21.28
Noise Free with 8° initial angle	**27.79**	40.48

and x_3 are the coordinate of x axis, y axis and z axis, respectively; 2) selected action $u(t) = [u_1, u_2, u_3, u_4, u_5, u_6]^T$, where u_1, u_2, u_3, u_4, u_5 and u_6 are the direction of forward, right, backward, left, up and down, respectively; 3) external reward $U(t) = 1$ if agent reaches the target position, $U(t) = -0.001$ if agent hits an obstacle or exceeds the board, and $U(t) = 0$ if agent moves in free space. In the SARSA(0) algorithm, the agent takes an action u_i, where $i = 1, 2, \ldots, 6$, to move from $x(t)$ to next state $(x(t))$, and it gains $U(t)$. There are many strategies to select actions. If the agent always chooses the action with the highest state-action pair value, it is a greedy strategy. An $\epsilon-$greedy strategy selects the greedy action with probability of (1-ϵ). Otherwise, the agent chooses a random action. In this situation, one says that the agent is exploring the environment. for other strategies, see [3], [13] and [36]. After learning trials are completed, the collected data is put in a lookup-table (Q table). The Q table values of all state-action pairs can be updated in the SARSA(0) algorithm by the Bellman formula:

$$Q(x(t), u_i(t)) = Q(x(t), u_i) + \ell\Big(U(t) + \gamma Q(x(t+1), u_i) - Q(x(t), u_i)\Big), \tag{60}$$

where $x(t)$ is state vector, ℓ is the learning rate and u_i, $i = 1, 2, \ldots, 6$, is a selected action (the direction of movement). Q(λ) is similar to SARSA except for two issues 1) the updating occurs as in (60) but with a greedy action $\max_{u_i \in u(t)} \big(Q(x(t+1), u_i\big)$ instead of $Q\big(x(t+1), u_i\big)$; 2) the eligibility trace is a temporary record to be stored. There are two main steps to update eligibility traces for the Q table. The first step is setting all state action pairs to zero when a non-greedy action is taken. Otherwise, they are declining by $\gamma\lambda$ as described in (8). In the second step, the eligibility trace is reset to one if it is identical to the current state-action pair. Updating Q-learning by using only a critic network was presented in [41]. We also use only a critic neural network to approximate the value function but for n-steps of the eligibility trace. Algorithm 1 generates and updates a table of Q values according to state-action pairs (step 2 in Algorithm 1) as well as generating greedy actions ($u_i(t)$) in step 3 of Algorithm 1. The same procedure is used to generate and update a Q table for the HDP by taking one critic network. In contrast with the previous two case studies that required minimizing the cost value, the agent has to maximize a reward in order to solve the maze problem. We assume that an agent starts at an initial location in the cube environment, which is (0,0,0) as shown in Fig. 20 with 12 obstacles included. The agent can learn online through interacting with its environment to obtain an optimal collision-free path from the start point to the target point, which is taken as (5,5,5). A declining $\epsilon-$greedy learning method is used in all approaches, which is used to balance between exploration and exploitation [3], [13]. We evaluate HDP(λ) by comparing with other methods according to the Q reference values. the Q reference evaluation method is presented in [43], [44]. The Q reference table is calculated depending on the distance between the current state location and target location. All states around a target are set to 1, while other states are assigned by finding $1/(L + W + D)$ for each step, where L, W and D is the maximum number of possible state in length, width and depth directions, respectively. In other words, Q references (Q_{ref}) can be calculated in 3-D

Algorithm 1 Critic(λ)-Only Q-Learning for Maze Navigation

```
$\hat{v}(t) \longleftarrow f_c(x_c(t), \hat{\omega}_c(t))$, value function approximation
$f_c$: the critic network
$x_c(t) = [x(t), u_i]^T$: input of critic network, where $i = 1, 2, \ldots, 6$
$\hat{\omega}_c(t)$: weights in critic network
  • Step 1:
      – $Q(x, u_i) = 0$ for all states and actions
      – $\bar{v}(x, u_i) = 0$ for all states and actions
      – $t = 0, T_c = 10^{-5}$ with $N_c = 110$ and $C_c = 0$ are stopping learning critic network
        parameters, $goal = [5, 5, 5]$, and set all other learning parameters
  • Step 2 (**Policy evaluation**) updating the Q-table by:
      – $\bar{v}(x(t), u_i) = U(t) + \gamma\Big(\lambda\bar{v}(x(t+1), u_i) + (1-\lambda)\hat{v}(t+1)\Big)$
      – **while** $(E_c(t) > T_c)$**or**$(C_c < N_c)$ **do**
      –     $\hat{\omega}_c(t) = \hat{\omega}_c(t) + \triangle\hat{\omega}_c(t)$; equations (71) - (84)
      –     $\hat{v}(t) = f_c(x_c(t), \hat{\omega}_c(t))$
      –     $D_{err}(t) = \bar{v}(x(t), u_i) - \hat{v}(t)$
      –     $E_c(t) = 0.5\Big(D_{err}(t)\Big)^2$
      –     $C_c = C_c + 1$
      – **end while**
      – $Q(x(t), u_i) = Q(x(t), u_i) + \ell D_{err}(t)$ ($\ell$ is a leaning rate)
  • Step 3 (**Policy improvement**) updating content policy while taking the $\epsilon$-learning
    strategy into a consideration:
      – $u_i = argmax_{u_i \in u(t)}\big(Q(x(t), u_i)\big)$
  • Step 4:
      –     **if** $x(t) = goal$ **then**
      –         stop and do other episode
      –     **else**
      –         $x(t) = x(t+1)$
      –         $t = t + 1$
      –         go back to step 2 and continue
      –     **end if**
```

as follows:

$$Q_{ref}(x_1, x_2, x_3) = 1 - \frac{1}{L+W+D}\Big(L - x_1 + W - x_2 + D - x_3 - 1\Big). \tag{61}$$

Equation (61) is calculated for the desired values of all cubes of the 3-D maze including the obstacle cubes. But obstacles are used in our 3-D maze benchmark (Fig. 20); therefore, we assign obstacle cubes as zero in Q_{ref} because the agent cannot enter to them to update Q_{ref}. Greedy Q table values (Q_{greedy}) are used to calculate mean-square-error (MSE), where $Q_{greedy}\big((x(t)\big) = \max_{u_i \in u(t)}\big(Q(x(t), u_i\big)$. MSE is obtained by

$$MSE = \frac{1}{2}\sum_{i=1}^{S_n}\Big(Q_{greedy}(i) - Q_{ref}(i)\Big)^2, \tag{62}$$

where S_n is the number of states of the 3-D maze ($L \times W \times D$). The common general parameters shared with all approaches are: learning rate for the Q table (ℓ) is 0.01; γ=0.95; λ=0.95 for Q(λ) and HDP(λ), and λ=0 for SARSA(0) and HDP(0); ϵ-greedy parameter starts at 1 and decreases by $\epsilon = \epsilon * 0.992$ after each episode, stopping at ϵ =0.05; we take a 20-run (loop,) and each run has 300 consecutive episodes (episode loops); moreover, at each episode the agent navigates in the maze until reaching the target. Certain learning parameters are related only to the HDP(0) and HDP(λ) approaches. Thus, the number of neurons in the critic network (h_c) is 24; the initial learning parameters are set at ($\ell_c = 0.001$) for the critic network. The stopping criteria for the critic networks is 110; the training for either network will be terminated if the error drops under $T_c = 10^{-5}$ or if the number of iterations meets the stopping threshold ($N_c = 110$). Fig. 21 demonstrates that the MSE of the HDP(0) and HDP(λ) approaches drop faster than those with SARSA(0), and Q(λ) methods. HDP(λ) can also converge faster than HDP(0) to achieve the best performance in

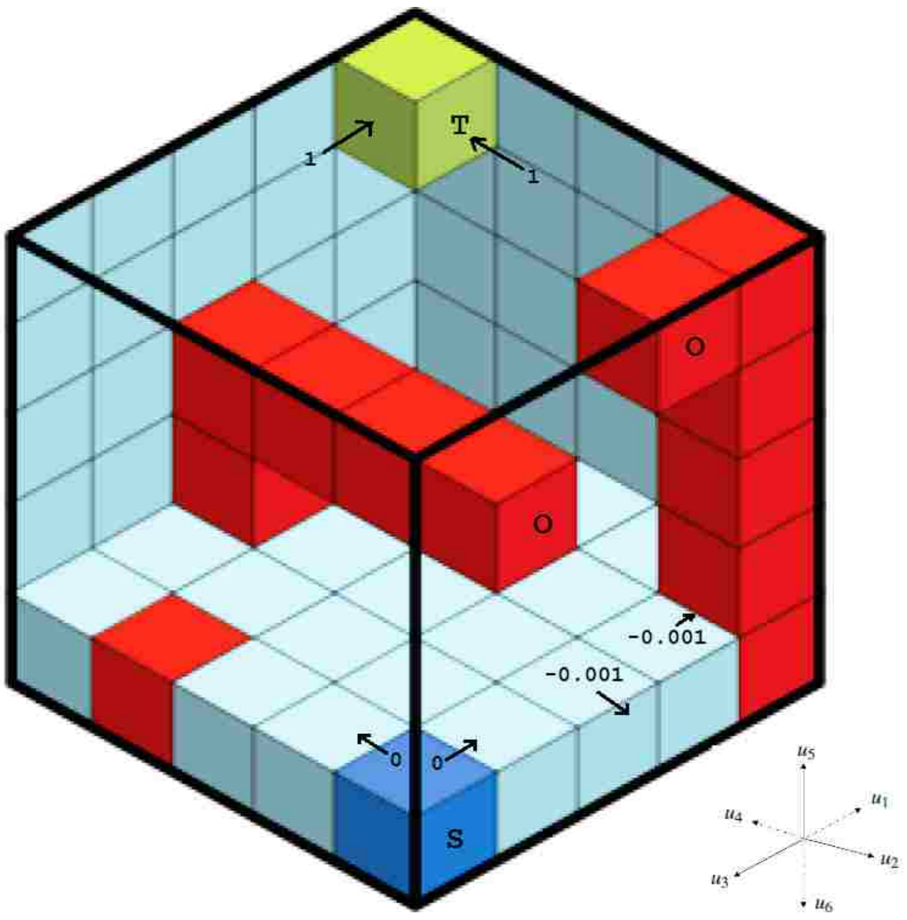


Figure 21. Diagram of 3-D maze ($5 \times 5 \times 5$) with obstacles. The dark blue cube $(0, 0, 0)$ represents the initial position. The green cube $(4, 4, 4)$ represents the target position. 12 obstacles are located in $(0, 3, 0)$, $(2, 4, 1)$, $(2, 4, 2)$, $(2, 3, 2)$, $(2, 2, 2)$, $(2, 1, 2)$, $(4, 0, 0 - 4)$ and $(3, 0, 4)$, which are represented by the red cubes. Otherwise, the agent can move in free space. Three modes allow the agent to receive reward/cost values. First, The agent will receive reward 1 when it arrives to the target cube. Second, the agent will be punished by receiving -0.001 if it hits obstacles or passes the boundary. Third, the agent will receive 0 value as a reward in a free space. At any position in the maze, the agent has to select 1 action (direction) out of six actions in order to to move one step. The six actions are forward, right, backward, left, up and down, which can be seen in the figure as u_1, u_2, u_3, u_4, u_5 and u_6, respectively. We sketch this figure by using the isometric drawing tool [42].

this test. Another test is related to the accumulative reward over episodes. The goal for this test is to discover, which algorithm can calculate a maximum of accumulated rewards over each episode fastest.

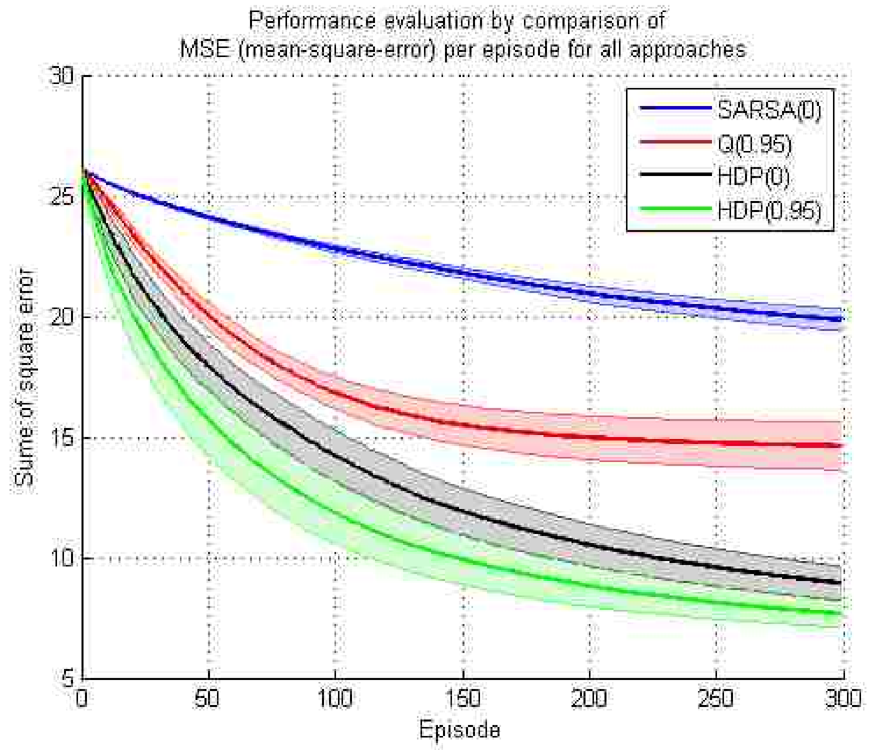


Figure 22. Mean-squared-error (MSE) learning curves for SARSA(0), Q(λ = 0.95), HDP(0) and HDP(λ = 0.95) for the 3-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The HDP(0.95) approach has the fastest learning with a lower MSE compared to other approaches.

Fig. 22 illustrates that HDP(λ) approach has a large accumulated reward value during exploration. Determination of the most likely exploration episodes is mostly done between episode number 0 until 200, which is shown in detail in Fig. 23. Fig. 23 shows the number of steps per episode over reducing the probability of exploration (via decreasing of ϵ value). Therefore, the Q(λ), HDP(0) and HDP(λ) approaches converge together to the optimal accumulative reward value after episode number 200 because of exploitation navigation behavior, while SARSA(0) method needs more episodes to reach the optimal.

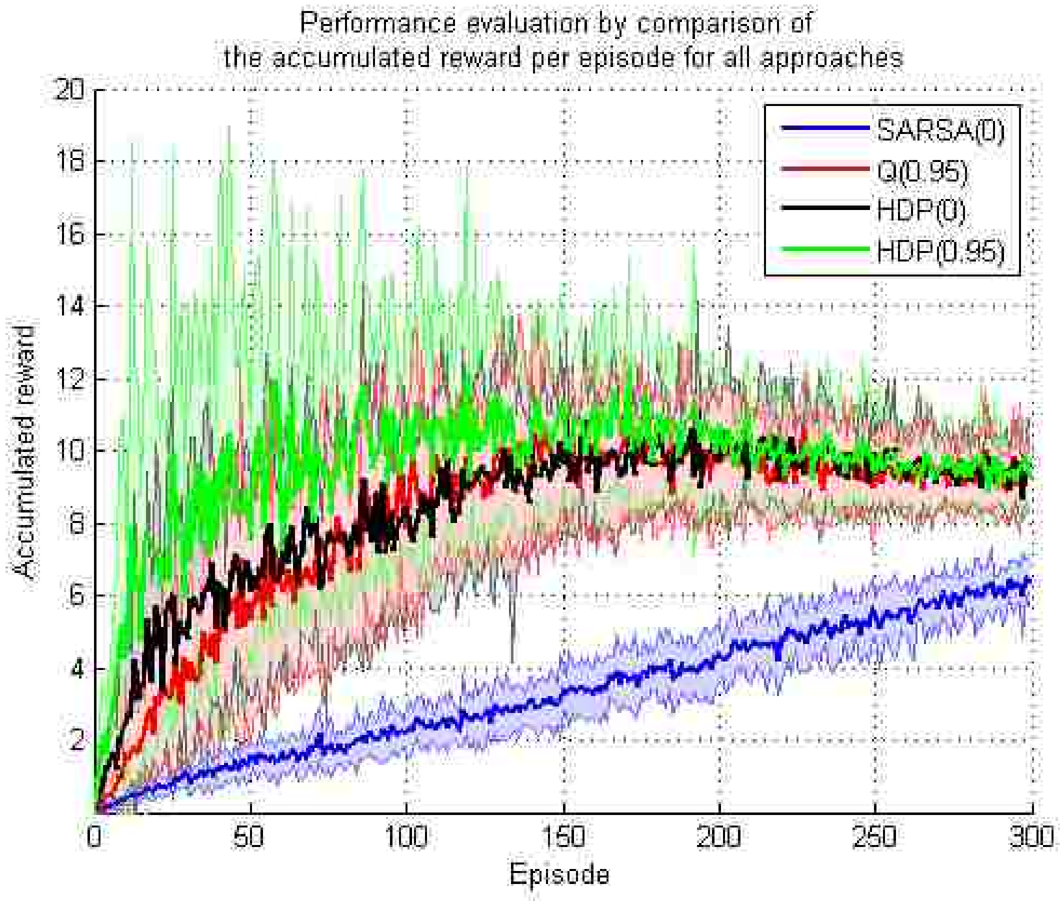


Figure 23. Summation of accumulative reward for every single episode of SARSA(0), Q(λ = 0.95), HDP(0) and HDP(λ = 0.95) approaches, which is applied in the 3-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The HDP(0.95) approach reaches the largest accumulative reward compared to other methods. Because $\epsilon-$greedy learning will reset the accumulative reward value every episode, the accumulative reward values for Q(0.95), HDP(0) and HDP(0.95) converge over episodes.

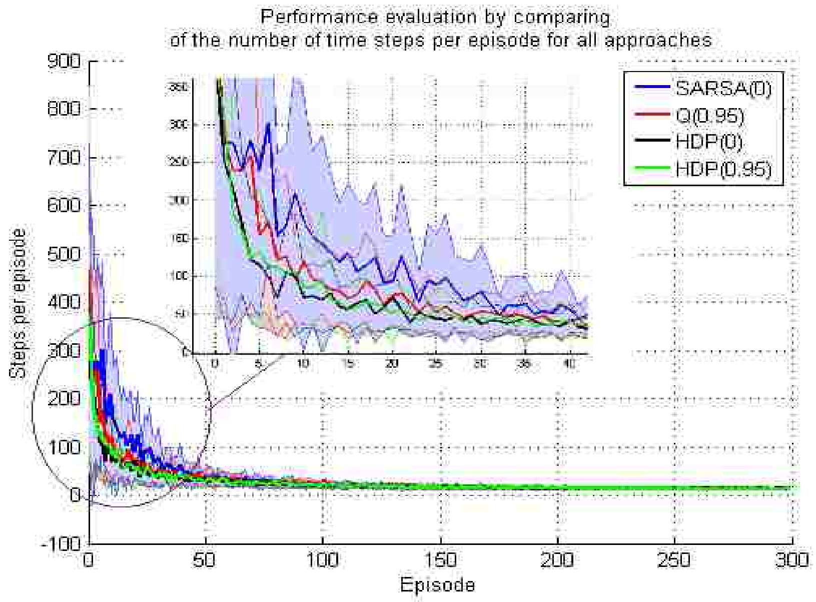


Figure 24. $\epsilon-$greedy learning curves for SARSA(0), Q(λ = 0.95), HDP(0) and HDP(λ = 0.95) approaches for 3-D maze navigation benchmark as shown in Fig. 20. These curves represent the number of steps per episode, where the agent returns back to the start cube only when it reaches the target cube. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. HDP(0.95) and HDP(0) have an almost identical number of steps over episodes, which are less than those in both Q(0.95) and SARSA(0) methods.

5. CONCLUSION

This work shows stability proofs for model-free HDP with arbitrary values of the eligibility trace long-term prediction parameter (λ). Previous works on HDP only apply when $\lambda = 0$. By using Lyapunov theory under reasonable conditions, we extend the stability proof for the HDP(λ) approach, proving that the weights and network outputs for both critic and actor networks are UUB. We examine the HDP(λ) approach with three simulation studies. With these results, we have made a step forward to improve the learning efficiency, computational complexity and robustness performance for ADP algorithms using eligibility traces.

BIBLIOGRAPHY

[1] R. Bellman, *Dynamic Programming*, Princeton, NJ, USA: Princeton Univ. Press, 1957.

[2] D. Prokhorov, and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 8, no. 5, pp. 997-1007, Sep. 1997.

[3] R. S. Sutton, and A. Barto, *Reinforcement Learning: An Introduction*, Cambridge, U.K.: Cambridge Univ. MIT Press, Mar. 1998.

[4] P. J. Werbos, *Approximate dynamic programming for real-time control and neural modeling*, Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches, 1992.

[5] J. Si, and Y. Wang, "Online learning control by association and reinforcement," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 12, no. 2, pp. 264-276, Mar. 2001.

[6] Z. Ni, H. He, X. Zhong, and D. Prokhorov, "Model-Free dual heuristic dynamic programming," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 26, no. 8, pp. 1834-1839, Aug. 2015.

[7] H. He, Z. Ni, and J. Fu, "A three-network architecture for on-line learning and optimization based on adaptive dynamic programming," *Neurocomputing*, vol. 78, no. 1, pp. 3-13, Feb. 2012.

[8] X. Fanga, D. Zhenga, H. He, and Z. Nib, "Data-driven heuristic dynamic programming with virtual reality," *Neurocomputing*, vol. 166, pp. 244-255, Oct. 2015.

[9] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, "GrDHP: A general utility function representation for dual heuristic dynamic programming," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 26, no. 3, pp 614-626, Mar. 2015.

[10] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Dual heuristic programming excitation neurocontrol for generators in a multimachine power system," *IEEE Trans. Applications Industry*, vol. 39, no. 2, pp. 382-394, Mar. 2003.

[11] N. Zhang, and D. C. Wunsch, "A Comparison of Dual Heuristic Programming (DHP) and neural network based stochastic optimization approach on collective robotic search problem," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 1, pp. 248-253, Jul. 2003.

[12] C. Lian, and X. Xu, "Motion planning of wheeled mobile robots based on heuristic dynamic programming," *IEEE Proc. World Congress Intelligent Control and Automation (WCICA)*, pp 576-580, Jul. 2014.

[13] S. Al-Dabooni, and D. Wunsch, "Heuristic dynamic programming for mobile robot path planning based on Dyna approach," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 3723-3730, Jul. 2016.

[14] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, no. 1, pp. 9-44, Aug. 1988.

[15] M. Fairbank, and E. Alonso, “Value-Gradient Learning,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, Jun. 2012.

[16] M. Fairbank, D. Prokhorov, and E. Alonso, “Approximating Optimal Control with Value Gradient Learning,” *Chapter 7 in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, New York, NY, USA: John Wiley and Sons, pp. 142-161, Jan. 2013.

[17] S. Al-Dabooni, and D. Wunsch, “Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 2820-2827, May 2017.

[18] X. Bai, D. Zhao, and J. Yi. “ADHDP (λ) strategies based coordinated ramps metering with queuing consideration,” *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 22-27, May 2009.

[19] T. Li, D. Zhao, and J. Yi. “Heuristic dynamic programming strategy with eligibility traces,” *IEEE American Control Conference*, pp. 4535-4540, Jun. 2008.

[20] X. Bai, D. Zhao, and J. Yi, “Ramp Metering Based on on-line ADHDP(λ) controller,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1847-1852, Jul. 2008.

[21] X. Bai, D. Zhao, and J. Yi, “Coordinated multiple ramps metering based on neuro-fuzzy adaptive dynamic programming,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 241-248, Jul. 2009.

[22] H. Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, “True On-line Temporal-Difference Learning,” *Journal of Machine Learning Research (JMLR)*, vol. 145, no. 17, pp. 1-40, Jan. 2016.

[23] R. S. Sutton, A. R. Mahmood, and M. White, “An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning,” *Journal of Machine Learning Research (JMLR)*, vol. 73, no. 17, pp. 1-29, Jan. 2016.

[24] H. Seijen, and R. S. Sutton, “True Online TD(λ),” *Proceedings of the* 31st *International Conference on Machine Learning*, pp. 692-700, Jan. 2014.

[25] Y. Sokolov, R. Kozma, L. D. Werbos, and P. J. Werbos, “Complete stability analysis of a heuristic approximate dynamic programming control design,” *Automatica*, vol. 59, pp. 9-18, Sep. 2015.

[26] F. Liu, J. Sun, J. Si, W. Guo, and S. Mei, “A boundedness result for the direct heuristic dynamic programming,” *Neural Networks*, vol. 32, pp. 229-235, Aug. 2012.

[27] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. NewYork, NY, USA: Wiley, Mar. 2012.

[28] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219-245, Jan. 2000.

[29] M. Fairbank, "Reinforcement learning by value gradients," *eprintarXiv:0803.3539*, Mar. 2008.

[30] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550-1560, Oct. 1990.

[31] Y. Zhu, D. Zhao , and D. Liu, "Convergence analysis and application of fuzzy-HDP for nonlinear discrete-time HJB systems," *Neurocomputing*, vol. 149, pp. 124-131, Feb. 2015.

[32] C. Yang, Y. Jiang, Z. Li, W. He, and C. Su, "Neural control of bimanual robots with guaranteed global stability and motion precision," *IEEE Trans. Industrial Informatics*, vol. 13, no. 3, pp. 1162-1171, Jun. 2017.

[33] C. Yang, X. Wang, L. Cheng, and H. Ma, "Neural-learning-based telerobot control with guaranteed performance," *IEEE Trans. Cybernetics*, vol. PP, no. 99, pp. 1-12, Jun. 2016.

[34] G. Zhang, M. Y. Hu, B. E. Patuwo, and D. C. Indro, "Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis," *European Journal of Operational Research*, vol. 116, no. 1, pp. 16-32, Jul. 1999.

[35] D. Liu, and D. Wang, "Optimal Control of Unknown Nonlinear Discrete-Time Systems Using the Iterative Globalized Dual Heuristic Programming Algorithm," *Chapter 3 in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, New York, NY, USA: John Wiley and Sons, pp. 52-77, Jan. 2013.

[36] C. Chen, D. Dong, H. Li, J. Chu, and T. Tarn, "Fidelity-Based Probabilistic Q-Learning for Control of Quantum Systems," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 5, no. 10, pp. 920-933, May 2014.

[37] P. J. Werbos, and X. Pang, "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot," *IEEE Proc. Conf. Systems, Man and Cybernetics (SMC)* , pp. 1764-1769, Oct. 1996.

[38] D. Wunsch, "The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 79-82, Jul. 2000.

[39] R. Ilin, R. Kozma, and P. J. Werbos,"Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 929-937, Jun. 2008.

[40] N. Zheng, and P. Mazumder, “Hardware-Friendly Actor-Critic Reinforcement Learning Through Modulation of Spike-Timing-Dependent Plasticity,” *IEEE Trans. on Computers*, vol. 66, no. 2, pp. 299-311, Feb. 2017.

[41] B. Luo, D. Liu, T Huang, and D. Wang, “Model-Free Optimal Tracking Control via Critic-Only Q-Learning,” *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 27, no. 5, pp. 2134-2144, Oct. 2016.

[42] The National Council of Teachers of Mathematics (NCTM): https://illuminations.nctm.org

[43] R. Ilin, R. Kozma, and P. Werbos, “Beyond feedforward models trained by backpropagation: A practical training tool for a more efficient universal approximator,” *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 19, no. 6, pp. 929-937, Jan. 2008.

[44] Z. Ni, H. He, J. Wen, and X. Xu, âĂİGoal Representation Heuristic Dynamic Programming on Maze Navigation," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 24, no. 12, pp. 2038-2050, Dec. 2013.

V. ONLINE MODEL-FREE N-STEP HDP WITH STABILITY ANALYSIS.

S. Al-Dabooni and Donald C. Wunsch
Department of Electrical & Computer Engineering
Missouri University of Science and Technology
Rolla, Missouri 65409–0050
Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

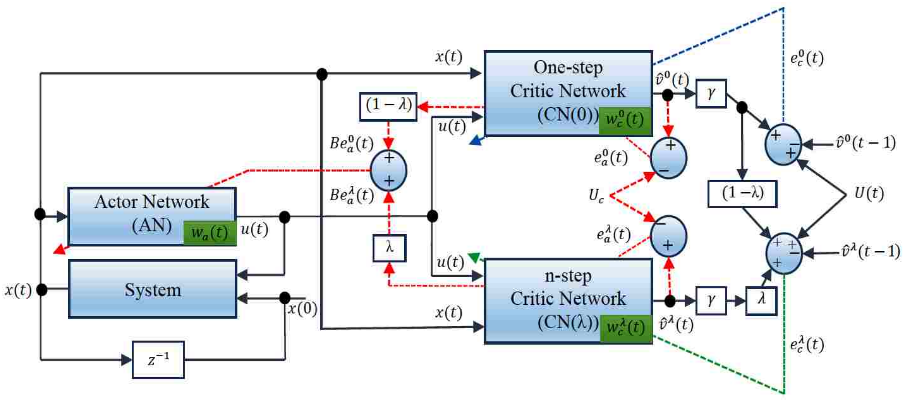
ABSTRACT

Since a *backward view* learning of eligibility traces requires pre-episode updating (off-line tuning), this paper presents a novel adaptive dynamic programming (ADP) architecture with a *forward view* learning that is useful for online updating. Three neural networks are used with this architecture: the critic network with one-step temporal-difference (TD) learning (TD(0)), a critic network with n-step TD learning (TD(λ)) and a action network. This design is called the online model-free n-step action-dependent heuristic dynamic programming (NSHDP(λ)). NSHDP(λ) has low computational costs and is memory efficient because it uses direct implementation without storing the trajectory for every state. The design architecture and their relative learning algorithms are illustrated in detail. Furthermore, stability is proved for NSHDP(λ) under certain conditions by using Lyapunov analysis to obtain the uniformly ultimately bounded (UUB) property. Moreover, a complex nonlinear system, an inverted pendulum and a 2-D maze problem are three simulation benchmarks that are used to examine NSHDP(λ) performance as it compares with other ADP methods.

Keywords: Approximate dynamic programming (ADP), action dependent heuristic dynamic programming (ADHDP), λ-return, Lyapunov stability, uniformly ultimately bounded (UUB).

1. INTRODUCTION

Because of the behavior of a nonlinear system, solving the Hamilton-Jacobi-Bellman (HJB) equation instead of the Riccati equation is very hard. Adaptive/approximate dynamic programming (ADP) is used to overcome this challenge via heuristic techniques with an approximate solution of the HJB equation [1]. ADP has three fundamental families [2] and [3]: heuristic dynamic programming (HDP), dual heuristic programming (DHP) and globalized DHP. Each of them consists of three neural networks: actor, critic and model that provide decision making, evaluation and prediction, respectively. If the action-dependent (AD) is used in ADP (ADHDP for HDP and ADDHP for DHP), the critic network has state-action pair input. In [5], Si and Wang implemented the online learning ADHDP, while the online learning of ADDHP is introduced by Ni *et al.* [6]. Online ADP learning is also improved by Haibo He via adding dual critic networks for ADHDP as in [7], [8] and for ADDHP as in [9]. Many applications have used the ADP techniques. Venayagamoorthy *ea al.* in [10] used DHP to control the operation of a turbo-generator that has a better performance than HDP. Collective robotic search problems are solved with DHP by N. Zhang and D. Wunsch [11]. Lian and Xu [12] applied HDP to allow a mobile robot to escape from sharp corners. Al-Dabooni and Wunsch [13] applied ADHDP to the Dyna algorithm to obtain an optimal path by cooperating multi-robot navigation in an unknown environment. Similar to [13], Volodymyr Mnih *et al.* [14] presented asynchronous actor networks in single multi-core CPU (threads) but with a common critic network in another thread, and they called it asynchronous deep reinforcement learning. The asynchronous deep reinforcement learning is applied in various state-of-the-art forms in the Atari domain, but the most important one is StarCraft II [15]. A temporal-difference (TD) with λ parameter is a more advanced learning algorithm than traditional TD learning. Sutton in [16], [17] illustrated a combination between basic TD learning with eligibility traces further accelerating learning. Inspired by [16], Fairbank and Alonso [18], [19] introduced new ADP algorithms that extend DHP by including a bootstrapping parameter (λ) for eligibility traces. They called it value-gradient learning (VGL(λ)). The VGL(λ) is used to track a reference trajectory under uncertainties to control a nonholonomic mobile robot [20]. As reviewed in [2], ADP trains the actor (controller) network to give optimal actions by minimizing the value function that is produced from the critic network. Both networks are approximated by using a feedforward artificial neural network of multilayer perceptron. The paper denotes the actor network with ω_a parameters as AN(x, ω_a), or only AN for simplicity, which produces an action vector (u). This work uses two critic networks. The first critic network is learned by using a one-step TD learning error. The function approximator for the one-step critic network with ω_c^0 parameters is CN(x, u, ω_c^0), or CN(0) for simplicity. The second critic network is learned by using an average of the n-step TD learning error. The function approximator for the average of the n-step critic network with ω_c^λ parameters is AN(x, u, ω_c^λ), or CN(λ) for simplicity. A simple interpretation and good performance are two well-known properties of TD(λ) as presented in [16], [17], [21] - [28]. But these works used an additional memory variable associated with each state to store the eligibility traces; therefore, they suffer from high computational complexity. In [29], Al-Dabooni and D. Wunsch solved this problem, but that was for batch-implementation learning. The NSHDP(λ) design is used for online-implementation learning. Thus, the NSHDP(λ) structure is memory efficient since it overcome the drawbacks of using eligibility-trace storage and online learning. The online learning aspect with low computational cost is the first contribution for this work. The second contribution is that it provides stability proofs for NSHDP(λ) architecture. The general stability of ADP is an open problem [30]. The stability of the one-step model-free ADHDP is introduced by Feng Liu *et al.* [31] and Yury Sokolov *et al.* [30]. Haibo He *et al.* [32] provided UUB proofs for critic/reference neural networks and a fuzzy logic controller. This work expands the stability of model free learning from one-step ($\lambda = 0$) ADHDP(0)



Parameters Tuning Paths:

One-step Critic Network Path: 🟦

n-step (Eligibility Traces) Critic Network Path: 🟩

Actor Network Path: 🟥

Backpropagation Calculation Error Signals:

One-step Critic Network Error (TD (0) Error): $e_c^0(t) = \left(\gamma\hat{v}^0(t) + U(t)\right) - (\hat{v}^0(t-1))$; $E_c^0(t) = \frac{1}{2}\left(e_c^0(t)\right)^2$; $\Delta\widehat{w}_c^0(t) = \ell_c^0[-\frac{\partial E_c^0(t)}{\partial w_c^0(t)}]$.

n-step Critic Network Error (TD (λ) Error): $e_c^\lambda(t) = \left(\gamma\lambda\hat{v}^\lambda(t) + \gamma(1-\lambda)\hat{v}^0(t) + U(t)\right) - (\hat{v}^\lambda(t-1))$; $E_c^\lambda(t) = \frac{1}{2}\left(e_c^\lambda(t)\right)^2$; $\Delta\widehat{w}_c^\lambda(t) = \ell_c^\lambda[-\frac{\partial E_c^\lambda(t)}{\partial w_c^\lambda(t)}]$.

Actor Network Error: $e_a^0(t) = \hat{v}^0(t) - U_c$; $e_a^\lambda(t) = \hat{v}^\lambda(t) - U_c$; $E_a^0(t) = \frac{1}{2}\left(e_a^0(t)\right)^2$, $E_a^\lambda(t) = \frac{1}{2}\left(e_a^\lambda(t)\right)^2$; $\Delta\widehat{w}_a(t) = -\ell_a\,[(1-\lambda)\frac{\partial E_a^0(t)}{\hat{v}^0(t)}\frac{\hat{v}^0(t)}{\partial w_a(t)} + \lambda\frac{\partial E_a^\lambda(t)}{\hat{v}^\lambda(t)}\frac{\hat{v}^\lambda(t)}{\partial w_a(t)}]$.

Figure 1. Schematic diagram for the adaptation of an online model-free n-step ADHDP (NSHDP(λ)). This design uses two critic networks: the one-step critic network (CN(0)) and the n-step critic network (CN(λ)). The CN(0) produces a one-step-return value function ($\hat{v}^0(t)$) based on the ordinary temporal-difference (TD) learning algorithm, while the CN(λ) produces the average of the n-step-return value function ($\hat{v}^\lambda(t)$) based on a TD(λ) learning algorithm [27]. The TD(λ) learned from the average of the n-step-return backups, where λ represents the proportional average weight. $\lambda-$return (R_t^λ) [16] is another name for the average of the n-serp-return. The $\hat{v}^\lambda(t)$ value is identical to R_t^λ, [29]. This design is equivalent to the one-step TD backup (λ=0). It focuses on the recent information to predict the value function via CN(0). Online learning is another advantage of this design, where it speeds up the tuning without requiring any initial backup for $\hat{v}^\lambda(t)$. Furthermore, this design is a model-free learning design that does not require prior knowledge about a mathematics dynamic model. Despite the bootstrapping eligibility trace parameters (λ and γ) give the CN(λ) the ability to determine a depth (effecting via λ) and a width (effecting via γ) from information during a sequence of events (i.e., the rewards in the backward view of TD(λ), [21]). The CN(0) provides the value function that concentrates on recent events. Therefore, the NSHDP(λ) design combines the details of the current information (real-time data) with a sequence of predicted events. This combination provides the optimal decisions [40] in the control/industry field as well as [41] in the consumer/marketing field (correlation between real time and history). The weights for CN(0) and CN(λ) are updated according to the TD(0) error (blue dashed line) and the TD(λ) error (green dashed line), respectively. The actor network (AN) that provides the action values is tuned by two paths (backpropagating errors): one through the CN(0) path ($e_a^0(t)$) and the other through the CN(λ) path ($e_a^\lambda(t)$). This strategy assists AN training to correlate and combine the fluid information from CN(λ) and CN(0). These two paths are filtered via a similar value of λ, and they combine to produce a total backpropagating actor error (red dashed line).

to ADHDP with λ. Section 2 shows the NSHDP(λ) structure. The remaining sections are organized as follows: Section 3 presents the UUB stability analysis for CN(0), CN(λ) and AN, Section 6 illustrates the simulation results, and Section 7 is the Conclusion.

2. THE ONLINE MODEL-FREE NSHDP(λ)

2.1. NSHDP(λ) Architecture. The ADP technique allows agents to select optimal actions to minimize their long-term cost, which is given [33]:

$$J\big(x(t)\big) = \sum_{k=t}^{\infty} \gamma^{k-t} U\Big(x(k), u(k)\Big), \tag{1}$$

where $J(x(t))$ is the value function (long-term cost) of the state vector ($x \in \mathbb{R}^m$) at time step t. γ denotes the discount factor, and $U(x(k), u(k)) \equiv U(k)$ is called the utility function at time step k for x after applying the action vector $u \in \mathbb{R}^n$. The TD learning with an eligibility traces parameter (λ) helps fill the gaps between the sequence of predicted events and the training data [17], [27], [28]. This work combines the TD(λ) learning technique with the model-free ADHDP. It uses two critical networks to criticize and learn the actor network. Haibo He *et al.* introduced dual-critic with ADHDP, which is called the goal representation HDP (GrHDP) [7], [34]. GrHDP used the one-step TD learning for both critic networks. In contrast, this paper uses the one-step TD for learning a first critic network, while the second critic is learned by using n-step TD learning. Fig. 1 illustrates the main architecture for this paper's approach, which is the online model-free n-step ADHDP (NSHDP(λ)), and the "AD" abbreviation is removed for simplicity. This design not only accomplished the GrHDP function, but it also applied the advantages of TD(λ) learning technique (good fast performance, low computational cost and simple interpretation). The details of all the blocks are explained in this section. The general discrete-time nonlinear system model is represented as:

$$x(t+1) = f\big(x(t), u(t)\big), \tag{2}$$

where x is the m-dimensional system state vector and u is the n-dimensional control vector. NSHDP(λ) is used to solve the Bellman equation [4], [35]

$$J(t) = U(t+1) + \gamma J(t+1), \tag{3}$$

to satisfy the optimal performance discrete-time HJB equation [1]:

$$J^*(t) = \min_{u(t)}\{U(t+1) + \gamma J^*(t+1)\}, \tag{4}$$

where J^* denotes the optimal value function and the instantaneous cost (the utility function) should be bounded ($U(t) \in [0,1]$). Equation (3) is called a one-step TD that learns the critic and actor in traditional ADHDP(0)[2]. As in [21], the Bellman equation for a one-step-return (total discounted future reward depending on one-step TD error) is given as $R_t^{(1)} = U(t)+\gamma J(t+1)$, and for a two-step-return is given as $R_t^{(2)} = U(t)+\gamma U(t+1)+\gamma^2 J(t+2)$, while for an n-step-return ($R_t^{(n)}$) is given as follows:

$$R_t^{(n)} = U(t) + \gamma U(t+1) + \ldots + \gamma^{n-1}U(t+n-1) + \gamma^n J(t+n) = \sum_{k=t}^{n} \gamma^{k-t} U(t+k), \tag{5}$$

where $R_t^{(i)}$ is the i-step-return value, which is the summation of instantaneous costs from t to i. An average of the n-step return is a technique to achieve the fair cost value distribution. For instance, the average calculation of the four-step return can be done via half of the two-step return and half of the four-step return such that $R_t^{Av(2,4)} = \omega_2 R_t^{(2)} + \omega_4 R_t^{(4)}$, where $\omega_2 = 0.5$ and $\omega_4 = 0.5$ are the proportional weights. The proportional weights (ω_i), $i = 1, 2...$, are positive and add up 1. A λ parameter represents these proportional weights. For instance, $\omega_1 = (1-\lambda)$ for an average of one-step return($R_t^{(1)}$); for an average of two-step return, $R_t^{Av(1,2)} = \omega_1 R_t^{(1)} + \omega_2 R_t^{(2)}$, where $\omega_1 = (1-\lambda)$ and $\omega_2 = \lambda$; and so on. Section 3 shows a stability proof for selecting λ to fit with NSHDP(λ), which should be $0 < \lambda < 1$

[2]the zero denotes a one-step learning (λ=0)

rather than $0 \leq \lambda \leq 1$, which is used frequently with the regular TD(λ) reinforcement learning algorithms (i.g. Q(λ)-learning). The λ-return is another name for the average of the n-step-return, which is defined in general as:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)}. \tag{6}$$

For simplicity, the average of the n-step-return is henceforth denoted as the n-step expression. The previous method is called the *forward view* of the TD(λ) learning algorithm. Since each step uses the knowledge of what will happen many steps later, the *forward view* is not directly implementable. The *backward view* provides an extra variable associated with each state, which is called an eligibility trace. NSHDP(λ) does not use any eligibility trace parameters by redriving the λ-return as in (5), which is used to train the parameters of the approximate value function (critic network). By substituting (5) into (5), the following is obtained [3]:

$$\begin{aligned} R_t^{\lambda} &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left(\Big[\sum_{k=0}^{n-1}\gamma^k U(t+k)\Big] + \gamma^n J(t+n)\right). \\ &= (1-\lambda)\left(\Big[\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{k=0}^{n-1}\gamma^k U(t+k)\Big] + \sum_{n=1}^{\infty}\lambda^{n-1}\gamma^n J(t+n)\right). \end{aligned} \tag{7}$$

Expanding and re-arranging (6) yield

$$\begin{aligned} R_t^{\lambda} =& U(t) + \lambda\gamma\Bigg(U(t+1) + \lambda\gamma\Bigg(U(t+2) + \lambda\gamma\Big(U(t+3) + \ldots + \lambda\gamma\big(U(\infty) + (1-\lambda)\gamma \\ & J(\infty)\big) + \ldots + (1-\lambda)\gamma J(t+4)\Big) + (1-\lambda)\gamma J(t+3)\Bigg) + (1-\lambda)\gamma J(t+2)\Bigg) + \\ & (1-\lambda)\gamma J(t+1), \end{aligned} \tag{8}$$

[3]The detail derived is presented in [29]

where $\bar{v}(\infty) = U(\infty)$. Then, the target-value according to (8) is given as follows:

$$\bar{v}(t) = U(t) + \gamma\lambda\bar{v}(t+1) + \gamma(1-\lambda)J(t+1), \tag{9}$$

where $\bar{v}(t)$ in (8) is identical to R_t^{λ}. The NSHDP(λ) design uses a λ-return value as a target value to train the weights in the critic neural network as an approximation function for the n-step value function. We called this network the n-step critic network (CN(λ)). CN(λ) uses $\hat{v}_c^{\lambda}(t)$ as the approximated output symbol. The other critic network provides $\hat{v}_c^{0}(t)$. $\hat{v}_c^{0}(t)$ is an approximated function value for $J(t)$ in (3), which is similar to $R_t^{(1)}$. For this reason, the other critic network is called a one-step critic network (CN($\lambda = 0$) or CN(0)). Therefore, the Bellman equations for the one- and n-step TD learning are given as:

$$\hat{v}^0(t) = U(t) + \gamma\hat{v}^0(t+1), \tag{10}$$

and

$$\hat{v}^{\lambda}(t) = U(t) + \gamma\lambda\hat{v}^{\lambda}(t+1) + \gamma(1-\lambda)\hat{v}^0(t+1), \tag{11}$$

respectively. The online free-model design for NSHDP(λ) was inspired from [5] and [37]. Both the previous $(t-1)$ step and the current (t) step are stored. Similar to [5], the delayed errors for the one- and n-step critic networks are adopted, as well as the actor network, and they use the gradient descent technique to update the weights in all the networks over time steps. Since structure of a fully connected neural network structure by Werbos [38] is so elegant and extendable, it is used in the all three networks (AN, CN(0), and CN(λ)) as a universal function approximator. All the weights that connect the input nodes with the output nodes were set to zero; therefore, it is similar to the traditional three-layer feedforward neural network structure. However, this structure might be useful for models that require direct connections between the input and output nodes without passing through the hidden layers.

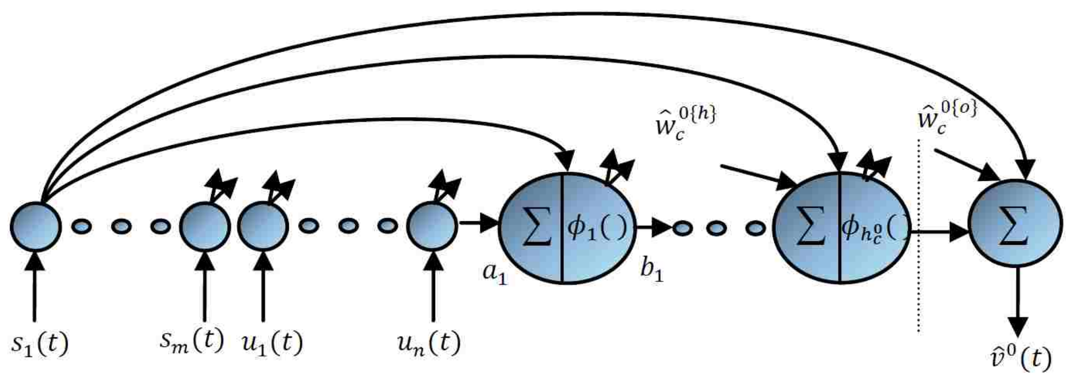


Figure 2. A schematic diagram of CN(0) in NSHDP(λ). As mentioned by Werbos in [38], this structure is more general than a traditional three-layer feed-forward neural network that is fully connected among all neurons. It models a variety of functional forms as demonstrated in [39]. All weights were set so that the connection input nodes with the output nodes were zero. $\hat{\omega}_c^{0\{h\}}$ represents hidden weights, which are connected to the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_c^{0\{o\}}$, which connect both the input and hidden layers with the output layer. $a_k(t)$ is the kth hidden node input of the critic network, and $b_k(t)$ is the corresponding output of the hidden node. A hyperbolic tangent threshold function ($\phi(.)$) is applied to the hidden neurons.

2.2. The One-Step Critic Network (CN(0)). The structure of the CN(0) consists of a three-layer feed-forward neural network including one hidden layer neural network. As shown in Fig. 2, the output of the CN(0) is $\hat{v}^0(t)$, which is an approximation value of $J(t)$ in (1). The inputs for the CN(0) are the system states ($s_1(t), s_2(t), \ldots, s_m(t)$) and the actions ($u_1(t), u_2(t), \ldots, u_\text{n}(t)$). h_c^0 is the number of hidden neurons, m is the number of system states, and n is the number of actions. The hidden weights are indicated as $\hat{\omega}_c^{0\{h\}}$, which can be represented in a $((m + \text{n}) \times h_c^0)$ dimension matrix. The output weights are indicated as $\hat{\omega}_c^{0\{o\}}$, which can be represented in a $(h_c^0 \times 1)$ dimension matrix. The activation function for the hidden nodes is the hyperbolic tangent threshold function ($\phi(x) = (1 - e^{-x})/(1 + e^{-x})$). The forward propagating output signal according to Fig. 1 and Fig. 2 is

$$a_{(k)}(t) = \sum_{i=1}^{m} \hat{\omega}_{c_{(k,i)}}^{0\{h\}} s_{(i)}(t) + \sum_{j=1}^{\text{n}} \hat{\omega}_{c_{(k,(j+m))}}^{0\{h\}} u_{(j)}(t), \tag{12}$$

$$b_{(k)}(t) = \phi(a_{(k)}(t)), \tag{13}$$

$$\hat{v}^0(t) = \sum_{k=1}^{h_c^0} \hat{\omega}_{c_{(k)}}^{0\{o\}} b_{(k)}(t), \tag{14}$$

where $k = 1, 2, \ldots, h_c^0$, $a_{(k)}(t)$ is the kth hidden node input of the CN(0) network, and $b_{(k)}(t)$ is the corresponding output in the hidden node. The weights for the hidden ($\hat{\omega}_c^{0\{h\}}$) and output ($\hat{\omega}_c^{0\{o\}}$) layers are tuned by backpropagating the prediction error of the critic network, which is given as follows:

$$e_c^0(t) = U(t) + \gamma \hat{v}^0(t) - \hat{v}^0(t-1). \tag{15}$$

The objective function for the CN(0) is to minimize $E_c^0(t) = 0.5\big(e_c^0(t)\big)^2$ by updating the value for the weights according to the gradient descent algorithm:

$$\hat{\omega}_c^0(t+1) = \hat{\omega}_c^0(t) + \triangle\hat{\omega}_c^0(t) = \hat{\omega}_c^0(t) - \ell_c^0 \frac{\partial E_c^0(t)}{\partial \hat{\omega}_c^0(t)}, \tag{16}$$

and the chain propagation path can be represented as

$$\frac{\partial E_c^0(t)}{\partial \hat{\omega}_c^0(t)} = \frac{\partial E_c^0(t)}{\partial \hat{v}_c^0(t)} \cdot \frac{\partial \hat{v}_c^0(t)}{\partial \hat{\omega}_c^0(t)}. \tag{17}$$

Then, the adaptation of the CN(0) output weight is

$$\hat{\omega}_c^0(t+1) = \hat{\omega}_c^0(t) - \ell_c^0 \gamma \phi_c^0(t) \Big[\gamma \hat{\omega}_c^{0^T}(t)\phi_c^0(t) + U(t) - \hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)\Big]^T, \tag{18}$$

where $\phi_c^0 = [b_1, b_2, \ldots, b_{h_c^0}]^T$, $\hat{\omega}_c^0 = \hat{\omega}_c^{0\{o\}}$, and $\hat{v}^0(t) = \hat{\omega}_c^{0^T}(t)\phi_c^0(t)$. Following [31], the initial hidden weights ($\hat{\omega}_c^{0\{h\}}$) were chosen randomly and kept constant while the output weights ($\hat{\omega}_c^{0\{o\}} \equiv \hat{\omega}_c^0$) were updated.

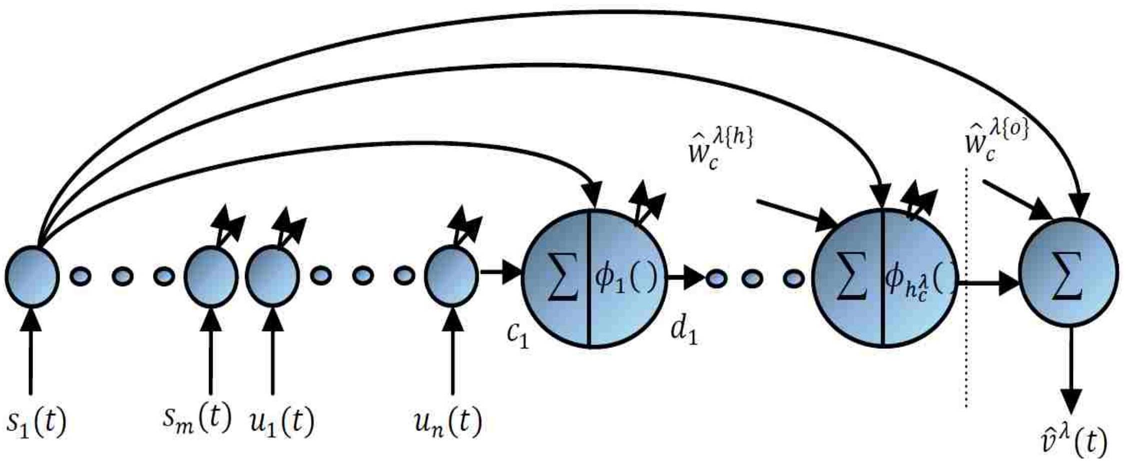


Figure 3. A schematic diagram of the average n-step learning critic network (CN(λ)) in NSHDP(λ). The $\hat{\omega}_c^{\lambda\{h\}}$ represents the hidden weights which are connected to the input layer through the hidden layer. The output weights are indicated as $\hat{\omega}_c^{\lambda\{o\}}$, which connect both the input and hidden layers with the output layer. $c_k(t)$ is the kth hidden node input of the critic network, and $d_k(t)$ is the corresponding output to the hidden node. Here, only apply a hyperbolic tangent threshold function ($\phi(.)$) is applied to the hidden neurons.

2.3. The N-Step Critic Network (CN(λ)). Fig. 3 illustrates the structure for CN(λ), which has a similar configuration to the CN(0), but with its own hidden and output weights. The output of CN(λ), $\hat{v}^\lambda(t)$, learns to approximate R_t^λ as in (8). The inputs for the CN(λ) are the actions $(u_1(t), u_2(t), \ldots, u_\text{n}(t))$ and the system states $(s_1(t), s_2(t), \ldots, s_m(t))$. h_c^λ is the number of hidden neurons of the CN(λ), m is the number of system states, and n is the number of actions. The hidden weights are indicated by $\hat{\omega}_c^{\lambda\{h\}}$, which can be represented in a $((m + \text{n}) \times h_c^\lambda)$ dimension matrix. The output weights are indicated as $\hat{\omega}_c^{\lambda\{o\}}$, which can be represented in a $(h_c^\lambda \times 1)$ dimension matrix. The activation function for the hidden nodes is the hyperbolic tangent threshold function. The forward propagating output signal according to Fig. 1 and Fig. 3 is expressed as follows:

$$c_{(k)}(t) = \sum_{i=1}^{m} \hat{\omega}_{c_{(k,i)}}^{\lambda\{h\}} s_{(i)}(t) + \sum_{j=1}^{\text{n}} \hat{\omega}_{c_{(k,(j+m))}}^{\lambda\{h\}} u_{(j)}(t), \tag{19}$$

$$d_{(k)}(t) = \phi(c_{(k)}(t)), \tag{20}$$

$$\hat{v}^{\lambda}(t) = \sum_{k=1}^{h_c^{\lambda}} \hat{\omega}_{c_{(k)}}^{\lambda\{o\}} q_{(k)}(t), \tag{21}$$

where $k = 1, 2, \ldots, h_c^{\lambda}$, $c_{(k)}(t)$ is the kth hidden node input of the CN(λ) network, and $d_{(k)}(t)$ is the corresponding output of the hidden node. The weights for the hidden ($\hat{\omega}_c^{\lambda\{h\}}$) and output ($\hat{\omega}_c^{\lambda\{o\}}$) layers are implemented by backpropagating the prediction error of the critic network

$$e_c^{\lambda}(t) = U(t) + \gamma\Big(\lambda\hat{v}^{\lambda}(t) + (1-\lambda)\hat{v}^{0}(t)\Big) - \hat{v}^{\lambda}(t-1). \tag{22}$$

The objective function for the CN(λ) is to minimize $E_c^{\lambda}(t) = 0.5\big(e_c^{\lambda}(t)\big)^2$ by updating the value for the weights according to the gradient descent algorithm:

$$\hat{\omega}_c^{\lambda}(t+1) = \hat{\omega}_c^{\lambda}(t) + \triangle\hat{\omega}_c^{\lambda}(t) = \hat{\omega}_c^{\lambda}(t) - \ell_c^{\lambda}\frac{\partial E_c^{\lambda}(t)}{\partial\hat{\omega}_c^{\lambda}(t)}, \tag{23}$$

and the chain propagation path can be represented as

$$\frac{\partial E_c^{\lambda}(t)}{\partial\hat{\omega}_c^{\lambda}(t)} = \frac{\partial E_c^{\lambda}(t)}{\partial\hat{v}_c^{\lambda}(t)} \cdot \frac{\partial\hat{v}_c^{\lambda}(t)}{\partial\hat{\omega}_c^{\lambda}(t)}. \tag{24}$$

Then, the adaptation of the CN(λ) output weights is

$$\begin{aligned}\hat{\omega}_c^{\lambda}(t+1) =& \hat{\omega}_c^{\lambda}(t) - \ell_c^{\lambda}\lambda\gamma\phi_c^{\lambda}(t)\Big[\gamma\lambda\hat{\omega}_c^{\lambda^T}(t)\phi_c^{\lambda}(t) + \gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^{0}(t) + U(t) - \\ & \hat{\omega}_c^{\lambda^T}(t-1)\phi_c^{\lambda}(t-1)\Big]^T,\end{aligned} \tag{25}$$

where $\phi_c^{\lambda} = [d_1, d_2, \ldots, d_{h_c^{\lambda}}]^T$, $\hat{\omega}_c^{\lambda} = \hat{\omega}_c^{\lambda\{o\}}$, and $\hat{v}^{\lambda}(t) = \hat{\omega}_c^{\lambda^T}(t)\phi_c^{\lambda}(t)$. As in CN(0), the initial hidden layer weights ($\hat{\omega}_c^{\lambda\{h\}}$) are kept constant while updating the output critic weights ($\hat{\omega}_c^{\lambda\{o\}} \equiv \hat{\omega}_c^{\lambda}$).

2.4. Actor Network (AN). The AN generates near-optimal control actions (policy) that are illustrated in Fig. 4. Werbos [38] suggests that there are extra weights connected among the outputs themselves. This work sets extra weights equal to zero to become similar to the outputs in a traditional multi-output feed-forward neural network. The system states are the input to AN, which are represented by $(s_1(t), s_2(t), \ldots, s_m(t))$, while the outputs are the control actions $(u_1(t), u_2(t), \ldots, u_\text{n}(t))$. h_a is the number of hidden neurons of the AN, m is the number of system states, and n is the number of action values. The hidden weights are indicated as $\hat{\omega}_a^{\{h\}}$, which can be represented in a $(m \times h_a)$ dimension matrix. The output weights are indicated as $\hat{\omega}_a^{\{o\}}$, which can be represented in a $(h_a \times \text{n})$ dimension matrix. The activation function for the hidden nodes is the hyperbolic tangent threshold function. The forward propagating output signal according to Fig. 1 and Fig. 4 can be expressed as follows:

$$p_{(k)}(t) = \sum_{i=1}^{m} \hat{\omega}_{a_{(k,i)}}^{\{h\}} s_{(i)}(t), \quad k = 1, 2, \ldots, h_a, \tag{26}$$

$$q_{(k)}(t) = \phi(p_{(k)}(t)), \quad k = 1, 2, \ldots, h_a, \tag{27}$$

$$u_{(j)}(t) = \sum_{k=1}^{h_a} \hat{\omega}_{a_{(j,(k))}}^{\{o\}} q_{(k)}(t), \quad j = 1, 2, \ldots, \text{n}, \tag{28}$$

where $p_{(k)}(t)$ is the kth hidden node input of the AN network and $q_{(k)}(t)$ is the corresponding output of the hidden node. The actor network weights are adapted by combining the backpropagating signals for errors of the actor network ($e_a^0(t) = \hat{v}^0(t) - U_c$ and $e_a^\lambda(t) = \hat{v}^\lambda(t) - U_c$), where U_c is the desired ultimate cost-to-go objective value. As in [5], U_c is set to "0," corresponding to "success." The objective function for this network is minimizing the actor error, which is given as follows:

$$E_a(t) = 0.5\Big((1-\lambda)E_a^0(t) + \lambda E_a^\lambda(t)\Big) = 0.5\Big(BE_a^0(t) + BE_a^\lambda(t)\Big), \tag{29}$$

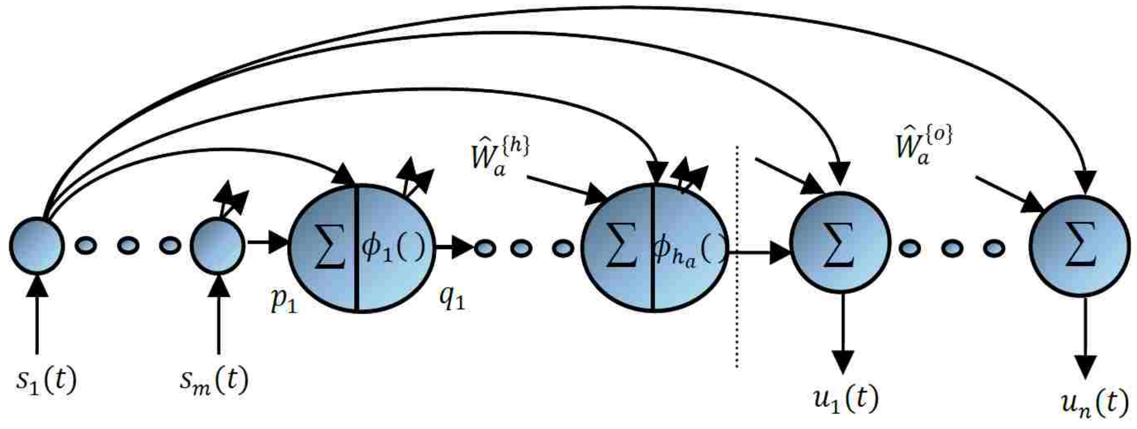


Figure 4. A schematic diagram of the actor network (AN) in NSHDP(λ). All of the weights that connect input nodes with output nodes are set to zero, as well as the connected weights between the outputs themselves. $\hat{\omega}_a^{\{h\}}$ represents the hidden weights which are connected the input layer with the hidden layer. The output weights are indicated as $\hat{\omega}_a^{\{o\}}$, which connect both input and hidden layers with the output layer. $p_k(t)$ is the kth hidden node input of the critic network, and $q_k(t)$ is the corresponding output to the hidden node. A hyperbolic tangent threshold function ($\phi(.)$) is applied in the hidden neurons.

where $E_a^0 = e_a^{0^2}$ and $E_a^\lambda = e_a^{\lambda^2}$. Updating the weight vector by applying the chain rule is given as follows:

$$\begin{aligned}\hat{\omega}_a(t+1) =&\hat{\omega}_a(t) + \triangle\hat{\omega}_a(t) = \hat{\omega}_a(t) - \ell_a \frac{\partial E_a(t)}{\partial \hat{\omega}_a(t)}\\ =&\hat{\omega}_a(t) - \frac{1}{2}\ell_a\Big((1-\lambda)\frac{\partial E_a^0(t)}{\partial \hat{v}_c^0(t)} \cdot \frac{\partial \hat{v}_c^0(t)}{\partial u(t)} \cdot \frac{\partial u(t)}{\partial \hat{\omega}_a(t)} + \lambda \frac{\partial E_a^\lambda(t)}{\partial \hat{v}_c^\lambda(t)} \cdot \frac{\partial \hat{v}_c^\lambda(t)}{\partial u(t)} \cdot \\ &\frac{\partial u(t)}{\partial \hat{\omega}_a(t)}\Big),\end{aligned} \tag{30}$$

where

$$\begin{aligned}\frac{\partial \hat{v}_c^0(t)}{\partial u_{(k)}(t)} &= \sum_{i=1}^{h_c^0} \frac{\partial \hat{v}_c^0(t)}{\partial a_{(i)}(t)} \cdot \frac{\partial a_{(i)}(t)}{\partial b_{(i)}(t)} \cdot \frac{\partial b_{(i)}(t)}{\partial u_{(k)}(t)},\\ \frac{\partial \hat{v}_c^\lambda(t)}{\partial u_{(k)}(t)} &= \sum_{i=1}^{h_c^\lambda} \frac{\partial \hat{v}_c^\lambda(t)}{\partial c_{(i)}(t)} \cdot \frac{\partial c_{(i)}(t)}{\partial d_{(i)}(t)} \cdot \frac{\partial d_{(i)}(t)}{\partial u_{(k)}(t)}, \quad k = 1, 2, \ldots, \mathfrak{n}.\end{aligned} \tag{31}$$

The final adaptation of the action network's weights between the hidden layer and the output later is

$$\begin{aligned}\hat{\omega}_a(t+1) =& \hat{\omega}_a(t) - \ell_a\Bigg((1-\lambda)\phi_a(t)\hat{\omega}_c^{0^T}(t)C^0[\hat{\omega}_c^{0^T}(t)\phi_c^0(t)]^T + \\ & \lambda\phi_a(t)\hat{\omega}_c^{\lambda^T}(t)C^\lambda[\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)]^T\Bigg),\end{aligned} \tag{32}$$

where $\phi_a = [q_1, q_2, \ldots, q_{h_a}]^T$, $\hat{\omega}_a = \hat{\omega}_a^{\{o\}}$, $\hat{u}(t) = \hat{\omega}_a^T(t)\phi_a(t)$, $C^0(t)$ is the feedback weighting values for CN(0), and $C^\lambda(t)$ are the feedback weighting values for CN(λ). $C^0(t)$ is a matrix of $h_c^0 \times \mathrm{n}$, and the elements for this matrix are

$$C^0_{(i,j)}(t) = \frac{1}{2}\Big(1 - \big(\phi^0_{c_{(i)}}(t)\big)^2\Big)\hat{\omega}^{0\{h\}}_{c_{(i,m+j)}}(t), \tag{33}$$

where $i = 1, 2, \ldots, h_c^0$ and $j = 1, 2, \ldots, \mathrm{n}$. Likewise $C^\lambda(t)$ is a matrix of $h_c^\lambda \times \mathrm{n}$, and the elements for this matrix are

$$C^\lambda_{(i,j)}(t) = \frac{1}{2}\Big(1 - \big(\phi^\lambda_{c_{(i)}}(t)\big)^2\Big)\hat{\omega}^{\lambda\{h\}}_{c_{(i,m+j)}}(t), \tag{34}$$

where $i = 1, 2, \ldots, h_c^\lambda$ and $j = 1, 2, \ldots, \mathrm{n}$. Similar to the updated strategy of the critic networks, hidden actor weights ($\hat{\omega}_a^{\{h\}}$) are kept at random constants while updating the output actor weights ($\hat{\omega}_a^{\{o\}} \equiv \hat{\omega}_a$). Werbos *et at.* [30] shows the details about backpropagating signals and the gradient decent learning algorithm.

3. STABILITY ANALYSIS FOR NSHDP(λ)

The Lyapunov function provides the UUB property for dynamical systems without solving the state equations. This section discusses the stability for CN(0), CN(λ) and AN networks by using the Lyapunov function.

3.1. Basics of The Lyapunov Approach. Let $\omega_c^{0^*}$ and $\omega_c^{\lambda^*}$ denote the optimal weights for the one- and n-step critic networks, ω_a^* is the optimal weight for the actor network. the optimal weights for the three networks are defined as follows:

$$\omega_c^{0^*} = argmin_{\hat{\omega}^0} \|U(t) + \gamma\hat{v}^0(t) - \hat{v}^0(t-1)\|, \tag{35}$$

$$\omega_c^{\lambda^*} = argmin_{\hat{\omega}^\lambda} \|U(t) + \gamma\Big(\lambda\hat{v}^\lambda(t) + (1-\lambda)\hat{v}^0(t)\Big) - \hat{v}^\lambda(t-1)\|, \tag{36}$$

and

$$\omega_a^* = argmin_{\hat{\omega}_a} \|\lambda\hat{v}^\lambda(t) + (1-\lambda)\hat{v}^0(t)\|. \tag{37}$$

The weight estimation error for all three networks (CN(0), CN(λ) and AN networks) is:

$$\tilde{\omega}(t) = \hat{\omega}(t) - \omega^*. \tag{38}$$

A more general discrete time dynamic system for weight update rules (equation (18) and for the one-step critic network, (25) n-step critic network and (32) actor network) defines a dynamic system of estimation errors for a general nonlinear function ($\Phi(.)$) as

$$\tilde{\omega}(t+1) = \tilde{\omega}(t) - \Phi\Big(\hat{\omega}(t), \hat{\omega}(t-1), \phi(t), \phi(t-1)\Big). \tag{39}$$

Therefore, the stability properties of the system in (39) express the asymptotic behavior of the estimation error of the weights ($\tilde{\omega}(t)$).

Definition 1. cf [31] Within a bounded positive value ($\varepsilon > 0$), a discrete time dynamic system (39) solution is UUB, if and only if for any $\delta > 0$ and $t_0 > 0$, there exists a positive number $N = N(\delta, \varepsilon)$ independent of t_0, such that $\|\tilde{\omega}(t)\| \leq \varepsilon$ for all $t \geq N + t_0$ when $\|\tilde{\omega}(t)\| \leq \delta$.

Theorem 1. The discrete time dynamic system (39) has a Lyapunov function $L(\tilde{\omega}(t))$ such that for all $\tilde{\omega}(t_0)$ in a compact set K, $L(\tilde{\omega}(t))$ is positive definite and the first difference,

$\triangle L(\tilde{\omega}(t)) < 0$ for $\|\tilde{\omega}(t)\| > \varepsilon$, where $\varepsilon > 0$, such that $\varepsilon-$ neighborhood of $\tilde{\omega}(t)$ is contained in K. Thus, the dynamic system is UUB and the norm of the state is bounded within a neighborhood of ε.

According to Theorem 1, an appropriate function L is selected to determine the UUB property for (39) subject to Assumptions C^0, C^λ and A below.

3.2. Assumptions. The main theorem are proven according to three lemmas as follows:

Assumptions $\boldsymbol{C^0}$. Let $\omega_c^{0^*}(t)$ be the optimal weights for CN(0):

$$\begin{aligned}\omega_c^{0^*} =& argmin_{\hat{\omega}_c^0}\|e_c^0(t)\| \\ =& argmin_{\hat{\omega}_c^0}\|U(t) + \gamma\hat{\omega}_c^{0^T}(t)\phi_c^0(t) - \hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)\|.\end{aligned} \tag{40}$$

Assume it is bounded by a positive constant (i.e., $\|\omega_c^{0^*}(t)\| \le \omega_{c_{max}}^0$) where $\|.\|$ represents 2-norm.

Lemma $\boldsymbol{C^0}$. Let Assumption C^0 hold. Then, the first difference of $L_c^0(t) = \frac{1}{\ell_c^0}tr\Big(\tilde{\omega}_c^{0^T}(t)\tilde{\omega}_c^0(t)\Big)$ for CN(0) is expressed as follows:

$$\begin{aligned}\triangle L_c^0(t) \le& -\gamma^2\|\xi_c^0(t)\|^2 - \gamma^2(1-\ell_c^0\gamma^2\|\phi_c^0(t)\|^2)\times\|\xi_c^0(t) + \omega_c^{0^{*^T}}\phi_c^0(t) + \gamma^{-1}U(t) - \\ &\gamma^{-1}\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)\|^2 + 2\|\gamma\omega_c^{0^{*^T}}\phi_c^0(t) + U(t) - \frac{1}{2}\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1) \\ &-\frac{1}{2}\omega_c^{0^{*^T}}\phi_c^0(t-1)\|^2 + \frac{1}{2}\|\xi_c^0(t-1)\|^2,\end{aligned} \tag{41}$$

where $\xi_c^0(t) = \Big(\hat{\omega}_c^0(t) - \omega_c^{0^*}\Big)\phi_c^0(t) = \tilde{\omega}_c^0(t)\phi_c^0(t)$, which is the approximation error of the CN(0) output.

Lemma $\boldsymbol{C^0}$ Proof. The first discrete difference of the nominated Lyapunov function is given as follows:

$$\triangle L_c^0(t) = \frac{1}{\ell_c^0}tr\Big(\tilde{\omega}_c^{0^T}(t+1)\tilde{\omega}_c^0(t+1) - \tilde{\omega}_c^{0^T}(t)\tilde{\omega}_c^0(t)\Big). \tag{42}$$

By updating rule (38) and $\tilde{\omega}_c^0(t+1)$ as in (18), the following is obtained:

$$\begin{aligned}\tilde{\omega}_c^0(t+1) =& \tilde{\omega}_c^0(t) - \ell_c^0\gamma\phi_c^0(t)\Big(\gamma\phi_c^{0^T}(t)\tilde{\omega}_c^0(t) + \big[\gamma\omega_c^{0^{*^T}}\phi_c^0(t) + U(t) - \hat{\omega}_c^{0^T}(t-1) \\ & \phi_c^0(t-1)\big]^T\Big)\end{aligned} \tag{43}$$

By assuming $P^0 = [I - \ell_c^0\gamma^2\phi_c^0(t)\phi_c^{0^T}(t)]$, and $Q^0 = [\gamma\omega_c^{0^{*^T}}\phi_c^0(t) + U(t) - \hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)]$ and substituting P^0, Q^0 and (43) into (42) yields

$$\begin{aligned}\triangle L_c^0(t) = & \frac{1}{\ell_c^0}tr\Big(\Big[\tilde{\omega}_c^{0^T}(t)P^{0^T} - \ell_c^0\gamma Q^0\phi_c^{0^T}(t)\Big]\Big[P^0\tilde{\omega}_c^0(t) - \ell_c^0\gamma\phi_c^0(t)Q^{0^T}\Big] - \\ & \tilde{\omega}_c^0(t)\tilde{\omega}_c^{0^T}(t)\Big).\end{aligned} \tag{44}$$

Applying the Cauchy-Schwarz inequality obtains

$$\begin{aligned}\triangle L_c(t) \le & -\gamma^2\|\xi_c^0(t)\|^2 - \gamma^2\Big(1 - \ell_c^0\gamma^2\|\phi_c^0(t)\|^2\Big)\|\xi_c^0(t)\|^2 - 2\Big(1 - \gamma^2\ell_c^0\|\phi_c^0(t)\|^2\Big) \\ & \|\gamma\xi_c^0(t)\|\|Q^0\| + \ell_c^0\gamma^2\|\phi_c^0(t)\|^2\|Q^0\|^2.\end{aligned} \tag{45}$$

With further arrangement (41) is acquired.

Assumptions C^λ. Let $\omega_c^{\lambda^*}(t)$ be the optimal weight for CN(λ), which is defined as follows:

$$\begin{aligned}\omega_c^{\lambda^*} =& argmin_{\hat{\omega}_c^\lambda}\|e_c^\lambda(t)\| = argmin_{\hat{\omega}_c^\lambda}\|U(t) + \gamma\lambda\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t) + \gamma(1-\lambda)\hat{\omega}_c^{0^T}(t) \\ & \phi_c^0(t) - \hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\|.\end{aligned} \tag{46}$$

Assume it is bounded by a positive constant (i.e., $\|\omega_c^{\lambda^*}(t)\| \le \omega_{c_{max}}^\lambda$).

Lemma C^λ. Let Assumption C^λ hold. Then, the first difference of $L_c^\lambda(t) = \frac{1}{\alpha\ell_c^\lambda}tr(\tilde{\omega}_c^{\lambda^T}(t)\tilde{\omega}_c^\lambda(t))$ for CN(λ) is expressed as follows:

$$\begin{aligned}\triangle L_c^\lambda(t) \leq & -\frac{\gamma^2\lambda^2}{2}\|\xi_c^\lambda(t)\|^2 - \frac{\gamma^2\lambda^2}{\alpha}(1-\ell_c^\lambda\gamma^2\lambda^2\|\phi_c^\lambda(t)\|^2)\times\|\xi_c^\lambda(t)+\lambda^{-1}(1-\lambda) \\ & \hat{\omega}_c^{0^T}(t)\phi_c^0(t)+\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t)+\gamma^{-1}\lambda^{-1}U(t)-\gamma^{-1}\lambda^{-1}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\|^2+ \\ & \frac{2}{\alpha}\|\gamma\lambda\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t)+U(t)+\gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t)-\frac{1}{2}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)- \\ & \frac{1}{2}\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t-1)\|^2+\frac{1}{2\alpha}\|\xi_c^\lambda(t-1)\|^2,\end{aligned} \tag{47}$$

where $\xi_c^\lambda(t) = (\hat{\omega}_c^\lambda(t) - \omega_c^{\lambda^*})\phi_c^\lambda(t) = \tilde{\omega}_c^\lambda(t)\phi_c^\lambda(t)$, which is the approximation error of the CN(λ) output, and α is a weighting factor ($\alpha > 0$).

Lemma C^λ Proof. The first discrete difference of $L_c^\lambda(t)$ is

$$\triangle L_c^\lambda(t) = \frac{1}{\alpha\ell_c^\lambda}tr(\tilde{\omega}_c^{\lambda^T}(t+1)\tilde{\omega}_c^\lambda(t+1)-\tilde{\omega}_c^{\lambda^T}(t)\tilde{\omega}_c^\lambda(t)). \tag{48}$$

By updating rule (38) and $\tilde{\omega}_c^\lambda(t+1)$ as in (25), which yields

$$\begin{aligned}\tilde{\omega}_c^\lambda(t+1) =& \tilde{\omega}_c^\lambda(t) - \ell_c^\lambda\gamma\lambda\phi_c^\lambda(t)\Big[U(t)+\gamma\lambda\tilde{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t)- \\ & \tilde{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\Big]^T\end{aligned} \tag{49}$$

By assuming $P^\lambda = [I - \ell_c^\lambda\gamma^2\lambda^2\phi_c^\lambda(t)\phi_c^{\lambda^T}(t)]$ and $Q^\lambda = [U(t) + \gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) + \gamma\lambda\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t) - \hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)]$ and substituting P^λ, Q^λ and (49) into (48), the following is obtained:

$$\begin{aligned}\triangle L_c^\lambda(t) =& \frac{-\gamma^2\lambda^2}{\alpha}\Big[\|\xi_c^\lambda(t)\|^2+(1-\ell_c^\lambda\|\phi_c^\lambda(t)\|^2)\|\xi_c^\lambda(t)\|^2\Big]-\frac{2\gamma\lambda}{\alpha}tr\Big((1-\gamma^2\lambda^2\ell_c^\lambda \\ & \|\phi_c^\lambda(t)\|^2)+\frac{\gamma^2\lambda^2\ell_c^\lambda}{\alpha}\|\phi_c^\lambda(t)\|^2\|U(t)+\gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t)+\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t)- \\ & \hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\|^2.\end{aligned} \tag{50}$$

By applying the Cauchy-Schwarz inequality, the $\triangle L_c^\lambda(t)$ becomes

$$\begin{aligned}\triangle L_c(t) \le & -\frac{1}{\alpha}\|\gamma\lambda\xi_c^\lambda(t)\|^2 - \frac{\gamma^2\lambda^2}{\alpha}\Big(1-\gamma^2\lambda^2\ell_c^\lambda\|\phi_c^\lambda(t)\|^2\Big)\|\xi_c^\lambda(t) + \gamma^{-1}\lambda^{-1}\Big[U(t) \\ & + \gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) + \gamma\lambda\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t) - \hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\Big]\|^2 + \frac{2}{\alpha} \\ & \|\gamma\lambda\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t) + \gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) + U(t) - \frac{1}{2}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1) - \\ & \frac{1}{2}\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t-1)\|^2 + \frac{1}{2\alpha}\|\xi_c^\lambda(t-1)\|^2.\end{aligned} \tag{51}$$

With further arrangement, (47) is reached.

Assumption A. Let $\tilde{\omega}_a^*$ be the optimal weight for AN, which is bounded by a positive constant, i.e., $\|\omega_a^*\| \le \omega_a^{max}$. Let $\xi_a(t) = [\hat{\omega}_a(t) - \omega_a^*]^T\phi_a(t) = \tilde{\omega}_a^T(t)\phi_a(t)$, which is the approximation error of the actor network output.

Lemma A. Let assumption A hold. Then, the first difference of $L_a(t) = \frac{1}{\alpha_1\ell_a}tr(\tilde{\omega}_a^T(t)\tilde{\omega}_a(t))$ is

$$\begin{aligned}\triangle L_a(t) \le \frac{1}{\alpha_1}\Bigg(&(1-\lambda)\Big(4\|\xi_c^0(t)\|^2 + 4\|\omega_c^{0^{*^T}}\phi_c^0(t)\|^2 + \|\xi_a\hat{\omega}_c^{0^T}(t)C^0\|^2 - \|\hat{\omega}_c^{0^T}(t) \\ & \phi_c^0(t)\|^2\Big) + \lambda\Big(4\|\xi_c^\lambda(t)\|^2 + 4\|\omega_c^{\lambda^{*^T}}\phi_c^\lambda(t)\|^2 + \|\xi_a\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2 - \|\hat{\omega}_c^{\lambda^T}(t) \\ & \phi_c^\lambda(t)\|^2\Big) + \ell_a(1-\lambda)^2\|\phi_a\|^2\|\hat{\omega}_c^{0^T}(t)C^0\|^2\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 + \ell_a\lambda^2\|\phi_a\|^2 \\ & \|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 + \frac{1}{2}\ell_a\lambda(1-\lambda)\|\phi_a(t)\|^2\Big(\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 \\ & \|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 + \|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2 + \|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2\|\hat{\omega}_c^{0^T}(t) \\ & C^0\|^2 + \|\hat{\omega}_c^{0^T}(t)C^0\|^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big)\Bigg),\end{aligned} \tag{52}$$

where $\alpha_1 > 0$, which is a weighting factor.

Lemma A Proof. As in (39), the asymptotic behavior of the estimation error of the actor weight ($\tilde{\omega}_a(t)$) is to be analyzed by studying the stability of (39). The first discrete difference of $L_a(t)$ is given as follows:

$$\triangle L_a(t) = \frac{1}{\alpha_1\ell_a}tr\Big(\tilde{\omega}_c^T(t+1)\tilde{\omega}_a(t+1) - \tilde{\omega}_a^T(t)\tilde{\omega}_a(t)\Big). \tag{53}$$

By updating rule (38) and $\tilde{\omega}_c(t+1)$ as in (32), the following is obtained:

$$\begin{aligned}\tilde{\omega}_a(t+1) =&\tilde{\omega}_a(t) - \ell_a\Bigg((1-\lambda)\phi_a(t)\hat{\omega}_c^{0^T}(t)C^0[\hat{\omega}_c^{0^T}(t)\phi_c^0(t)]^T + \lambda\phi_a(t)\hat{\omega}_c^{\lambda^T}(t)C^\lambda \\ &[\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)]^T\Bigg).\end{aligned} \tag{54}$$

Substituting (54) into (53) yields

$$\begin{aligned}\triangle L_a(t) = \frac{1}{\alpha_1\ell_a}tr\Bigg(&-2\ell_a(1-\lambda)\xi_a(t)\hat{\omega}_c^{0^T}(t)C^0[\hat{\omega}_c^{0^T}(t)\phi_c^0(t)]^T - 2\ell_a\lambda\xi_a(t) \\ &\hat{\omega}_c^{\lambda^T}(t)C^\lambda[\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)]^T + 2\ell_a^2\lambda(1-\lambda)\|\phi_a(t)\|^2\hat{\omega}_c^{0^T}(t)\phi_c^0(t) \\ &[\hat{\omega}_c^{0^T}(t)C^0]^T\hat{\omega}_c^{\lambda^T}(t)C^\lambda[\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)]^T + \ell_a^2(1-\lambda)^2\|\phi_a(t)\|^2\|\hat{\omega}_c^{0^T}(t) \\ &C^0\|^2\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 + \ell_a^2\lambda^2\|\phi_a(t)\|^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2\Bigg).\end{aligned} \tag{55}$$

By applying the Cauchy-Schwarz inequality, the following is obtained:

$$\begin{aligned}\triangle L_a(t) \leq \frac{1}{\alpha_1\ell_a}\Bigg(&\ell_a(1-\lambda)\Big(4\|\xi_a(t)\|^2 + 4\|\omega_c^{0^{*T}}\phi_c^0(t)\|^2 + \|\xi_a(t)\hat{\omega}_c^{0^T}(t)C^0\|^2 - \\ &\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2\Big) + \ell_a\lambda\Big(4\|\xi_a(t)\|^2 + 4\|\omega_c^{\lambda^{*T}}\phi_c^\lambda(t)\|^2 + \|\xi_a(t)\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2 \\ &-\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2\Big) + 2\ell_a^2\lambda(1-\lambda)\|\phi_a(t)\|^2\Big(-\frac{1}{2}\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 - \frac{1}{2} \\ &\|\hat{\omega}_c^{0^T}(t)C^0\|^2\Big)\Big(-\frac{1}{2}\|^2\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 - \frac{1}{2}\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big) + \ell_a^2(1-\lambda)^2 \\ &\|\phi_a(t)\|^2\|\hat{\omega}_c^{0^T}(t)C^0\|^2\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 + \ell_a^2\lambda^2\|\phi_a(t)\|^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2 \\ &\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2\Bigg).\end{aligned} \tag{56}$$

With further arrangement, (52) is reached.

3.3. The Stability Analyses for The Dynamical System. The Lyapunov candidate functions for all networks in NSHDP(λ) are analyzed in this subsection to prove the bound estimation errors.

Theorem 2 (UUB for CN(0), CN(λ) and AN). Let Assumptions C^0, C^λ and A hold with a bounded reinforcement signal. Let gradient descent be used to update the weights of both one-step and n-step critic networks as (18), (25), respectively. The gradient descent is also used to update the weight of the actor network by using (32). The errors between the optimal weights for all networks ($\omega_c^{0^*}$, $\omega_c^{\lambda^*}$, ω_a^*) and their estimates ($\hat{\omega}_c^0(t)$, $\hat{\omega}_c^\lambda(t)$, $\hat{\omega}_a(t)$) are UUB if the following conditions are met

$$\begin{aligned}
&\ell_c^0 < \frac{1}{\gamma^2\|\phi_c^0(t)\|^2}, \quad \ell_c^\lambda < \frac{1}{\gamma^2\lambda^2\|\phi_c^\lambda(t)\|^2},\\
&\ell_a < \frac{(1-\lambda)\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 + \lambda\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2}{\|\phi_a(t)\|^2\Psi}, 0 < \lambda < 1, \quad \frac{1}{\sqrt{2}} < \gamma \leq 1,
\end{aligned} \tag{57}$$

where Ψ is described in (65), and the constraints on the weighting factors are

$$\alpha < \frac{(1-\lambda)(1-2\gamma^2\lambda^2)}{\lambda(1-2\gamma^2)}, \tag{58}$$

and

$$\alpha_1 < \frac{-8(1-\lambda)}{1-2\gamma^2}. \tag{59}$$

Theorem 2 Proof. The definition of the Lyapunov function for NSHDP(λ) is given as follows:

$$L(t) = L_c^0(t) + L_c^\lambda(t) + L_a(t) + L_p^0(t) + L_p^\lambda(t), \tag{60}$$

where $L_p^0(t) = \frac{1}{2}\|\xi_c^0(t-1)\|^2$, and $L_p^\lambda(t) = \frac{1}{2}\|\xi_c^\lambda(t-1)\|^2$. The first difference for $L_p^0(t)$ and $L_p^\lambda(t)$ are given as

$$\triangle L_p^0(t) = \frac{1}{2}\Big(\|\xi_c^0(t)\|^2 - \|\xi_c^0(t-1)\|^2\Big), \tag{61}$$

and

$$\triangle L_p^\lambda(t) = \frac{1}{2}\Big(\|\xi_c^\lambda(t)\|^2 - \|\xi_c^\lambda(t-1)\|^2\Big), \tag{62}$$

respectively. The first difference of the Lyapunov function (60) is

$$\begin{aligned}\triangle L(t) \leq &-\Big(\gamma^2 - \frac{4(1-\lambda)}{\alpha_1} - \frac{1}{2}\Big)\|\xi_c^0(t)\|^2 - \Big(\frac{\gamma^2\lambda^2}{\alpha} - \frac{4\lambda}{\alpha_1} - \frac{1}{2\alpha}\Big)\|\xi_c^\lambda(t)\|^2 - \gamma^2\Big(1 - \gamma^2 \\ &\ell_c^0\|\phi_c^0(t)\|^2\Big)\|\xi_c^0(t) + \omega_c^{0^{*T}}\phi_c^0(t) + \gamma^{-1}U(t) - \gamma^{-1}\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)\|^2 - \\ &\frac{\gamma^2\lambda^2}{\alpha}\Big(1 - \gamma^2\lambda^2\ell_c^\lambda\|\phi_c^\lambda(t)\|^2\Big)\|\xi_c^\lambda(t) + \omega_c^{\lambda^{*T}}\phi_c^\lambda(t) + \gamma^{-1}\lambda^{-1}U(t) + \lambda^{-1} \\ &(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) - \gamma^{-1}\lambda^{-1}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\|^2 - \frac{1}{\alpha_1}\Big((1-\lambda)\|\hat{\omega}_c^{0^T}(t) \\ &\phi_c^0(t)\|^2 + \lambda\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 - \ell_a\|\phi_a(t)\|^2\Psi\Big) + 2\|\gamma\omega_c^{0^{*T}}\phi_c^0(t) + U(t) - \\ &\frac{1}{2}\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1) - \frac{1}{2}\omega_c^{0^{*T}}\phi_c^0(t-1)\|^2 + \frac{2}{\alpha}\|\gamma\lambda\omega_c^{\lambda^{*T}}\phi_c^\lambda(t) + U(t) + \\ &\gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) - \frac{1}{2}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1) - \frac{1}{2}\omega_c^{\lambda^{*T}}\phi_c^\lambda(t-1)\|^2 + \\ &\frac{(1-\lambda)}{\alpha_1}\Big(4\|\omega_c^{0^{*T}}\phi_c^0(t)\|^2 + \|\xi_c^0(t)\hat{\omega}_c^{0^T}(t)C^0\|^2\Big) + \frac{\lambda}{\alpha_1}\Big(4\|\omega_c^{\lambda^{*T}}\phi_c^\lambda(t)\|^2 \\ &+ \|\xi_c^\lambda(t)\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big),\end{aligned} \tag{63}$$

where Ψ is given as follows:

$$\begin{aligned}\Psi = &(1-\lambda)^2\|\hat{\omega}_c^{0^T}(t)C^0\|^2\|\xi_c^0(t)\|^2 - \lambda^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\|\xi_c^\lambda(t)\|^2 - \frac{1}{2}\lambda(1-\lambda)\Big(\\ &\|\xi_c^\lambda(t)\|^2 + \|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big)\Big(\|\xi_c^0(t)\|^2 + \|\hat{\omega}_c^{0^T}(t)C^0\|^2\Big),\end{aligned} \tag{64}$$

which Ψ is equivalent to

$$\begin{aligned}\Psi = &(1-\lambda)^2\|\hat{\omega}_c^{0^T}(t)C^0\|^2\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 - \lambda^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 - \\ &\frac{1}{2}\lambda(1-\lambda)\Big(\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 + \|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2 + \\ &\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2\|\hat{\omega}_c^{0^T}(t)C^0\|^2 + \|\hat{\omega}_c^{0^T}(t)C^0\|^2\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big).\end{aligned} \tag{65}$$

To guarantee that the third and fourth terms of (63) are negative, the learning rate for the one-step critic network (ℓ_c^0) and the n-step critic network (ℓ_c^λ) have to be $1-\gamma^2\ell_c^0\|\phi_c^0(t)\|^2 > 0$ and $1-\gamma^2\lambda^2\ell_c^\lambda\|\phi_c^\lambda(t)\|^2 > 0$, respectively; similar procedure is taken to guarantee the fifth term of (63). From the first and second terms of (63), the weighting factors are calculated. The first difference of $L(t)$ can be rewritten as follows:

$$\begin{aligned}\triangle L(t) \leq & -\Big(\gamma^2 - \frac{4(1-\lambda)}{\alpha_1} - \frac{1}{2}\Big)\|\xi_c^0(t)\|^2 - \Big(\frac{\gamma^2\lambda^2}{\alpha} - \frac{4\lambda}{\alpha_1} - \frac{1}{2\alpha}\Big)\|\xi_c^\lambda(t)\|^2 - \gamma^2\Big(1-\gamma^2 \\ & \ell_c^0\|\phi_c^0(t)\|^2\Big)\|\xi_c^0(t) + \omega_c^{0^{*T}}\phi_c^0(t) + \gamma^{-1}U(t) - \gamma^{-1}\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)\|^2 - \\ & \frac{\gamma^2\lambda^2}{\alpha}\Big(1-\gamma^2\lambda^2\ell_c^\lambda\|\phi_c^\lambda(t)\|^2\Big)\|\xi_c^\lambda(t) + \omega_c^{\lambda^{*T}}\phi_c^\lambda(t) + \gamma^{-1}\lambda^{-1}U(t) + \lambda^{-1} \\ & (1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) - \gamma^{-1}\lambda^{-1}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1)\|^2 - \frac{1}{\alpha_1}\Big((1-\lambda)\|\hat{\omega}_c^{0^T}(t) \\ & \phi_c^0(t)\|^2 + \lambda\|\hat{\omega}_c^{\lambda^T}(t)\phi_c^\lambda(t)\|^2 - \ell_a\Psi\Big) + \Gamma^2,\end{aligned} \tag{66}$$

where Γ^2 is defined as a positive term in (63), which is given as follows:

$$\begin{aligned}\Gamma^2 = & 2\|\gamma\omega_c^{0^{*T}}\phi_c^0(t) + U(t) - \frac{1}{2}\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1) - \frac{1}{2}\omega_c^{0^{*T}}\phi_c^0(t-1)\|^2 + \frac{2}{\alpha}\|\gamma\lambda \\ & \omega_c^{\lambda^{*T}}\phi_c^\lambda(t) + U(t) + \gamma(1-\lambda)\hat{\omega}_c^{0^T}(t)\phi_c^0(t) - \frac{1}{2}\hat{\omega}_c^{\lambda^T}(t-1)\phi_c^\lambda(t-1) - \frac{1}{2}\omega_c^{\lambda^{*T}} \\ & \phi_c^\lambda(t-1)\|^2 + \frac{(1-\lambda)}{\alpha_1}\Big(4\|\omega_c^{0^{*T}}\phi_c^0(t)\|^2 + \|\xi_c^0(t)\hat{\omega}_c^{0^T}(t)C^0\|^2\Big) + \frac{\lambda}{\alpha_1}\Big(4\|\omega_c^{\lambda^{*T}} \\ & \phi_c^\lambda(t)\|^2 + \|\xi_c^\lambda(t)\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big).\end{aligned} \tag{67}$$

With further simplifying, Γ^2 can be rewritten as follows:

$$\begin{aligned}\Gamma^2 \leq & 4\Big(\gamma^2\|\omega_c^{0^{*T}}\phi_c^0(t)\|^2 + U(t)^2 + \frac{1}{4}\|\hat{\omega}_c^{0^T}(t-1)\phi_c^0(t-1)\|^2\frac{1}{4}\|\omega_c^{0^{*T}}\phi_c^0(t-1)\|^2\Big) + \\ & \frac{2}{\alpha}\Big(\gamma^2\lambda^2\|\omega_c^{\lambda^{*T}}\phi_c^\lambda(t)\|^2 + U(t)^2 + \gamma^2(1-\lambda)^2\|\hat{\omega}_c^{0^T}(t)\phi_c^0(t)\|^2 + \frac{1}{4}\|\hat{\omega}_c^{\lambda^T}(t-1) \\ & \phi_c^\lambda(t-1) + \frac{1}{4}\|\omega_c^{\lambda^{*T}}\phi_c^\lambda(t-1)\|^2\Big) + \frac{(1-\lambda)}{\alpha_1}\|\hat{\omega}_c^{0^T}(t)C^0\|^2\Big(\|\omega_a^T\phi_a(t)\|^2 + \|\omega_a^{*^T} \\ & \phi_a(t)\|^2\Big) + \frac{\lambda}{\alpha_1}\|\hat{\omega}_c^{\lambda^T}(t)C^\lambda\|^2\Big(\|\omega_a^T\phi_a(t)\|^2 + \|\omega_a^{*^T}\phi_a(t)\|^2\Big) + \frac{4(1-\lambda)}{\alpha_1}\|\omega_c^{0^{*T}} \\ & \phi_c^0(t)\|^2 + \frac{4\lambda}{\alpha_1}\|\omega_c^{\lambda^{*T}}\phi_c^\lambda(t)\|^2.\end{aligned} \tag{68}$$

The upper bounds for ω_c^0 ($\omega_c^{0^*}$ and $\hat{\omega}_c^0$), ω_c^λ ($\omega_c^{\lambda^*}$ and $\hat{\omega}_c^\lambda$), ω_a (ω_a^* and $\hat{\omega}_a$), ϕ_c^0, ϕ_c^λ, ϕ_a, C^0, C^λ and $U(t)$ are substituted to ω_{cm}^0, ω_{cm}^λ, ω_{am}, ϕ_{cm}^0, ϕ_{cm}^λ, ϕ_{am}, C_m^0, C_m^λ and U_m, respectively. Therefore, Γ^2 can be rewritten as follows:

$$\begin{aligned}\Gamma^2 \leq\ & \omega_{cm}^{0^2}\phi_{cm}^{0^2}\Big(4\gamma^2 + 1 + \frac{2}{\alpha}\gamma^2(1-\lambda)^2 + \frac{4(1-\lambda)}{\alpha_1}\Big) + \omega_{cm}^{\lambda^2}\phi_{cm}^{\lambda^2}\Big(\frac{2}{\alpha}\gamma^2\lambda^2 + \frac{1}{\alpha} + \frac{\lambda}{\alpha_1}\Big) + \\ & \frac{2(1-\lambda)}{\alpha_1}\omega_{cm}^{0^2}C_m^{0^2}\omega_{am}^2\phi_{am}^2 + \frac{2\lambda}{\alpha_1}\omega_{cm}^{\lambda^2}C_m^{\lambda^2}\omega_{am}^2\phi_{am}^2 = \Gamma_m^2.\end{aligned} \tag{69}$$

If (57) holds, then any

$$\|\xi_c^0\| > \frac{\Gamma_m}{\sqrt{\gamma^2 - \frac{4(1-\lambda)}{\alpha_1} - \frac{1}{2}}} \tag{70}$$

or,

$$\|\xi_c^\lambda\| > \frac{\Gamma_m}{\sqrt{\frac{\gamma^2\lambda^2}{\alpha} - \frac{4\lambda}{\alpha_1} - \frac{1}{2\alpha}}}, \tag{71}$$

where $0 < \lambda < 1$, $\frac{1}{\sqrt{2}} < \gamma \leq 1$, α is defined in (58), and α_1 is defined (59), making $\triangle L(t) \leq 0$, meaning that the estimated errors, which are the deference between $\omega_c^{0^*}$, $\omega_c^{\lambda^*}$, ω_a^* and their estimates ($\hat{\omega}_c^0(t)$, $\hat{\omega}_c^\lambda(t)$, $\hat{\omega}_a(t)$) are UUB.

4. SIMULATION STUDY

The trajectories of the internal reinforcement signal 2-D nonlinear system are considered the first case study to verify the effectiveness of the NSHDP(λ). Here, the results are compared with the performance of ADHDP and GrHDP. The second case study is a single link inverted pendulum. The performance of the inverted pendulum is investigated by comparing NSHDP(λ) with GrHDP in different noise exposure effects. The third case study is a 2-D maze benchmark. NSHDP(λ) and ADHDP are also compared, along with other reinforcement algorithms, which are SARSA(0)[4] and Q(λ) [21].

[4]The SARSA (State-Action-Reward-State-Action) algorithm is used to find the series state-action pairs by using one-step learning; therefore, it is written as SARSA(0).

4.1. First Case: Nonlinear System Problem. Consider the following nonlinear system derived from [42]:

$$\begin{aligned} x_1(t+1) &= -sin(0.5x_1(t) + u(t)) \\ x_2(t+1) &= -sin(x_1(t) + 0.5x_2(t)), \end{aligned} \tag{72}$$

where $x(t) = [x_1(t) \quad x_2(t)]^T \in \mathbb{R}^2$, is the state vector ($m = 2$), and $u(t) \in \mathbb{R}^1$ is the control action ($\mathfrak{n} = 1$).

The external instantaneous cost function is $U(t) = x^T(t)x(t)+u^T(t)u(t)$. The discount factor ($\gamma$) is 0.9, and λ is 0.95. The number of hidden nodes in both critic networks is 7 ($h_c^0 = h_c^\lambda = 7$).

The number of hidden nodes for AN is also 5 ($h_a = 5$), and the initial learning parameters for all the networks are $\ell_c^0 = \ell_c^\lambda = 0.05$ for both the CN(0) and the CN(λ), and $\ell_a = 0.01$ for the action network. The training for either network will be terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold for the internal cycle (40 iterations for both critic networks and 50 iterations for the actor network).

The initial weights for all the networks are randomly within [-0.3, 0.3]. As previously mentioned, the hidden weights ($\hat{w}_c^{0\{h\}}$, $\hat{w}_c^{\lambda\{h\}}$, $\hat{w}_a^{\{h\}}$) are fixed, and the output weights ($\hat{w}_c^{0\{o\}}$, $\hat{w}_c^{\lambda\{o\}}$, $\hat{w}_a^{\{o\}}$) are trained. The NSHDP(λ), GrHDP and ADHDP are compared with similar learning parameters. An initial CN(0) in the NSHDP(λ) is chosen that is similar to the reference network in the GrHDP structure, and the initial CN(λ) is the same as the critic network in the GrHDP, while the ADHDP has one critic network that has an initial weight that is similar to the CN(0) in the NSHDP(λ) with $\lambda = 0$. The actor networks for NSHDP(λ), GrHDP and ADHDP have similar initial weights.

Fig. 5 shows that NSHDP(λ) is more efficient because it learn faster than the ADHDP and the GrHDP.

Fig. 5 shows also mean squared errors (MSEs) during iteration. Fig. 6 illustrates the state trajectories and the control actions for 25 time steps.

The initial state is set to $x(0) = [-0.5 \quad 0.5]^T$. The NSHDP($\lambda$) and the GrHDP are derived from the system states to converge faster than the ADHDP. Compareding to the ADHDP, the improvement (according to an MSE technique) is 5.12% for the NSHDP(λ), while it is 3.506% for the GrHDP.

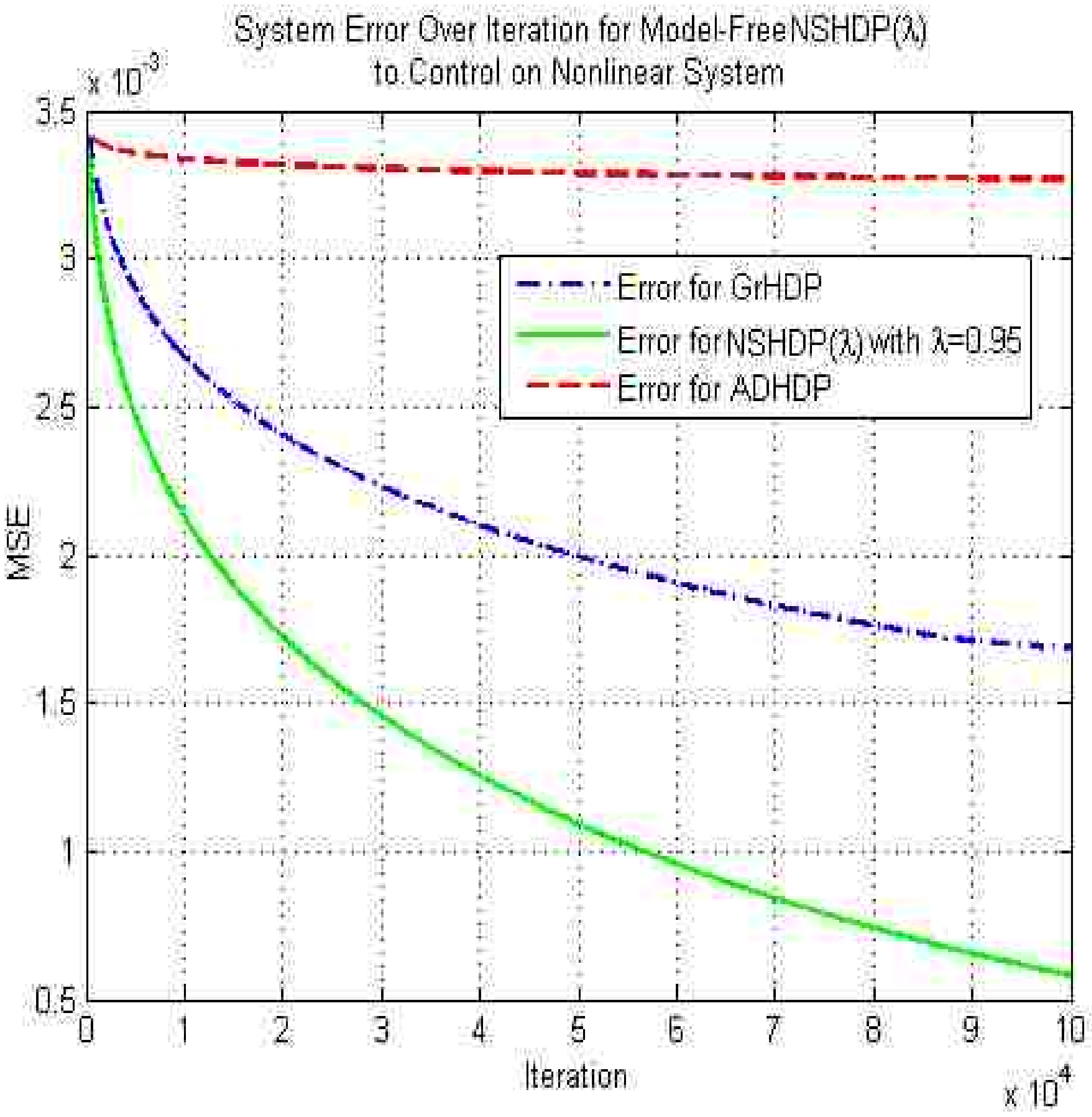


Figure 5. Mean squared error comparisons over iterations among the NSHDP(λ), the GrHDP and the traditional ADHDP. NSHDP(λ) has a faster learning speed than the GrHDP and the ADHDP.

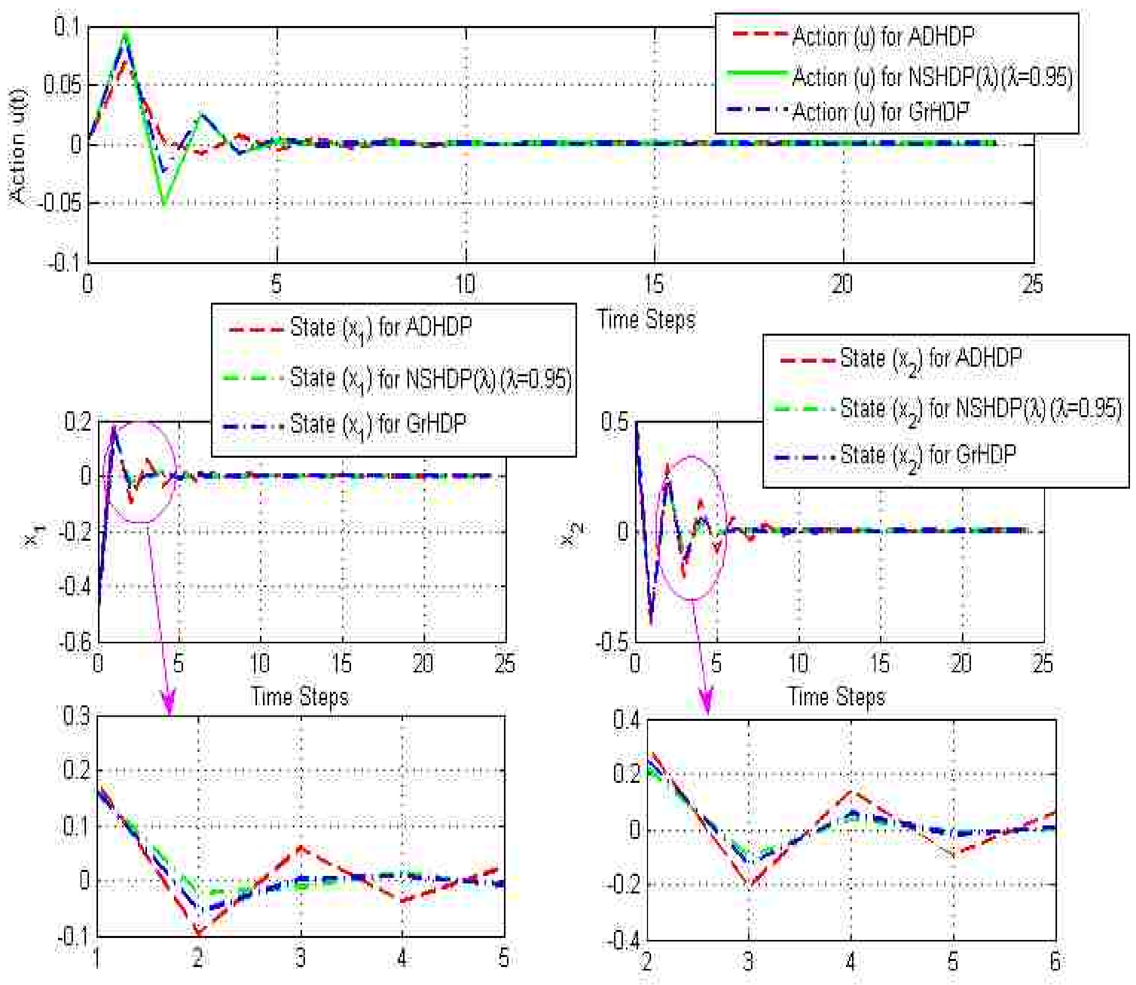


Figure 6. Comparisons of system response trajectories, which are the two system states and the actions for the NSHDP(λ), GrHDP and ADHDP.

The value functions and their targets for the NSHDP(λ) and the GrHDP are shown in Fig. 7. After three online learning time steps, the critic errors for CN(0) and CN(λ) have converged to its equilibrium value, which is clearly shown in the lowest figure of Fig. 8. The GrHDP needs four time steps (as shown in the upper figures of Fig. 7 and Fig. 8). The critic error for the NSHDP(λ = 0) is shown in the middle figure in Fig. 8. Fig. 9 demonstrates the squared backpropagation actor errors through critic networks for all methods (GrHDP, ADHDP and NSHDP(λ)). The corresponding weight trajectory for the NSHDP(λ) and the GrHDP are shown in Fig. 10, which illustrate that the weights for both networks reach their optimal fixed values.

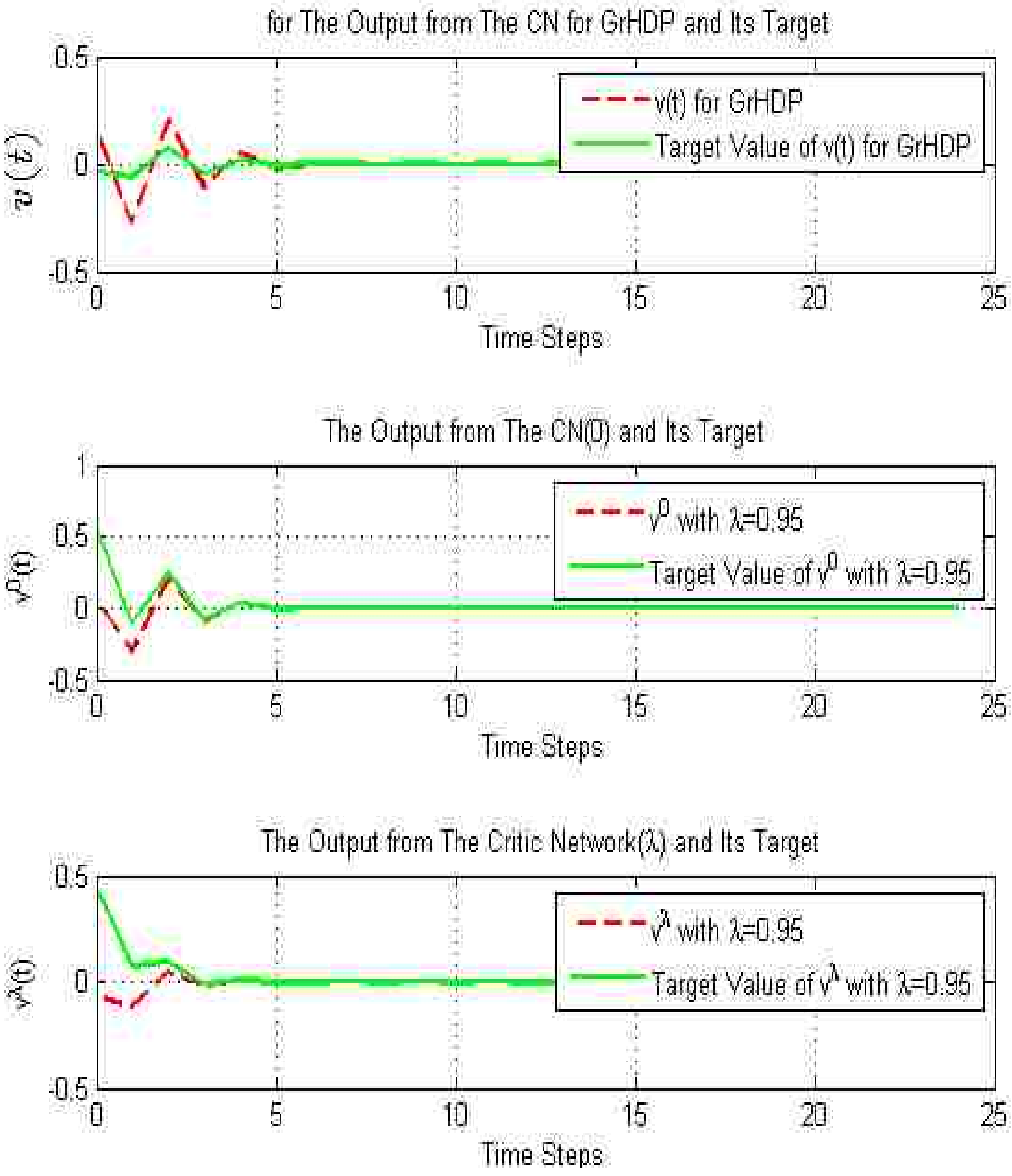


Figure 7. Comparisons for the value functions with their targets between the NSHDP(λ) and the GrHDP. The upper figure shows the value function and its target for the GrHDP (see [7]), which is represented by a red dashed line in the upper figure part of Fig. 8. The middle and upper figures show the value functions of CN(0) and CN(λ) of the NSHDP(λ) with their targets (the left sides of Equations (10) and (11)), respectively. The solid green line in the lower part of Fig. 8 represents the difference of the middle figure, while the dashed red line represents the difference of the lower figure.

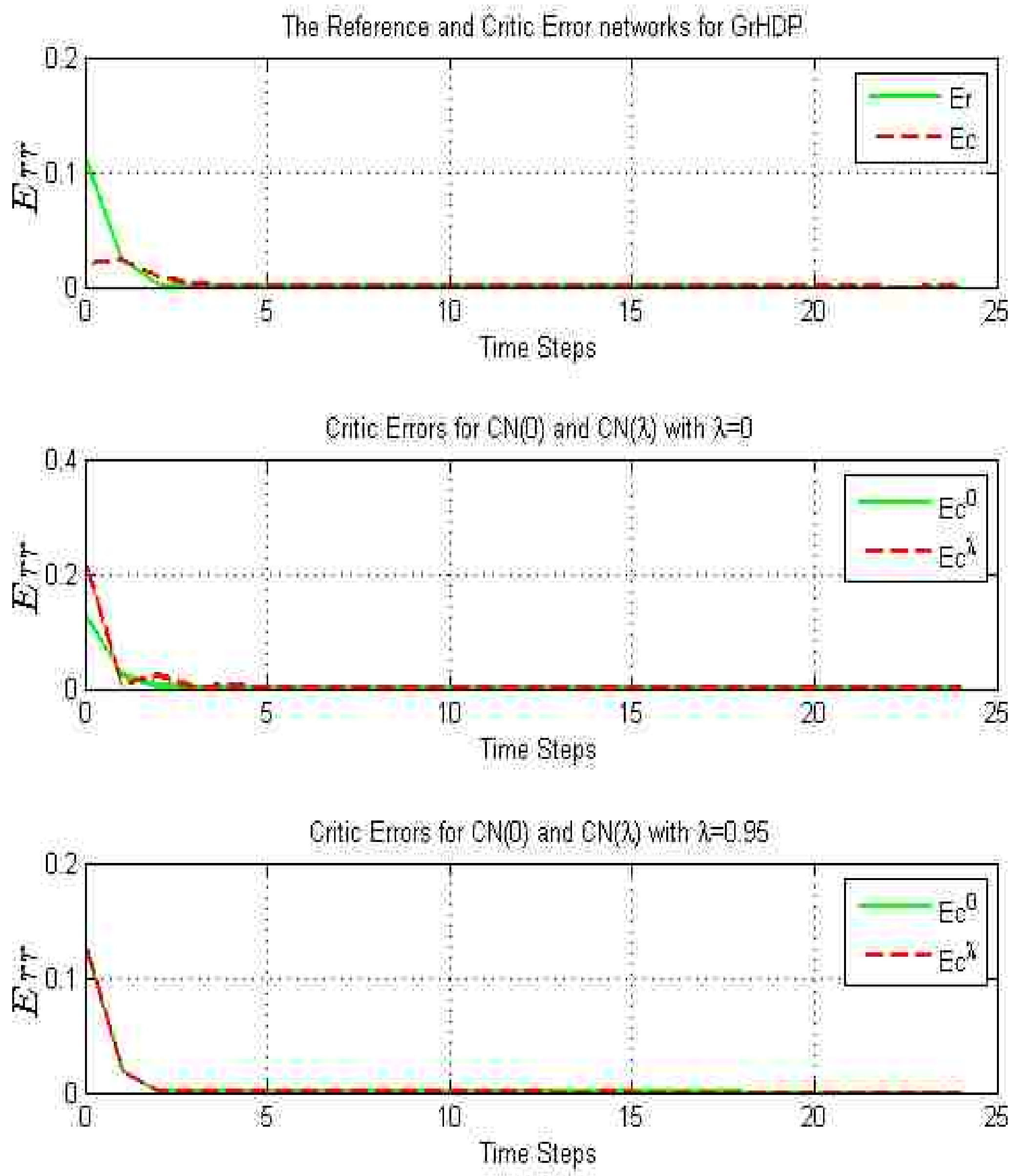


Figure 8. The squared critic errors. The upper figure shows the squared errors for critic and reference networks in the GrHDP structure (see [7] Equations(4) and (5)). The squared critic errors for NSHDP(λ) for both critic networks (CN(0) and CN(λ)) are illustrated in the middle and upper figures when λ = 0.95 and λ = 0, respectively. The squared error for the reference network in the GrHDP is represented by "Er".

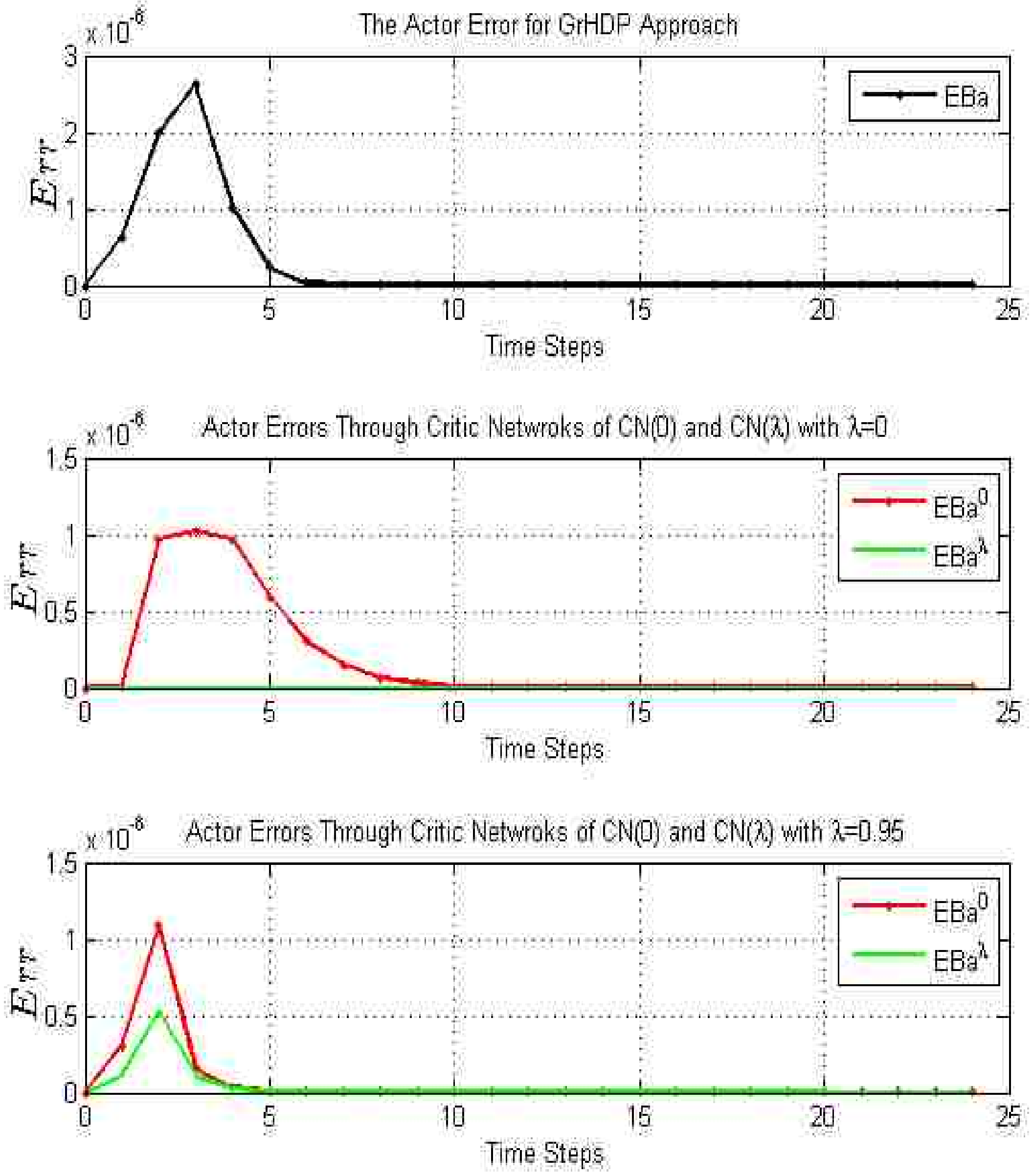


Figure 9. The upper figure shows the squared backpropagating actor error after passing a critic network in the GrHDP. The middle and lower figures show $BE_a^0(t)$ and $BE_a^\lambda(t)$ (see (29)) of the NSHDP(λ) when $\lambda = 0.95$ and $\lambda = 0$, respectively. The lower figure (NSHDP(0.95)) converges faster than the upper (GrHDP) and middle (ADHDP) figures

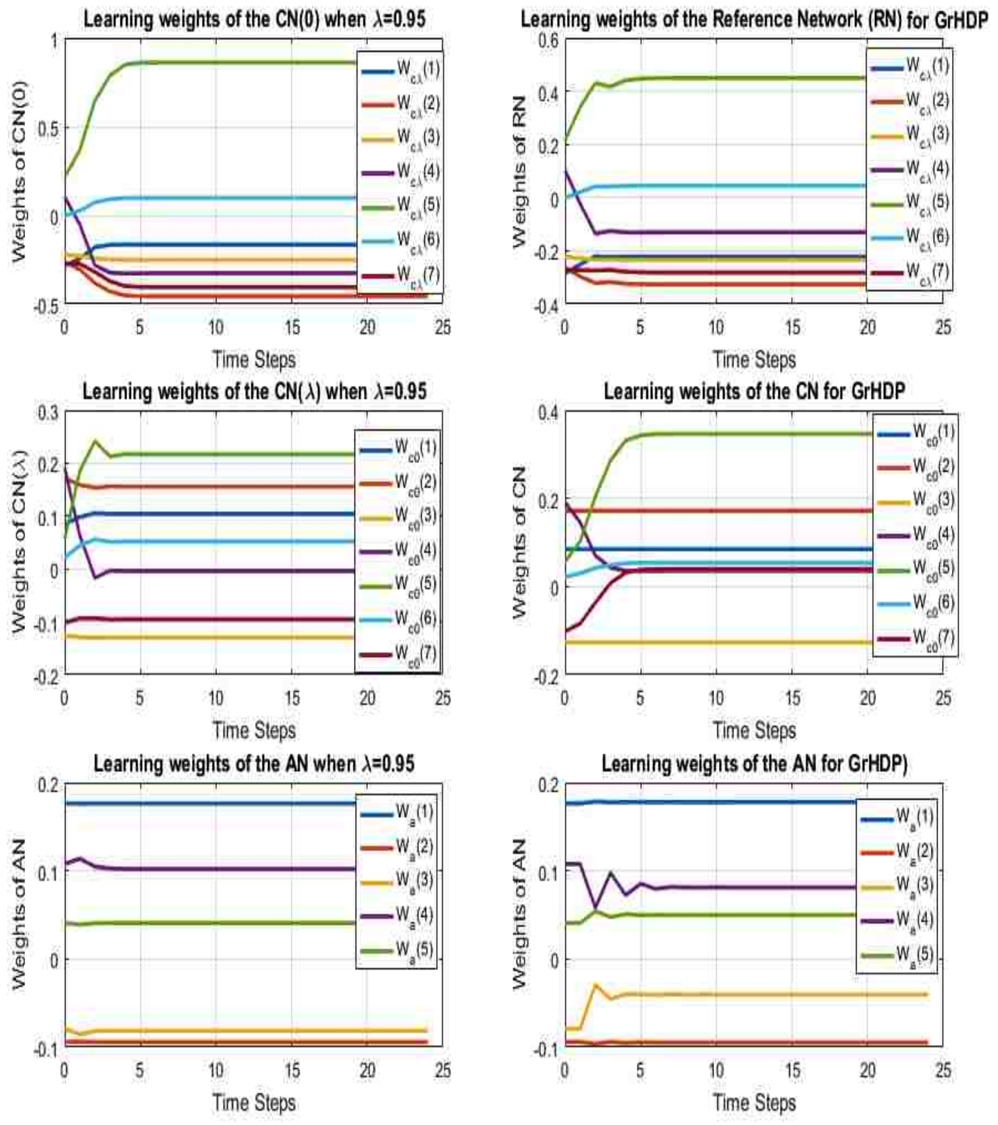


Figure 10. Learning weights for the NSHDP(λ) and the GrHDP. The upper, middle and lower figures in the first column show the learning weights during the time steps for CN(0), CN(λ) and AN in the NSHDP(λ), respectively. The upper, middle and lower figures of the second column show the learning weights during the time steps for goal representation or the reference network (RN), critic network and actor network of GrHDP.

4.2. Second Case: Inverted Pendulum.

4.2.1. Description for The Inverted Pendulum Dynamic Model. The inverted pendulum dynamic model is simulated for the second case study. The NSHDP(λ) is self-learning without *prior* knowledge of the system model. The actor network allows the cart to be moved to the left or right in order to balance a pole (inverted pendulum) mounted on the cart. A binary reinforcement signal is used to learn the actor network. the reinforcement signal is either “-1” or “0” in correspondence to a fallen or balanced pole, respectively. As in [5] and [29], the cart pole system model shown in Fig. 11 is given by

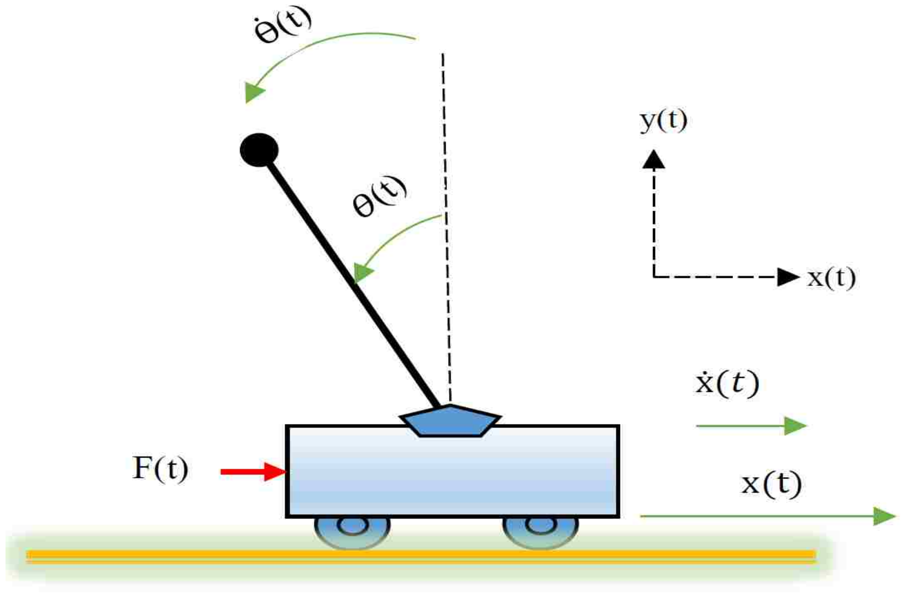


Figure 11. The configuration schematic diagram of the inverted pendulum balancing system.

$$\ddot{\theta} = \frac{g\sin(\theta) + \cos(\theta)\Big(-F - ml\dot{\theta}^2\sin(\theta) + \mu_c\sigma(\dot{x})\Big) - \dfrac{\mu_p\dot{\theta}}{ml}}{l\Big(\dfrac{4}{3} - \dfrac{m\cos^2(\theta)}{m_c + m}\Big)}, \tag{73}$$

$$\ddot{x} = \frac{F + ml\Big(\dot{\theta}^2\sin(\theta) - \ddot{\theta}\cos(\theta)\Big) - \mu_c\sigma(\dot{x})}{m_c + m}, \tag{74}$$

where $g = 9.8[m/sec^2]$, the acceleration due to gravity; $m_c = 1.0[kg]$, the mass of the cart; $m = 0.1[kg]$, the mass of the pole; $l = 0.5[m]$, the half-pole length; $\mu_c = 0.0005$, the coefficient of friction of the cart on the track; $\mu_p = 0.000002$, the coefficient of friction of the pole on the cart; $F = \pm 10[N]$, the force applied to the cart's center of mass; and $\sigma(.)$ is a Sigmund function. The fourth-order Runge-Kutta method is used to solve nonlinear differential equations (73) and (74) with 0.02 sec for the sample step. The inverted pendulum model has four states: $\begin{bmatrix} x(t) & \theta(t) & \dot{x}(t) & \dot{\theta}(t) \end{bmatrix}$, where $x(t)$ is the position of the cart, $\theta(t)$ is the angle of the pole with respect to the vertical axis, $\dot{x}(t)$ is the linear velocity for the cart, and $\dot{\theta}(t)$ is the angular velocity of the pole.

4.2.2. Simulation Results. In this simulation, the NSHDP(λ) and the GrHDP are compared. This comparison focuses on various uniform random noise and initial pole angles. The pole has fallen if it is beyond the range of $(-12°, 12°)$, and also, if the cart moves outside the range $(-2.4, 2.4$ meter) its initial position. The control action ($u(t)$) is a continuous signal, although the cart requires a binary value. The force (F= 10 [N]) is applied on the cart if $u(t) \geq 0$; otherwise, F= -10 [N]. Each run has 60 consecutive trials. The run is considered successful if the last trial has balanced the pole. Each successful trial has 2000 time steps to complete the balancing task. γ is 0.9, and λ is 0.95. The number of hidden nodes is 22 for h_c^0 and h_c^λ. The number of hidden nodes for AN is 20 ($h_a = 20$), and the initial learning parameters are set to $\ell_c^0 = \ell_c^\lambda = 0.005$ for both critic networks and $\ell_a = 0.003$ for the action network. The learning rates are decreased by dividing by 3 every 30 time steps. The training for either network is terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold for the internal cycle (100 iterations for both critic networks and 130 iterations for the actor network). The initial weights for all networks are within [-0.3, 0.3] by fixing the hidden weights and training the output weights. All figures are shown for the last iteration. Fig. 12 - Fig. 15 show the noise-free system responses (states, action, and the cost-to-go values,) for the NSHDP(λ) during 2000 time steps with an initial angle $\theta(t)$ of 0.1°. The bottom figure of Fig. 14 and Fig. 15 show the backpropagating actor error through both CN(0) and CN(λ), which are defined as: $Be_a^0(t) = (1 - \lambda)e_a^0(t)$ and $Be_a^\lambda(t) = \lambda e_a^\lambda(t)$.

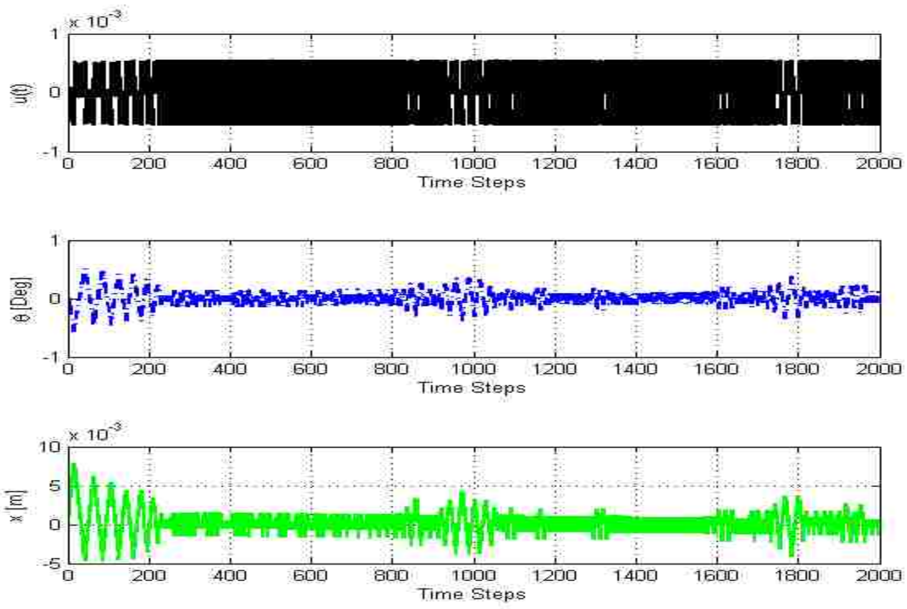


Figure 12. Simulated results of balancing the inverted pendulum for $u(t)$, $\theta(t)$, and $x(t)$ when the system is free of noise. The initial angle $\theta(t)$ is $0.1°$.

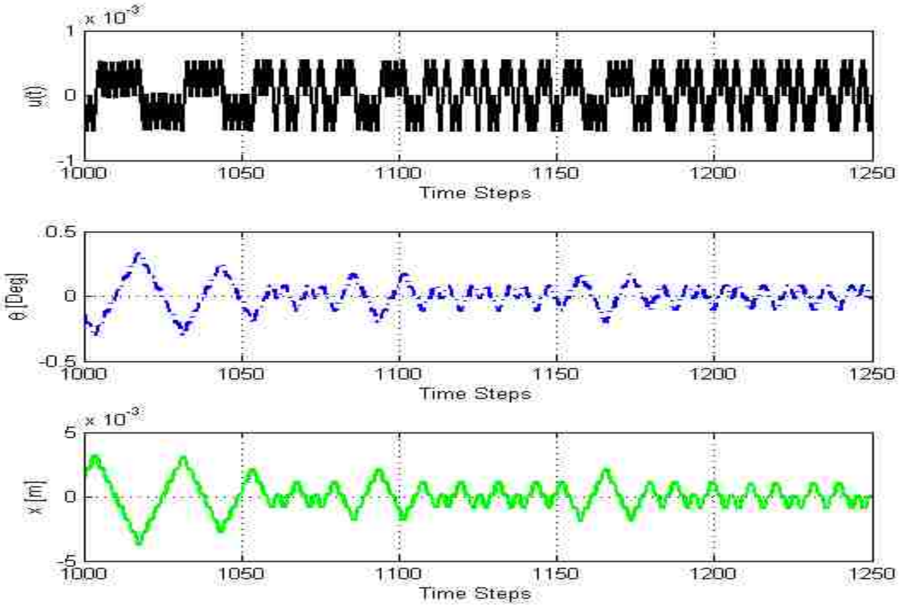


Figure 13. Zoom-in between 1000 to 1250 time steps for Fig. 12.

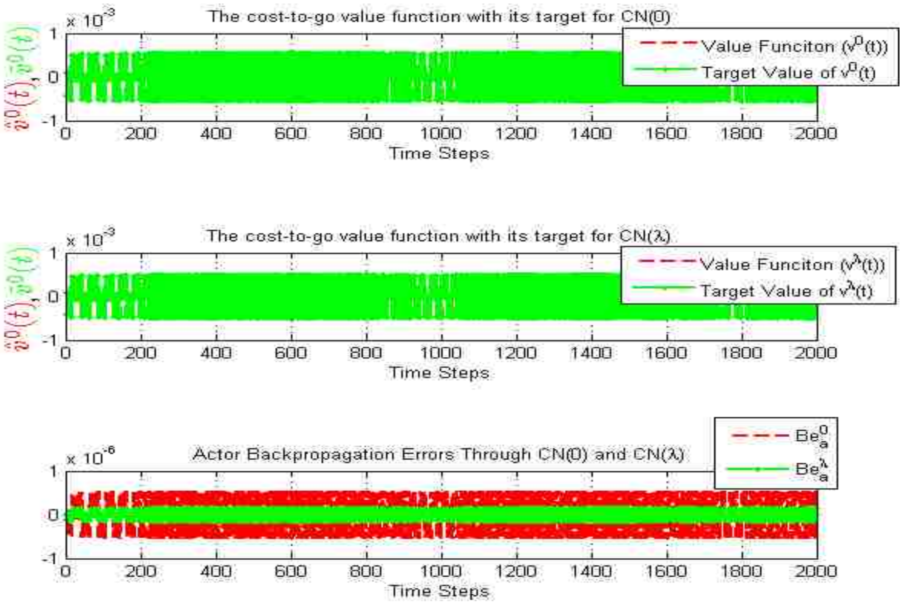


Figure 14. The value functions and their targets without noise. The backpropagating actor errors are Be_a^0 and Be_a^λ through CN(0) and CN(λ), respectively. The initial angle $\theta(t)$ is 0.1°.

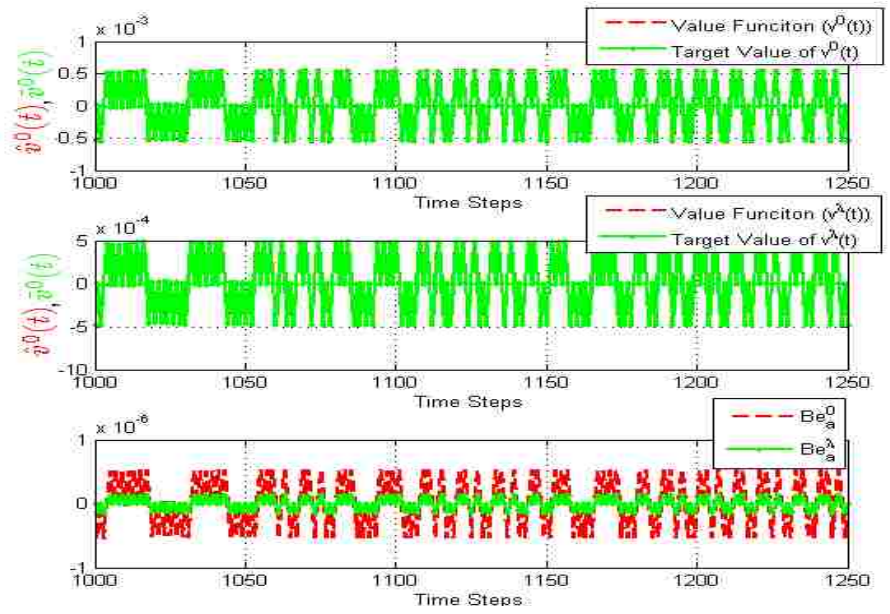


Figure 15. Zoom-in between 1000 to 1250 time steps for Fig. 14.

In order to simulate realistic behavior, the NSHDP(λ) was challenged with $\pm 12°$ as the initial deviation of the pole angle. Furthermore, random noise was uniformly injected into the actuator and sensor. Particularly, the sensor is $\theta(t) = \theta(t) + \eta$, while the actuator noises are determined by $u(t) = u(t) + \eta$. Fig. 16 illustrates the situation results for the system responses $(u(t), \theta(t), x(t))$ with disturbances: 1) $\theta(0) = -12°$. 2) sensor-effected noise is 3%. 3) actuator-effected noise is 3%. Fig. 17 shows the value functions (CN(0) and CN(λ)) and backpropagating actor errors with similar disturbances in Fig. 16. Fig. 18 and Fig. 19 demonstrate the system responses, value functions and backpropagation actor errors with the challenge of effecting sensor and actuator noises similar to Fig. 16 but with $\theta(0) = 12°$

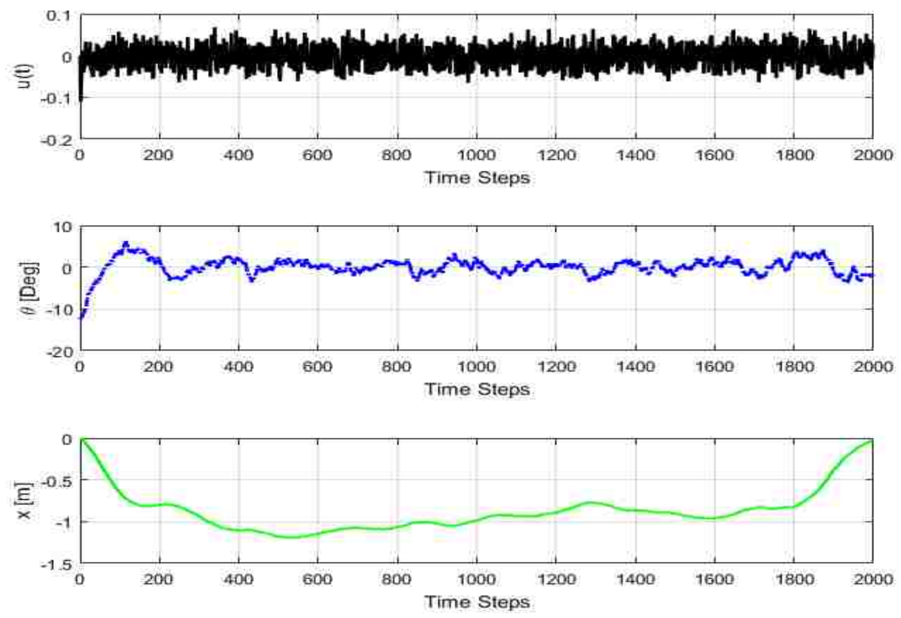


Figure 16. Simulated results of balancing the inverted pendulum to show u(t), $\theta(t)$ and $x(t)$, when the system is injected with 3% uniform noise to the actuator and sensor. The initial angle $\theta(t)$ is $-12°$.

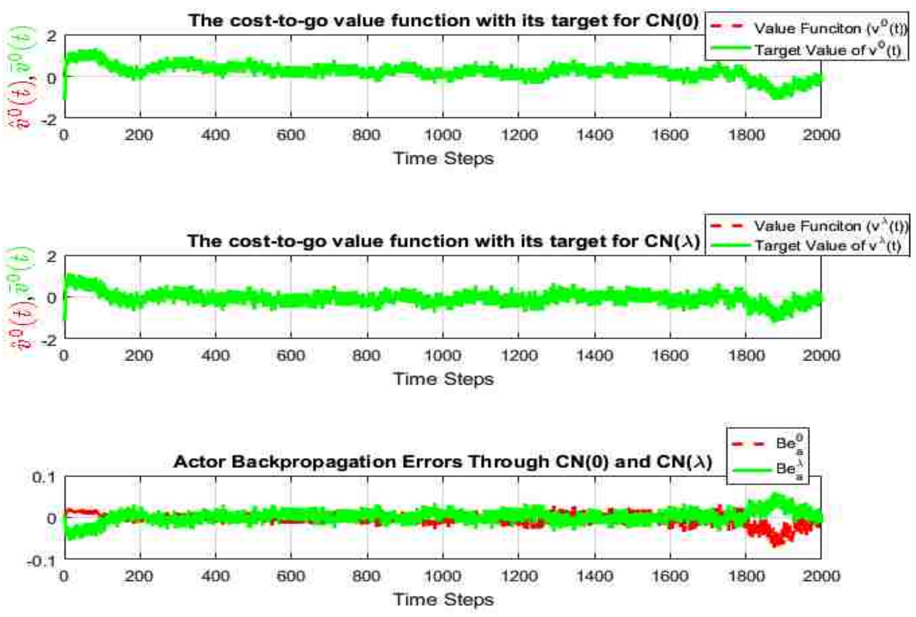


Figure 17. The value functions and their target values with similar disturbances are in the caption of Fig. 16 (upper and lower figures for CN(0) and CN(λ), respectively). The backpropagation actor errors through CN(0) (Be_a^0) and CN(λ) (Be_a^λ) are shown in the lower figure. The initial angle $\theta(t)$ is $-12°$.

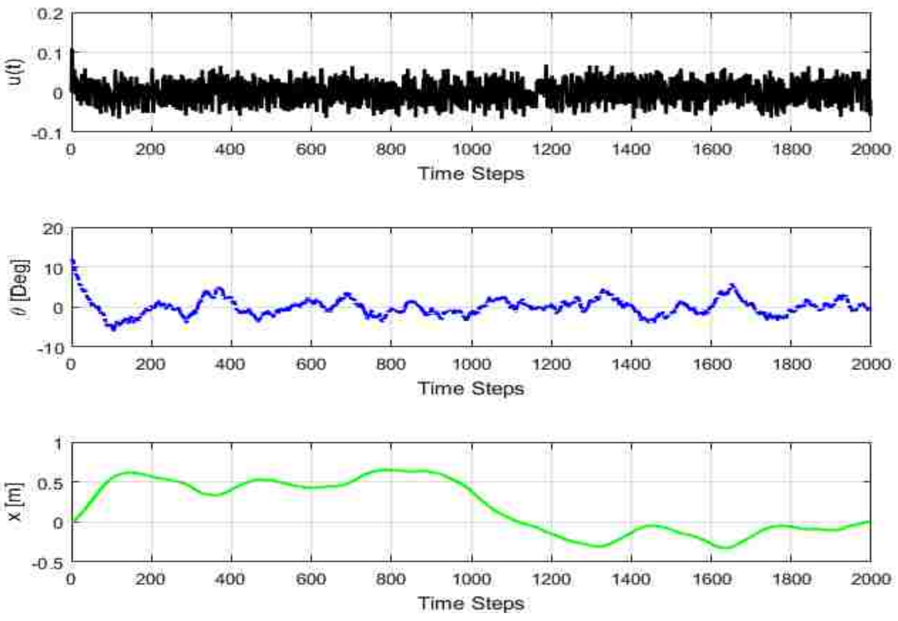


Figure 18. Simulated results of balancing the inverted pendulum to show u(t), $\theta(t)$ and $x(t)$, when the system is injected with 3% uniform noise to the actuator and sensor. The initial angle $\theta(t)$ is $-12°$.

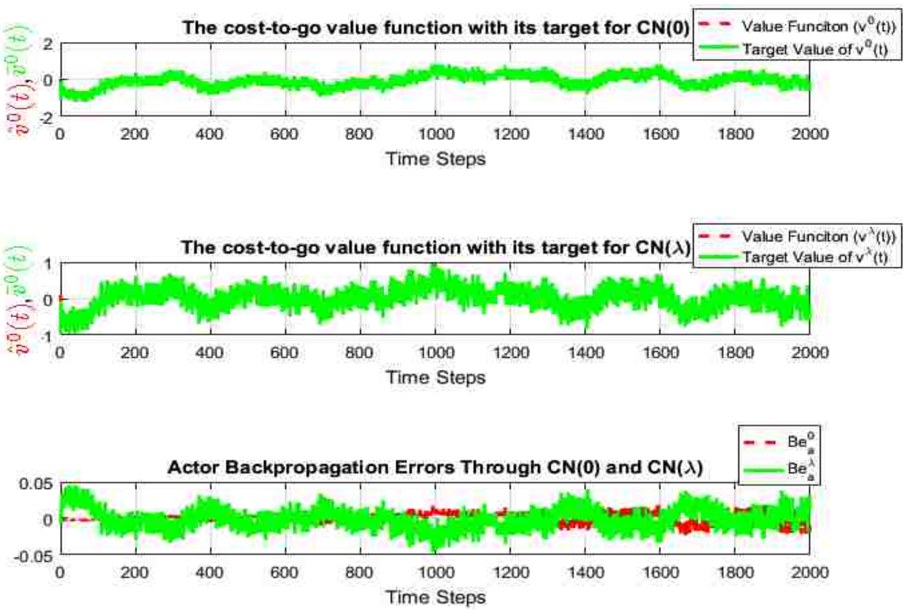


Figure 19. The function values with errors with the initial angle $\theta(t)$ is 12°. The actuator and sensor disturbances are similar to what in the caption of Fig. 18

For further comparison, the NSHDP(λ) is compared with the GrHDP as they are examined under different noise/θ levels as shown in Table I. Table I summarizes the simulation results through 100 averaged runs for 100 time steps and 10 iterations for each run. The average number of iterations for all columns in Table 1 is 3.62 and 4.19 for the NSHDP(λ) and GrHDP, respectively. Therefore, the NSHDP(λ) improves by 13.48% more than the GrHDP by reducing the average number of iterations at various noise levels.

4.3. Third Case: 2-D Maze Problem. Maze navigation has been proposed as an ADP benchmark [44]-[47]. In this case, the NSHDP(λ) is examined in a 2-D maze navigation benchmark with various other approaches (e.g. SARSA(0), Q(λ), ADHDP). Sutton [21] presents a perfect explanation about SARSA(0) and Q(λ). Briefly, the SARSA algorithm is used to find the series state-action pairs by using one-step learning; therefore, SARSA(0) is written. In this case study, the data that the agent uses to learn is: 1) current state vector $x(t) = [x_1(t), x_2(t)]^T$, where x_1 and x_2 are the coordinates of the x axis and the y axis, respectively; 2)selected action $u(t) = [u_1, u_2, u_3, u_4]^T$, where u_1, u_2, u_3 and u_4 are the

Table 1. Performance evaluation of the NSHDP(λ) learning controller when balancing the inverted pendulum dynamic system. The third and fourth columns depict the average number of trials it took to learn to balance the pole for 100 time steps for the GRHDP and the NSHDP(λ), respectively. The average is taken over 100 successful runs for 5 iterations each. * actuators are subjected to noise; $^\#$ sensors are subjected to noise

Noise Type	**Initial Angle ($\theta(0)^\circ$)**	**NSHDP(λ)**	**GrHDP**
Noise free	0.1°	**2.44**	3.33
Noise free	5°	**3.42**	3.95
Uniform* 5%	0.1°	**3.04**	3.37
Uniform* 10%	0.1°	**3.84**	4.49
Uniform* 5%	5°	**2.99**	4.1
Uniform* 10%	5°	**3.15**	4.25
Uniform$^\#$ 5%	0.1°	**3.56**	4.23
Uniform$^\#$ 10%	0.1°	**4.47**	4.62
Uniform$^\#$ 5%	5°	**4.12**	4.42
Uniform$^\#$ 10%	5°	**4.36**	4.51
Uniform$^{*\#}$ 5%	0.1°	**4.44**	4.86

direction of forward, right, backward and left, respectively; 3) external reward $U(t) = 1$ if the agent reaches the target position, $U(t) = -0.001$ if the agent hits the obstacle or exceeds the board and $U(t) = 0$ if the agent moves in a free space. In the SARSA(0) algorithm, the agent takes an action u_i, where $i = 1, 2, ..., 4$, to move from $x(t)$ to the next state $(x(t+1))$, and it gains $U(t)$. There are many strategies to select actions. If the agent always chooses the action with the highest value of the value function (state-action pair value), it is a greedy strategy. An $\epsilon-$greedy strategy selects the greedy action with a probability of (1-ϵ). Otherwise, the agent chooses a random action. In a non-greedy action strategy, the robot selects a random action that the agent is exploring inside the environment. There are other strategies that can be found in the literature [13], [21] and [29]. After the learning trials have been completed, the collected data is put in a lookup-table (Q table). The Q table values of all of the state-action pairs can be updated in the SARSA(0) algorithm learning by the

Bellman formula: $Q(x(t), u_i(t)) = Q(x(t), u_i) + \ell\Big(U(t) + \gamma Q(x(t+1), u_i) - Q(x(t), u_i)\Big)$, where $x(t)$ is the state vector, ℓ is the learning rate and u_i, $i = 1, 2, ..., 4$, is the selected action (the direction of movement). Q(λ)-learning is similar to the SARSA-learning, except: 1) The updating occurs as in SARSA-learning but with a greedy action $\max_{u_i \in u(t)} \big(Q(x(t+1), u_i\big)$ instead of $Q(x(t+1), u_i$ and 2) the eligibility trace is a temporary record to be stored. There are two main steps to update the eligibility traces for the Q table. The frist step involves setting all state action pairs to zero when a non-greedy action is taken. Otherwise, they are declining. Second, the eligibility trace is reset to one if it is identical with the current state-action pair. Updating Q-learning by using only a critic network was presented by [40]. In this study case, only critic neural networks are used to approximate the value function for the n-step as well as the one-step. Algorithm 1 is used to avoid solving discrete-time HJB (4). Algorithm 1 generates and updates a table of Q values according to the state-action pairs (step 2 in Algorithm 1). It also as generates greedy actions ($u_i(t)$) as in step 3 of Algorithm 1. The same procedure is used to generate and update the Q table for ADHDP by using one critic network. In the previous two case studies, the cost value should minimize over time, while here, the agent has to maximize a reward intake in order to solve the maze problem. This paper assumes that an agent starts at an initial location in an environment, which is (0,0) as shown in Fig. 20 with 11 obstacles included. The agent can learn through by online techniques by through interacting with its environment to obtain an optimal collision-free path from the starting point to the target point, which is taken (6,6). The declining $\epsilon-$greedy learning method is used in all approaches as a way of balancing between exploration and exploitation [13] and [21]. The NSHDP(λ) method is evaluated by comparing it with other methods according to Q reference value table. The Q reference evaluation method is presented [49], *et al.*, [50]. The Q reference table is calculated dependent on a distance between the current state location and the target location. All states around the target are set to 1, while the other states are assigned by droning $1/(L + W)$ for each step, where L and W are the maximum number of possible states in the length and width directions, respectively.

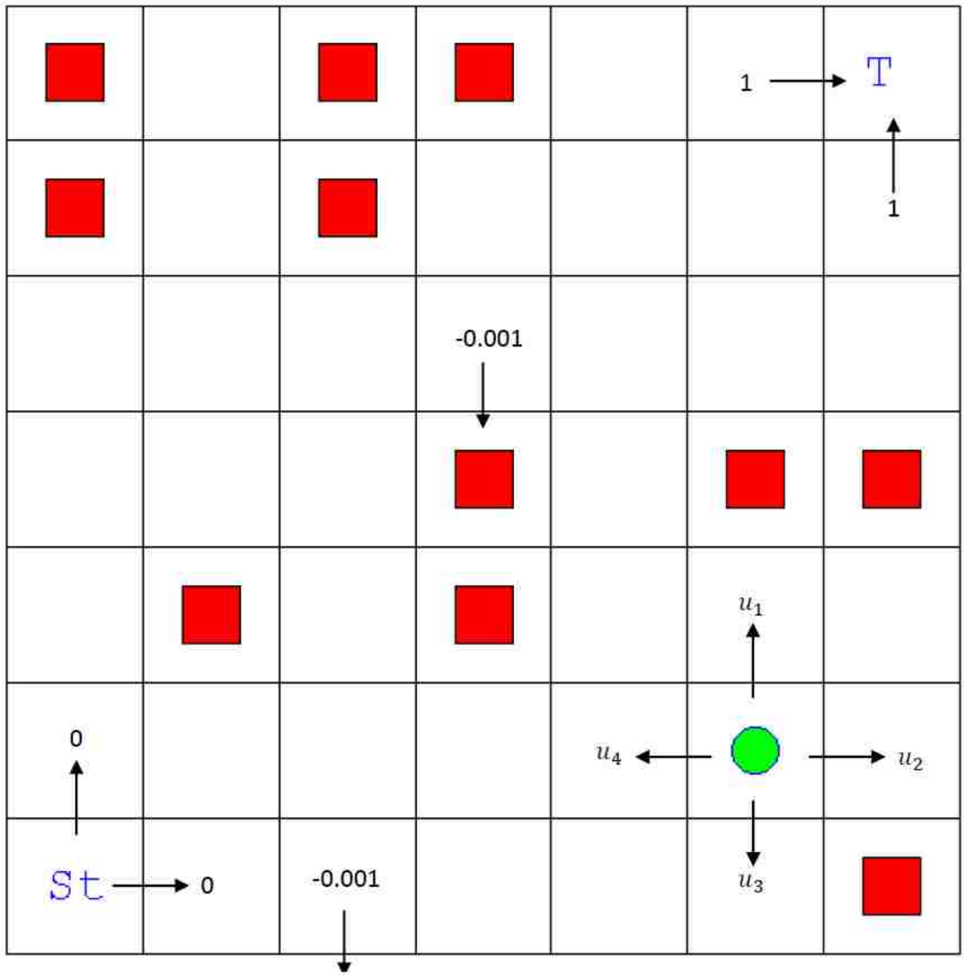


Figure 20. Diagram of a 2-D maze (7 × 7) with obstacles. The point(0,0) represents the initial position. The point (6,6) represents the target position. Eleven obstacles are located at (5,0), (6,0), (2,1), (5,2), (6,2), (2,3), (3,3), (6,3), (3,6), (0,6) and (3,6), which are represented as red squares. Otherwise, the agent can move in the free space. There are three modes where that the agent can receive reward/cost values. First, the agent will a receive reward of 1 when it arrives at the target. Second, the agent will be punished by receiving -0.001 if hits obstacles or passes the bound. Third, the agent will receive a 0 value as a reward in a free space. At any position in the maze, the agent has to select 1 action (direction) out of 4 actions in order to move one step. The 4 actions are forward, right, backward and left, which can be seen in the figure as u_1, u_2, u_3 and u_4, respectively.

The Q reference (Q_{ref}) is calculated as

$$Q_{ref}(x_1, x_2) = 1 - \frac{1}{L+W}\Big(L - x_1 + W - x_2 - 1\Big). \tag{75}$$

Equation (76) is calculated for the desired values of all of all squares of the 2-D maze, including the obstacle square. However, obstacles are used in the 2-D maze benchmark (Fig. 20); therefore, the obstacle square is assigned as zero in Q_{ref} because the agent cannot enter them to update Q_{ref}. Greedy Q table values (Q_{greedy}) are used to calculate the MSE, where $Q_{greedy}((x(t)) = \max_{u_i \in u(t)} \big(Q(x(t), u_i\big)$. The MSE is obtained

$$MSE = \frac{1}{2}\sum_{i=1}^{S_n}\Big(Q_{greedy}(i) - Q_{ref}(i)\Big)^2, \tag{76}$$

where S_n is the number of states of the 2-D maze ($L * W$). The common general parameters that are shared with all testing algorithms are: the learning rate for Q table (ℓ) is 0.01; γ=0.95; λ=0.95 for Q(λ) and the NSHDP(λ) and λ=0 for SARSA(0) and the ADHDP; ϵ-greedy parameter starts at 1 and decreases by $\epsilon = \epsilon * 0.99$ after each episode and stops at ϵ =0.05. There are 20 runs (run loop), and each run has 300 consecutive episodes (episode loop); moreover, at each episode the agent navigates in the maze until reaching to the target (time steps loop). The learning parameters related with only-critic ADHDP and NSHDP(λ) are: the number of neurons in both critic networks ($h_c^0 = h_c^\lambda$) is 24; the initial learning parameters are set as ($\ell_c^0 = \ell_c^\lambda = 0.001$) for both critic networks. The stopping threshold for both critic networks are 110 iterations; the training for either network will be terminated if the error drops under $T_c^0 = T_c^\lambda = 10^{-5}$ or if the number of iterations meets the stopping threshold ($N_c^0 = 110$ and $N_c^\lambda = 110$). Fig. 21 demonstrates that the MSE of the ADHDP and the NSHDP(λ) has a faster dropping rate than those that used the SARSA(0) and Q(λ) methods. Those using the NSHDP(λ) can also converge faster than the ADHDP to achieve the best performance in this test. Both the ADHDP and the NSHDP(λ) fuse together after

Algorithm 2 Markov-Critic Networks To Support Q-Learning in Maze Navigation

```
$\hat{v}^0(t) \leftarrow f_c^0(x_c(t), \hat{\omega}_c^0(t))$
$\hat{v}^\lambda(t) \leftarrow f_c^\lambda(x_c(t), \hat{\omega}_c^\lambda(t))$
$f_c^0$: one-step critic network for value function approximation
$f_c^\lambda$: $n$-step critic network for value function approximation
$x_c(t) = [x(t), u_i]^T$ : input of both critic networks, where $i = 1, 2, \ldots, 4$
$\hat{\omega}_c^0(t)$: weights in one-step critic network
$\hat{\omega}_c^\lambda(t)$: weights in $n$-step critic network
    • Step 1:
        – $Q(x, u_i) = 0$ for all states and actions
        – $\bar{v}^\lambda(x, u_i) = 0$ for all states and actions
        – $t = 0, T_c^0 = T_c^\lambda = 10^{-5}$ with $N_c^0 = N_c^\lambda = 110$ are parameters for stopping learning for both critic networks, initial counters are $C_c^0 = 0$ and $C_c^\lambda = 0$,
          $goal = [6, 6]$, and set all other learning parameters
    • Step 2 (**Policy evaluation**) updating the Q-table by:
        – $\bar{v}^0(x(t), u_i) = U(t) + \gamma\hat{v}^0(t+1)$
        – **while** $(E_c^0(t) > T_c^0)$**or**$(C_c^0 < N_c^0)$ **do**
        –     $\hat{\omega}_c^0(t) = \hat{\omega}_c^0(t) + \triangle\hat{\omega}_c^0(t)$; equations (84) - (18)
        –     $\hat{v}^0(t) = f_c^0(x_c(t), \hat{\omega}_c^0(t))$
        –     $D_{err}^0(t) = \bar{v}^0(x(t), u_i) - \hat{v}^0(t)$
        –     $E_c^0(t) = 0.5\Big(D_{err}^0(t)\Big)^2$
        –     $C_c^0 = C_c^0 + 1$
        – **end while**
        – $\bar{v}^\lambda(x(t), u_i) = U(t) + \gamma\Big(\lambda\hat{v}^\lambda(t+1) + (1-\lambda)\hat{v}^0(t+1)\Big)$
        – **while** $(E_c^\lambda(t) > T_c^\lambda)$**or**$(C_c^\lambda < N_c^\lambda)$ **do**
        –     $\hat{\omega}_c^\lambda(t) = \hat{\omega}_c^\lambda(t) + \triangle\hat{\omega}_c^\lambda(t)$; equations (23) - (25)
        –     $\hat{v}^\lambda(t) = f_c(x_c(t), \hat{\omega}_c^\lambda(t))$
        –     $D_{err}^\lambda(t) = \bar{v}^\lambda(x(t), u_i) - \hat{v}^\lambda(t)$
        –     $E_c^\lambda(t) = 0.5\Big(D_{err}^\lambda(t)\Big)^2$
        –     $C_c^\lambda = C_c^\lambda + 1$
        – **end while**
        – $Q(x(t), u_i) = Q(x(t), u_i) + \ell D_{err}^\lambda(t)$
    • Step 3 (**Policy improvement**) updating content policy while taking the $\epsilon$-learning strategy into a consideration:
        – $u_i = argmax_{u_i \in u(t)}\big(Q(x(t), u_i\big)$
    • Step 4:
        –     **if** $x(t) = goal$ **then**
        –         stop and do other episode
        –     **else**
        –         $x(t) = x(t+1)$
        –         $t = t + 1$
        –         go back to step 2 and continue
        –     **end if**
```

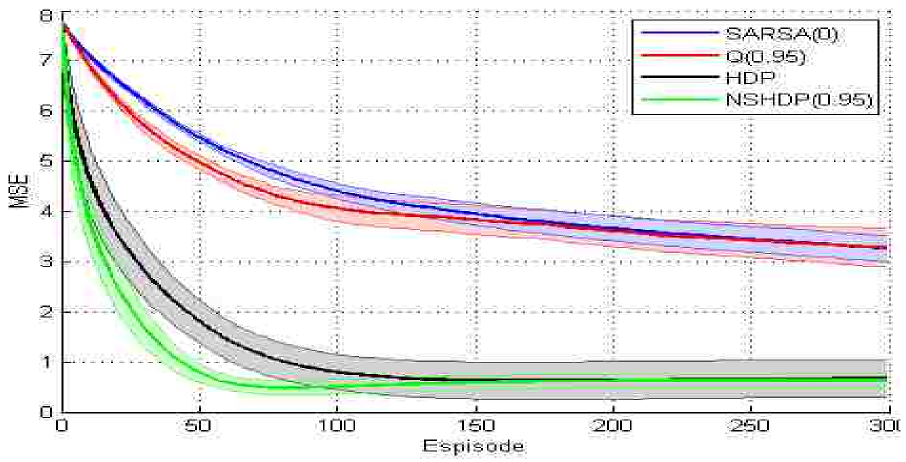


Figure 21. MSE Learning curves for the SARSA(0), Q(λ = 0.95), ADHDP and NSHDP(λ = 0.95) methods for a 2-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The NSHDP(0.95) method learns the fastest and has the MSE compared to other methods.

150 episodes to the sub-optimal values (near zero). Those using the SARSA(0) and Q(λ) methods merge after 180 episodes, and they need more episodes for learning by increasing the exploration mode (decreasing the rate of the ϵ doping value). Another test is related with accumulative rewards over episodes. The goal for this test is to discover which algorithm can calculate the maximum accumulative rewards over each episode the fastest. Fig. 22 illustrates that the NSHDP(λ) has a large accumulative reward value during the exploration episodes. Most likely, the exploration episodes are mostly done between episode 0 and 100, as shown in detail in Fig. 23. Fig. 23 shows the number of steps per episodes over reducing the probability of exploration (by decreasing the ϵ rate value). Therefore, the NSHDP(λ) and the ADHDP converge together to reach the optimal accumulative reward value after episode 100 because of the exploitation navigation behavior, while the Q(λ) and SARSA(0) methods need more exploration episodes to reach the optimal value.

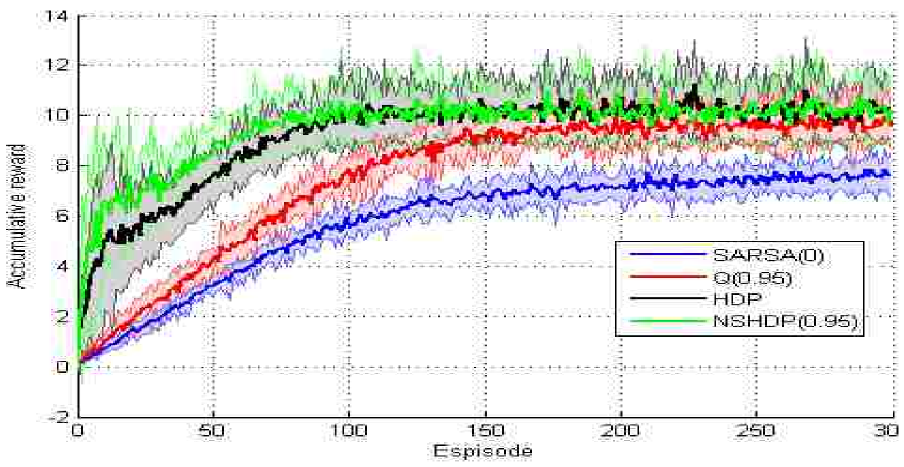


Figure 22. A summation of the accumulative reward for every single episode for the SARSA(0), Q(λ = 0.95), ADHDP and NSHDP(λ = 0.95) methods, which are applied in a 2-D maze navigation benchmark as shown in Fig. 20. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The NSHDP(0.95) method has the largest accumulative reward compared to the other methods in the exploration moving mode. Because the $\epsilon-$greedy deceasing rate learning will reset the accumulative reward value at every episode, the accumulative reward values for the Q(0.95), ADHDP and NSHDP(0.95) are convergence over episodes.

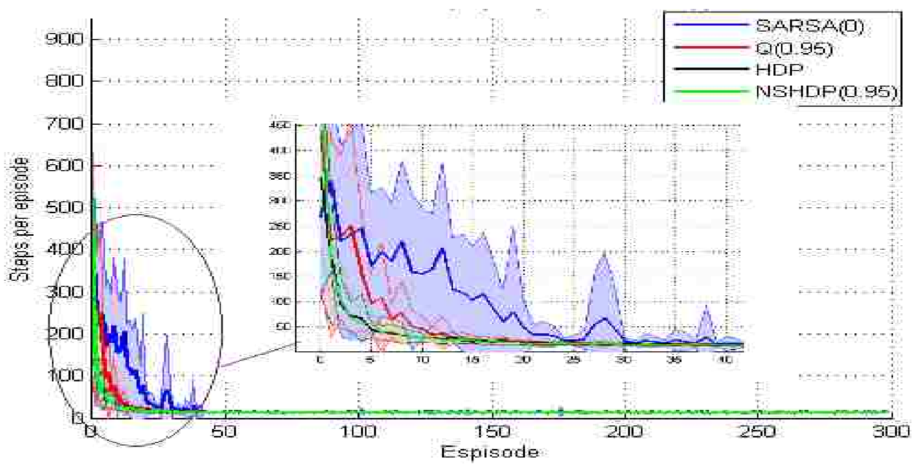


Figure 23. $\epsilon-$greedy learning curves for the SARSA(0), Q(λ = 0.95), ADHDP and NSHDP(λ = 0.95) methods for a 2-D maze navigation benchmark as shown in Fig. 20. These curves represent the number of steps per episode, where the agent returns to the starting point only when it reaches the target cube. The mean values from 20 independent runs are taken for all methods. The shaded color represents the 20 runs, while the solid line represents the mean for all 20 runs. The NSHDP(0.95) and the ADHDP have a nearly similar number of steps over episodes, which are less than both the Q(0.95) and the SARSA(0).

5. CONCLUSION

This paper presents a new ADP architecture (NSHDP(λ)) that consists of one-step critic, *n*-steps critic, and actor networks. NSHDP(λ) presents the combination between TD(0) and TD(λ) via a structure of NSHDP(λ). Good performance is demonstrated by examining a simulation analysis on a nonlinear system and a inverted pendulum benchmark problem in various circumstances, as well as solving a 2-D maze problem. Moreover, this work proves all three networks' stability by using a Lyapunov approach with reasonable conditions. It also shows that the weights and network outputs for both critic networks, as well as the actor network, are bounded. With these results, this paper has made steps to improve ADP learning efficiency, robustness, and control performance.

BIBLIOGRAPHY

[1] X. Zhong, Z. Ni, and H. He, "A Theoretical Foundation of Goal Representation Heuristic Dynamic Programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[2] D. Prokhorov, and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 8, no. 5, pp. 997-1007, Sep. 1997.

[3] P. J. Werbos, *Approximate dynamic programming for real-time control and neural modeling*, Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches, 1992.

[4] R. Bellman, *Dynamic Programming*, Princeton, NJ, USA: Princeton Univ. Press, 1957.

[5] J. Si, and Y. Wang, "Online learning control by association and reinforcement," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 12, no. 2, pp. 264-276, Mar. 2001.

[6] Z. Ni, H. He, X. Zhong, and D. Prokhorov, "Model-Free dual heuristic dynamic programming," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 26, no. 8, pp. 1834-1839, Aug. 2015.

[7] H. He, Z. Ni, and J. Fu, "A three-network architecture for on-line learning and optimization based on adaptive dynamic programming," *Neurocomputing*, vol. 78, no. 1, pp. 3-13, Feb. 2012.

[8] X. Fanga, D. Zhenga, H. He, and Z. Nib, "Data-driven heuristic dynamic programming with virtual reality," *Neurocomputing*, vol. 166, pp. 244-255, Oct. 2015.

[9] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, “GrDHP: A general utility function representation for dual heuristic dynamic programming,” *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 26, no. 3, pp 614-626, Mar. 2015.

[10] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Dual heuristic programming excitation neurocontrol for generators in a multimachine power system,” *IEEE Trans. Applications Industry*, vol. 39, no. 2, pp. 382-394, Mar. 2003.

[11] N. Zhang, and D. C. Wunsch, “A Comparison of Dual Heuristic Programming (DHP) and neural network based stochastic optimization approach on collective robotic search problem,” *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 1, pp. 248-253, Jul. 2003..

[12] C. Lian, and X. Xu, “Motion planning of wheeled mobile robots based on heuristic dynamic programming,” *IEEE Proc. World Congress Intelligent Control and Automation (WCICA)*, pp 576-580, Jul. 2014.

[13] S. Al-Dabooni, and D. Wunsch, “Heuristic dynamic programming for mobile robot path planning based on Dyna approach,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 3723-3730, Jul. 2016.

[14] V. Mnih, A. P. Badia , M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” *International Conference on Machine Learning,* pp. 1928-1937, Jul. 2016.

[15] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. KÃijttler, J. Agapiou, J. Schrittwieser, and J. Quan, “StarCraft II: A New Challenge for Reinforcement Learning,” arXiv preprint arXiv:1708.04782. Aug. 2017.

[16] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, no. 1, pp. 9-44, Aug. 1988.

[17] H. Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, “True Online Temporal-Difference Learning,” *Journal of Machine Learning Research (JMLR)*, vol. 145, no. 17, pp. 1-40, Jan. 2016.

[18] M. Fairbank, and E. Alonso, “Value-Gradient Learning,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, Jun. 2012.

[19] D. Liu, and D. Wang, “Optimal Control of Unknown Nonlinear Discrete-Time Systems Using the Iterative Globalized Dual Heuristic Programming Algorithm,” *Chapter 3 in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, New York, NY, USA: John Wiley and Sons, pp. 52-77, Jan. 2013.

[20] S. Al-Dabooni, and D. Wunsch, “Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 2820-2827, May 2017.

[21] R. S. Sutton, and A. Barto, *Reinforcement Learning: An Introduction*, Cambridge, U.K.: Cambridge Univ. MIT Press, Mar. 1998.

[22] X. Bai, D. Zhao, and J. Yi. "ADHDP (λ) strategies based coordinated ramps metering with queuing consideration," *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 22-27, May 2009.

[23] T. Li, D. Zhao, and J. Yi. "Heuristic dynamic programming strategy with eligibility traces," *IEEE American Control Conference*, pp. 4535-4540, Jun. 2008.

[24] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219-245, Jan. 2000.

[25] X. Bai, D. Zhao, and J. Yi, "Ramp Metering Based on on-line ADHDP(λ) controller," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1847-1852, Jul. 2008.

[26] X. Bai, D. Zhao, and J. Yi, "Coordinated multiple ramps metering based on neuro-fuzzy adaptive dynamic programming," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 241-248, Jul. 2009.

[27] R. S. Sutton, A. R. Mahmood, and M. White, "An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 73, no. 17, pp. 1-29, Jan. 2016.

[28] H. Seijen, and R. S. Sutton, "True Online TD(λ)," *Proceedings of the* 31st *International Conference on Machine Learning*, pp. 692-700, Jan. 2014.

[29] S. Al-Dabooni, and D. Wunsch, "The Boundedness Conditions for Model-Free HDP(λ)," Under reviewing for *IEEE Trans. Neural Netw. and Learn Syst.*

[30] Y. Sokolov, R. Kozma, L. D. Werbos, and P. J. Werbos, "Complete stability analysis of a heuristic approximate dynamic programming control design," *Automatica*, vol. 59, pp. 9-18, Sep. 2015.

[31] F. Liu, J. Sun, J. Si, W. Guo, and S. Mei, "A boundedness result for the direct heuristic dynamic programming," *Neural Networks*, vol. 32, pp. 229-235, Aug. 2012.

[32] Y. Tang, H. He, Z. Ni, X. Zhong, D. Zhao, and X. Xu, "Fuzzy-Based Goal Representation Adaptive Dynamic Programming," *IEEE Trans. on Fuzzy Sys.*, vol. 24, no. 5, pp. 1156 - 1176, Oct. 2016.

[33] F. Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," IEEE Computational Intelligence Magazine, vol. 4, no. 2, pp. 39-47, May 2009.

[34] H. He, Self-Adaptive Systems for *Machine Intelligence*. New York, NY, USA: Wiley, 2011.

[35] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. NewYork, NY, USA: Wiley, Mar. 2012.

[36] S. P. Singh, and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1, pp. 123 - 158, Mar. 1996.

[37] Z. Ni, H. He, J. Wen, and X. Xu, “Goal representation heuristic dynamic programming on maze navigation,” *IEEE Trans. Neural Netw. and Learn Syst.,* vol. 24, no. 12, pp. 2038 - 2050, Dec. 2013.

[38] P. J. Werbos, “Backpropagation through time: What it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550-1560, Oct. 1990.

[39] G. Zhang, M. Y. Hu, B. E. Patuwo, and D. C. Indro, “Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis,” *European Journal of Operational Research*, vol. 116, no. 1, pp. 16-32, Jul. 1999.

[40] Y. Xiao, M. Wasei, P. Hu, P. Wieringa, and F. Dexter, “Dynamic Management of Perioperative Processes: A Modeling and Visualization Paradigm,” *International Federation of Automatic Control (IFAC)*, vol. 39, no. 3, pp. 647 - 652, Jan. 2006.

[41] K. L. Keller, and R. Staelin, “Effects of Quality and Quantity of Information on Decision Effectiveness,” *Journal of Consumer Research*, vol. 14, no. 2, pp. 200 - 213, Sep. 1987.

[42] X. Zhong, Z. Ni, and H. He, “A Theoretical Foundation of Goal Representation Heuristic Dynamic Programming,” *IEEE Trans. Neural Netw. and Learn Syst.,* vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[43] C. Chen, D. Dong, H. Li, J. Chu, and T. Tarn, “Fidelity-Based Probabilistic Q-Learning for Control of Quantum Systems,” *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 5, no. 10, pp. 920-933, May 2014.

[44] P. J. Werbos, and X. Pang, “Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot,” *IEEE Proc. Conf. Systems, Man and Cybernetics (SMC)* , pp. 1764-1769, Oct. 1996.

[45] D. Wunsch, “The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 79-82, Jul. 2000.

[46] R. Ilin, R. Kozma, and P. J. Werbos,“Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator,” *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 929-937, Jun. 2008.

[47] N. Zheng, and P. Mazumder, “Hardware-Friendly Actor-Critic Reinforcement Learning Through Modulation of Spike-Timing-Dependent Plasticity,” *IEEE Trans. on Computers*, vol. 66, no. 2, pp. 299-311, Feb. 2017.

[48] B. Luo, D. Liu, T Huang, and D. Wang, “Model-Free Optimal Tracking Control via Critic-Only Q-Learning,” *IEEE Trans. Neural Netw. and Learn Syst.,* vol. 27, no. 5, pp. 2134-2144, Oct. 2016.

[49] R. Ilin, R. Kozma, and P. Werbos, “Beyond feedforward models trained by backpropagation: A practical training tool for a more efficient universal approximator,” *IEEE Trans. Neural Netw. and Learn Syst.,* vol. 19, no. 6, pp. 929-937, Jan. 2008.

[50] Z. Ni, H. He, J. Wen, and X. Xu, “Goal Representation Heuristic Dynamic Programming on Maze Navigation,” *IEEE Trans. Neural Netw. and Learn Syst.,* vol. 24, no. 12, pp. 2038-2050, Dec. 2013.

VI. AN IMPROVED N-STEP VALUE GRADIENT LEARNING ADAPTIVE DYNAMIC PROGRAMMING ALGORITHM FOR ONLINE LEARNING, WITH CONVERGENCE PROOF AND CASE STUDIES

S. Al-Dabooni and Donald C. Wunsch

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

ABSTRACT

Adaptive dynamic programming (ADP) is the dominant approach to reinforcement learning because of its ability to respond to noise, uncertainty, time lags, disturbances, high dimensionality and other challenges for control, optimization and decision problems. In problems with complex dynamics and challenging state spaces, the dual heuristic programming (DHP) algorithm has been shown theoretically and experimentally to perform well. This was recently extended by an approach called value gradient learning (VGL). VGL was inspired by a version of temporal difference (TD) learning that uses eligibility traces. The eligibility traces create an exponential decay of older observations with a decay parameter (λ). This approach is known as TD(λ) and its DHP extension is known as VGL(λ), where VGL(0) is identical to DHP. VGL has presented convergence and other desirable properties, but is primarily useful for batch learning. Online learning requires an eligibility-trace-work-space matrix, the dimensions of which are $x \times w$, where x is the dimensionality of the input vector and w is the number of weights in the entire neural network. This is not required for the batch learning version of VGL. Since online learning is desirable for many applications, it is important to remove this computational and memory impediment. This paper introduces a dual-critic version of VGL, called N-Step VGL (NSVGL) that does not need the eligibility-trace-work-space matrix, thereby allowing online learning. This paper includes convergence proofs for NSVGL and case studies that demonstrate its superior performance.

Keywords: Adaptive dynamic programming (ADP), value-gradient learning (VGL), online reinforcement learning, eligibility traces, convergence analysis, temporal differences (TD).

1. INTRODUCTION

Adaptive dynamic programming (ADP) is a powerful tool that allows an agent to learn by interacting with its environment to obtain an optimal control policy [1] - [6]. It is a heuristic to solve the Hamilton-Jacobi-Bellman (HJB) equation instead of the Riccati equation [7] - [10]. ADP allows agents to select an action to minimize their long-term cost:

$$J(x_i) = \langle \sum_{k=i}^{\infty} \gamma^{k-i} U\Big(x_k, u(x_k)\Big) \rangle, \tag{1}$$

where $\langle . \rangle$ is the expectation symbol, $J(x_i)$ is a value function (cost-to-go value or long-term cost) for a state vector ($x \in \mathbb{R}^m$) at initial time i, γ is a constant discount factor, and $U(x_k, u(x_k))$ is an instantaneous utility function at time step k for x after applying an action vector $u \in \mathbb{R}^n$. Heuristic dynamic programming (HDP), dual heuristic programming (DHP) and globalized DHP (GDHP) are three fundamental categories for ADP [12] - [15]. These use three approximation function networks, which are actor, critic and model networks that provide decision making, evaluation and prediction, respectively. Because a model network, which predicts the future system state, is included, these ADP categories are model-based ADP [12] - [17]. If the action-dependent (AD) prefix is used (i.e., ADHDP for HDP and ADDHP for DHP), the critic network has the state and the action inputs in these model-free variants. In [18] - [22], model-free ADP designs were presented for online learning.

There are many applications have used ADP. In [23], DHP controlled a turbo-generator more efficiently than HDP. Collective robotic search problems can be solved with improved performance by using DHP as in [24]. Lian and Xu [25] applied HDP to allow a mobile robot to escape from sharp corners. Al-Dabooni and Wunsch [26] applied model-free ADHDP in the Dyna algorithm to obtain an optimal path by using multi-robot navigation in an unknown environment. Other theoretical and practical works in ADP are presented in [27] - [31].

Stability of ADP in the general case is an open problem [32]. Stability of the one-step model-free ADHDP learning approach is introduced by Liu *et al.* [33] and by Werbos *et al.* [32] with critic/actor neural networks and by He *et al.* [34] with critic/reference neural networks and a fuzzy logic controller. Al-Tamimi *et al.* in [35] demonstrate a convergence analysis of value-iteration based on HDP for general discrete-time nonlinear systems. Many other publications on the theoretical analysis and proofs for ADP are shown in [36] - [40]. The GDHP convergence analysis proof and comparison with the HDP and the DHP approaches is presented by Liu [41], [42].

Sutton *et al.* in [43] - [46] show the efficient performance of temporal difference (TD) learning with eligibility trace long-predation parameter denoted by λ. Inspired by [43], Fairbank and Alonso [47], [48] introduced a new ADP algorithm to extend DHP by including λ. They called it value-gradient learning. The value-gradient learning approach was used in [49] to track a reference trajectory under uncertainties by computing the optimal left and right torques for a nonholonomic mobile robot. As reviewed in [12], the ADP technique is used to train an actor network to give optimal actions based on minimizing a value function that is produced from a critic network. Both networks are approximated by using a multilayer perceptron.

In this paper, we present a novel ADP structure, which is called online n-step value-gradient learning with a eligibility trace parameter (NSVGL(λ)). This design uses two critic networks. The first critic network is called a one-step critic network, which is learned based on the gradient of TD. The second critic network is called an n-step critic network, which is learned based on the gradient of TD(λ). The one-step critic network provides a gradient of the one-step value function with respect to system states, while the n-step critic network provides a gradient of the n-step value function with respect to system states. The NSVGL(λ) uses one actor network that learns from both critic networks. In this work, we also provide a theoretical analysis of convergence for the NSVGL(λ). The fundamental contributions of this paper are summarized as follows:

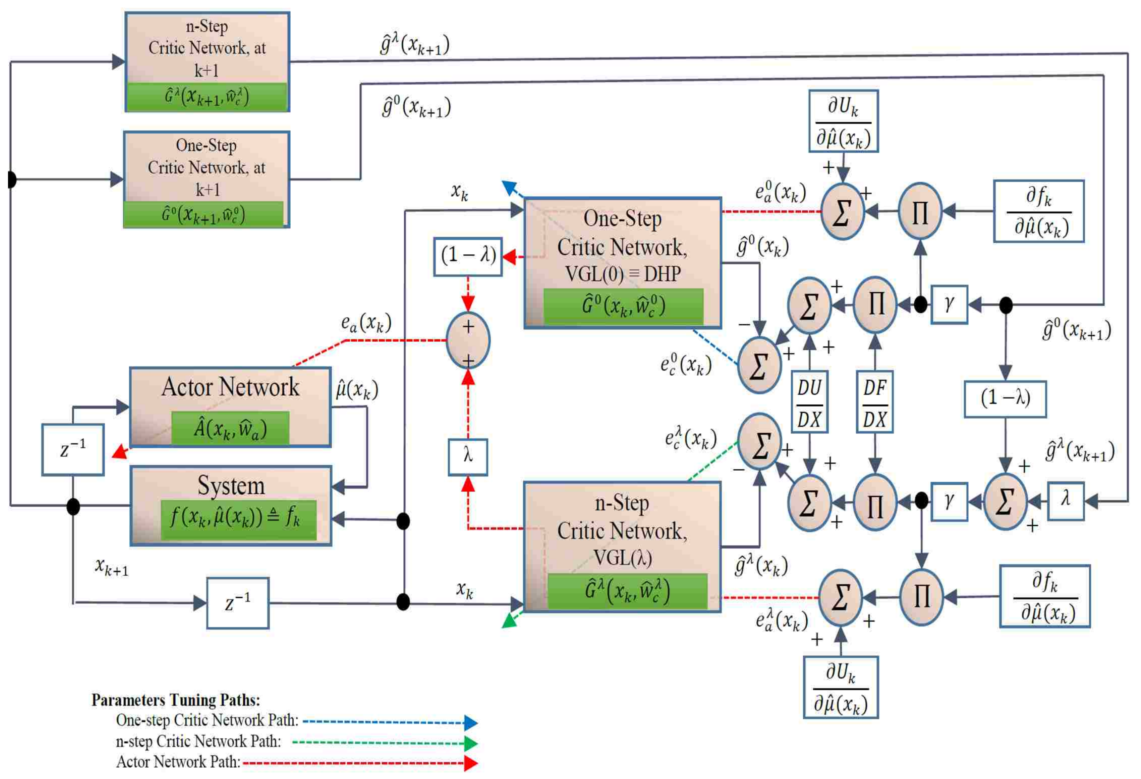
1. The theoretical foundation analysis for NSVGL(λ) architecture is presented designing how the agent receives better information about the control action than traditional DHP. Memory efficiency is provided by NSVGL(λ) via online learning in contest with online VGL(λ) that uses a matrix for eligibility trace parameters to store every signal state trajectory.

2. A theoretical convergence analysis is provided for the NSVGL(λ) structure. Gradients of the one-step and n-step value functions are learned. We demonstrate that both gradients are monotonically nondecreasing and converges to their optimal values.

3. These advantages of NSVGL(λ) are verified by simulation in two case studies.

4. Pseudocode of NSVGL(λ) is provided in this paper.

The schematic structure for NSVGL(λ) is presented in Section 2. Section 3 provides a convergence stability analysis for the NSVGL(λ) design. The neural network architecture design used in NSVGL(λ) is presented in Section 4. The simulation results and the conclusion are presented in Section 6 and Section 7, respectively.


n-Step Critic Network, at k+1
$\hat{G}^{\lambda}(x_{k+1},\hat{w}_c^{\lambda})$
One-Step Critic Network, at k+1
$\hat{G}^{0}(x_{k+1},\hat{w}_c^{0})$
$\hat{g}^{\lambda}(x_{k+1})$
$\hat{g}^{0}(x_{k+1})$
One-Step Critic Network, VGL(0) ≡ DHP
$\hat{G}^{0}(x_k,\hat{w}_c^{0})$
n-Step Critic Network, VGL(λ)
$\hat{G}^{\lambda}(x_k,\hat{w}_c^{\lambda})$
Actor Network
$\hat{A}(x_k,\hat{w}_a)$
System
$f(x_k,\hat{\mu}(x_k)) \triangleq f_k$
z^{-1}
x_k
x_{k+1}
$\hat{\mu}(x_k)$
$e_a(x_k)$
$(1-\lambda)$
λ
$e_a^0(x_k)$
$\hat{g}^0(x_k)$
$e_c^0(x_k)$
$e_c^{\lambda}(x_k)$
$\hat{g}^{\lambda}(x_k)$
$e_a^{\lambda}(x_k)$
$\frac{\partial U_k}{\partial \hat{\mu}(x_k)}$
$\frac{\partial f_k}{\partial \hat{\mu}(x_k)}$
$\frac{DU}{DX}$
$\frac{DF}{DX}$
γ
Σ
Π
Parameters Tuning Paths:
One-step Critic Network Path:
n-step Critic Network Path:
Actor Network Path:

Backpropagation Calculation Errors:

One-step Critic Network Error (Gradient of TD (0) Error): $e_c^0(x_k) = \bar{g}^0(x_k) - \hat{g}^0(x_k)$; $\bar{g}^0(x_k) = \left(\frac{DU}{DX}\right)_k + \gamma \hat{g}^0(x_{k+1}) \left(\frac{Df}{DX}\right)_k$; $\Delta\hat{w}_c^0 = \ell_c^0 [\frac{\partial \hat{g}^0(x_k)}{\partial w_c^0}] e_c^0(x_k)$.

n-step Critic Network Error (Gradient of TD (λ) Error): $e_c^{\lambda}(x_k) = \bar{g}^{\lambda}(x_k) - \hat{g}^{\lambda}(x_k)$; $\bar{g}^{\lambda}(x_k) = \left(\frac{DU}{DX}\right)_k + \gamma\kappa \left(\frac{Df}{DX}\right)_k$; $\Delta\hat{w}_c^{\lambda} = \ell_c^{\lambda} [\frac{\partial \hat{g}^{\lambda}(x_k)}{\partial w_c^0}] e_c^{\lambda}(x_k)$.

Actor Network Error: $e_a(x_k) = \lambda e_a^{\lambda}(x_k) + (1-\lambda) e_a^0(x_k)$; $e_a^0(x_k) = \frac{\partial U_k}{\partial \hat{\mu}(x_k)} + \gamma \hat{g}^0(x_{k+1}) \frac{\partial f_k}{\partial \hat{\mu}(x_k)}$; $e_a^{\lambda}(x_k) = \frac{\partial U_k}{\partial \hat{\mu}(x_k)} + \gamma\kappa \frac{\partial f_k}{\partial \hat{\mu}(x_k)}$; $\Delta\hat{w}_a = \ell_a [\frac{\partial \hat{A}(x_k)}{\partial \hat{w}_a}] e_a(x_k)$,

where $\kappa = \lambda \hat{g}^{\lambda}(x_{k+1}) + (1-\lambda)\hat{g}^0(x_{k+1})$, $\left(\frac{DU}{DX}\right)_k = \frac{\partial U_k}{\partial x_k} + \frac{\partial U_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}$, and $\left(\frac{DF}{DX}\right)_k = \frac{\partial f_k}{\partial x_k} + \frac{\partial f_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}$.

Figure 1. Schematic diagram for the adaptation of a novel online n-step value gradient learning (NSVGL(λ)). Two critic networks and one actor network are used in NSVGL(λ). A combination of two critic networks is presented to speed up the tuning for online learning without needing the initial backup value function and eligibility trace parameters. The weights for the one-step critic network ($\hat{G}^0(x_k, \hat{\omega}_c^0)$) and the n-step critic network ($\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$) are updated according to gradient of TD(0) error (blue dashed line) and gradient of TD(λ) error (green dashed line), respectively. The actor network ($\hat{A}(x_k, \hat{\omega}_a)$) is tuned by two paths (backpropagating errors): one through $\hat{G}^0(x_k, \hat{\omega}_c^0)$ path (e_a^0) and the other through $(\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$ path (e_a^{λ}). This strategy can correlate and combine the information from the two critic networks. These two paths are filtered via the same values of λ, and they are added together to produce a total backpropagating actor error (red dashed line).

2. THE ONLINE NSVGL(λ) STRUCTURE DESIGN

Fig. 1 is shown a main schematic diagram for the online NSVGL(λ) structure. The online NSVGL(λ) approach has five fundamental advantages described in subsections II. A-E:

2.1. Improved Leaning of Temporal Sequences. NSVGL(λ) fills the gaps between a sequence of predicted events and tuning data via TD(λ) learning. TD(λ) is a fundamental contribution in reinforcement learning [44] - [46]. ADP is used to solve the recursive form of the Bellman equation [11], [50]:

$$J^*(x_k) = \min_{u(x_k)}\{U_{k+1} + \gamma J^*(x_{k+1})\}, \tag{2}$$

where J^* denotes the optimal value fuction, and the instantaneous cost $(U(x_k, u(x_k)) \equiv U_k)$ is bounded. As in [52], the Bellman equations for one-step (total discounted future reward depending on one-step TD error) and for two-steps are given as

$$R_k^{(1)} = J(x_k) = U_k + \gamma J(x_{k+1}), \tag{3}$$

and

$$R_k^{(2)} = U_k + \gamma U_{k+1} + \gamma^2 J(x_{k+1}), \tag{4}$$

while for n-steps, $R_k^{(n)}$ is given as follows:

$$R_k^{(n)} = U_k + \gamma U_{k+1} + \ldots + \gamma^{n-1} U_{k+n-1} + \gamma^n J(x_{k+n}) = \sum_{i=k}^{n} \gamma^{i-k} U_i, \tag{5}$$

where $R_k^{(i)}$ denotes the actual return for the value function from k to i. An average of an n-step-return is a technique for accelerating the optimization. For instance, a 4-step average can be done via half of a two-step-return and a half of a four-step-return such that $R_k^{Av(2,4)} = \omega_2 R_k^{(2)} + \omega_4 R_k^{(4)}$, where $\omega_2 = 0.5$ and $\omega_4 = 0.5$. The proportional weights (ω_i),

$i = 1, 2 \ldots$, are positive and sum to 1, which is controlled by λ, where λ represents the proportional weight for the actual return value function. For instance, $\omega_1 = (1-\lambda)$ for a one-step average return ($\lambda = 0$), which is equivalent to $R_k^{(1)}$; for a two-steps average return, $R_k^{Av(1,2)} = \omega_1 R_k^{(1)} + \omega_2 R_k^{(2)}$, where $\omega_1 = (1-\lambda)$ and $\omega_2 = \lambda$; for a three-steps average return, $R_k^{Av(1,2,3)} = \omega_1 R_k^{(1)} + \omega_2 R_k^{(2)} + \omega_3 R_k^{(3)}$, where $\omega_1 = (1-\lambda)$, $\omega_2 = (1-\lambda)\lambda$, and $\omega_3 = \lambda^2$. This procedure continues until n-steps average return. [55] shows a stability proof for selecting λ, which should be $0 < \lambda < 1$ (in contrast to previous literature, which are included bounded from 0 to 1). The λ-return is another name for the average of the n-step-return, which is defined in general as

$$R_k^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_k^{(n)}. \tag{6}$$

By substituting (5) into (5), we obtain:

$$R_k^{\lambda} = (1-\lambda)\left(\sum_{n=1}^{\infty} \lambda^{n-1}\Big[\sum_{i=0}^{n-1} \gamma^i U_{k+i}\Big] + \sum_{n=1}^{\infty} \lambda^{n-1}\gamma^n J(x_{k+n})\right). \tag{7}$$

Expanding and rearranging (6) yields:

$$R_k^{\lambda} = \sum_{n=0}^{\infty} \gamma^n U_{k+n}\Big[\big(\lambda^n + \lambda^{n+1} + \ldots + \lambda^{\infty}\big) - \big(\lambda^{n+1} + \lambda^{n+2} + \ldots + \lambda^{\infty}\big)\Big] + (1-\lambda)\sum_{n=0}^{\infty} \Big[\lambda^n \gamma^{n+1} J(x_{k+n+1})\Big], \tag{8}$$

with $\bar{v}(\infty) = U(\infty)$ (the final target-value at the infinite horizon terminal state). Then, the target value is given as follows:

$$\bar{v}(x_k) = U_k + \gamma\lambda\bar{v}(x_{k+1}) + \gamma(1-\lambda)J(x_{k+1}), \tag{9}$$

(derived in [56]), where $\bar{v}(x_k)$ in (8) is identical to R_k^λ. The Bellman equation for one-step TD learning (TD(0)) with $J = \hat{v}^0$ is given as

$$\hat{v}^0(x_k) = U_k + \gamma\hat{v}^0(x_{k+1}), \tag{10}$$

and for n-step TD learning[5] (TD(λ)) with $\bar{v} = \hat{v}^\lambda$ and $J = \hat{v}^0$, it is given as

$$\hat{v}^\lambda(x_k) = U_k + \gamma\lambda\hat{v}^\lambda(x_{k+1}) + \gamma(1-\lambda)\hat{v}^0(x_{t+1}), \tag{11}$$

where $\hat{v}^0(x_k)$ and $\hat{v}^\lambda(x_k)$ are function approximators for the one-step value function and n-step value function, respectively.

2.2. Improved Exploration\Exploitation Trade-off. Fairbank and Alonso [47] illustrate that the value function has to be learned over all immediately neighboring trajectories in order to reach a locally-extremity optimal trajectory. Therefore, a stochastic exploration (a randomization of trajectory starting points, or policy) should be supplemented with any value function approaches. A gradient of a value function handles this requirement. In VGL, the Bellman equation is solved via a greedy policy, with few trajectories (even a single one), instead of learning in all of the state space. Therefore, the value function approaches have a high probability to fail without exploration, while the gradient of a value function has perfect learning without any exploration, as well as fast stable learning. Therefore, the second advantage of NSVGL(λ) is that it uses the gradient of both the one-step and n-step value functions. VGL has a critic network that estimates the gradients of the one-step value function (J in (3)) with respect to the vector of the state (x_k). VGL(λ) also has one critic network that estimates the derivatives of the n-step target-value function ($\bar{v}(x_k)$ in (8)) with respect to the state vector. In NSVGL(λ), we combine two critic networks (one-step and n-step critics). The gradient of the Bellman equation with respect

[5]We henceforth denote the average of the n-step by only n-step for simplicity

to the system states for the one-step critic is:

$$\frac{\partial \hat{v}^0(x_k)}{\partial x_k} = \frac{\partial U(x_k, \hat{\mu}(x_k)) + \gamma \hat{v}^0(f(x_k, \hat{\mu}(x_k)), \hat{\omega}_c^0)}{\partial x_k}, \tag{12}$$

where $\hat{\mu}(x_k)$ is a control action vector, which is provided from the approximator actor network $\hat{A}(x_k, \hat{\omega}_a)$ with parameter vector $\hat{\omega}_a$, and the general discrete-time nonlinear system model is represented as

$$x_{k+1} = f(x_k, \hat{\mu}(x_k)). \tag{13}$$

For shorthand notation, we denote $U(x_k, \hat{\mu}(x_k)) \equiv U_k$, $\hat{A}(x_k, \hat{\omega}_a) \equiv \hat{A}(x_k)$ and $f(x_k, \hat{\mu}(x_k)) \equiv f_k$. By applying the chain rules in Equation (12), $\partial \hat{v}^0(x_k)/\partial x_k$ is given as

$$\frac{\partial \hat{v}^0(x_k)}{\partial x_k} = \Big(\frac{\partial U_k}{\partial x_k} + \frac{\partial U_k}{\partial \hat{\mu}(x_k)}\frac{\partial \hat{A}(x_k)}{\partial x_k}\Big) + \gamma \frac{\partial \hat{v}^0(x_{t+1})}{\partial x_{t+1}}\Big(\frac{\partial f_k}{\partial x_k} + \frac{\partial f_k}{\partial \hat{\mu}(x_k)}\frac{\partial \hat{A}(x_k)}{\partial x_k}\Big). \tag{14}$$

The one-step critic network $\hat{G}^0(x_k, \hat{\omega}_c^0)$ is a function approximator with parameter vector $\hat{\omega}_c^0$. $\hat{G}^0(x_k, \hat{\omega}_c^0)$ function network provides the estimated gradient of a one-step value approximator function with respect to the system state vector ($\partial \hat{v}^0(x_k)/\partial x_k$). There are two different ways to implement the critic network for DHP [49]: a *scalar critic* method and a *vector critic* method. The *scalar critic* method makes the output critic network equal to $\partial \hat{v}^0(x_k)/\partial x_k$, while the *vector critic* method makes $\partial \hat{v}^0(x_k)/\partial x_k$ equal to $\hat{G}^0(x_k, \hat{\omega}_c^0)$ and provides one-step gradient value function ($\hat{g}^0(x_k)$). We use the *vector critic* method because it is a smooth and stable vector output [47], [48], [51]. In other words, the critic network in the VGL scheme estimates the partial derivatives of the value function with respect to the system's state vector. In order to learn $\hat{G}^0(x_k, \hat{\omega}_c^0)$, the left-hand side of (14) has to be a target value $g^0(x_k)$ (a one-step target value), which is given as

$$g^0(x_k) = \Big(\frac{\partial DU}{\partial DX}\Big)_k + \gamma \hat{g}^0(x_{k+1})\Big(\frac{\partial DF}{\partial DX}\Big)_k, \tag{15}$$

where

$$\Big(\frac{\partial DU}{\partial DX}\Big)_k = \frac{\partial U_k}{\partial x_k} + \frac{\partial U_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}, \tag{16}$$

and

$$\Big(\frac{\partial DF}{\partial DX}\Big)_k = \frac{\partial f_k}{\partial x_k} + \frac{\partial f_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}. \tag{17}$$

During each iteration, a one-step critic error (e_c^0) should be minimized, where e_c^0 is given as

$$e_c^0(x_k) = g^0(x_k) - \hat{g}^0(x_k), \tag{18}$$

where $\hat{g}^0(x_k)$ is an estimated output vector from $\hat{G}^0(x_k, \hat{\omega}_c^0)$ network. The gradient of the Bellman equation with respect to system states for the *n*-step critic network is

$$\frac{\partial \hat{v}^\lambda(x_k)}{\partial x_k} = \frac{\partial}{\partial x_k}\Bigg(U_k + \gamma\Big(\lambda \hat{v}^\lambda(f_{k+1}) + (1-\lambda)\hat{v}^0(f_{k+1})\Big)\Bigg). \tag{19}$$

Applying the chain rules to (9) yields

$$\begin{aligned} \frac{\partial \hat{v}^\lambda(x_k)}{\partial x_k} =& \Big(\frac{\partial U_k}{\partial x_k} + \frac{\partial U_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}\Big) + \gamma\Big(\lambda \frac{\partial \hat{v}^\lambda(x_{k+1})}{\partial x_{k+1}}\big(\frac{\partial f_k}{\partial x_k} + \frac{\partial f_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}\big) + \\ & (1-\lambda)\frac{\partial \hat{v}^0(x_{k+1})}{\partial x_{k+1}}\big(\frac{\partial f_k}{\partial x_k} + \frac{\partial f_k}{\partial \hat{\mu}(x_k)} \frac{\partial \hat{A}(x_k)}{\partial x_k}\big). \end{aligned} \tag{20}$$

The *n*-step critic network $\hat{G}^\lambda(x_k, \hat{\omega}_c^\lambda)$ is a function approximator with parameter vector $\hat{\omega}_c^\lambda$. The $\hat{G}^\lambda(x_k, \hat{\omega}_c^\lambda)$ function network provides the estimated gradient of an *n*-step value approximator function with respect to system state vector ($\partial \hat{v}^\lambda(x_k)/\partial x_k$). During training, the *n*-step critic error (e_c^λ) should be minimized, where e_c^λ is given as

$$e_c^\lambda(x_k) = g^\lambda(x_k) - \hat{g}^\lambda(x_k), \tag{21}$$

where $\hat{g}^{\lambda}(x_k)$ is an estimated output vector from $\hat{G}^{\lambda}(x_k, \hat{\omega}_c^0)$ network, and $g^{\lambda}(x_k)$ is the n-step target value, which is given (according to the *vector critic* method):

$$g^{\lambda}(x_k) = \left(\frac{\partial DU}{\partial DX}\right)_k + \gamma\big(\lambda \hat{g}^{\lambda}(x_{k+1}) + (1-\lambda)\hat{g}^{0}(x_{k+1})\big)\left(\frac{\partial DF}{\partial DX}\right)_k. \tag{22}$$

2.3. Memory Efficiency. A direct implementation without the requirement to store the trajectory is the third advantage of NSVGL(λ). In other words, all the approximator parameters for the one-step and n-step critic networks, as well as the actor network update in a single forward pass of the trajectory. Therefore, higher memory efficiency is achieved than the batch-mode method implementation. Furthermore, NSVGL(λ) has a *forward view* property. The *forward view* is defined by Sutton [43], [52], which is a knowledge of far future rewards for each state. Eligibility trace variables are suggested by Sutton [53] by adding an extra memory variable associated with each state, tuning in backward iteration; therefore, it is known as *backward view*. Many strategies shown in the literature use eligibility trace method. For instance, incremental method provides an incremental mechanism for approximating the *forward view* [43]. A replacing traces method that provides a solution to the accumulating traces by truncating the valve of eligibility trace variably. [53]. Doya [54] derives continuous time domain eligibility traces. Fairbank and Alonso [47] show that the eligibility trace method has high computational complexity in online implementation, which requires and consumes resources to carry out matrix-matrix multiplications over the entire time for the whole trajectory. NSVGL(λ) has no extra required variables for storing eligibility traces for each state. NSVGL(λ) is used only the bootstrapping eligibility trace parameters (λ and γ) to learn the $\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$ network directly via implementation of (22) through minimizing a gradient-TD(λ) error, which will be illustrated in Section 4.

2.4. Improved Actor Network Training. A procedure that is used in NSVGL(λ) to learn the actor network gives a fourth advantage. The actor network (controller) is used to make a system learn to behave optimally. The optimal decision is obtained from a combination between a quantity of current information (the details recent data) and a quality of near history [60], [61]. The $\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$ network has the bootstrapping eligibility traces parameters λ and γ that have the ability to determine a depth and a width of information. For instance, Equation (4) has two future steps that are determined by the λ value via λ-return, and γ controls the amount of summation U value for each step. The $\hat{G}^{0}(x_k, \hat{\omega}_c^{0})$ network provides a gradient of value function that concentrates on recent events. Therefore, a combination between $\hat{G}^{0}(x_k, \hat{\omega}_c^{0})$ and $\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$ provide an optimal decision. The NSVGL(λ) design achieves this combination via learning the actor network. The actor network is tuned by minimizing an actor error ($e_a(x_k)$) by summation of two backpropagating error paths: one path through $\hat{G}^{0}(x_k, \hat{\omega}_c^{0})$ ($e_a^0(x_k)$) and the other path through $\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$ ($e_a^{\lambda}(x_k)$). These two paths are filtered via a similar value of λ. A final actor error $e_a(x_k)$ is calculated by combination between one- and n-step actor errors as follows:

$$e_a(x_k) = \lambda e_a^{\lambda}(x_k) + (1-\lambda) e_a^{0}(x_k), \tag{23}$$

where

$$e_a^{0}(x_k) = \frac{\partial \hat{v}_c^{0}(x_k, \hat{\omega}_c^{0})}{\partial \hat{\mu}(x_k)} = \frac{\partial U_k}{\partial \hat{\mu}(x_k)} + \gamma \hat{g}^{0}(x_{k+1}) \frac{\partial f_k}{\partial \hat{\mu}(x_k)}, \tag{24}$$

and

$$e_a^{\lambda}(x_k) = \frac{\partial \hat{v}_c^{\lambda}(x_k, \hat{\omega}_c^{\lambda})}{\partial (x_k)} = \frac{\partial U_k}{\partial \hat{\mu}(x_k)} + \gamma \big(\lambda \hat{g}^{\lambda}(x_{k+1}) + (1-\lambda)\hat{g}^{0}(x_{k+1})\big) \frac{\partial f_k}{\partial \hat{\mu}(x_k)}. \tag{25}$$

The combination of the critic errors is also found in GDHP [41], [42] to minimize an actor error, which is used to tune a critic network by integrating a critic error with respect to the value function and its derivative.

GDHP uses a merging parameter to adjust how HDP and DHP combine in GDHP, while our approach merges VGL (or DHP) and VGL(λ) (or DHP(λ)) to tune an actor network. A merging parameter, λ, is used to adjust flow information between VGL and VGL(λ).

2.5. Faster Convergence Via Two-Critic Iteration. The fifth advantage is that NSVGL(λ) cooperates iterative learning between $\hat{G}^0(x_k, \hat{\omega}_c^0)$ and $\hat{G}^\lambda(x_k, \hat{\omega}_c^\lambda)$ that can accelerate learning rather than working individually. NSVGL(λ) has an efficient convergence process to obtain the optimal cost function quite rapidly. The following section proves that NSVGL(λ) converges both the cost function and control law sequences to their optimal values.

3. CONVERGENCE PROOF

In this section, we present convergence proofs for the online value-iteration-based NSVGL(λ) to solve the value function and hence the optimal control policy for a nonlinear discrete-time system.

3.1. One-step and *n*-step DT-HJB Equations. Consider an affine nonlinear discrete-time dynamical system described by

$$x_{k+1} = f(x_k) + g(x_k)u(x_k), \tag{26}$$

where k is a discrete time index, $x \in \mathbb{R}^m$ is the state vector, $u(x_k) \in \mathbb{R}^n$ is the control vector, and both $f(x_k)$ and $g(x_k)$ are differential functions with an equilibrium state at $x = 0$ (e.g., $f(0) = g(0) = 0$). Assume that (26) is Lipschitz continuous and stable on a compact set $\Omega_0 \in \mathbb{R}^m$.

Definition 1 cf [35] and [42] (stabilization system): A nonlinear system is stable on a compact set $\Omega_0 \in \mathbb{R}^m$, if a control input $u(x_k) \in \mathbb{R}^n$ exists for all initial conditions $x_0 \in \mathbb{R}^m$

such that the state $x_k \longrightarrow 0$ as $k \longrightarrow 0$. The infinite-horizon cost function is given as

$$J(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} U(x_i, u_i), \tag{27}$$

where the utility function (U) is chosen as the quadratic form, $U(x_i, u_i) = x_i^T Q x_i + u_i^T R u_i$, where $Q = Q^T > 0 \in \mathbb{R}^{m \times m}$ and $R = R^T > 0 \in \mathbb{R}^{\mathfrak{n} \times \mathfrak{n}}$. Then, the objective function to minimize (39) is to find the admissible control action, which is given in the following definition.

Definition 2: If $u(x_k)$ is continuous on a compact set $\Omega_u \in \mathbb{R}^{\mathfrak{n}}$ with $u(0) = 0$, and $J(x_k)$ is finite for $\forall x_0 \in \Omega_0$, then $u(x_k)$ is defined to be admissible with respect to (39). The definition of the infinite horizon optimal cost function is

$$J^*(x_k) = \inf_{u \in \Lambda} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma J^*(x_{k+1}) \right\}, \tag{28}$$

where Λ is a set of all infinite horizon admissible control sequences of x_k. According to Bellman's optimality principle, the optimal cost function in finite horizon that satisfies the discrete time HJB (DT-HJP) is

$$J^*(x_k) = \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma J^*(x_{k+1}) \right\}. \tag{29}$$

For one-step TD learning (TD(0) learning method) with $J = v^0$, (39) is written as

$$v^0(x_k) = x_k^T Q x_k + u_k^T R u_k + \gamma v^0(x_{k+1}). \tag{30}$$

Then, the optimal value function for one-step TD learning is

$$v^{0^*}(x_k) = \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma v^{0^*}(x_{k+1}) \right\}. \tag{31}$$

The gradient of the right-hand side of (31) with represent to u_k gives the optimal control u^*, which satisfies the first-order necessary condition

$$\frac{\partial\big(x_k^T Q x_k + u_k^T R u_k\big)}{\partial u_k} + \gamma\Big(\frac{\partial x_{k+1}}{\partial u_k}\Big)^T \frac{\partial\big(v^{0^*}(x_{k+1})\big)}{\partial x_{k+1}} = 0. \tag{32}$$

Thus,

$$u^*(x_k) = -\frac{\gamma}{2} R^{-1} g^T(x_k) \frac{\partial v^{0^*}(x_{k+1})}{\partial x_{k+1}}. \tag{33}$$

For the n-step TD learning (TD(λ) learning method) with $\bar{v} = v^\lambda$ ($\bar{v}$ in (8)) and $J = v^0$, (39) can be written as

$$v^\lambda(x_k) = x_k^T Q x_k + u_k^T R u_k + \gamma\big(\lambda v^\lambda(x_{k+1}) + (1-\lambda) v^0(x_{k+1})\big). \tag{34}$$

The optimal value function for the n-step TD learning is

$$v^{\lambda^*}(x_k) = \min_{u_k} \Big\{ x_k^T Q x_k + u_k^T R u_k + \gamma\big(\lambda v^{\lambda^*}(x_{k+1}) + (1-\lambda) v^{0^*}(x_{k+1})\big)\Big\}. \tag{35}$$

Because $v^{\lambda^*}(x_k)$, $v^{0^*}(x_k)$ and $J^*(x_k)$ represent the optimal value functions, Equations (31) and (35) are equivalent. The gradient of the right-hand side of (35) with represent to u_k gives the optimal control u^* as

$$0 = \frac{\partial}{\partial u_k}\big(x_k^T Q x_k + u_k^T R u_k\big) + \gamma\Big(\frac{\partial x_{k+1}}{\partial u_k}\Big)^T \frac{\partial}{\partial x_{k+1}}\Big(\lambda v^{\lambda^*}(x_{k+1}) + (1-\lambda) v^{0^*}(x_{k+1})\Big). \tag{36}$$

Therefore, $u^*(x_k)$ for the n-step, which is equivalent to (33) because of $v^{\lambda^*}(x_k) = v^{0^*}(x_k) = J^*(x_k)$ is given as

$$u^*(x_k) = -\frac{\gamma}{2} R^{-1} g^T(x_k)\Big(\lambda \frac{\partial v^{\lambda^*}(x_{k+1})}{\partial x_{k+1}} + (1-\lambda)\frac{\partial v^{0^*}(x_{k+1})}{\partial x_{k+1}}\Big). \tag{37}$$

The optimal control policy with the n-step method is used to prove convergence of the NSVGL(λ) iteration algorithm.

3.2. Derivation of Iteration NSVGL(λ) Algorithm. The initial values of one-step and n-step cost functions are $v_0^0(.) = 0$ and $v_0^\lambda(.) = 0$, respectively. These solve the initial policy μ_0 by

$$\mu_0(x_k) = arg \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma \left(\lambda v_0^\lambda(x_{k+1}) + (1-\lambda) v_0^0(x_{k+1}) \right) \right\}. \tag{38}$$

Then, the iteration on the one-step value function is

$$\begin{aligned} v_1^0(x_k) &= \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma v_0^0(x_{k+1}) \right\}, \\ &= x_k^T Q x_k + \mu_0^T(x_k) R \mu_0(x_k) + \gamma v_0^0(f(x_k) + g(x_k)\mu_0(x_k)), \end{aligned} \tag{39}$$

and the iteration on the n-step value function is

$$\begin{aligned} v_1^\lambda(x_k) &= \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma \left(\lambda v_0^\lambda(x_{k+1}) + (1-\lambda) v_0^0(x_{k+1}) \right) \right\}, \\ &= x_k^T Q x_k + \mu_0(x_k)^T R \mu_0(x_k) + \gamma \Big(\lambda v_0^\lambda(f(x_k) + g(x_k)\mu_0(x_k)) + \\ &\quad (1-\lambda) v_0^0(f(x_k) + g(x_k)\mu_0(x_k)) \Big). \end{aligned} \tag{40}$$

For $i = 1, 2, \ldots$, the action policy, μ_i, in the value iteration algorithm uses a greedy update as follows:

$$\mu_i(x_k) = arg \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma \left(\lambda v_i^\lambda(x_{k+1}) + (1-\lambda) v_i^0(x_{k+1}) \right) \right\}, \tag{41}$$

and the one-step value function, which is

$$\begin{aligned} v_{i+1}^0(x_k) &= \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma v_i^0(x_{k+1}) \right\}, \\ &= x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma v_i^0(f(x_k) + g(x_k)\mu_i(x_k)), \end{aligned} \tag{42}$$

as well as the n-step value function, which is

$$\begin{aligned} v_{i+1}^{\lambda}(x_k) =& \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma \left(\lambda v_i^{\lambda}(x_{k+1}) + (1-\lambda) v_i^0(x_{k+1}) \right) \right\}, \\ =& x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma \Big(\lambda v_i^{\lambda}(f(x_k) + g(x_k)\mu_i(x_k)) + \\ & (1-\lambda) v_i^0(f(x_k) + g(x_k)\mu_i(x_k)) \Big), \end{aligned} \tag{43}$$

where i is the iteration index of the action policy as well as the value functions (one-step and n-step value functions), while k is the time index of state and control for system trajectories.

3.3. Convergence of Iterative NSVGL(λ) Algorithm. The convergence proof depends on the iterations of (42) to the optimal value function $v_i^0 \to J^*$ and (43) to the optimal value function $v_i^{\lambda} \to J^*$ as $i \to \infty$. We prove that both v_i^{λ} and v_i^0 reach the optimal value J^*. Furthermore, we present the convergence of the iteration process of (41) to the optimal control value (i.e., $\mu_i \to u^*$ as $i \to \infty$).

Lemma 1: Let υ_i be any arbitrary sequence of control policies, and μ_i be the control policy sequence described in (41). Let Υ_i^0 and Υ_i^{λ} be

$$\Upsilon_{i+1}^0(x_k) = x_k^T Q x_k + \upsilon_i^T(x_k) R \upsilon_i(x_k) + \gamma \Upsilon_i^0(f(x_k) + g(x_k)\upsilon_i(x_k)), \tag{44}$$

and

$$\begin{aligned} \Upsilon_{i+1}^{\lambda}(x_k) =& x_k^T Q x_k + \upsilon_i^T(x_k) R \upsilon_i(x_k) + \gamma \Big(\lambda v_i^{\lambda}(f(x_k) + g(x_k)\upsilon_i(x_k)) + \\ & (1-\lambda) \Upsilon_i^0(f(x_k) + g(x_k)\upsilon_i(x_k)) \Big), \end{aligned} \tag{45}$$

respectively. v_i^0 and v_i^{λ} are defined as in (42) and (43), respectively. If $v_0^0 = v_0^{\lambda} = \Upsilon_0^0 = \Upsilon_0^{\lambda} = 0$, then $v_{i+1}^0(x_k) \le \Upsilon_{i+1}^0(x_k)$ and $v_{i+1}^{\lambda}(x_k) \le \Upsilon_{i+1}^{\lambda}(x_k)$, $\forall i$.

Proof: Because $\mu_i(x_k)$ minimizes the right-hand side of (42) with respect to the control μ_i, and because $v_0^0 = \Upsilon_0^0 = 0$, then by induction it follows that $v_{i+1}^0(x_k) \le \Upsilon_{i+1}^0(x_k), \forall i$.

Because $\mu_i(x_k)$ minimizes the right-hand side of (43) with respect to the control μ_i, and because $v_0^{\lambda} = \Upsilon_0^{\lambda} = 0$, then by induction it follows that $v_{i+1}^{\lambda}(x_k) \le \Upsilon_{i+1}^{\lambda}(x_k), \forall i$. ■

Lemma 2: Let the one-step value function v_i^0 be defined as in (42). If the system is controllable, then there is an upper bound $Y1$ such that $0 \leq v_i^0 \leq Y1$, $\forall i$. Similarly, set the n-step value function v_i^λ be defined as in (43). If the system is controllable, then there is an upper bound $Y2$ such that $0 \leq v_i^\lambda \leq Y2$, $\forall i$

Proof: First, similar to [42], we prove the one-step value function v_i^0, which is defined as in (42), has an upper bound $Y1$ such that $0 \leq v_i^0 \leq Y1$, $\forall i$ as follows: Let $\zeta_i(x_k)$ be a sequence of admissible control policies, and $v_0^0(.) = \Lambda_0^0(.) = 0$, where v_i^0 is defined and updated as in (42) and Λ_i^0 is defined and updated by

$$\Lambda_{i+1}^0(x_k) = x_k^T Q x_k + \zeta_i^T(x_k) R \zeta_i(x_k) + \gamma \Lambda_i^0(x_{k+1}), \tag{46}$$

where $x_{k+1} = f(x_k) + g(x_k)\zeta_i(x_k)$. Because the DT-HJB equation develops backward ($i = i - 1$) for the next time index ($k = k + 1$), then

$$\Lambda_i^0(x_{k+1}) = x_{k+1}^T Q x_{k+1} + \zeta_{i-1}^T(x_{k+1}) R \zeta_{i-1}(x_{k+1}) + \gamma \Lambda_{i-1}^0(x_{k+2}), \tag{47}$$

where $x_{k+2} = f(x_{k+1}) + g(x_{k+1})\zeta_{i-1}(x_{k+1})$.

Equation (46) expands with $U(x_k, \zeta_i(x_k)) = x_k^T Q x_k + \zeta_i^T(x_k) R \zeta_i(x_k)$ to be

$$\begin{aligned}
\Lambda_{i+1}^0(x_k) =& U(x_k, \zeta_i(x_k)) + \gamma \Lambda_i^0(x_{k+1}) \\
=& U(x_k, \zeta_i(x_k)) + \gamma \Big(U(x_{k+1}, \zeta_{i-1}(x_{k+1})) + \gamma \Lambda_{i-1}^0(x_{k+2}) \Big) \\
=& U(x_k, \zeta_i(x_k)) + \gamma \Big(U(x_{k+1}, \zeta_{i-1}(x_{k+1})) + \gamma \Big(U(x_{k+2}, \\
& \zeta_{i-2}(x_{k+2})) + \gamma \Lambda_{i-2}^0(x_{k+3}) \Big) \Big) \\
& \vdots \\
=& U(x_k, \zeta_i(x_k)) + \gamma U(x_{k+1}, \zeta_{i-1}(x_{k+1})) + \gamma^2 U(x_{k+2}, \\
& \zeta_{i-2}(x_{k+2})) + \ldots + \gamma^i U(x_{k+i}, \zeta_0(x_{k+i})) + \gamma^{i+1} \Lambda_0^0(x_{k+i+1}),
\end{aligned} \tag{48}$$

where $\Lambda_0^0(x_{k+i+1}) = 0$ $\Big($because $\Lambda_0^0(.) = 0\Big)$. Therefore

$$\begin{aligned}\Lambda_{i+1}^0(x_k) &= \sum_{j=0}^{i} \gamma^j U(x_{k+j}, \zeta_{i-j}(x_{k+j})) \\ &= \sum_{j=0}^{i} \gamma^j \Big(x_{k+j}^T Q x_{k+j} + \zeta_{i-j}^T(x_{k+j}) R \zeta_{i-j}(x_{k+j})\Big).\end{aligned} \tag{49}$$

Taking $\lim_{i\to\infty}$ of (49), we obtain

$$\Lambda_{i+1}^0(x_k) \le \lim_{i\to\infty} \sum_{j=0}^{i} \gamma^j \Big(x_{k+j}^T Q x_{k+j} + \zeta_{i-j}^T(x_{k+j}) R \zeta_{i-j}(x_{k+j})\Big). \tag{50}$$

Because Definition 1 shows that $\zeta_i(x_k)$ is an admissible control policies and $x_k \longrightarrow 0$ as $k \longrightarrow \infty$, there exists an upper bound $Y1$ such that

$$\forall i : \Lambda_{i+1}^0(x_k) \le \lim_{i\to\infty} \sum_{j=0}^{i} \gamma^j U(x_{k+j}, \zeta_{i-j}(x_{k+j})) = Y1. \tag{51}$$

Therefore, by using Lemma 1 with $\mu_i(x_k) = \zeta_i(x_k)$ and $\Upsilon_{i+1}^0(x_k) = \Lambda_{i+1}^0(x_k)$, we obtain $0 \le v_{i+1}^0(x_k) \le \Lambda_{i+1}^0(x_k) \le Y1$, $\forall i$. Second, we will prove the n-step value function v_i^λ, which is defined as in (43), has an upper bound Y such that $0 \le v_i^\lambda \le Y2$, $\forall i$ as follows: Let $\zeta_i(x_k)$ be a sequence of admissible control policy, and $v_0^0(.) = v_0^\lambda(.) = \Lambda_0^0(.) = \Lambda_0^\lambda(.) = 0$, where v_i^0 and v_i^λ are defined and updated as in (42) and (43), respectively, while Λ_i^0 is defined and updated as in (46), and Λ_i^λ is defined and updated by

$$\Lambda_{i+1}^\lambda(x_k) = x_k^T Q x_k + \zeta_i^T(x_k) R \zeta_i(x_k) + \gamma\Big(\lambda \Lambda_i^\lambda(x_{k+1}) + (1-\lambda)\Lambda_i^0(x_{k+1})\Big), \tag{52}$$

where $x_{k+1} = f(x_k) + g(x_k)\zeta_i(x_k)$. Because the DT-HJB equation develops backward for the next time index and $U(x_k, \zeta_i(x_k)) = x_k^T Q x_k + \zeta_i^T(x_k) R \zeta_i(x_k)$, then equation (52) expands to be

$$\begin{aligned}\Lambda_{i+1}^{\lambda}(x_k) =&U(x_k,\zeta_i(x_k))+\gamma\Bigg[\lambda\Lambda_i^{\lambda}(x_{k+1})+(1-\lambda)\Lambda_i^0(x_{k+1})\Bigg]=U(x_k,\zeta_i(x_k))+\\ &\gamma\Bigg[\lambda\Bigg(U(x_{k+1},\zeta_{i-1}(x_{k+1}))+\gamma\Big[\lambda\Lambda_{i-1}^{\lambda}(x_{k+2})+(1-\lambda)\Lambda_{i-1}^0(x_{k+2})\Big]+\\ &(1-\lambda)\Lambda_i^0(x_{k+1})\Bigg)\Bigg]=U(x_k,\zeta_i(x_k))+\gamma\Bigg[\lambda\Bigg(U(x_{k+1},\zeta_{i-1}(x_{k+1}))+\\ &\gamma\Big[\big(U(x_{k+2},\zeta_{i-2}(x_{k+2}))+\gamma\big[\lambda\Lambda_{i-2}^{\lambda}(x_{k+3})+(1-\lambda)\Lambda_{i-2}^0(x_{k+3})\big]\big)+\\ &(1-\lambda)\Lambda_{i-1}^0(x_{k+2})\Big]+(1-\lambda)\Lambda_i^0(x_{k+1})\Bigg)\Bigg],\\ =&U(x_k,\zeta_i(x_k))+(\gamma\lambda)U(x_{k+1},\zeta_{i-1}(x_{k+1}))+(\gamma\lambda)^2U(x_{k+2},\zeta_{i-2}(x_{k+2}))+\\ &(\gamma\lambda)^3\Lambda_{i-2}^{\lambda}(x_{k+3})+(\gamma\lambda)^2\gamma(1-\lambda)\Lambda_{i-2}^0(x_{k+3})+(\gamma\lambda)\gamma(1-\lambda)\\ &\Lambda_{i-1}^0(x_{k+2})+\gamma(1-\lambda)\Lambda_i^0(x_{k+1}),\\ &\vdots\\ =&U(x_k,\zeta_i(x_k))+(\gamma\lambda)U(x_{k+1},\zeta_{i-1}(x_{k+1}))+(\gamma\lambda)^2U(x_{k+2},\zeta_{i-2}(x_{k+2}))+\\ &\ldots+(\gamma\lambda)^iU(x_{k+i},\zeta_0(x_{k+i}))+(\gamma\lambda)^{i+1}\Lambda_0^{\lambda}(x_{k+i+1})+(\gamma\lambda)^i\gamma(1-\lambda)\\ &\Lambda_0^0(x_{k+i+1})+(\gamma\lambda)^{i-1}\gamma(1-\lambda)\Lambda_1^0(x_{k+i})+\ldots+(\gamma\lambda)\gamma(1-\lambda)\Lambda_{i-1}^0(x_{k+2})\\ &+\gamma(1-\lambda)\Lambda_i^0(x_{k+1}),\end{aligned} \tag{53}$$

where $\Lambda_0^{\lambda}(x_{k+i+1})=0$, $\Big(\text{because } \Lambda_0^{\lambda}(.)=0\Big)$, and $\Lambda_0^0(x_{k+i+1})=0$, $\Big(\text{because } \Lambda_0^0(.)=0\Big)$. Therefore,

$$\begin{aligned}\Lambda_{i+1}^{\lambda}(x_k)=&\sum_{j=0}^{i}\Bigg[(\gamma\lambda)^jU(x_{k+j},\zeta_{i-j}(x_{k+j}))\Bigg]+\gamma(1-\lambda)\sum_{j=0}^{i-1}\Bigg[(\gamma\lambda)^j\\ &\Lambda_{i-j}^0(x_{k+j+1})\Bigg].\end{aligned} \tag{54}$$

According to (49), Equation (54) is rewritten as

$$\Lambda_{i+1}^{\lambda}(x_k) = \sum_{j=0}^{i}\left[(\gamma\lambda)^j U(x_{k+j}, \zeta_{i-j}(x_{k+j}))\right] + \gamma(1-\lambda)\sum_{j=0}^{i-1}\left[(\gamma\lambda)^j \sum_{l=0}^{i-j-1}\left[(\gamma^l U(x_{k+l}, \zeta_{i-j-l-1}(x_{k+l}))\right]\right]. \tag{55}$$

Because Definition 1 demonstrates that $\zeta_i(x_k)$ is an admissible control policy and $x_k \to 0$ as $k \to \infty$, there exists an upper bound $Y2$ such that

$$\forall i : \Lambda_{i+1}^{\lambda}(x_k) \leq \lim_{i\to\infty}\Bigg(\sum_{j=0}^{i}\left[(\gamma\lambda)^j\left(x_{k+j}^T Q x_{k+j} + \zeta_{i-j}^T(x_{k+j})R\zeta_{i-j}(x_{k+j})\right)\right] + \gamma(1-\lambda)\sum_{j=0}^{i-1}\left[(\gamma\lambda)^j\sum_{l=0}^{i-j-1}\left[\gamma^l\left(x_{k+l}^T Q x_{k+l} + \zeta_{i-j-l-1}^T(x_{k+l}) R\zeta_{i-j-l-1}(x_{k+l})\right)\right]\right]\Bigg) = Y2. \tag{56}$$

Therefore, by using Lemma 1 with $\mu_i(x_k) = \zeta_i(x_k)$ and $\Upsilon_{i+1}^{\lambda}(x_k) = \Lambda_{i+1}^{\lambda}(x_k)$, we obtain $0 \leq v_{i+1}^{\lambda}(x_k) \leq \Lambda_{i+1}^{\lambda}(x_k) \leq Y2, \forall i$. ■

Lemma 3: If Lemma 2 holds, then both upper bounds ($Y1$ and $Y2$) are equal, such that $\lim_{i\to\infty}\Lambda_i^0(x_k) = \lim_{i\to\infty}\Lambda_i^{\lambda}(x_k) = Y1 = Y2 = Y$.

Proof: Let $U(.) = 1$ for $\forall j$ (or, let $U(.)$ be any constant value for the entire trajectory), then the difference between (51) and (56) is zero, such that $\varrho = \lim_{i\to\infty}\Lambda_{i+1}^{\lambda}(x_k) - \lim_{i\to\infty}\Lambda_{i+1}^{0}(x_k) = 0$, which is illustrated as follows:

$$\begin{aligned}\varrho &= \lim_{i\to\infty}\left(\sum_{j=0}^{i}\left[(\gamma\lambda)^j\right] + \gamma(1-\lambda)\sum_{j=0}^{i-1}\left[(\gamma\lambda)^j\sum_{l=0}^{i-j-1}\left[\gamma^l\right]\right]\right) - \lim_{i\to\infty}\left(\sum_{j=0}^{i}\left[(\gamma)^j\right]\right),\\ &= \lim_{i\to\infty}\left(\frac{1-(\gamma\lambda)^{i+1}}{1-\gamma\lambda} + \frac{(1-\lambda)\gamma}{1-\gamma}\left[\frac{1-(\gamma\lambda)^i}{1-\gamma\lambda} - \frac{(1-\lambda)\gamma^i}{1-\lambda}\right]\right) - \lim_{i\to\infty}\left(\frac{1-\gamma^{i+1}}{1-\gamma}\right)\\ &= \frac{1-\gamma\lambda}{1-\gamma-\gamma\lambda+\gamma^2\lambda} - \frac{1}{1-\gamma} = 0.\end{aligned} \tag{57}$$

Theorem 1: Consider sequences for the action policy μ_i, the one-step value function v_i^0 and the n-step value function v_i^λ defined as in (41), (42) and (43), respectively. Then, v_i^0 and v_i^λ are non-decreasing sequences such that $v_i^0 \leq v_{i+1}^0$ and $v_i^\lambda \leq v_{i+1}^\lambda$, $\forall i$.

Proof: First, we will prove $v_i^0 \leq v_{i+1}^0$. Define μ_i and v_i^0 from (41) and (42), respectively. A new one-step value function Γ_i^0 is defined by

$$\Gamma_{i+1}^0(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma \Gamma_i^0(x_{k+1}), \tag{58}$$

where $x_{k+1} = f(x_k) + g(x_k)\mu_i(x_k)$ and $\Gamma_0^0 = v_0^0 = 0$. Mathematical induction illustrates that $\Gamma_i^0 \leq v_{i+1}^0$ as follows: For $i = 0$, consider

$$v_1^0(x_k) = x_k^T Q x_k + \mu_0^T(x_k) R \mu_0(x_k) + \gamma v_0^0(x_{k+1}), \tag{59}$$

with $v_0^0(.) = \Gamma_0^0(.) = 0$. Then,

$$v_1^0(x_k) - \Gamma_0^0(x_k) = x_k^T Q x_k + \mu_0^T(x_k) R \mu_0(x_k) \geq 0. \tag{60}$$

Thus, $\Gamma_0^0(x_k) \leq v_1^0(x_k)$. A similar procedure is continuous, and assuming it reaches and holds for $i-1$, then $\Gamma_{i-1}^0(x_k) \leq v_i^0(x_k)$. For i, considering that

$$v_{i+1}^0(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma v_i^0(x_{k+1}), \tag{61}$$

and

$$\Gamma_i^0(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma \Gamma_{i-1}^0(x_{k+1}), \tag{62}$$

we obtain

$$v_{i+1}^0(x_k) - \Gamma_i^0(x_k) = \gamma\Big(v_i^0(x_{k+1}) - \Gamma_{i-1}^0(x_{k+1})\Big) \geq 0. \tag{63}$$

Then, $\Gamma_i^0(x_k) \leq v_{i+1}^0(x_k)$. Because of $\Gamma_i^0(x_k) \geq v_i^0(x_k)$ according to Lemma 1 and $\Gamma_i^0(x_k) \leq v_{i+1}^0(x_k)$, we obtain $v_i^0(x_k) \leq v_{i+1}^0(x_k)$.

Second, we will prove $v_i^\lambda \leq v_{i+1}^\lambda$. Define μ_i and v_i^λ from (41) and (43), respectively. A new n-step value function Γ_i^λ is defined by

$$\Gamma_{i+1}^\lambda(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma\Big(\lambda \Gamma_i^\lambda(x_{k+1}) + (1-\lambda)\Gamma_i^0(x_{k+1})\Big), \tag{64}$$

where $x_{k+1} = f(x_k) + g(x_k)\mu_i(x_k)$. Mathematical induction illustrates that $\Gamma_i^\lambda \leq v_{i+1}^\lambda$ as follows: For $i = 0$,

$$v_1^\lambda(x_k) = x_k^T Q x_k + \mu_0^T(x_k) R \mu_0(x_k) + \gamma\Big(\lambda v_0^\lambda(x_{k+1}) + (1-\lambda) v_0^0(x_{k+1})\Big), \tag{65}$$

with $\Gamma_0^0 = v_0^0 = \Gamma_0^\lambda = v_0^\lambda = 0$. Then,

$$v_1^\lambda(x_k) - \Gamma_0^\lambda(x_k) = x_k^T Q x_k + \mu_0^T(x_k) R \mu_0(x_k) \geq 0. \tag{66}$$

Thus, $\Gamma_0^\lambda(x_k) \leq v_1^\lambda(x_k)$. A similar procedure is continuous, and assuming it reaches and holds for $i-1$, then $\Gamma_{i-1}^\lambda(x_k) \leq v_i^\lambda(x_k)$. We note that $\Gamma_{i-1}^0(x_k) \leq v_i^0(x_k)$. Therefore, for i, after considering that

$$v_{i+1}^\lambda(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma\Big(\lambda v_i^\lambda(x_{k+1}) + (1-\lambda) v_i^0(x_{k+1})\Big), \tag{67}$$

and

$$\Gamma_i^\lambda(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k) + \gamma\Big(\lambda \Gamma_{i-1}^\lambda(x_{k+1}) + (1-\lambda)\Gamma_{i-1}^0(x_{k+1})\Big), \tag{68}$$

we obtain

$$v_{i+1}^{\lambda}(x_k) - \Gamma_i^{\lambda}(x_k) = \gamma\Big(\lambda v_i^{\lambda}(x_{k+1}) + (1-\lambda)v_i^0(x_{k+1}) - \lambda\Gamma_{i-1}^{\lambda}(x_{k+1}) - (1-\lambda) \\ \Gamma_{i-1}^{0}(x_{k+1})\Big) \geq 0. \tag{69}$$

Then, $\Gamma_i^{\lambda}(x_k) \leq v_{i+1}^{\lambda}(x_k)$. Because $\Gamma_i^{\lambda}(x_k) \geq v_i^{\lambda}(x_k)$ according to Lemma 1 and $\Gamma_i^{\lambda}(x_k) \leq v_{i+1}^{\lambda}(x_k)$, we obtain $v_i^{\lambda}(x_k) \leq v_{i+1}^{\lambda}(x_k)$. ■

Theorem 2: The one-step value function (v_i^0) and the n-step value function (v_i^{λ}) can be defined as in (42) and (43), respectively, and $v_i^0 = v_i^{\lambda} = 0$. Define $\lim_{i\to\infty} v_i^0(x_k) = v_\infty^0(x_k)$ and $\lim_{i\to\infty} v_i^{\lambda}(x_k) = v_\infty^{\lambda}(x_k)$ as the infinite limits of the one-step and n-step value functions. With controllable system states, both v_i^0 and v_i^{λ} are limited by $J^*(x_k)$, where J^* is described in (28). That is, $v_\infty^0(x_k) = v_\infty^{\lambda}(x_k) = J^*$.

Proof: First, let $\zeta_i(x_k)$ be a sequence of admissible control policies, and $v_0^0(.) = H_0^0(.) = 0$, where v_i^0 is defined and updated as in (42) and H_{i+1}^0 is defined and updated by

$$H_{i+1}^0(x_k) = x_k^T Q x_k + \zeta_i^T(x_k) R \zeta_i(x_k) + \gamma H_i^0(x_{k+1}). \tag{70}$$

From the first part of Lemma 1 and Lemma 2, we have $v_{i+1}^0(x_k) \leq H_{i+1}^0(x_k) \leq Y1$, $\forall i$. By defining $\lim_{i\to\infty} H_i^0(x_k) = H_\infty^0(x_k)$, we obtain $v_\infty^0(x_k) \leq H_\infty^0(x_k) \leq Y1$ for all admissible control policy sequences. If $\zeta(x_k) = u^*(x_k)$, and the first part of Lemma 2 is applied, we obtain

$$Y1 = \lim_{i\to\infty} \sum_{j=0}^{i} \left(\gamma^j \left(x_k^T Q x_k + \zeta_{i-j}^T(x_k) R \zeta_{i-j}(x_k)\right)\right), \tag{71}$$

such that $J^*(x_k) \leq Y1$, where J^* is described in (28). Therefore, from first part of Lemma 2 and $v_\infty^0(x_k) \leq H_\infty^0(x_k) \leq Y1$, we obtain $v_\infty^0(x_k) \leq J^*(x_k)$. Because $J^*(x_k)$ is the infimum of the cost function that derives from all other admissible control sequences, we conclude that $J^*(x_k) \leq v_\infty^0(x_k)$. This implies that $J^*(x_k) \leq v_\infty^0(x_k) \leq J^*(x_k)$, and hence, $v_\infty^0(x_k) = J^*(x_k)$.

Second, let $\zeta_i(x_k)$ be a sequence of admissible control policies, and $v_0^0(.) = v_0^\lambda(.) = H_0^0(.) = H_0^\lambda(.) = 0$, where v_i^0 and v_i^λ are defined and updated as in (42) and (43). Equation (70) is used to update H_{i+1}^0 and H_{i+1}^λ:

$$H_{i+1}^\lambda(x_k) = x_k^T Q x_k + \zeta_i^T(x_k) R \zeta_i(x_k) + \gamma\Big(\lambda H_i^\lambda(x_{k+1}) + (1-\lambda) H_i^0(x_{k+1})\Big). \tag{72}$$

From the second part of both Lemma 1 and Lemma 2, we have $v_{i+1}^\lambda(x_k) \le H_{i+1}^\lambda(x_k) \le Y2$, $\forall i$. By defining $\lim_{i\to\infty} H_i^\lambda(x_k) = H_\infty^\lambda(x_k)$, we obtain $v_\infty^\lambda(x_k) \le H_\infty^\lambda(x_k) \le Y2$ for all admissible control policy sequences. If $\zeta(x_k) = u^*(x_k)$ with applying the second part of Lemma 2, then

$$\begin{aligned} Y2 = \lim_{i\to\infty}\Bigg(&\sum_{j=0}^{i}\Bigg[(\gamma\lambda)^j\Big(x_{k+j}^T Q x_{k+j} + \zeta_{i-j}^T(x_{k+j}) R \zeta_{i-j}(x_{k+j})\Big)\Bigg] + \gamma(1-\lambda)\sum_{j=0}^{i-1}\Bigg[\\ &(\gamma\lambda)^j \sum_{l=0}^{i-j-1}\Bigg[\gamma^l\Big(x_{k+l}^T Q x_{k+l} + \zeta_{i-j-l-1}^T(x_{k+l}) R \zeta_{i-j-l-1}(x_{k+l})\Big)\Bigg]\Bigg]\Bigg), \end{aligned} \tag{73}$$

such that $J^*(x_k) \le Y2$, where J^* is described in (28). Therefore, from the second part of Lemma 2 and $v_\infty^\lambda(x_k) \le H_\infty^\lambda(x_k) \le Y2$, we obtain $v_\infty^\lambda(x_k) \le J^*(x_k)$. Because $J^*(x_k)$ is the infimum of the cost function that derives from all other admissible control sequences, we conclude that $J^*(x_k) \le v_\infty^\lambda(x_k)$. This implies that $J^*(x_k) \le v_\infty^\lambda(x_k) \le J^*(x_k)$, and hence, $v_\infty^\lambda(x_k) = J^*(x_k)$. According to Lemma 3 and by applying $\zeta(x_k) = \mu(x_k)$ in (51) and (56), we conclude that $v_\infty^0(x_k) = v_\infty^\lambda(x_k) = J^*(x_k)$.

According to Theorem 2, $v_i^0(x_k)$ and $v_i^\lambda(x_k) \to J^*(x_k)$ as $i \to \infty$. Because $g_i^0(x_k) = \dfrac{\partial v_i^0(x_k)}{\partial x_k}$ and $g_i^\lambda(x_k) = \dfrac{\partial v_i^\lambda(x_k)}{\partial x_k}$, we conclude that $g_i^0(x_k)$ and $g_i^\lambda(x_k) \to g^*(x_k)$ as $i \to \infty$, where g^* is an optimum of the value gradient function.

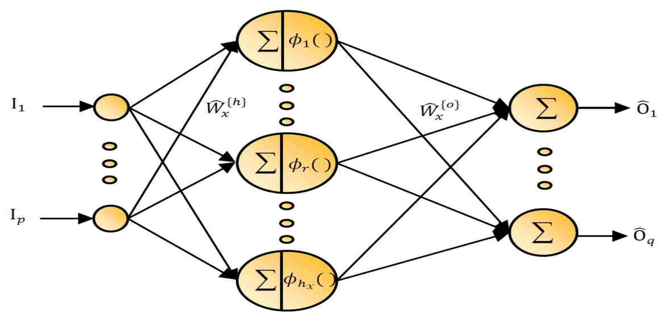


Figure 2. General feed-forward neural network, which is used in the one-step critic network, the n-step critic network, and the actor network in NSVGL(λ).

Corollary 1: The action policy μ_i, the one-step value function v_i^0 and the n-step value function v_i^λ are defined as in (41), (42) and (43), respectively. If the system state x_k is controllable, then the v_i^0 and v_i^λ force the controller (actor network) to converge to the optimal control u^* as $i \to \infty$ (i.e., $\lim_{i\to\infty} \mu_i(x_k) = u^*(x_k)$). Similar conclusions can apply with gradients of $v_i^\lambda(x_k)$ and $v_i^0(x_k)$ with respect to x_k (i.e., $g_i^0(x_k)$ and $g_i^\lambda(x_k)$ reach an optimal value gradient function as $i \to \infty$).

4. NEURAL NETWORK ARCHITECTURE DESIGN

We use a feed-forward neural network, which is shown in Fig. 2, for all three networks $\left(\hat{G}_c^0(.), \hat{G}_c^\lambda(.), \text{and } \hat{A}(.)\right)$ as universal function approximator, which is explained in detail in the following subsections.

4.1. The One-Step Critic Network $\left(\hat{G}^0\left(x_k, \hat{\omega}_c^0\right)\right)$. The structure of $\hat{G}^0\left(x_k, \hat{\omega}_c^0\right)$ is shown in Fig. 2, where $\hat{\omega}_c^0$ represents a combination of hidden and output weights for the one-step critic network. In Fig. 2, the inputs are the system state ($I_k = [I_1, \ldots, I_p]^T \equiv x_k = [x_1, \ldots, x_m]^T$); the output is $\hat{g}^0(x_k)$, which is an approximation m dimension vector of a gradient of the one-step value function ($\hat{O} = [\hat{O}_1, \ldots, \hat{O}_q]^T \equiv \hat{g}^0(x_k) = [\hat{g}_1^0, \ldots, \hat{g}_m^0]^T$); h_c^0 ($\equiv h_x$) represents the number of hidden neurons; the hidden weights are indicated as $\hat{\omega}_c^{0\{h\}}$ ($\equiv \hat{\omega}_x^{\{h\}}$), which can be represented in matrix form with ($m \times h_c^0$) dimension; and the output weight matrix is indicated as $\hat{\omega}_c^{0\{o\}}$ ($\equiv \hat{\omega}_x^{\{o\}}$), which can be represented in matrix form with ($h_c^0 \times m$) dimension. The activation function for the hidden nodes is the hyperbolic tangent threshold function ($\phi(x) = (1 - e^{-x})/(1 + e^{-x})$). The forward propagating output signal is expressed as:

$$\hat{g}_i^0(x_k) = \big(\hat{\omega}_c^{0\{o\}}\big)_i^T \phi\Big(\big(\hat{\omega}_c^{0\{h\}}\big)_i^T x_k\Big), \tag{74}$$

where i is the iteration index and k is the time index. With quadratic form of the utility function ($U(x_k, \mu_i(x_k)) = x_k^T Q x_k + \mu_i^T(x_k) R \mu_i(x_k)$)) and (3) for the system state equation, the one-step target value, which is defined in (15) in the general form, is given:

$$\begin{aligned} g_i^0(x_k) =& \frac{\partial}{\partial x_k}\Big(x_k^T Q x_k + \mu_{i-1}^T(x_k) R \mu_{i-1}(x_k) + \gamma \hat{v}_{i-1}^0(f(x_k, \hat{\mu}_{i-1}(x_k))\Big), \\ =& 2Qx_k + 2\Big(\frac{\partial \mu_{i-1}(x_k)}{\partial x_k}\Big)^T R \mu_{i-1}(x_k) + \gamma g_{i-1}^0(x_{k+1}) \times \\ & \Big(\frac{\partial f(x_k, \hat{\mu}_{i-1}(x_k))}{\partial x_k} + \frac{\partial f(x_k, \hat{\mu}_{i-1}(x_k))}{\partial \hat{\mu}_{i-1}(x_k)} \frac{\partial \hat{A}_{i-1}(x_k)}{\partial x_k}\Big). \end{aligned} \tag{75}$$

The weights for hidden ($\hat{\omega}_c^{0\{h\}}$) and output ($\hat{\omega}_c^{0\{o\}}$) layers are tuned by backpropagating the prediction error of the critic network, which is given as:

$$e_{ci}^0(x_k) = g_i^0(x_k) - \hat{g}_i^0(x_k). \tag{76}$$

The objective function for the $\hat{G}_{ci}^0(x_k, \hat{\omega}_c^0)$ network is to minimize $E_{ci}^0(x_k) = 0.5\big(e_{ci}^0(x_k)\big)^2$ by updating the value for the weights ($\hat{\omega}_c^0$) according to the gradient descent algorithm inside the local inner-loop, which is given by:

$$\begin{aligned} \hat{\omega}_{ci}^{0\{h\}} &= \hat{\omega}_{ci}^{0\{h\}} + \ell_c^0 \frac{\partial E_{ci}^0}{\partial \hat{\omega}_{ci}^{0\{h\}}} = \hat{\omega}_{ci}^{0\{h\}} + \ell_c^0 \frac{\partial \hat{G}_{ci}^0}{\partial \hat{\omega}_{ci}^{0\{h\}}} e_{ci}^0(x_k), \\ \hat{\omega}_{ci}^{0\{o\}} &= \hat{\omega}_{ci}^{0\{o\}} + \ell_c^0 \frac{\partial E_{ci}^0}{\partial \hat{\omega}_{ci}^{0\{o\}}} = \hat{\omega}_{ci}^{0\{o\}} + \ell_c^0 \frac{\partial \hat{G}_{ci}^0}{\partial \hat{\omega}_{ci}^{0\{o\}}} e_{ci}^0(x_k), \end{aligned} \tag{77}$$

where ℓ_c^0 is the one-step critic learning rate.

4.2. The *n*-step Critic Network ($\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$). The structure of $\hat{G}^{\lambda}(x_k, \hat{\omega}_c^{\lambda})$ is shown in Fig. 2, where $\hat{\omega}_c^{\lambda}$ represents a combination of hidden and output weights for the n-step critic network. In Fig. 2, the inputs are the system state ($I_k = [I_1, \ldots, I_p]^T \equiv x_k = [x_1, \ldots, x_m]^T$); the output is $\hat{g}^{\lambda}(x_k)$, which is an approximation m dimension vector of a gradient of the n-step value function ($\hat{O} = [\hat{O}_1, \ldots, \hat{O}_q]^T \equiv \hat{g}^{\lambda}(x_k) = [\hat{g}_1^{\lambda}, \ldots, \hat{g}_m^{\lambda}]^T$); h_c^{λ} ($\equiv h_x$) represents the number of hidden neurons; the hidden weights are indicated as $\hat{\omega}_c^{0\{h\}}$ ($\equiv \hat{\omega}_x^{\{h\}}$), which can be represented in matrix form with ($m \times h_c^{\lambda}$) dimension; and the output weights are indicated as $\hat{\omega}_c^{\lambda\{o\}}$ ($\equiv \hat{\omega}_x^{\{o\}}$), which can be represented in matrix form with ($h_c^{\lambda} \times m$) dimension.

The activation function for the hidden nodes is the hyperbolic tangent threshold function. The forward propagating output signal is expressed as follows:

$$\hat{g}_i^{\lambda}(x_k) = \left(\hat{\omega}_c^{\lambda\{o\}}\right)_i^T \phi\left(\left(\hat{\omega}_c^{\lambda\{h\}}\right)_i^T x_k\right), \tag{78}$$

where i is the iteration index and k is the time index. With quadratic form of the utility function and (3) for the system state equation, the one-step target value, which is defined in (22) in the general form, is

$$\begin{aligned} g_i^{\lambda}(x_k) =& \frac{\partial}{\partial x_k}\Bigg(x_k^T Q x_k + \mu_{i-1}^T(x_k) R \mu_{i-1}(x_k) + \gamma\Big(\lambda \hat{v}^{\lambda}(f(x_k, \hat{\mu}_{i-1}(x_k)) + (1-\lambda) \\ & \hat{v}^0(f(x_k, \hat{\mu}_{i-1}(x_k))\Big)\Bigg), \\ =& 2Qx_k + 2\Big(\frac{\partial \mu_{i-1}(x_k)}{\partial x_k}\Big)^T R\mu_{i-1}(x_k) + \gamma\Big(\lambda \hat{g}_{i-1}^{\lambda}(f(x_k, \hat{\mu}_{i-1}(x_k)) + (1-\lambda) \\ & \hat{g}_{i-1}^{0}(f(x_k, \hat{\mu}_{i-1}(x_k))\Big)\Big(\frac{\partial f(x_k, \hat{\mu}_{i-1}(x_k))}{\partial x_k} + \frac{\partial f(x_k, \hat{\mu}_{i-1}(x_k))}{\partial \hat{\mu}_{i-1}(x_k)} \frac{\partial \hat{A}_{i-1}(x_k)}{\partial x_k}\Big). \end{aligned} \tag{79}$$

The weights for hidden and output layers are tuned by back-propagating the prediction error of the critic network, which is

$$e^{\lambda}_{ci}(x_k) = g^{\lambda}_i(x_k) - \hat{g}^{\lambda}_i(x_k). \tag{80}$$

The objective function for n-step critic network is to minimize $E^{\lambda}_{ci}(x_k) = 0.5\big(e^{\lambda}_{ci}(x_k)\big)^2$ by updating the related critic weights ($\hat{\omega}^{\lambda}_c$) according to the gradient descent algorithm inside the local inner-loop, which is given by

$$\begin{aligned} \hat{\omega}^{\lambda\{h\}}_{ci} &= \hat{\omega}^{\lambda\{h\}}_{ci} + \ell^{\lambda}_c \frac{\partial E^{\lambda}_{ci}}{\partial \hat{\omega}^{\lambda\{h\}}_{ci}} = \hat{\omega}^{\lambda\{h\}}_{ci} + \ell^{\lambda}_c \frac{\partial \hat{G}^{\lambda}_{ci}}{\partial \hat{\omega}^{\lambda\{h\}}_{ci}} e^{\lambda}_{ci}(x_k), \\ \hat{\omega}^{\lambda\{o\}}_{ci} &= \hat{\omega}^{\lambda\{o\}}_{ci} + \ell^{\lambda}_c \frac{\partial E^{\lambda}_{ci}}{\partial \hat{\omega}^{\lambda\{o\}}_{ci}} = \hat{\omega}^{\lambda\{o\}}_{ci} + \ell^{\lambda}_c \frac{\partial \hat{G}^{\lambda}_{ci}}{\partial \hat{\omega}^{\lambda\{o\}}_{ci}} e^{\lambda}_{ci}(x_k), \end{aligned} \tag{81}$$

where ℓ^{λ}_c is the n-step critic learning rate.

4.3. Actor Network ($\hat{A}\big(x_k, \hat{\omega}_a\big)$)). The main goal for an actor network is to generate a near-optimal control policy. We also use the general multilayer perceptron neural network to represent the structure of $\hat{A}(x_k) \equiv \hat{A}\big(x_k, \hat{\omega}_a\big)$ as shown in Fig. 2, where $\hat{\omega}^{\lambda}_c$ represents a combination of the hidden and output actor network weights. In Fig. 2, the inputs are the system state ($I_k = [I_1, \ldots, I_p]^T \equiv x_k = [x_1, \ldots, x_m]^T$); the outputs are $\hat{\mu}(x_k)$, which is an approximation $\mathfrak{n}$ dimension vector of control actions ($\hat{O} = [\hat{O}_1, \ldots, \hat{O}_q]^T \equiv \hat{g}^0(x_k) = [\hat{\mu}_1, \ldots, \hat{\mu}_{\mathfrak{n}}]^T$); h_a ($\equiv h_x$) represents the number of hidden neurons; the hidden weights are indicated as $\hat{\omega}^{\{h\}}_a$ ($\equiv \hat{\omega}^{\{h\}}_x$), which can be represented in matrix form with $(m \times h_a)$ dimension; and the output weights are indicated as $\hat{\omega}^{\{o\}}_a$ ($\equiv \hat{\omega}^{\{o\}}_x$), which can be represented in matrix form with $(h_a \times \mathfrak{n})$ dimension. The activation function for the hidden nodes is the hyperbolic tangent threshold function. The forward propagating output signal is expressed as follows:

$$\hat{\mu}_{i-1}(x_k) = \big(\hat{\omega}^{\{o\}}_a\big)^T_{i-1} \phi\Big(\big(\hat{\omega}^{\{h\}}_a\big)^T_{i-1} x_k\Big). \tag{82}$$

As mentioned in (23), $e_a(x_k)$ combines the actor error path through the one-step critic network ($e_a^0(x_k)$) as in (13) and the actor error path through the n-step critic network ($e_a^0(x_k)$) as in (25). With quadratic form of the utility function and (3) for the system state equation, $e_a(x_k)$ is

$$\begin{aligned} e_{a(i-1)}(x_k) =&\lambda\left(2R\hat{\mu}_{i-1}(x_k) + \gamma\left(\lambda g_{i-1}^{\lambda}(x_{k+1}) + (1-\lambda)\lambda g_{i-1}^{0}(x_{k+1})\right)\frac{\partial f(x_k, \hat{\mu}_{i-1}(x_k))}{\partial \hat{\mu}_{i-1}(x_k)}\right) \\ &+ (1-\lambda)\left(2R\hat{\mu}_{i-1}(x_k) + \gamma\lambda g_{i-1}^{0}(x_{k+1})\frac{\partial f(x_k, \hat{\mu}_{i-1}(x_k))}{\partial \hat{\mu}_{i-1}(x_k)}\right). \end{aligned} \tag{83}$$

The objective function for the $\hat{A}\big(x_k, \hat{\omega}_a\big))$ network is to minimize $e_{a(i-1)}$ by updating the actor weights ($\hat{\omega}_a$) according to the gradient descent algorithm (with $E_{a(i-1)}(x_k) = \Big(e_{a(i-1)}(x_k)\Big)^2$) inside the local inner-loop, which is given by

$$\begin{aligned} \hat{\omega}_{a(i-1)}^{\{h\}} &= \hat{\omega}_{a(i-1)}^{\{h\}} - \ell_a \frac{\partial \hat{A}_{(i-1)}(x_k)}{\partial \hat{\omega}_{a(i-1)}^{\{h\}}} e_{a(i-1)}(x_k), \\ \hat{\omega}_{a(i-1)}^{\{o\}} &= \hat{\omega}_{a(i-1)}^{\{o\}} - \ell_a \frac{\partial \hat{A}_{(i-1)}(x_k)}{\partial \hat{\omega}_{a(i-1)}^{\{o\}}} e_{a(i-1)}(x_k), \end{aligned} \tag{84}$$

where ℓ_a^0 is the actor learning rate. If a system type is affine, as in (26), then a greedy action can be calculated for a one-step critic network as follows:

$$\mu_{i-1}^{0}(x_k) = -\frac{\gamma}{2}R^{-1}g^T(x_k)\hat{g}_{i-1}^{0}(x_{k+1}), \tag{85}$$

and for n-step critic network as

$$\mu_{i-1}^{\lambda}(x_k) = -\frac{\gamma}{2}R^{-1}g^T(x_k)\left(\lambda\hat{g}_{i-1}^{\lambda}(x_{k+1}) + (1-\lambda)\hat{g}_{i-1}^{0}(x_{k+1})\right), \tag{86}$$

and then used instead of estimated control actions from critic networks. (85) can also be used as the one-step target control to train the one-step critic network similar to [41]. Likewise, (86) can be used as the n-step target control to train the one-step critic network

as follows:

$$\begin{aligned} e^{0}_{a(i-1)}(x_k) &= \mu^{0}_{i-1}(x_k) - \hat{\mu}_{i-1}(x_k), \\ e^{\lambda}_{a(i-1)}(x_k) &= \mu^{\lambda}_{i-1}(x_k) - \hat{\mu}_{i-1}(x_k). \end{aligned} \tag{87}$$

The objective function for this network is minimizing the actor error, which is given by

$$E_{a(i-1)}(x_k) = 0.5\Big((1-\lambda)E^{0}_{a(i-1)}(x_k) + \lambda E^{\lambda}_{a(i-1)}(x_k)\Big), \tag{88}$$

where $E^{0}_{a(i-1)}(x_k) = \Big(e^{0}_{a(i-1)}(x_k)\Big)^2$ and $E^{\lambda}_{a(i-1)}(x_k) = \Big(e^{\lambda}_{a(i-1)}(x_k)\Big)^2$. The actor weights are updated by minimizing $E_{a(i-1)}(x_k)$ according to the gradient descent algorithm inside the local inner-loop as follows:

$$\begin{aligned} \hat{\omega}^{\{h\}}_{a(i-1)} =& \hat{\omega}^{\{h\}}_{a(i-1)} - \ell_a \frac{\partial E_{a(i-1)}(x_k)}{\partial \hat{\omega}^{\{h\}}_{a(i-1)}} = \hat{\omega}^{\{h\}}_{a(i-1)} - \ell_a \frac{\partial \hat{A}_{(i-1)}(x_k)}{\partial \hat{\omega}^{\{h\}}_{a(i-1)}} e_{a(i-1)}(x_k), \\ \hat{\omega}^{\{o\}}_{a(i-1)} =& \hat{\omega}^{\{o\}}_{a(i-1)} - \ell_a \frac{\partial E_{a(i-1)}(x_k)}{\partial \hat{\omega}^{\{o\}}_{a(i-1)}} = \hat{\omega}^{\{o\}}_{a(i-1)} - \ell_a \frac{\partial \hat{A}_{(i-1)}(x_k)}{\partial \hat{\omega}^{\{o\}}_{a(i-1)}} e_{a(i-1)}(x_k), \end{aligned} \tag{89}$$

where $e_{a(i-1)}(x_k) = (1-\lambda)e^{0}_{a(i-1)}(x_k) + \lambda e^{\lambda}_{a(i-1)}(x_k)$, which is similar to (23).

5. SIMULATION STUDIES

Two case studies are taken to verify and demonstrate the effectiveness of the NSVGL(λ) method. The trajectories of 2-D nonlinear system responses are considered as the first case. The second case study is considered a two-wheeled dynamic, nonholonomic mobile robot model. The theoretical analysis is provided for both cases to demonstrate the performance index during learning. We compare the performance results for two cases with DHP and NSVGL(λ) with $\lambda = 0.5$ and $\lambda = 0.99$. For a fair comparison, we set similar values of global parameters during ten independent runs.

5.1. Case I: Nonlinear System Problem . Consider the following nonlinear system derived from [62]:

$$x_{k+1} = \begin{bmatrix} x1_{k+1} \\ x2_{k+1} \end{bmatrix} = \begin{bmatrix} -sin(0.5x1_k + \mu(x_k) \\ -sin(x1_k + 0.5x2_k) \end{bmatrix}, \tag{90}$$

where $x_k = [x1_k \quad x2_k]^T \in \mathbb{R}^2$ is the state vector ($m = 2$), and $\mu(x_k) \in \mathbb{R}^1$ is the control action (n = 1). The external instantaneous cost function is chosen as $U_k = x_k^T Q x_k + \mu^T(x_k) R \mu(x_k)$, where Q is 2-D identity matrix and $R = 1$. The discount factor (γ) is 0.95. The number of hidden nodes is 5, which is equal to h_c^0 and h_c^λ for the one-step critic network ($\hat{G}^0(x_k, \hat{\omega}_c^0)$)) and the n-step critic network ($\hat{G}^\lambda(x_k, \hat{\omega}_c^0)$)), respectively. The number of hidden nodes for actor network ($\hat{A}(x_k, \hat{\omega}_a)$)) is also 5 ($h_a$). Therefore, the structure of the three-layer feed-forward neural network for the critic networks is 2-5-2 (two input nodes, five hidden nodes, and two output nodes), and the structure for the actor network is 2-5-1. The initial learning-rate parameters are set to $\ell_c^0 = \ell_c^\lambda = \ell_a = 0.01$ for all networks. The training will be terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold for the internal cycle (20 iterations for both critic networks and 30 iterations for the actor network). The initial weights for all networks are randomly chosen within [-0.3, 0.3] range. The initial state is $x_0 = [1, 1]^T$. We compare NSVGL(λ) with $\lambda = 0.5$ and $\lambda = 0.99$ and the traditional DHP for ten independent runs. Each run has 400 iterations to train ADP to control the system for 10 time steps. As demonstrated in Fig. 3, which is the mean squared error (MSE) values over iteration for ten runs, NSVGL(λ) is more efficient and faster than DHP. Fig. 4 - Fig. 6 illustrate the state trajectory curves and the corresponding control action inputs for 10 time steps. We can see the improvement for the system responses as it learns.

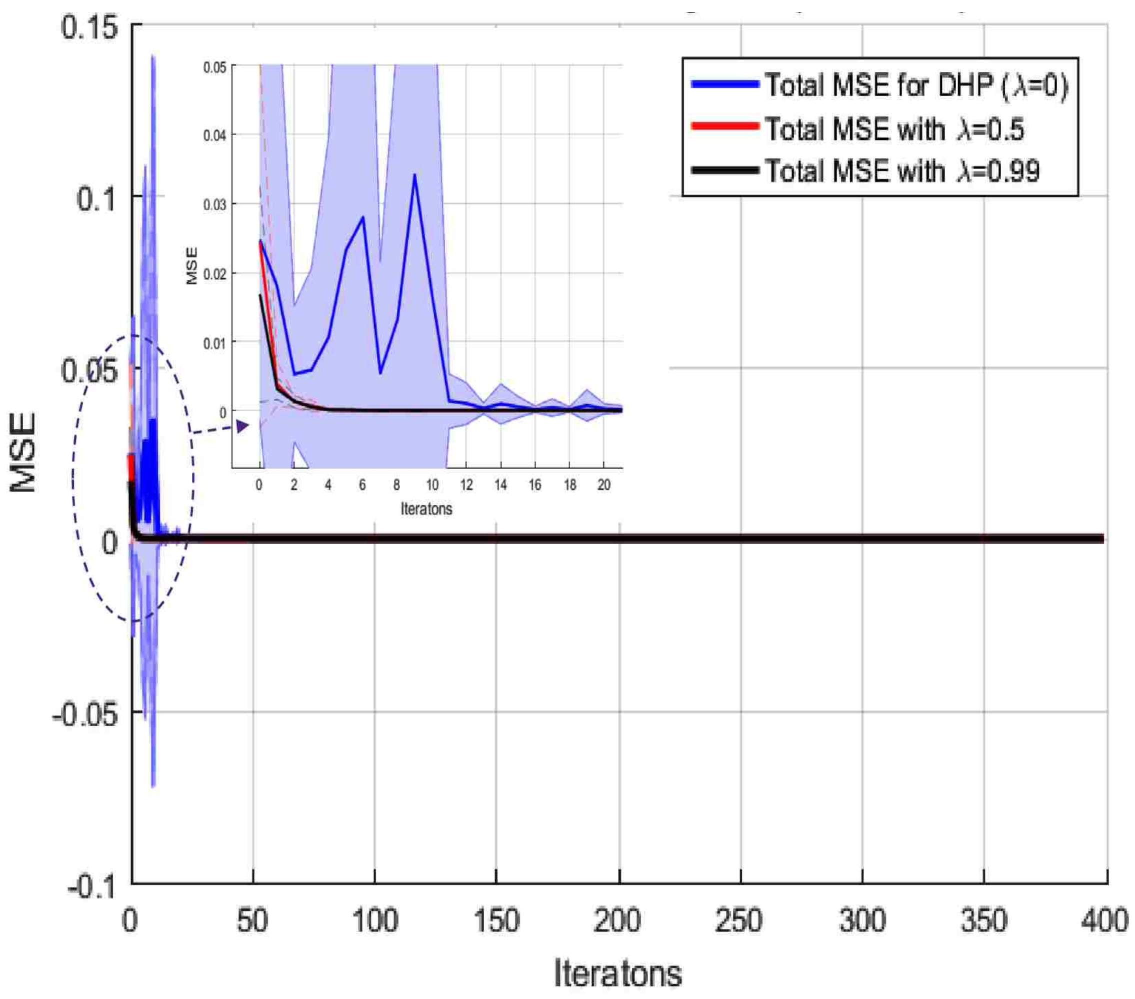


Figure 3. The mean-squared-error (MSE) comparisons over iteration among NSVGL(λ = 0.99), NSVGL(λ = 0.5) and the DHP approaches. The mean values from 10 independent runs are taken for all methods. The shaded region represents 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) provides a faster learning speed than DHP.

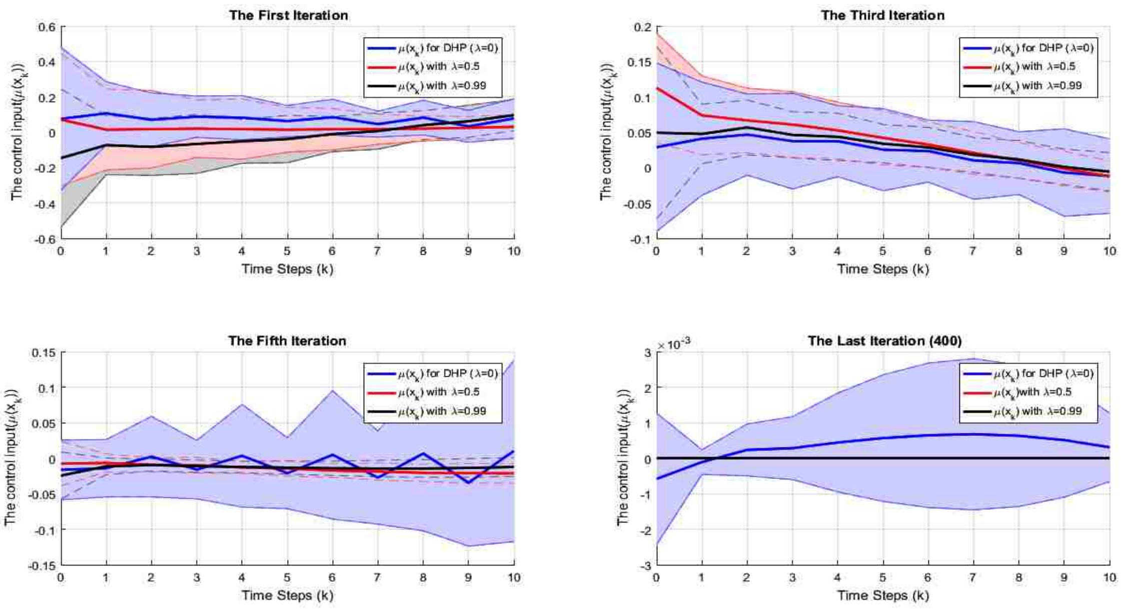


Figure 4. The control input during iteration for NSVGL(λ = 0.99), NSVGL(λ = 0.5) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs.

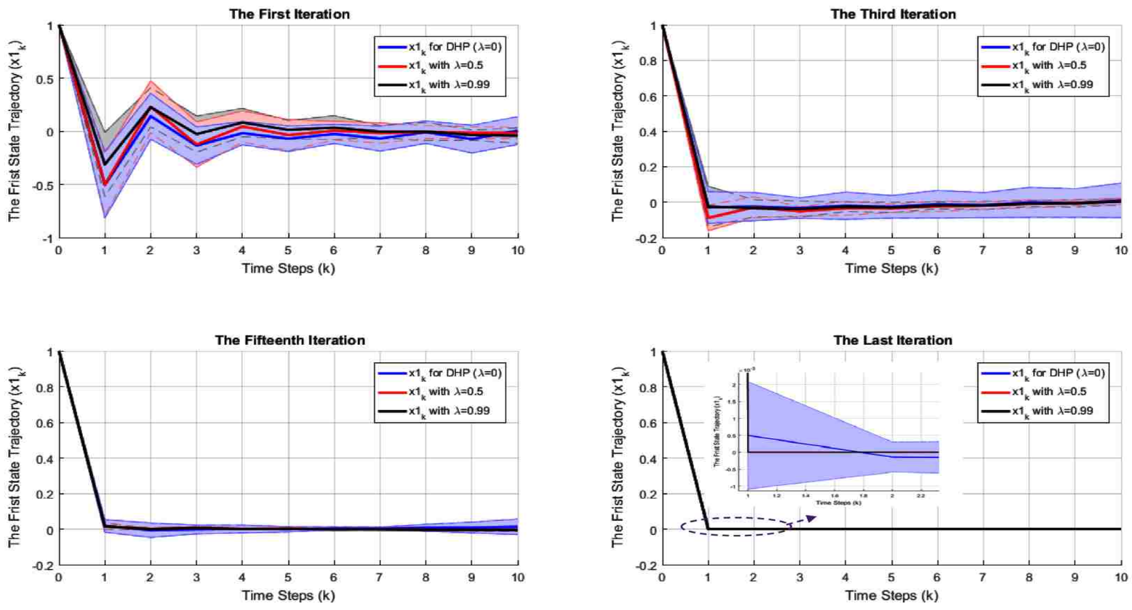


Figure 5. The state trajectories for the $x1$ state during iteration for NSVGL(λ = 0.99), NSVGL(λ = 0.5) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) improves faster than DHP.

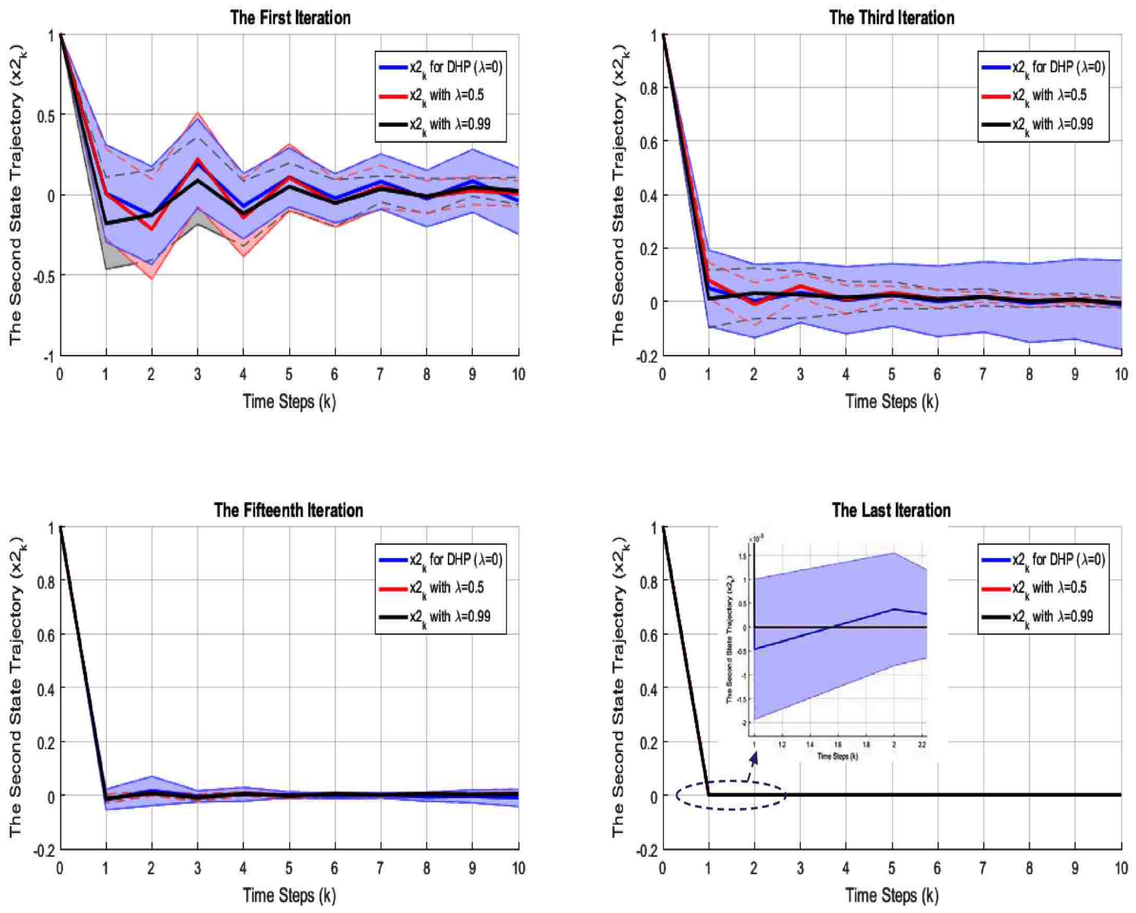


Figure 6. The state trajectories for the $x2$ state during iteration for NSVGL($\lambda = 0.99$), NSVGL($\lambda = 0.5$) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) improves faster than DHP.

Fig. 7 and Fig. 8 demonstrate the convergence process of the gradient of the cost functions during iterations for NSVGL($\lambda = 0.99$). The convergence of the gradient of the cost functions ($g_i^0(x_k)$ and $g_i^\lambda(x_k)$) to the optimal cost function indicates the effectiveness of NSVGL(λ).

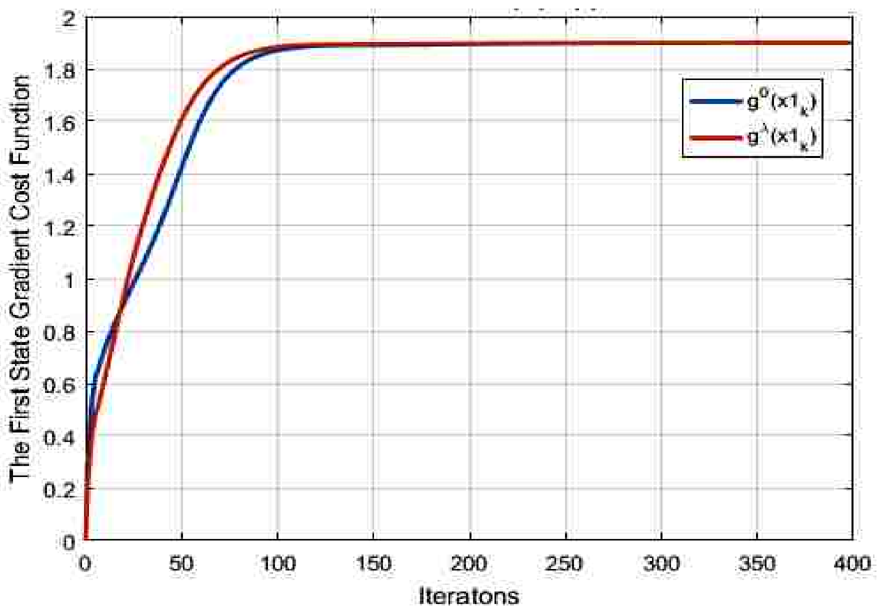


Figure 7. The average gradient trajectories of the value functions for the first system state in NSVGL(λ = 0.99. The gradient of the cost functions $\left(g_i^0(x1_k)\text{ and }g_i^\lambda(x1_k)\right)$ converge to the optimal value function.

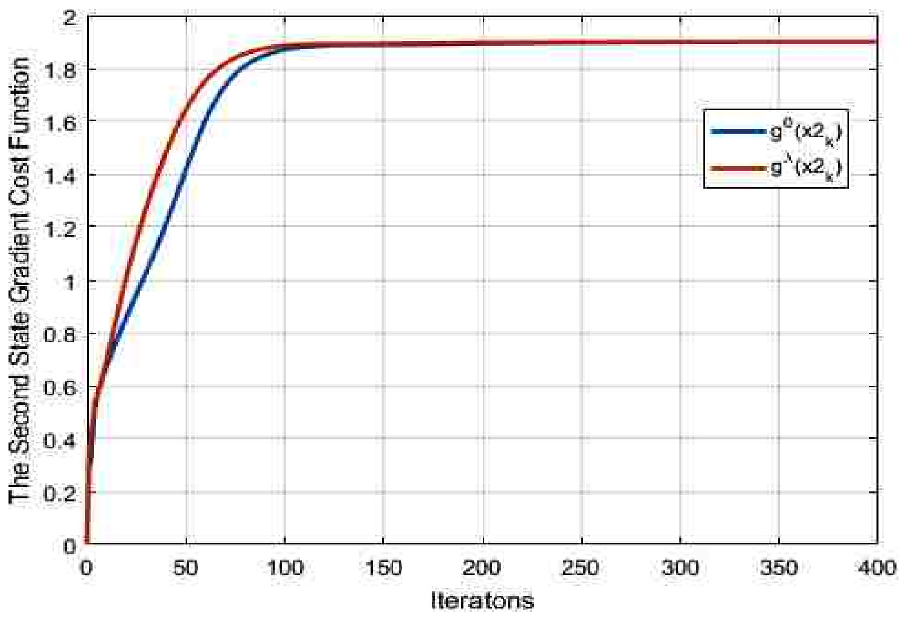


Figure 8. The average gradient of the second cost function trajectories for the both critic networks in the NSVGL(λ = 0.99) approaches. The convergence of the gradient of the value functions for the second system state $\left(g_i^0(x2_k)\text{ and }g_i^\lambda(x2_k)\right)$ to the optimal cost function is clearly shown starting from iteration number 200.

5.2. Case II: Mobile Robot Dynamic Model. A differential-drive mobile robot contains two independently driven wheels mounted on the left and right of its chassis at the same axis, and a castor wheel (free rotating wheel) mounted at the front for balancing the mobile robot. An inertial Cartesian frame represents the position of the mobile robot, while $q = [x_c, y_c, \theta]^T$ is the set of coordinates for the center of mass of the robot and the robot orientation with respect to the Cartesian frame.

Two independent driving wheels are provided with the necessary torque for generating a left angular velocity (w_L) and a right angular velocity (w_R), which in turn generate a linear velocity (v_1) and angular velocity (v_2) for the mobile robot as follows:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.5\bar{r} & -0.5\bar{r} \\ \dfrac{\bar{r}}{2b} & -\dfrac{\bar{r}}{2b} \end{bmatrix} \begin{bmatrix} w_R \\ w_L \end{bmatrix}, \tag{91}$$

where $\bar{r}$ is wheel radius and b is half of the robot width. The different forces for the mobile robot mechanical motion are considered in the literature for the dynamic model but not the kinematic model. The kinematic model is considered only for the motion.

As stated in [49], [66] - [68], the dynamic model of the mobile robot has $\bar{n}$ dimensional configuration space subjected to r constraints as described by

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)u + A^T(q)\Psi, \tag{92}$$

with $A(q)\dot{q} = 0$ as a constrained kinematic wheel, where $q \in \mathbb{R}^{\bar{n}}$ is coordinate vector, $M(q) \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is a is a symmetric positive definite inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is the centripetal and Coriolis matrix, $F(\dot{q}) \in \mathbb{R}^{\bar{n}}$ is a surface friction force vector, $G(q) \in \mathbb{R}^{\bar{n}}$ is a gravity vector, $\tau_d \in \mathbb{R}^{\bar{n}}$ is a bounded unknown disturbance, $B(q) \in \mathbb{R}^{\bar{n}\times \mathrm{n}}$ is a input transformation matrix, $u \in \mathbb{R}^{\mathrm{n}}$ is the input torque vector, $A(q) \in \mathbb{R}^{r\times\bar{n}}$ is the full rank matrix associated with constraints, and $\Psi \in \mathbb{R}^r$ is the Lagrange multiplier (constraint forces) vector. In this case study, There are two control inputs, which are a left torque (τ_L) and a right torque (τ_R).

Since the system does not change in vertical position and has a constant value for potential energy, $G(q)$ is set to zero. Using Lagrange multipliers to reduce the dynamic model from $\bar{n}$ to $m = \bar{n} - r$, (76) is pre-multiplied by spanning the linear independent null space of the $A(q)\dot{q}$ matrix, (which is denoted as the Jacobian matrix of $S_c(q) \in \mathbb{R}^{\bar{n}\times m}$). In this case, a kinematic equation is given as follows:

$$\dot{q} = S_c(q)v, \tag{93}$$

where

$$\dot{q} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix}, \quad S_c(q) = \begin{bmatrix} cos(\theta) & -dsin(\theta) \\ sin(\theta) & dcos(\theta) \\ 0 & 1 \end{bmatrix}, \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},$$

and d is a center of gravity. The final affine dynamic model is obtained from the kinematic equation (77) as fallows: By taking the derivative of (77)

$$\ddot{q} = \dot{S}_c(q)v + S_c(q)\dot{v}. \tag{94}$$

Substitute (75), (77) and (78) into (76) to obtain

$$\dot{v} = -\bar{M}(q)^{-1}\Big(\bar{V}(q,\dot{q})v + \bar{F}(\dot{q}) + \bar{\tau}_d\Big) + \bar{M}(q)^{-1}\bar{\tau}. \tag{95}$$

$\bar{M}(q)$ is an invertible matrix:

$$\bar{M}(q) = \begin{bmatrix} m_T + 2\frac{I_{YY}^b}{\bar{r}^2} & 0 \\ 0 & m_T d^2 + I_T + 2I_{YY}^b \frac{b^2}{\bar{r}^2} - 4m_w d^2 \end{bmatrix},$$

$$S_c(q) = \begin{bmatrix} 0 & -dv_2(m_T - 2m_w) \\ dv_2(m_T - 2m_w) & 0 \end{bmatrix}, \tag{96}$$

$$\bar{\tau}_d = \begin{bmatrix} \bar{\tau}_R \\ \bar{\tau}_L \end{bmatrix}, \bar{F}(\dot{q}) = \frac{1}{\bar{r}} \begin{bmatrix} f_v(w_R + w_L) + f_c(\Delta(w_R) + \Delta(w_L)) \\ bf_v(w_R - w_L) + bf_c(\Delta(w_R) - \Delta(w_L)) \end{bmatrix},$$

and

$$\bar{\tau} = \bar{B}\tau = \begin{bmatrix} 0.5\bar{r} & -0.5\bar{r} \\ \frac{\bar{r}}{2b} & -\frac{\bar{r}}{2b} \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}, (\Delta(.) \text{ is a sigmoid function}). \tag{97}$$

All mobile robot dynamic mode parameters are given in Table 2. We assume a noise-free environment and therefore ignore unknown disturbances impacting the left and the right wheels by setting $\bar{\tau}_L = 0$ and $\bar{\tau}_R = 0$, respectively. In this case study, the first state ($x1_k$) and the second state ($x2_k$) are the linear velocity (v_1) and the angular velocity (v_2), respectively, whereas $x_k = [x1_k \quad x2_k]^T \in \mathbb{R}^2$ is the state vector ($m = 2$). The $\mu(x_k) = [\tau_R \quad \tau_L]^T \in \mathbb{R}^2$ is the control action ($\mathfrak{n} = 2$). The external instantaneous cost function is chosen as $U_k = x_k^T Q x_k + \mu^T(x_k) R \mu(x_k)$, where Q and R are 2-D identity matrices. The discount factor (γ) is 0.95. The number of hidden nodes is 24, which is equal to h_c^0 and h_c^λ for the one-step critic network ($\hat{G}^0(x_k, \hat{\omega}_c^0)$)) and the n-step critic network ($\hat{G}^\lambda(x_k, \hat{\omega}_c^0)$)), respectively. The number of hidden nodes for the actor network ($\hat{A}(x_k, \hat{\omega}_a)$)) is also 30 ($h_a$). The initial learning rate parameters are set to $\ell_c^0 = \ell_c^\lambda = 0.0001$ for the critic networks and $\ell_a = 0.001$ for actor network. The training for each iteration of either network will be terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold for the internal cycle (30 iterations for actor and critic networks). The initial weights for all networks are randomly chosen within [-0.3, 0.3] range. The initial state is $x_0 = [0.5, 0.5]^T$. We compare NSVGL(λ) with $\lambda = 0.5$ and $\lambda = 0.99$ and DHP for ten independent runs. Each run has 5000 iterations to train the ADP algorithms to control the system for 100 time steps. Fig. 9 illustrates the average MSE for the two velocity states during iterations for ten runs. NSVGL(λ) is more efficient and faster to learn than DHP.

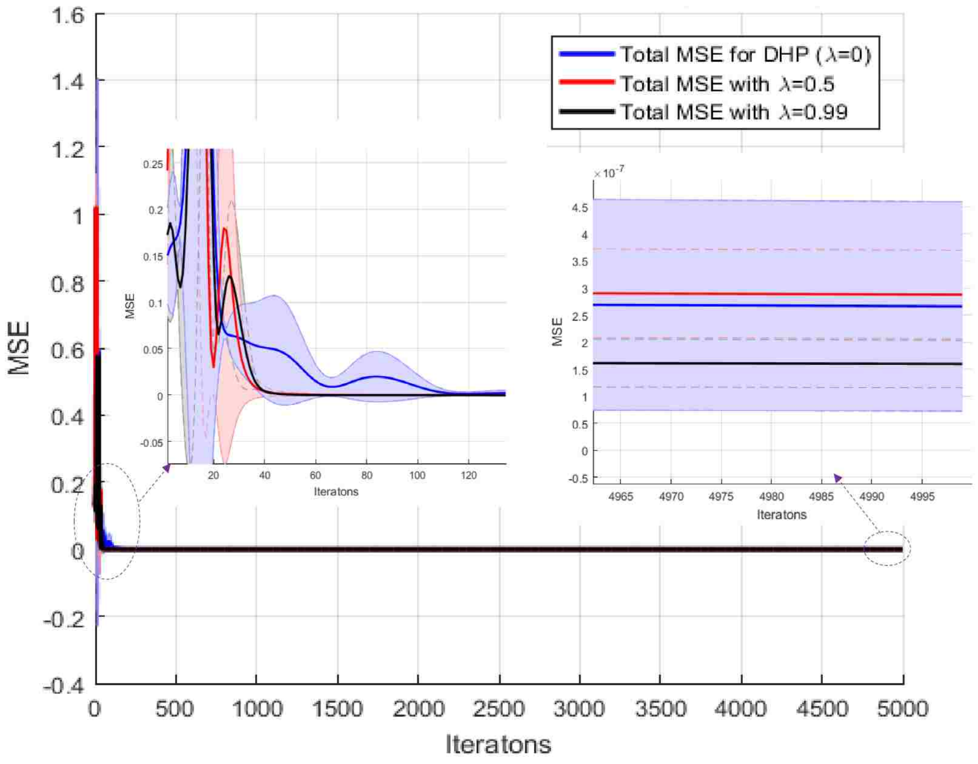


Figure 9. The average mean-squared-error for two velocity states during iterations among NSVGL(λ = 0.99), NSVGL(λ = 0.5) and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) illustrates faster learning than DHP.

Fig. 10 - Fig. 13 illustrate the state trajectory curves and the corresponding control action inputs for 100 time steps. We can see the improvement for system responses during increasing the iterations.

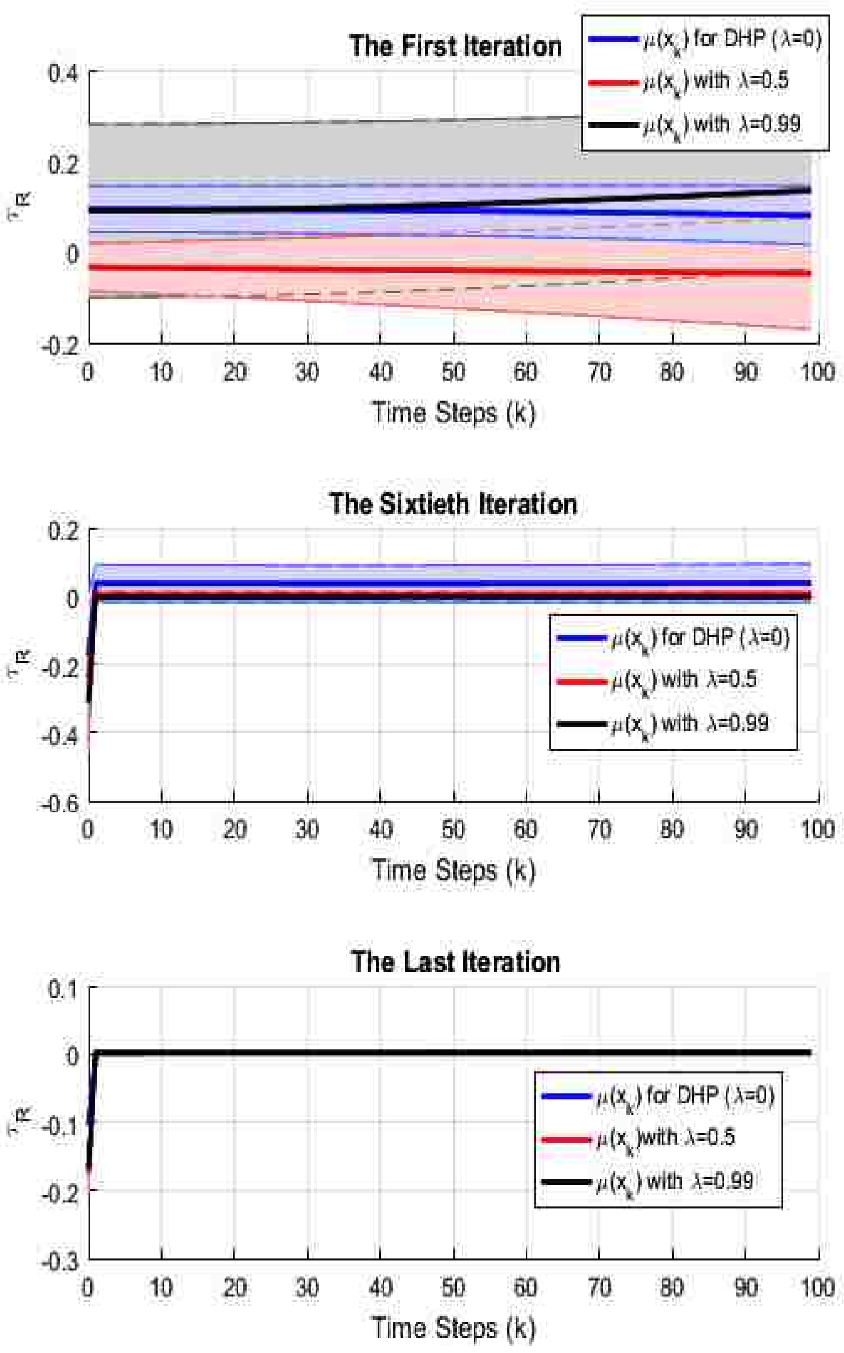


Figure 10. The first control input (left torque) during iterations among NSVGL(λ = 0.99), NSVGL(λ = 0.5), and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs.

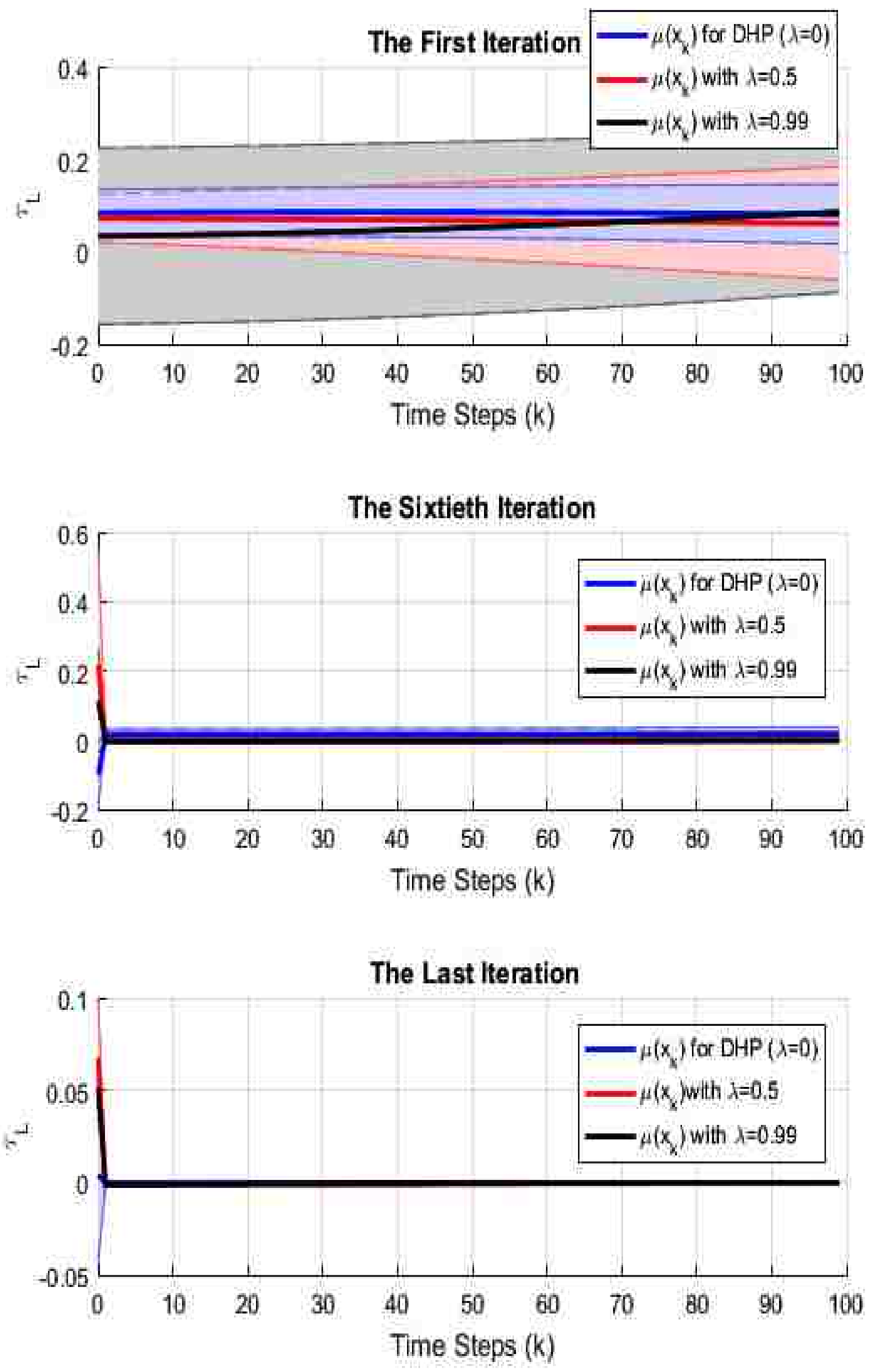


Figure 11. The second control input (right torque) during iterations among NSVGL(λ = 0.99), NSVGL(λ = 0.5), and the DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs.

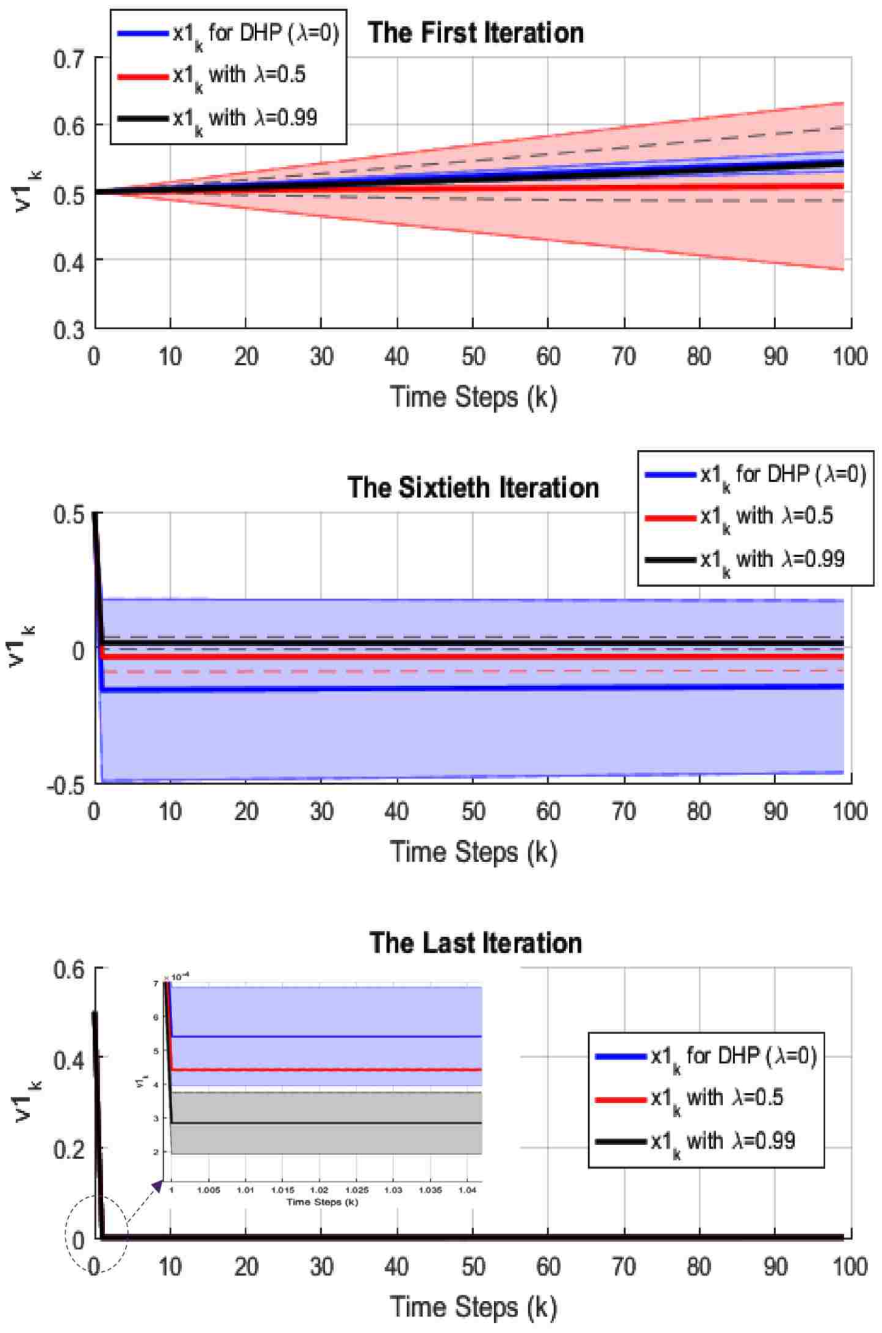


Figure 12. Comparisons of state trajectory for the first state (linear velocity) during iterations for NSVGL(λ = 0.99), NSVGL(λ = 0.5), and DHP approaches. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) has better performance than DHP, whereas it is faster improved during iteration.

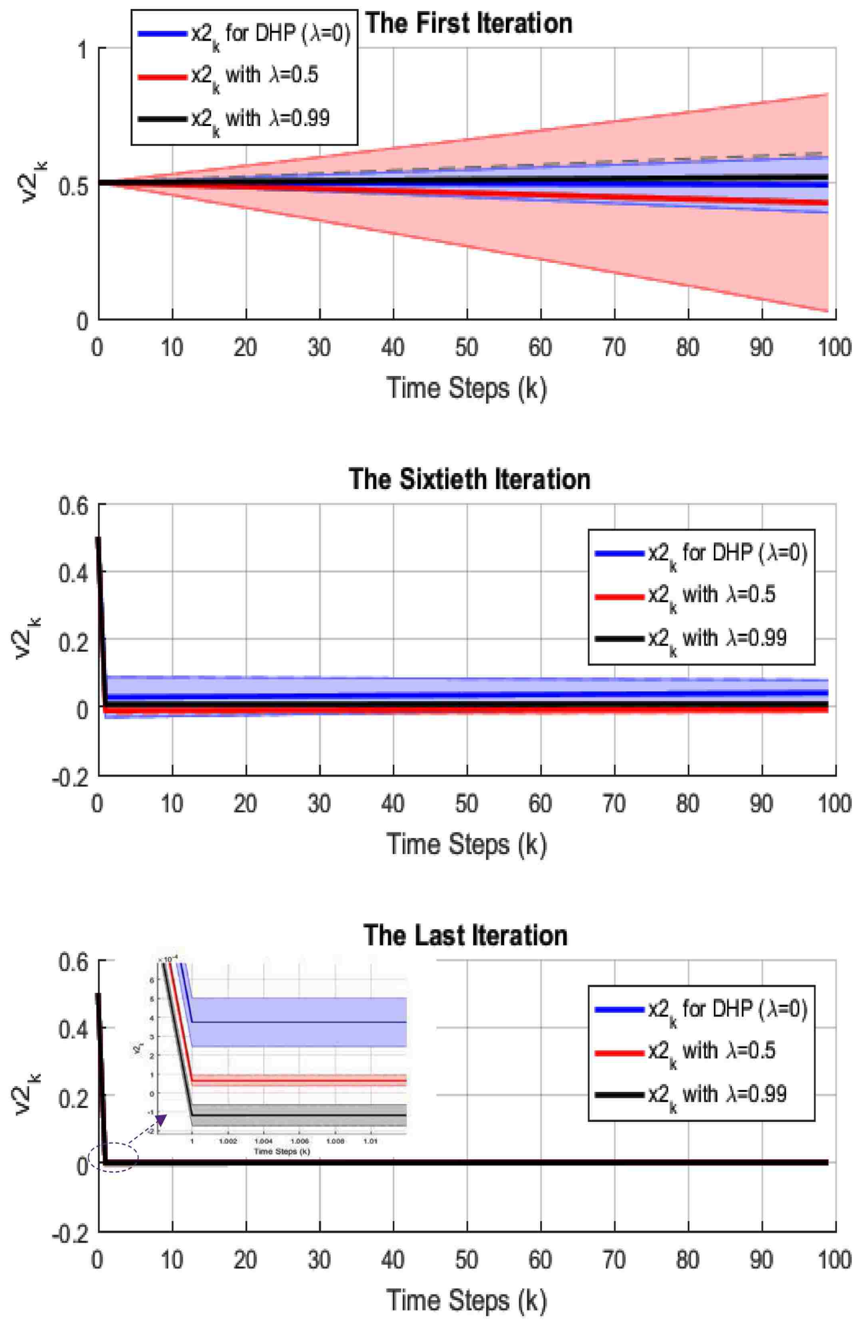


Figure 13. Comparisons of the state trajectories for the second state (angular velocity) for NSVGL(λ = 0.99), NSVGL(λ = 0.5), and DHP. The mean values from 10 independent runs are taken for all methods. The shaded region represents the 10 runs, while the solid line represents the mean for all 10 runs. NSVGL(λ) learns faster than DHP.

Fig. 14 and Fig. 15 demonstrate the convergence of the gradient of the cost functions during iterations for NSVGL($\lambda = 0.99$). The convergence of the gradient of the cost functions ($g_i^0(x_k)$ and $g_i^\lambda(x_k)$) to the optimal cost function indicates the effectiveness of the iterative NSVGL(λ).

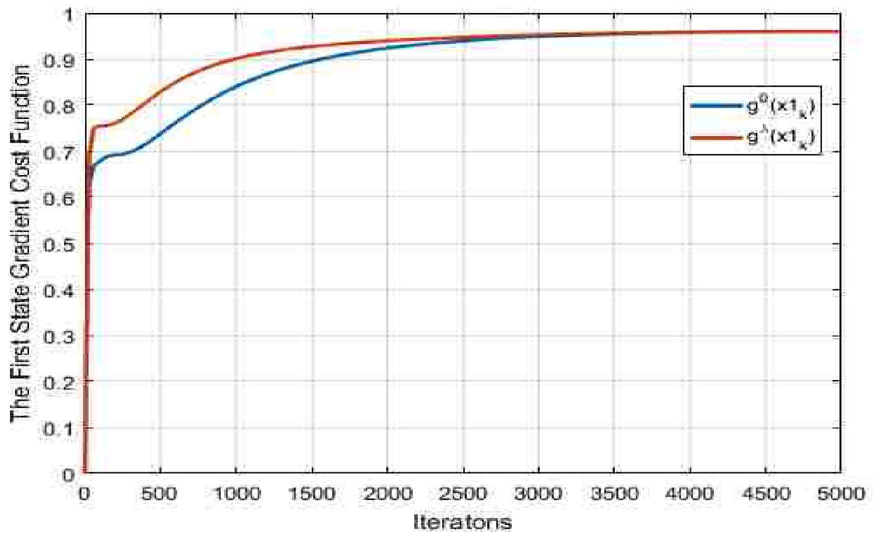


Figure 14. The average gradient of the first state of value function trajectories for both critic networks in the NSVGL($\lambda = 0.99$). The gradient of the value functions ($g_i^0(x1_k)$ and $g_i^\lambda(x1_k)$) are converged to the optimal cost function.

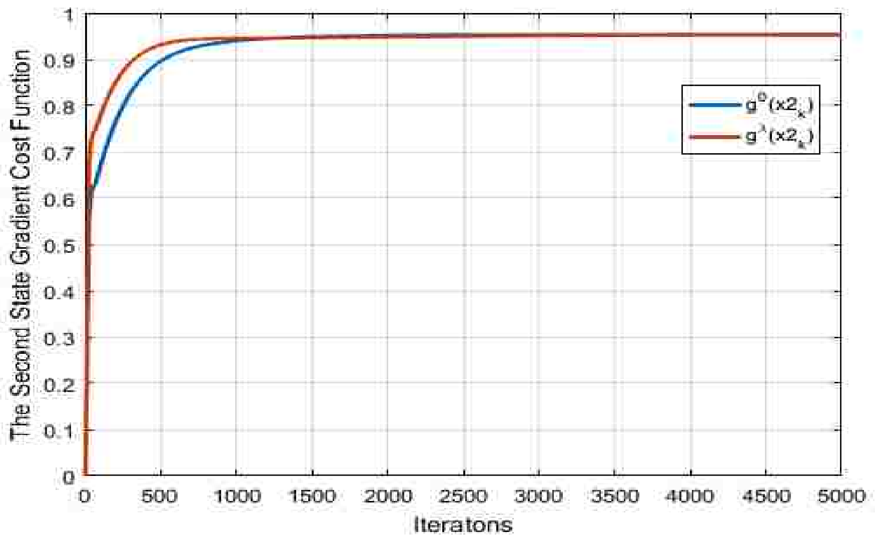


Figure 15. The average gradient of the second state of the value function trajectories for both critic networks for NSVGL($\lambda = 0.99$).

Algorithm 3 Pseudocode of Online NSVGL(λ)

```
Initializing MaxEpisode parameter
$Cnt_I = 0$ % counter for Iteration (Episode)
while ($Cnt_I < MaxEpisode$) do
    Init. $x_0, \hat{w}_c^0, \hat{w}_c^\lambda, \hat{w}_a, T_c^0, T_c^\lambda, N_c^\lambda, \Omega, \ell_c^0, \ell_c^\lambda, \ell_a$ and $N_c^0$
    $Cnt_k = 0$ % step counter from x(0) until finial state x(F)
    while ($Cnt_k < F$) do
        $\hat{\mu}(x_k) \leftarrow \hat{A}(x_k, \hat{w}_a)$
        $x_{k+1} \leftarrow f(x_k, \hat{\mu}(x_k))$
        $\Big[U_k, \dfrac{\partial U_k}{\partial x_k}, \dfrac{\partial U_k}{\partial \mu(x_k)}\Big]$ ; $(i.e. U_k = x_k^T Q x_k + \mu^T(x_k) R \mu(x_k))$
        $\hat{g}^0(x_{k+1}) \leftarrow \hat{G}^0(x_{k+1}, \hat{w}_c^0)$
        $\hat{g}^\lambda(x_{k+1}) \leftarrow \hat{G}^\lambda(x_{k+1}, \hat{w}_c^\lambda)$
        ----------------------------------------
        $P^0 = \hat{g}^0(x_{k+1})$
        $Cnt_c^0 = 0$ % counter for one-step critic network loop
        while ($Cnt_c^0 < N_c^0$) || ($E_c^0(x_k) > T_c^0$) do
            $D_{UX}(x_k) = \Big(\dfrac{\partial U_k}{\partial x_k}\Big) + \Big(\dfrac{\partial U_k}{\partial \mu(x_k)}\Big)\Big(\dfrac{\partial \hat{A}(x_k)}{\partial x_k}\Big)$
            $D_{FX}(x_k) = \Big(\dfrac{\partial f_k}{\partial x_k}\Big) + \Big(\dfrac{\partial f_k}{\partial \hat{\mu}(x_k)}\Big)\Big(\dfrac{\partial \hat{A}(x_k)}{\partial x_k}\Big)$
            $g^0(x_k) = D_{UX}(x_k) + \gamma D_{FX}(x_k) P^0$
            $e_c^0(x_k) = g^0(x_k) - \hat{g}^0(x_k)$
            $E_c^0(x_k) = 0.5(e_c^0(x_k))^2$
            $\hat{\omega}_c^0(x_k) = \hat{\omega}_c^0(x_k) + \ell_c^0\Big(\dfrac{\partial \hat{G}^0(x_k)}{\partial \hat{\omega}_c^0(x_k)}\Big) e_c^0(x_k)$
            $P^0 = \hat{g}^0(x_{k+1})$
            $Cnt_c^0 = Cnt_c^0 + 1$
        end while
        $P^\lambda = \lambda \hat{g}^\lambda(x_{k+1}) + (1-\lambda)\hat{g}^0(x_{k+1})$
        $Cnt_c^\lambda = 0$ % counter for $n$-step critic network loop
        while ($Cnt_c^\lambda < N_c^\lambda$) || ($E_c^\lambda(x_k) > T_c^\lambda$) do
            $D_{UX}(x_k) = \Big(\dfrac{\partial U_k}{\partial x_k}\Big) + \Big(\dfrac{\partial U_k}{\partial \mu(x_k)}\Big)\Big(\dfrac{\partial \hat{A}(x_k)}{\partial x_k}\Big)$
            $D_{FX}(x_k) = \Big(\dfrac{\partial f_k}{\partial x_k}\Big) + \Big(\dfrac{\partial f_k}{\partial \hat{\mu}(x_k)}\Big)\Big(\dfrac{\partial \hat{A}(x_k)}{\partial x_k}\Big)$
            $g^\lambda(x_k) = D_{UX}(x_k) + \gamma D_{FX}(x_k) P^\lambda$
            $e_c^\lambda(x_k) = g^\lambda(x_k) - \hat{g}^\lambda(x_k)$
            $E_c^\lambda(x_k) = 0.5(e_c^\lambda(x_k))^2$
            $\hat{\omega}_c^\lambda = \hat{\omega}_c^\lambda + \ell_c^\lambda\Big(\dfrac{\partial \hat{G}^\lambda(x_k)}{\partial \hat{\omega}_c^\lambda}\Big) e_c^\lambda(x_k)$
            $P^\lambda = \lambda \hat{g}^\lambda(x_{k+1}) + (1-\lambda)\hat{g}^0(x_{k+1})$
            $Cnt_c^\lambda = Cnt_c^\lambda + 1$
        end while
        $Cnt_a = 0$ % counter for the actor network loop
        while ($Cnt_a < N_a$) || ($E_a(x_k) > T_a$) do
            $P^0 = \hat{g}^0(x_{k+1})$
            $e_a^0(x_k) = \Big(\dfrac{\partial U_k}{\partial \hat{\mu}(x_k)}\Big) + \gamma\Big(\dfrac{\partial f_k}{\partial \hat{\mu}(x_k)}\Big) P^0$
            $P^\lambda = \lambda \hat{g}^\lambda(x_{k+1}) + (1-\lambda)\hat{g}^0(x_{k+1})$
            $e_a^\lambda(x_k) = \Big(\dfrac{\partial U_k}{\partial \hat{\mu}(x_k)}\Big) + \gamma\Big(\dfrac{\partial f_k}{\partial \hat{\mu}(x_k)}\Big) P^\lambda$
            $e_a(x_k) = \lambda e_a^\lambda(x_k) + (1-\lambda) e_a^0(x_k)$
            $E_a(x_k) = 0.5(e_a(x_k))^2$
            $\hat{\omega}_a = \hat{\omega}_a - \ell_a\Big(\dfrac{\partial \hat{A}(x_k)}{\partial \hat{\omega}_a}\Big) e_a(x_k)$
            $Cnt_a = Cnt_a + 1$
        end while
        $Cnt_k = Cnt_k + 1$
    end while
    $Cnt_I = Cnt_I + 1$
end while
```

Table 1. Parameters of the dynamic mobile robot

Symbol	Description	Value
m_T	Mass of the chassis	$10kg$
m_w	The mass of each wheel	$2kg$
$\bar{r}$	The wheel radius	$0.05m$
b	Half of the robot width	$0.1m$
d	The center of gravity offset form the rear axle	$0.1m$
I_{YY}^{b}	The wheel moment of inertial	$1kg.m^2$
I_T	The platform total moment of inertia	$5kg.m^2$
f_v	The viscous friction coefficient	$0.001N.m.s$
f_c	The Coulomb friction coefficient	$0.001N.m.s$

6. CONCLUSION

This paper presents a new ADP architecture, which merges between one- and n-step critic networks. The gradient of TD(0) error is used to learn the one-step critic network, while the gradient of TD(λ) error is used to learn the n-step critic notwork. The actor network is tuned by using these two TD errors via two filtering paths with a similar λ value. In addition to the gradient of TD(λ) providing the fast convergence learning to a locally-extreme optimal trajectory without exploration, this design provides direct implementation (online-mode) without the trajectory storage as in batch-mode implementation. Moreover, our design is more memory efficient by overcoming the drawback of using eligibility-trace storage for system states in online-mode implementation, which requires high computational complexity. Convergence proofs are provided for both gradients of one- and n-step value functions with respect to system states. We apply neural networks to implement our approach in two simulation case studies to verify the theoretical analyses in this work.

BIBLIOGRAPHY

[1] D. P. Bertsekas, “Approximate policy iteration: A survey and some new methods,” *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310 - 335, Aug. 2011.

[2] F. L. Lewis, and D. Vrabie, “Reinforcement learning and adaptive dynamic programming for feedback control,” *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32 - 50, Aug. 2009.

[3] R. Padhi, N. Unnikrishnan, X. Wang, and S. N. Balakrishnan, “A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems,” *Neural Netw.*, vol. 19, no. 10, pp. 1648 - 1660, Dec. 2006.

[4] B. Luo, D. Liu, T. Huang, and D.Wang, Member, “Model-Free Optimal Tracking Control via Critic-Only Q-Learning,” *Neural Netw.*, vol. 27, no. 10, pp. 2134 - 2144, Oct. 2016.

[5] J. Fu, H. He, and X. Zhou, “Adaptive learning and control for MIMO system based on adaptive dynamic programming,” *IEEE Trans. Neural Netw.*, vol. 22, no. 7, pp. 1133 - 1148, Jul. 2011.

[6] Z. Ni, H. He, and J. Wen, “Adaptive learning in tracking control based on the dual critic network design,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 6, pp. 913 - 928, Jun. 2013.

[7] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, Eds., *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2004.

[8] F. L. Lewis, and D. Liu, Eds., *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. Hoboken, NJ, USA: Wiley, 2013.

[9] Q. Wei, and D. Liu, “Adaptive dynamic programming for optimal tracking control of unknown nonlinear systems with application to coal gasification,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 4, pp. 1020 - 1036, Oct. 2014.

[10] X. Zhong, Z. Ni, and H. He, “A Theoretical Foundation of Goal Representation Heuristic Dynamic Programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[11] R. Bellman, *Dynamic Programming. Princet*on, NJ, USA: Princeton Univ. Press, 1957.

[12] D. Prokhorov, and D. C. Wunsch, “Adaptive critic designs,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 8, no. 8, pp. 997 - 1007, Sep. 1997.

[13] P. J. Werbos, “Approximate dynamic programming for real-time control and neural modeling,” Handbook of Intelligent Control: *Neural, Fuzzy, and Adaptive Approaches (Chapter 13)*, Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[14] F.-Y. Wang, H. Zhang, and D. Liu, “Adaptive dynamic programming: An introduction,” *IEEE Comput. Intell. Mag.*, vol. 4, no. 2, pp. 39 - 47, May 2009.

[15] F. L. Lewis, and D. Vrabie, “Reinforcement learning and adaptive dynamic programming for feedback control,” *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32 - 50, Sep. 2009.

[16] S. Ray, G. K. Venayagamoorthy, B. Chaudhuri, and R. Majumder, “Comparison of adaptive critic-based and classical wide-area controllers for power systems,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 4, pp. 1002 - 1007, Aug. 2008.

[17] H. Zhang, Y. Luo, and D. Liu, “Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints,” *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490 - 1503, Sep. 2009.

[18] J. Si, and Y. Wang, “Online learning control by association and reinforcement,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 12, no. 2, pp. 264 - 276, Mar. 2001.

[19] Z. Ni, H. He, X. Zhong, and D. Prokhorov, “Model-Free dual heuristic dynamic programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 8, pp. 1834 - 1839, Aug. 2015.

[20] H. He, Z. Ni, and J. Fu, “A three-network architecture for on-line learning and optimization based on adaptive dynamic programming,” Neurocomputing, vol. 78, no. 1, pp. 3 - 13, Feb. 2012.

[21] X. Fanga, D. Zhenga, H. He, and Z. Nib, “Data-driven heuristic dynamic programming with virtual reality,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 166, no. 6, pp. 244 - 255, Oct. 2015.

[22] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, “GrDHP: A general utility function representation for dual heuristic dynamic programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 3, pp 614 - 626, Mar. 2015.

[23] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Dual heuristic programming excitation neurocontrol for generators in a multimachine power system,” *IEEE Trans. Indus. Applying*, vol. 39, no. 2, pp. 382 - 394, Mar. 2003.

[24] N. Zhang, and D. C. Wunsch, “A Comparison of Dual Heuristic Programming (DHP) and neural network based stochastic optimization approach on collective robotic search problem,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 1, pp. 248 - 253, Jul. 2003.

[25] C. Lian, and X. Xu, “Motion planning of wheeled mobile robots based on heuristic dynamic programming,” *IEEE Proceeding of the World Congress on Intelligent Control and Automation Shenyang*, pp 576 - 580, Jul. 2014.

[26] S. Al-Dabooni, and D. Wunsch, “Heuristic dynamic programming for mobile robot path planning based on Dyna approach,” *IEEE, International Joint Conference on Neural Networks (IJCNN)*, pp. 3723 - 3730, Jul. 2016.

[27] B. Xu, C. Yang, and Z. Shi, “Reinforcement learning output feedback NN control using deterministic learning technique,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 635 - 641, Mar. 2014.

[28] Q. Wei, and D. Liu, “Data-driven neuro-optimal temperature control of waterâĂŞgas shift reaction using stable iterative adaptive dynamic programming,” *IEEE Trans. Ind. Electron.*, vol. 61, no. 11, pp. 6399 - 6408, Nov. 2014.

[29] D. Liu, D. Wang, F.-Y. Wang, H. Li, and X. Yang, “Neural-networkbased online HJB solution for optimal robust guaranteed cost control of continuous-time uncertain nonlinear systems,” *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2834 - 2847, Dec. 2014.

[30] H. He, Self-Adaptive Systems for *Machine Intelligence*. New York, NY, USA: Wiley, 2011.

[31] D. Liu, and Q. Wei, “Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 621 - 634, Mar. 2014.

[32] Y. Sokolov, R. Kozma, L. D. Werbos, and P. J. Werbos, “Complete stability analysis of a heuristic approximate dynamic programming control design,” *Automatica*, vol. 59, pp. 9 - 18, Sep. 2015.

[33] F. Liu, J. Sun, J. Si, W. Guo, and S. Mei, “A boundedness result for the direct heuristic dynamic programming,” *Neural Networks*, vol. 32, pp. 229-235, Aug. 2012.

[34] Y. Tang, H. He, Z. Ni, X. Zhong, D. Zhao, and X. Xu, “Fuzzy-Based Goal Representation Adaptive Dynamic Programming,” *IEEE Trans. on Fuzzy Sys.*, vol. 24, no. 5, pp. 1156 - 1176, Oct. 2016.

[35] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, “Discrete-time nonlinear hjb solution using approximate dynamic programming: convergence proof,” *IEEE Trans. on Sys., Man, and Cyb., Part B*, vol. 38, no. 4, pp. 943 - 949, Aug. 2008.

[36] D. Liu, and Q. Wei, “Finite-approximation-error-based optimal control approach for discrete-time nonlinear systems,” *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 779 - 789, Apr. 2013.

[37] D. Liu, and Q. Wei, “Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 621 - 634, Mar. 2014.

[38] H. Zhang, Y. Luo, and D. Liu, “Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints,” *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490 - 1503, Sep. 2009.

[39] X. Zhong , Z. Ni , and H. He, “A theoretical foundation of goal representation heuristic dynamic programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[40] Y. Yang, D. Wunsch, and Y. Yin, “Hamiltonian-driven adaptive dynamic programming for continuous nonlinear dynamical systems,” *IEEE Trans. Neural Netw. Learn. Syst.* vol. PP, no. 12, pp. 1 - 12, Feb. 2017 (to be published).

[41] D. Liu, D. Wang, D. Zhao, Q. Wei, and N. Jin, “Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming,” *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 3, pp. 628 - 634, Jul. 2012.

[42] D. Liu, and D. Wang, “Optimal control of unknown nonlinear discretetime systems using the iterative globalized dual heuristic programming algorithm,” in *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. New York, NY, USA: Wiley, pp. 52 - 74, Jan. 2013,

[43] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, no. 1, pp. 9 - 44, Aug. 1988.

[44] H. Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, “True Online Temporal-Difference Learning,” *Journal of Machine Learning Research (JMLR)*, vol. 145, no. 17, pp. 1 - 40, Jan. 2016.

[45] R. S. Sutton, A. R. Mahmood, and M. White, “An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning,” *Journal of Machine Learning Research (JMLR)*, vol. 73, no. 17 pp. 1 - 29, Jan. 2016.

[46] H. Seijen, and R. S. Sutton, “True Online TD(λ),” *Proceedings of the 31 st International Conference on Machine Learning*, pp. 692 - 700, Jan. 2014.

[47] M. Fairbank, and E. Alonso, “Value-Gradient Learning,” *IEEE, International Joint Conference on Neural Networks (IJCNN)*, pp. 1 - 8, Jun. 2012.

[48] F. L. Lewis, and D. Liu, Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, Chapter 7, John Wiley and Sons, Jan. 2013.

[49] S. Al-Dabooni, and D. Wunsch, “Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning,” *IEEE, International Joint Conference on Neural Networks (IJCNN)*, pp. 2820 - 2827, May 2017.

[50] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. NewYork, NY, USA: Wiley, 2012.

[51] M. Fairbank, E. Alonso, and D. Prokhorov, “An Equivalence Between Adaptive Dynamic Programming With a Critic and Backpropagation Through Time,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 12, pp. 2088 - 2100, Dec. 2013.

[52] R. S. Sutton, and A. Barto, *Reinforcement Learning: An Introduction*, Cambridge, U.K.: Cambridge Univ. Press, 1998.

[53] S. P. Singh, and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” *Machine Learning*, vol. 22, no. 1, pp. 123 - 158, Mar. 1996.

[54] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219 - 245, Jan. 2000.

[55] S. Al-Dabooni, and D. Wunsch, "Online Model-Free N-Step HDP with Stability Analysis," Under Preparing.

[56] S. Al-Dabooni, and D. Wunsch, "The Boundedness Conditions for Model-Free HDP(λ)," Under reviewing for *IEEE Trans. Neural Netw. Learn. Syst.*

[57] Z. Ni, H. He, J. Wen, and X. Xu, "Goal representation heuristic dynamic programming on maze navigation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 12, pp. 2038 - 2050, Dec. 2013.

[58] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550 - 1560, Oct. 1990.

[59] G. Zhang, M. Y. Hu, B. E. Patuwo, and D. C. Indro, "Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis," *European Journal of Operational Research*, vol. 116, no. 1, pp. 16 - 32, Jul. 1999.

[60] Y. Xiao, M. Wasei, P. Hu, P. Wieringa, and F. Dexter, "Dynamic Management of Perioperative Processes: A Modeling and Visualization Paradigm," *International Federation of Automatic Control (IFAC)*, vol. 39, no. 3, pp. 647 - 652, Jan. 2006.

[61] K. L. Keller, and R. Staelin, "Effects of Quality and Quantity of Information on Decision Effectiveness," *Journal of Consumer Research*, vol. 14, no. 2, pp. 200 - 213, Sep. 1987.

[62] X. Zhong, Z. Ni, and H. He, "A Theoretical Foundation of Goal Representation Heuristic Dynamic Programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[63] C. Chen, D. Dong, H. Li, J. Chu, and T. J. Tarn, "Fidelity-Based Probabilistic Q-Learning for Control of Quantum Systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol.5, no. 10, pp. 920 - 933, May 2014.

[64] R. Ilin, R. Kozma, and P. Werbos, "Beyond feedforward models trained by backpropagation: A practical training tool for a more efficient universal approximator," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 19, no. 6, pp. 929 - 937, Jan. 2008.

[65] Z. Ni, H. He, J. Wen, and X. Xu, "Goal Representation Heuristic Dynamic Programming on Maze Navigation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 12, pp. 2038 - 2050, Dec. 2013.

[66] R. Fierro, and F. L. Lewis, "Control of a Nonholonomic Mobile Robot Using Neural Networks," *IEEE Trans. Neural Networks*, vol. 9, no. 4, pp. 589 - 600, July 1998.

[67] W. S. Lin, L. H. Chang, and P. C. Yang, "Adaptive critic anti-slip control of wheeled autonomous robot," *IET Control Theory and Applications*, vol. 1, no. 1, pp. 51 - 57, Jan. 2007.

[68] T. Dierks, and S. Jagannathan, “Neural Network Output Feedback Control of Robot Formations,” *IEEE Trans. on Sys, Man, and Cyb.*, vol. 40, no.2, pp. 383 - 399, Apr. 2010.

VII. CONVERGENCE ANALYSIS PROOFS FOR RECURRENT NEURO-FUZZY VALUE-GRADIENT LEARNING WITH AND WITHOUT ACTOR

S. Al-Dabooni and Donald C. Wunsch

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–202–0445; 573–341–4521 Email: sjamw3@mst.edu; dwunsch@mst.edu

ABSTRACT

A gradient of the n-step temporal-difference (TD(λ)) is utilized as a learning algorithm to train an advanced Adaptive Dynamic programming (ADP) algorithm, which is called value-gradient learning (VGL(λ)). The VGL(λ) architecture with a single adaptive actor network (SNVGL(λ)) is derived and implemented to be compared with the regular VGL(λ). Moreover, a recurrent hybrid neuro-fuzzy (RNF) and a first-order Takagi-Sugeno RNF (TSRNF) are two structures that are presented to build the critic and actor networks for VGL(λ) and the critic network for SNVGL(λ). The fuzzy rules and the membership functions' (MFs) adjusted parameters are trained by a backpropagation gradient descent technique. Furthermore, convergence theoretical analysis proofs are demonstrated based on the iterative ADP strategy. A mobile robot simulation case study is presented with various amount of uncertainties and frictions to verify the performance and the theoretical analysis.

Keywords: Adaptive dynamic programming (ADP), recurrent neuro-fuzzy, Takagi-Sugeno neuro-fuzzy, eligibility traces, convergence analysis, mobile robot, single network adaptive critic.

1. INTRODUCTION

Adaptive dynamic programming (ADP) is a useful mechanism tool for solving the Hamilton-Jacobi-Bellman (HJB) equation instead of the Riccati equation [1] - [11].

Heuristic dynamic programming (HDP), dual heuristic programming (DHP) and globalized DHP (GDHP) are three fundamental categories for ADP [11] - [14]. Three approximation function networks are used to perform actor, critic and model networks that provide decision making, evaluation and prediction, respectively. Since a model network that predicts the future system state, is included, these ADP categories are model based ADP [11] - [17]. If the action-dependent (AD) prefix is used (i.e., ADHDP for HDP and ADDHP for DHP), the critic network has the state and the action inputs and model-free variants. In [18] - [22], model-free ADP designs were presented for online learning. As reviewed in [11], the ADP technique is used to train an actor network to give optimal actions by minimizing a value function that is produced from a critic network. Both networks are approximated by using a multilayer perceptron. Many applications have used ADP. In [23], DHP controlled a turbo-generator more efficient than HDP. Collective robotic search problems can be solved with improved performance by using DHP as in [24]. Lian and Xu [25] applied HDP to allow a mobile robot to escape from sharp corners. Maze navigation has been proposed as an ADP benchmark [26]-[31], but most of mazes have been 2-D. Al-Dabooni and Wunsch [32] use ADHDP(λ) in the 3-D maze navigation benchmark. Also, the previous authers [33] applied model-free ADHDP in the Dyna algorithm to obtain theoptimal path by using multi-robot navigation in an unknown environment. Other theoretical and practical works in ADP are presented in [34] - [38].

The stability of ADP in general cases is an open problem [39]. The stability of the one-step model-free ADHDP learning approach is introduced by Liu *et al.* [40] and by Werbos *et al.* [39] with critic/actor neural networks and by He *et al.* [41], [42] with critic/reference neural networks and a fuzzy logic controller. The stability for an n-step model-free ADHDP is presented in [32], [29]. Al-Tamimi *et al.* in [43] demonstrates a convergence analysis of value iteration based on HDP for general discrete-time nonlinear systems. Many other publications regarding the theoretical analysis and proofs for ADP are shown in [44] - [47]. The GDHP convergence analysis proof and its comparison with the HDP and the DHP approaches is presented by Liu [48], [49]. Sutton *et al.* in [50] - [54] show the efficient performance of temporal difference (TD) learning with an eligibility trace long-predation parameter denoted by λ.

Inspired by [50], Fairbank and Alonso [55] - [57] introduced a new ADP algorithm to extend DHP by including λ. They called it value-gradient learning (VGL(λ)). VGL(λ) was used in [58] to track a reference trajectory under uncertainties by computing the optimal left and right torques for a nonholonomic mobile robot. Al-Dabooni and Wunsch in [59] use on-line learning of VGL(λ) without requiring an eligibility-trace-work-space matrix.

Other papers exist in the literature that use hybrid neuro fuzzy (NF) systems for ADP. D. Zhao *et al.* in [60] explains how to use an NF with monotonic membership functions in the first (premise/ antecedent) layer with regular connected weights in the output (consequent) layer. They use ADHDP with a traditional feed-forward neural network for the critic network and NF with an actor network for controlling the two benchmarks: a two-link robot arm and cart-pole balancing. The authors in [60] illustrate how the systems become steady and robust with uncertainties. S. Mohagheghi *et al.* in [61] and [62] use ADHDP to control on a multi-machine power system via a static compensator. The authors use an adaptive neuro-fuzzy inference system (ANFIS) in the actor network with a zero-order Takagi-Sugeno fuzzy. Y. Zhu *et al.* in [63] also uses ADHDP mixed with an NF network for control on a rotational inverted pendulum. Y. Tang *et al.* in [41] presents a new structure that uses ADHDP and adds another network. The new structure is called a goal representation HDP (GrHDP); therefore, GrHDP has three networks: actor, critic and goal networks. Both critic and actor are represented by using a regular feedforward neural network, while the actor network is built by using monotonic membership functions in the premise layer with regular connected weights in the consequent layer. They test their structure with a cart-pole plant, a ball-and-beam system and a multi-machine power system control. Model-free GrHDP mixing with NF strucure is also presented in [64] by Y. Tang *et al.* It uses a damping controller for superconducting magnetic energy storage. H. Zhang *et al.* in [65] shows how a consensus problem of multiagent differential games is solved by using this paper and an NF for the critic network. The authors implement the actor network and use a policy iteration algorithm to find the optimal action sequence that provides uniformly ultimately bounded proofs. The NF structure in a critic network without an actor network is also shown by J. Zhang *et al.* in [66] by solving HJB. Using ANFIS with eligibility traces of ADP is introduced in [67] by X. Bai *et al.* to solve a multiple ramps metering problem. The authors use a backward view for eligibility traces with an ADHDP scheme. ANFIS is also used with ADHDP for both critic and actor networks, which is presented by X. Luo *et al.* in [68] to evaluate the quality of a wen service by providing ultimately uniformly bounded stability proofs.

This paper uses the VGL(λ) approach with various NF structures to summarize the following fundamental contributions of this paper

1. A theoretical foundation analysis for an n-step adaptive actor-critic approach of VGL(λ) architecture with NF (NF-VGL(λ)) is presented that illustrates how the agent receives better information from the control action than traditional DHP.

2. A single adaptive n-step critic approach of VGL(λ) (SNVGL(λ)) is derived to create a pioneer architecture of SNVGL(λ). SNVGL(λ) uses NF structures (NF-SNVGL(λ)) to compare with the first contribution.

3. One of the NF structure that uses in VGL(λ) and SNVGL(λ) is similar to [60], [63] - [64] but with a different membership function (MF) and with a feedback loop in one of the premise parameters as a recurrent network. The other NF structure is similar

to ANFIS as in [61], [62], [67], [68] but with a feedback loop in one of the premise parameters to store history information. It uses backpropagation gradient descent technique to train the fuzzy rules and MFs as well.

4. A theoretical convergence analysis proofs are provided for the VGL(λ) and SNVGL(λ) architectures by using an iterative ADP algorithm. This paper demonstrates that gradients are monotonically nondecreasing and converge to optimal values.

5. These advantages of VGL(λ) and SNVGL(λ) with and without recurrent feedback parameters of NF structures are verified by simulation with a high-nonlinear dynamic model case study with various uncertainties.

The schematic architectures for VGL(λ) and SNVGL(λ) are presented in Section 2. The NF structures are shown in Section 3. Section 4 provides a convergence stability analysis for the VGL(λ) and SNVGL(λ) designs. Harmonizing an iterative algorithm for VGL(λ) and SNVGL(λ) with NF structures is presented in Section 5. The simulation results and the conclusion are presented in Section 6 and Section 7, respectively.

2. THE ACTOR-CRITIC AND SINGLE-CRITIC OF VGL(λ) ARCHITECTURE DESIGNS

ADP allows agents to select an action to minimize their long-term cost:

$$J(i) = \langle \sum_{k=i}^{\infty} \gamma^{k-i} U\Big(x(k), u\big(x(k)\big)\Big) \rangle, \tag{1}$$

where $\langle . \rangle$ is the expectation symbol, $J(i)$ is a value function (cost-to-go value or long-term cost) for a state vector ($x \in \mathbb{R}^{\mathrm{p}}$) at initial time i, γ is a constant discount factor, and $U\Big(x(k), u\big(x(k)\big)\Big)$ is an instantaneous utility cost function at time step k for x after applying an action vector $u \in \mathbb{R}^{\mathrm{q}}$. NF-VGL($\lambda$) uses TD($\lambda$) learning that helps to fill the gaps between predicted events and training parameters. [51] - [54] illustrate that TD(λ) learning has fundamental advantages in reinforcement learning. In this paper, a gradient of TD(λ) is used with ADP to solve the recursive form of the optimal Bellman equation [10], [69]:

$$J^*(k) = \min_{u\left(x(k)\right)} \{U\left(x(k), u\left(x(k)\right)\right) + \gamma J^*(k+1))\}, \tag{2}$$

where J^* denotes the optimal value function, and the instantaneous cost, $U\left(.\right)$, is bounded. For simplicity, we denote $J^*\left(k\right) = J^*\left(x(k)\right)$, and $J^*\left(k+1\right) = J^*\left(x(k+1)\right)$. $x(k+1)$ is a next state that is provided from discrete-time nonlinear system as follows:

$$x(k+1) = f\left(x(k), u\left(x(k)\right)\right). \tag{3}$$

As in [70], the Bellman equations for one-step ($R^{(1)}(k)$), two-step ($R^{(2)}(k)$) until n-step ($R^{(n)}(k)$) are

$$\begin{aligned}
J(k) =& R^{(1)}(k) = U\left(x(k), u\left(x(k)\right)\right) + \gamma J(k+1), \\
&= R^{(2)}(k) = U\left(x(k), u\left(x(k)\right)\right) + \gamma U\left(x(k+1), u\left(x(k+1)\right)\right) \\
&\quad + \gamma^2 J(k+2), \\
&\vdots \\
&= R^{(n)}(k) = U\left(x(k), u\left(x(k)\right)\right) + \gamma U\left(x(k+1), u\left(x(k+1)\right)\right) \\
&\quad + \ldots + \gamma^{n-1} U\left(x(k+n-1), u\left(x(k+n-1)\right)\right) + \\
&\quad \gamma^n J(k+n) = \sum_{j=0}^{n} \left[\gamma^j U\left(x(k+j), u\left(x(k+j)\right)\right)\right].
\end{aligned} \tag{4}$$

where $R^{(i)}(k) = R^{(i)}\left(x(k)\right)$, which is denoted as an actual return for the value function from state time k to i. The average of an n-step-return is a technique for accelerating the optimization [32], [52], [70]. The λ-return, $R^\lambda(k) = R^\lambda\left(x(k)\right)$, is another name for the average of the n-step-return [70], which is defined in general as

$$R^{\lambda}(k) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R^{(n)}(k). \tag{5}$$

This paper's authors' previous work [32] shows a stability proof for selecting λ, which should be $0 \leq \lambda < 1$ by using TD(λ) learning (in contrast to previous literature, which are included bounded from 0 to 1). By substituting (4) into (5), authors obtain:

$$\begin{aligned} R^{\lambda}(k) &= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\left(\sum_{l=0}^{n-1}\left[\gamma^{l} U\Big(x(k+l), u\big(x(k+l)\big)\Big)\right] + \gamma^{n} J(k+n)\right). \\ &= (1-\lambda)\left(\sum_{n=1}^{\infty}\left(\lambda^{n-1}\sum_{l=0}^{n-1}\left[\gamma^{l} U\Big(x(k+l), u\big(x(k+l)\big)\Big)\right]\right) + \sum_{n=1}^{\infty}\left[\lambda^{n-1}\gamma^{n} J(k+n)\right]\right). \end{aligned} \tag{6}$$

Expanding and rearranging (6) yields:

$$\begin{aligned} R^{\lambda}(k) = &\sum_{n=0}^{\infty}\Bigg[\lambda^{n}\gamma^{n}\Big[U\Big(x(k+n), u\big(x(k+n)\big)\Big) + \\ &(1-\lambda)\gamma J(k+n+1)\Big]\Bigg], \\ =&U\Big(x(k), u\big(x(k)\big)\Big) + \lambda\gamma\Bigg[U\Big(x(k+l), u\big(x(k+l)\big)\Big) + \\ &\lambda\gamma\Bigg[U\Big(x(k+2), u\big(x(k+2)\big)\Big) + \lambda\gamma\Bigg[U\Big(x(k+3), \\ &u\big(x(k+3)\big)\Big) + \ldots + \lambda\gamma\Big[U\Big(x(\infty), u\big(x(\infty)\big)\Big) + \\ &(1-\lambda)\gamma J(\infty)\Big] + \ldots + (1-\lambda)\gamma J(k+4)\Bigg] + (1-\lambda) \\ &\gamma J(k+3)\Bigg] + (1-\lambda)\gamma J(k+2)\Bigg] + (1-\lambda)\gamma J(k+1), \end{aligned} \tag{7}$$

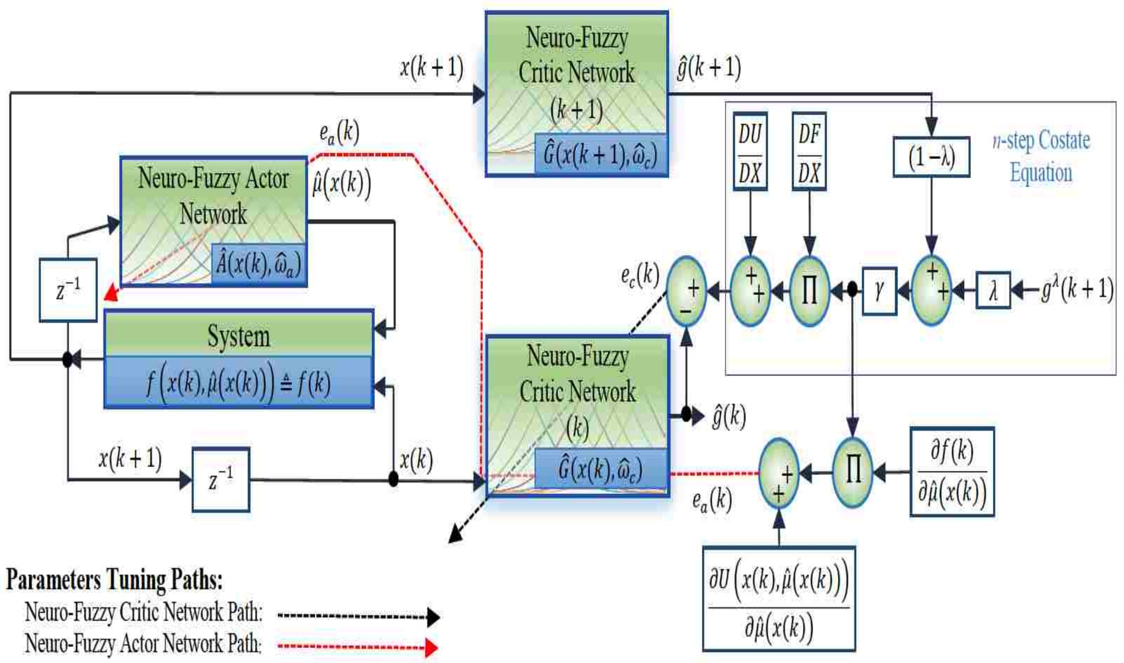
where $U\Big(x(\infty), u\big(x(\infty)\big)\Big) = R^{\lambda}(\infty)$, which is the instantaneous cost at the infinite horizon terminal state. The final target-value according to (7) is

$$R^{\lambda}(k) = U\Big(x(k), u\big(x(k)\big)\Big) + \lambda\gamma R^{\lambda}(k+1) + (1-\lambda)\gamma J(k+1), \tag{8}$$

(derived in [32]). A stochastic exploration should be supplemented with any value function that approaches as shown by Fairbank and Alonso [55]. But in the VGL method, a gradient of a value function handles this requirement via a greedy policy, with few trajectories, instead of learning in all of the state space [55]. The FN-VGL(λ) uses the gradient of the n-step value function. The n-step value function, which is the output vector from critic network $\hat{G}(x(k), \hat{\omega}_c)$ is a function approximator with parameter vector $\hat{\omega}_c$. $\hat{G}(x(k), \hat{\omega}_c)$. The function network provides the estimated gradient of the n-step value approximator function with respect to the system state vector $(\partial J(k)/\partial x(k))$. There are two different ways to implement [58]: The *scalar critic* method makes the output critic network equal to $\partial J(k)/\partial x(k)$. The *vector critic* method takes a direct vector output from $\hat{G}(x(k), \hat{\omega}_c)$ to provide the n-step gradient value function. This paper uses the *vector critic* method because it provides a smooth and stable vector output [55], [57], [56], specifically. The VGL(λ) with two hybrid recurrent fuzzy neural network structures for both the critic and actor networks. Furthermore, this work eliminates the actor network from NF-SNVGL(λ) structure by calculating the optimal control and co-state equations.

2.1. Adaptive Actor-Critic Approach. Fig. 1 shows a schematic diagram for the NF-VGL(λ) structure.

2.1.1. The n-step Critic Network. The critic network in the FN-VGL(λ) scheme estimates the partial derivatives of the value function with respect to the system's state vector. In order to learn a critic network, which is represented as $\hat{G}(x(k), \hat{\omega}_c)$, the left-hand


Neuro-Fuzzy Critic Network (k+1)
$\hat{G}(x(k+1),\hat{\omega}_c)$
$x(k+1)$
$\hat{g}(k+1)$
$e_a(k)$
Neuro-Fuzzy Actor Network
$\hat{A}(x(k),\hat{\omega}_a)$
$\hat{\mu}(x(k))$
z^{-1}
System
$f(x(k),\hat{\mu}(x(k))) \triangleq f(k)$
$x(k+1)$
z^{-1}
$x(k)$
Neuro-Fuzzy Critic Network (k)
$\hat{G}(x(k),\hat{\omega}_c)$
$\hat{g}(k)$
$e_a(k)$
$e_c(k)$
$\frac{DU}{DX}$
$\frac{DF}{DX}$
$(1-\lambda)$
n-step Costate Equation
Π
γ
λ
$g^{\lambda}(k+1)$
Π
$\frac{\partial f(k)}{\partial \hat{\mu}(x(k))}$
$\frac{\partial U(x(k),\hat{\mu}(x(k)))}{\partial \hat{\mu}(x(k))}$
Parameters Tuning Paths:
Neuro-Fuzzy Critic Network Path:
Neuro-Fuzzy Actor Network Path:

Backpropagation Calculation Errors:

Neuro-Fuzzy Critic Network Error (Gradient of TD (λ)-Error): $e_c(k) = g^{\lambda}(k) - \hat{g}(k)$; $g^{\lambda}(k) = \left(\frac{DU}{DX}\right)_k + \gamma\left(\frac{DF}{DX}\right)_k \kappa$;

$\Delta\hat{\omega}_c = l_c[\frac{\partial \hat{G}(x(k),\hat{\omega}_c)}{\partial \hat{w}_c}]\, e_c(k)$.

Neuro-Fuzzy Actor Network Error: $e_a(k) = \frac{\partial U(x(k),\hat{\mu}(x(k)))}{\partial \hat{\mu}(x(k))} + \gamma\left[\frac{\partial f(k)}{\partial \hat{\mu}(x(k))}\right]^T \kappa$; $\Delta\hat{\omega}_a = \ell_a[\frac{\partial \hat{A}(x(k),\hat{\omega}_a)}{\partial \hat{w}_a}]e_a(k)$.

Note that $\kappa = \lambda g^{\lambda}(k+1) + (1-\lambda)\hat{g}(k+1)$, $\left(\frac{DU}{DX}\right)_k = \frac{\partial U(x(k),\hat{\mu}(x(k)))}{\partial x(k)} + \left[\frac{\partial \hat{A}(x(k),\hat{\omega}_a)}{\partial x(k)}\right]^T \frac{\partial U(x(k),\hat{\mu}(x(k)))}{\partial \hat{\mu}(x(k))}$, and

$\left(\frac{DF}{DX}\right)_k = \left[\frac{\partial f(k)}{\partial x(k)} + \frac{\partial f(k)}{\partial \hat{\mu}(x(k))}\frac{\partial \hat{A}(x(k),\hat{\omega}_a)}{\partial x(k)}\right]^T$.

Figure 1. A schematic diagram for the adaptation of actor-critic VGL(λ). The weights for the critic network ($\hat{G}(x(k),\hat{\omega}_c)$) are updated according to the gradient of the TD(λ) error (black dashed line). The actor network ($\hat{A}(x(k),\hat{\omega}_a)$) is tuned by backpropagating the actor error (e_a) through $\hat{G}(x(k),\hat{\omega}_c)$ network (red dashed line).

side of (8) is derived with respect to the state vector for a target value of $g^{\lambda}(k)$:

$$\frac{\partial R^{\lambda}(k)}{\partial x(k)} = g^{\lambda}(k) = \frac{\partial}{\partial x(k)}\left(U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big) + \gamma\lambda R^{\lambda}(k+1) + \gamma(1-\lambda)J(k+1)\right), \tag{9}$$

where $\hat{\mu}\big(x(k)\big)$ is a control action vector, which is provided from the approximating actor network $\hat{A}\big(x(k), \hat{\omega}_a\big)$ with parameter vector $\hat{\omega}_a$ (details about the actor network will be presented in the next subsection). Applying the chain rules to (9) yields an *n-step costate equation*, which is

$$\begin{aligned} g^{\lambda}(k) =& \left(\frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} + \left[\frac{\partial \hat{A}\Big(x(k), \hat{\omega}_a\Big)}{\partial x(k)}\right]^T \frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)}\right) \\ & + \gamma\left(\lambda\left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} + \frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)}\right.\right. \\ & \left.\frac{\partial \hat{A}\Big(x(k), \hat{\omega}_a\Big)}{\partial x(k)}\right]^T g^{\lambda}(k+1) + (1-\lambda)\left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} + \right. \\ & \left.\left.\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} \frac{\partial \hat{A}\Big(x(k), \hat{\omega}_a\Big)}{\partial x(k)}\right]^T g(k+1)\right). \end{aligned} \tag{10}$$

As shown in Fig. 1, the n-step critic error ($e_c(k)$) should be minimized during training by following

$$e_c(k) = g^{\lambda}(k) - \hat{g}(k), \tag{11}$$

where $\hat{g}(k)$ is an estimated output vector that is provided from the $\hat{G}(x(k), \hat{\omega}_c^0)$ network.

2.1.2. The Actor Network. The actor is used to make a system learn to select an optimal decision. The adaptive actor-critic approach has two separated networks (actor and critic) as shown in Fig. 1. As in [11], [55], [58], [59], the actor network function learns via minimizing $\frac{\partial J(k)}{\partial \hat{\mu}\big(x(k)\big)}$ to zero. Because using J value, it was called as a one-step optimal control equation. In this work, the actor network function learns via minimizing $\frac{\partial R^{\lambda}(k)}{\partial \hat{\mu}\big(x(k)\big)}$

to zero (an n-step optimal control equation). Therefore the actor error e_a is defined by

$$e_a(k) = \frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} + \gamma \left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} \right]^T \kappa, \tag{12}$$

where

$$\kappa = \lambda g^{\lambda}(k+1) + (1-\lambda)\hat{g}(k+1). \tag{13}$$

2.2. Single Adaptive n-step Critic Approach. Fig. 2 shows a schematic diagram for a neuro-fuzzy single n-step adaptive critic network VGL(λ) (NF-SNVGL(λ)) structure. Because of the elimination of the actor network, the NF-SNVGL(λ) has a low computational load that comes from the simple structure.

2.2.1. The n-step Optimal Control Equation. The necessary condition for the n-step optimal control equation is given by

$$\frac{\partial R^{\lambda}(k)}{\partial \hat{\mu}\big(x(k)\big)} = \frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} + \gamma \left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} \right]^T \kappa = 0. \tag{14}$$

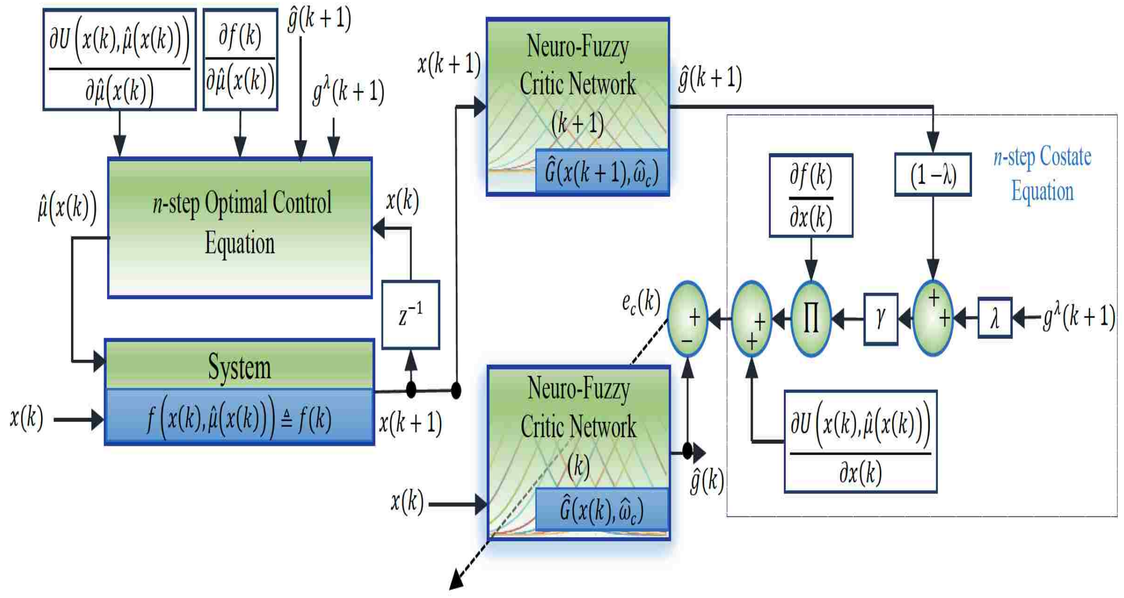
2.2.2. The n-step Critic Network. The n-step *costate equation* is similar to (10) with further arrangement, which becomes

$$\begin{aligned} g^{\lambda}(k) = & \left(\frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} + \gamma \left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} \right]^T \kappa \right) \\ & + \left[\frac{\partial \hat{A}\Big(x(k), \hat{\omega}_a\Big)}{\partial x(k)} \right]^T \left(\frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} + \gamma \left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial \hat{\mu}\big(x(k)\big)} \right]^T \kappa \right), \end{aligned} \tag{15}$$

where κ is defined in (13). By using (14) in (15), the n-step *costate equation* is [6]

$$g^{\lambda}(k) = \left(\frac{\partial U\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} + \gamma \left[\frac{\partial f\Big(x(k), \hat{\mu}\big(x(k)\big)\Big)}{\partial x(k)} \right]^T \kappa \right). \tag{16}$$

[6]During training, the n-step critic network is tuned by minimizing $e_c(k) = g^{\lambda}(k) - \hat{g}(k)$ (see Fig. 2).


$\frac{\partial U(x(k),\hat\mu(x(k)))}{\partial\hat\mu(x(k))}$
$\frac{\partial f(k)}{\partial\hat\mu(x(k))}$
$\hat g(k+1)$
$g^\lambda(k+1)$
$x(k+1)$
Neuro-Fuzzy Critic Network $(k+1)$
$\hat G(x(k+1),\hat\omega_c)$
$\hat g(k+1)$
n-step Costate Equation
$\hat\mu(x(k))$
n-step Optimal Control Equation
$x(k)$
$\frac{\partial f(k)}{\partial x(k)}$
$(1-\lambda)$
z^{-1}
$e_c(k)$
Π
γ
λ
$g^\lambda(k+1)$
System
$x(k)$
$f(x(k),\hat\mu(x(k))) \triangleq f(k)$
$x(k+1)$
Neuro-Fuzzy Critic Network (k)
$\frac{\partial U(x(k),\hat\mu(x(k)))}{\partial x(k)}$
$\hat g(k)$
$x(k)$
$\hat G(x(k),\hat\omega_c)$

Parameters Tuning Paths:

Neuro-Fuzzy Critic Network Path: $\dashrightarrow$

Backpropagation Calculation Error for only critic network:

Neuro-Fuzzy Critic Network Error (Gradient of TD (λ)-Error): $e_c(k) = g^\lambda(k) - \hat g(k)$; $g^\lambda(k) = \frac{\partial U(x(k),\hat\mu(x(k)))}{\partial x(k)} + \left[\frac{\partial f(k)}{\partial x(k)}\right]^T \kappa$

$\Delta\hat\omega_c = l_c\left[\frac{\partial \hat G(x(k),\hat\omega_c)}{\partial \hat w_c}\right] e_c(k)$, where $\kappa = \lambda g^\lambda(k+1) + (1-\lambda)\hat g(k+1)$.

Figure 2. A schematic diagram for the adaptation of a single n-step critic network of VGL(λ)(SNVGL(λ)). The weights for the critic network ($\hat G(x(k),\hat\omega_c)$) are updated according to the gradient of the TD(λ) error (black dashed line). The general n-step optimal control equation (14) (or (73) for Affine systems with the quadratic form of a utility function) is used to generate an optimal control signal.

3. NF STRUCTURES FOR VGL(λ) AND SNVGL(λ) APPROACHES

3.1. Recurrent Neuro-Fuzzy (RNF) Structure. For simplicity, Fig. (3) illustrates a structure for RNF for three inputs ($\mathfrak{n} = 3$), two outputs ($p = 2$) and two MFs ($m = 2$) for each input. This structure is similar to [71] but with a "grid" input space partitioning, which is used in many existing NF in ADP [61], [41], [64]. The grid-type partition divides the input space according to the number of MFs for each input variable (see [72]). This structure has three layers:

1. The first layer (premise layer) has premise parameters ($\mathbf{\mathfrak{m}}_{\mathfrak{n}\times m}$, $\boldsymbol{\sigma}_{\mathfrak{n}\times m}$, $\boldsymbol{\theta}_{\mathfrak{n}\times m}$), where the bold symbol is the matrix with $\mathfrak{n}\times m$ dimension. This layer has Gaussian membership functions (GMFs), which is denoted as G_i^f, represented by

$$O_i^f\left(x_i(k)\right) = e^{-\left(\frac{x_i(k) + O_i^f\left(x_i(k-1)\right)\theta_i^f(k) - \mathfrak{m}_i^f(k)}{\sigma_i^f(k)}\right)^2}, \tag{17}$$

where $i = 1, 2, \ldots, \mathfrak{n}$ and $f = 1, 2, \ldots, m$, x_i is the i input variable, $\mathfrak{m}_i^f \in U_i$ is the mean parameter that belong to the i input universe of discourse with the MF number f ($U_i \subset \mathbb{R}^{\mathfrak{n}}$), and $\sigma_i^f > 0$ are the variance parameters at f MF for input i. As shown in Fig. (3), the θ_i^f is a recurrent parameter for f MF in i input. θ_i^f works as memory in the NF structure to store the past information and merge with the current information.

2. The second layer (rule layer) has $m^{\mathfrak{n}}$ rule nodes (grid-type partitioning), which applies the AND operation in each rule node to yield a fired strength of a rule as follows:

$$\pi_\ell(k) = \prod_{i=1}^{\mathfrak{n}} \left(O_i^{\alpha_i^\ell}\left(x_i(k)\right)\right), \tag{18}$$

where $\ell = 1, 2, \ldots, m^{\mathfrak{n}}$, and α_i^ℓ is illustrated in Table 1.

Table 1. The values for α_i^ℓ. α_i^ℓ depend on i and ℓ. The rows for the last column ($i = \mathfrak{n}$) increase by one with each ℓ and replicate each m. The rows for the column with $i = \mathfrak{n} - 1$ increase by one each with each $\ell = m$ and republicate each m^2 and so on.

	α_i^ℓ	i				
		1	2	...	$\mathfrak{n}-1$	$\mathfrak{n}$
ℓ	1	1	1	...	1	1
	2	1	1	...	1	2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$m-1$	1	1	...	1	$m-1$
	m	1	1	...	1	m
	$m+1$	1	1	...	2	1
	$m+2$	1	1	...	2	2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$2m-1$	1	1	...	2	$m-1$
	$2m$	1	1	...	2	m
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$m^{\mathfrak{n}}-m$	m	m	...	m	1
	$m^{\mathfrak{n}}-m+1$	m	m	...	m	2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$m^{\mathfrak{n}}-1$	m	m	...	m	$m-1$
	$m^{\mathfrak{n}}$	m	m	...	m	m

3. The third layer (consequent/output layer) has consequent parameters ($\omega_{p\times m^{\mathfrak{n}}}$). This layer proceeds the defuzzification operation by a linear combination of the consequent parameters with firing strengths to yield the output fuzzy value as follows:

$$y_j(k) = \sum_{\ell=1}^{m^{\mathfrak{n}}} \Big(\omega_{j\ell}\pi_\ell(k)\Big), \tag{19}$$

where $j = 1, 2, \ldots, p$.

Equations (17) - (19) illustrate the feedforward propagation of the signal, while the backpropagation gradient algorithm with the chain rule is used as backward propagation of the signal to train both the consequent and premise parameters. In both NF-VGL(λ) or

NF-SNVGL(λ) structures, the partial derivatives of the outputs (y_j, where $j = 1, 2, \ldots, p$), with respect to the premise and consequent parameters, are required as well as the partial derivatives of the outputs, with respect to the inputs (x_i, where $i = 1, 2, \ldots, \mathfrak{n}$) as follows:

1. Tuning the consequent parameters ($\boldsymbol{\omega}_{p\times m^{\mathfrak{n}}}$):

$$\frac{\partial y_j(k)}{\partial \omega_{j\ell}(k)} = \pi_\ell, \tag{20}$$

where $j = 1, 2, \ldots, p$ and $\ell = 1, 2, \ldots, m^{\mathfrak{n}}$ ($m^{\mathfrak{n}}$ is the total number of the fuzzy rule).

2. Tuning The premise parameters ($\mathbf{\mathfrak{m}}_{\mathfrak{n}\times m}$, $\boldsymbol{\sigma}_{\mathfrak{n}\times m}$, $\boldsymbol{\theta}_{\mathfrak{n}\times m}$):

$$\begin{aligned}\frac{\partial y(k)}{\partial \mathfrak{m}_i^f(k)} &= \sum_{j=1}^{p}\left(\frac{\partial y_j(k)}{\partial \Pi_\ell(k)}\frac{\partial \Pi_\ell(k)}{\partial O_i^f\left(x_i(k)\right)}\frac{\partial O_i^f\left(x_i(k)\right)}{\partial \mathfrak{m}_i^f(k)}\right),\\ &= \sum_{j=1}^{p}\left(\omega_{j\ell}(k).\prod_{r=1,r\neq i}^{n}\left(O_r^{\alpha_r^\ell}\left(x_i(k)\right)\right).2O_i^f\left(x_i(k)\right)\right.\\ &\qquad \left.\frac{\left(x_i(k) + O_i^f\left(x_i(k-1)\right)\theta_i^f(k) - \mathfrak{m}_i^f(k)\right)}{\left(\sigma_i^f(k)\right)^2}\right),\end{aligned} \tag{21}$$

where $y(k) = \sum_{j=1}^{p}\left(y_j(k)\right)$.

$$\begin{aligned}\frac{\partial y(k)}{\partial \sigma_i^f(k)} &= \sum_{j=1}^{p}\left(\frac{\partial y_j(k)}{\partial \Pi_\ell(k)}\frac{\partial \Pi_\ell(k)}{\partial \sigma_i^f(k)}\right) = \sum_{j=1}^{p}\left(\omega_{j\ell}(k).2\Pi_\ell(k)\right.\\ &\qquad \left..\frac{\left(x_i(k) + O_i^f\left(x_i(k-1)\right)\theta_i^f(k) - \mathfrak{m}_i^f(k)\right)^2}{\left(\sigma_i^f(k)\right)^3}\right),\end{aligned} \tag{22}$$

where

$$\begin{aligned}\frac{\partial \Pi_\ell(k)}{\partial \sigma_i^f(k)} &= \frac{\partial \Pi_\ell(k)}{\partial O_i^f\left(x_i(k)\right)}\frac{\partial O_i^f\left(x_i(k)\right)}{\partial \sigma_i^f(k)} = \prod_{r=1,r\neq i}^{n}\left(O_r^{\alpha_r^\ell}\left(x_i(k)\right)\right).2O_i^f\left(x_i(k)\right)\\ &= 2\Pi_\ell(k).\end{aligned} \tag{23}$$

$$\begin{aligned}\frac{\partial y(k)}{\partial \theta_i^f(k)} &= \sum_{j=1}^{p} \left(\frac{\partial y_j(k)}{\partial \Pi_\ell(k)} \frac{\partial \Pi_\ell(k)}{\partial \theta_i^f(k)} \right), \\ &= -\sum_{j=1}^{p} \Bigg(\omega_{j\ell}(k).2\Pi_\ell O_i^f\big(x_i(k-1)\big) \\ &\quad . \frac{\Big(x_i(k) + O_i^f\big(x_i(k-1)\big)\theta_i^f(k) - \mathfrak{m}_i^f(k) \Big)}{\Big(\sigma_i^f(k)\Big)^2} \Bigg).\end{aligned} \tag{24}$$

3. The i, j element for the Jacobean matrix of the partial derivatives of the j output with respect to the i input is derived by

$$\begin{aligned}\frac{\partial y_j(k)}{\partial x_i(k)} =& \frac{\partial y_j(k)}{\partial \Pi_\ell(k)} \frac{\partial \Pi_\ell(k)}{\partial O_i^f\big(x_i(k)\big)} \frac{\partial O_i^f\big(x_i(k)\big)}{\partial x_i^f(k)} = -\Bigg(\omega_{j\ell}(k) \\ & .2\Pi_\ell(k) \frac{\Big(x_i(k) + O_i^f\big(x_i(k-1)\big)\theta_i^f(k) - \mathfrak{m}_i^f(k) \Big)}{\Big(\sigma_i^f(k)\Big)^2} \Bigg).\end{aligned} \tag{25}$$

3.2. Takagi-Sugeno Recurrent Neuro-Fuzzy (TSRNF) Structure. Fig. (4) illustrates a structure for TSRNF for three inputs ($\mathfrak{n} = 3$), two outputs ($p = 2$) and two MFs ($m = 2$) for each input with a grid-type partition for the input space. This structure is similar to [72] - [76], with except for the feedback premise recurrent parameters in the first layer, where this structure has five layers:

1. The first layer (premise layer) has premise's parameters, which is similar to the first layer of RFN structure as follows:

$$O_i^f\big(x_i(k)\big) = e^{-\left(\frac{x_i(k) + O_i^f\big(x_i(k-1)\big)\theta_i^f(k) - \mathfrak{m}_i^f(k)}{\sigma_i^f(k)} \right)^2}. \tag{26}$$

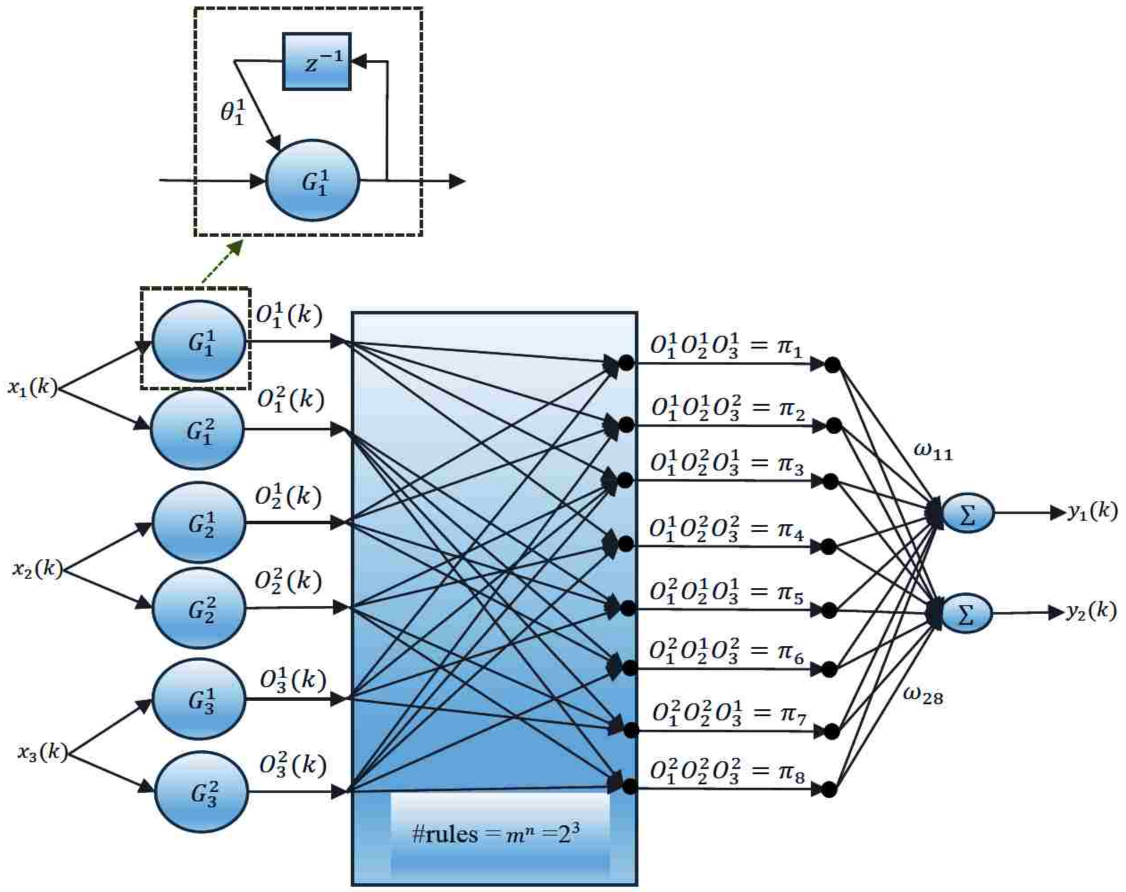


Figure 3. A structure for RNF that uses both actor and critic networks for VGL(λ) and NSVGL(λ). RNF consists of three layers. The first layer is the premise layer, which has premise parameters ($\mathbf{m}_{\mathrm{n}\times m}, \boldsymbol{\sigma}_{\mathrm{n}\times m}, \boldsymbol{\theta}_{\mathrm{n}\times m}$). The second layer is the rule layer, which has m^{n} rule nodes. The third layer is the consequent/output layer, which has consequent parameters ($\boldsymbol{\omega}_{p\times m^{\mathrm{n}}}$). The premise and consequent parameters are trained by using a backpropagation gradient algorithm. To simplify the appearance of this diagram, the number of inputs, MFs and outputs are represented by $\mathrm{n} = 3$, $m = 2$ and $p = 2$, respectively.

2. The second layer (rule layer) has $m^{\mathfrak{n}}$ rule nodes, which is also similar to the second layer of the RFN structure as follows:

$$\pi_\ell(k) = \prod_{i=1}^{\mathfrak{n}} \left(O_i^{\alpha_i^\ell} \left(x_i(k) \right) \right). \tag{27}$$

3. The third layer (normalization layer) is used to calculate the ratio of the *i*th rule node output (*i*th rule's firing strength) to get the sum of all the rules' node outputs:

$$\bar{\omega}_\ell(k) = \frac{\Pi_\ell(k)}{\sum_{r=1}^{m^{\mathfrak{n}}} \left(\Pi_r(k) \right)}, \tag{28}$$

   where $\ell = 1, 2, \ldots, m^{\mathfrak{n}}$,

4. The fourth layer (consequent/defuzzification layer) has consequent parameters ($\boldsymbol{a}^{m^{\mathfrak{n}}}_{p\times(\mathfrak{n}+1)}$, where $\boldsymbol{a}$), which is a 3-dimension matrix with $p \times m^{\mathfrak{n}} \times (\mathfrak{n}+1)$. This layer performs the defuzzification operation by using a linear combination of the consequent parameters as follows:

$$f_j^\ell(k) = \sum_{i=1}^{\mathfrak{n}} \left(a_{ji}^\ell x_i(k) \right) + a_{j(\mathfrak{n}+1)}^\ell, \tag{29}$$

   where $\ell = 1, 2, \ldots, m^{\mathfrak{n}}$, and $j = 1, 2, \ldots, p$.

5. The fifth layer (output layer) calculates the summation of the normalized firing strengths from the normalization layer with the outputs of the consequent layer:

$$y_j(k) = \sum_{\ell=1}^{m^{\mathfrak{n}}} \left(f_j^\ell(k) \bar{\omega}_\ell(k) \right), \tag{30}$$

   where $j = 1, 2, \ldots, p$.

Equations (26) - (30) illustrate the feedforward propagation of the signal, while the chain backpropagation gradient algorithm is used as a backward propagation of the signal to train both the consequent and premise parameters. As previously mentioned, the NF-

VGL(λ) and NF-SNVGL(λ) structures require the partial derivatives of the outputs with respect to the premise parameters, consequent parameters and the inputs to adapt the critic and actor approximation parameters as follows:

1. Tuning the consequent parameters ($\boldsymbol{a}^{m^{\mathfrak{n}}}_{p\times(\mathfrak{n}+1)}$):

$$\frac{\partial y_j(k)}{\partial a^{\ell}_{ji}(k)} = \frac{\partial y_j(k)}{\partial f^{\ell}_j(k)} \frac{\partial f^{\ell}_j(k)}{\partial a^{\ell}_{ji}(k)} = \bar{\omega}_{\ell}(k) x_i(k), \tag{31}$$

   where $i = 1, 2, \ldots, \mathfrak{n}$, $\ell = 1, 2, \ldots, m^{\mathfrak{n}}$, and $j = 1, 2, \ldots, p$. The free coefficients' consequence parameters, which are not related with input variables, can be updated by

$$\frac{\partial y_j(k)}{\partial a^{\ell}_{j(\mathfrak{n}+1)}(k)} = \frac{\partial y_j(k)}{\partial f^{\ell}_j(k)} \frac{\partial f^{\ell}_j(k)}{\partial a^{\ell}_{j(\mathfrak{n}+1)}(k)} = \bar{\omega}_{\ell}(k). \tag{32}$$

2. Tuning the premise parameters ($\mathfrak{m}_{\mathfrak{n}\times m}$, $\boldsymbol{\sigma}_{\mathfrak{n}\times m}$, $\boldsymbol{\theta}_{\mathfrak{n}\times m}$):

$$\begin{aligned} \frac{\partial y(k)}{\partial \mathfrak{m}^{f}_{i}(k)} &= \sum_{j=1}^{p} \left(\frac{\partial y_j(k)}{\partial \bar{\omega}_{\ell}(k)} \frac{\partial \bar{\omega}_{\ell}(k)}{\partial \Pi_{\ell}(k)} \frac{\partial \Pi_{\ell}(k)}{\partial \mathfrak{m}^{f}_{i}(k)} \right), \\ &= \sum_{j=1}^{p} \left(f^{\ell}_j(k) . \frac{\left(\sum_{r=1}^{m^{\mathfrak{n}}} \left[\Pi_r(k) \right] - \Pi_{\ell}(k) \right)}{\left(\sum_{r=1}^{m^{\mathfrak{n}}} \left[\Pi_r(k) \right] \right)^2} .2\Pi_{\ell}(k) \right. \\ &\quad \left. \frac{\left(x_i(k) + O^{f}_{i}\left(x_i(k-1) \right) \theta^{f}_{i}(k) - \mathfrak{m}^{f}_{i}(k) \right)}{\left(\sigma^{f}_{i}(k) \right)^2} \right), \end{aligned} \tag{33}$$

   where $y(k) = \sum_{j=1}^{p} \left(y_j(k) \right)$.

$$
\begin{aligned}
\frac{\partial y(k)}{\partial \sigma_i^f(k)} =& \sum_{j=1}^{p}\left(\frac{\partial y_j(k)}{\partial \bar{\omega}_\ell(k)}\frac{\partial \bar{\omega}_\ell(k)}{\partial \Pi_\ell(k)}\frac{\partial \Pi_\ell(k)}{\partial \sigma_i^f(k)}\right),\\
=&\beta\sum_{j=1}^{p}\left(f_j^\ell(k).\frac{\left(\sum_{r=1}^{m^{\mathfrak{n}}}\left[\Pi_r(k)\right]-\Pi_\ell(k)\right)}{\left(\sum_{r=1}^{m^{\mathfrak{n}}}\left[\Pi_r(k)\right]\right)^2}.2\Pi_\ell(k)\right.\\
&\left.\frac{\left(x_i(k)+O_i^f\left(x_i(k-1)\right)\theta_i^f(k)-\mathfrak{m}_i^f(k)\right)^2}{\left(\sigma_i^f(k)\right)^3}\right).
\end{aligned}
\tag{34}
$$

$$
\begin{aligned}
\frac{\partial y(k)}{\partial \theta_i^f(k)} =& \sum_{j=1}^{p}\left(\frac{\partial y_j(k)}{\partial \bar{\omega}_\ell(k)}\frac{\partial \bar{\omega}_\ell(k)}{\partial \Pi_\ell(k)}\frac{\partial \Pi_\ell(k)}{\partial \theta_i^f(k)}\right),\\
=&-\sum_{j=1}^{p}\left(f_j^\ell(k).\frac{\left(\sum_{r=1}^{m^{\mathfrak{n}}}\left[\Pi_r(k)\right]-\Pi_\ell(k)\right)}{\left(\sum_{r=1}^{m^{\mathfrak{n}}}\left[\Pi_r(k)\right]\right)^2}.2\Pi_\ell(k)\right.\\
&\left.\frac{\left(x_i(k)+O_i^f\left(x_i(k-1)\right)\theta_i^f(k)-\mathfrak{m}_i^f(k)\right)}{\left(\sigma_i^f(k)\right)^2}.O_i^f\left(x_i(k-1)\right)\right).
\end{aligned}
\tag{35}
$$

3. As illustrated in Fig. (4), the inputs ($x_i(k)$, $i = 1, 2, \ldots, \mathfrak{n}$) deal with the premise and consequent layers. Therefore, the i, j element for input/output Jacobean matrix is derived by combining two paths: the first path, $\left(\frac{\partial y_j(k)}{\partial x_i(k)}\right)^{(p_1)}$, derives from backpropagating the output signal through the consequent fuzzy part, while the second path, $\left(\frac{\partial y_j(k)}{\partial x_i(k)}\right)^{(p_2)}$, is derived from backpropagating the output signal through the premise fuzzy part as follows:

$$\begin{aligned}\frac{\partial y_j(k)}{\partial x_i(k)} =&\left(\frac{\partial y_j(k)}{\partial x_i(k)}\right)^{(p_1)} + \left(\frac{\partial y_j(k)}{\partial x_i(k)}\right)^{(p_2)} = \left(\frac{\partial y_j(k)}{\partial f_j^{\ell}(k)}\frac{\partial f_j^{\ell}(k)}{\partial x_i(k)}\right)^{(p_1)} + \\ &\left(\frac{\partial y_j(k)}{\partial \bar{\omega}_{\ell}(k)}\frac{\partial \bar{\omega}_{\ell}(k)}{\partial \Pi_{\ell}(k)}\frac{\partial \Pi_{\ell}(k)}{\partial x_i(k)}\right)^{(p_2)} = \left(\bar{\omega}_{\ell}(k)a_{ji}^{\ell}(k)\right) - \Bigg(f_j^{\ell}(k) \\ &.\frac{\left(\sum_{r=1}^{m^{\mathfrak{n}}}\left[\Pi_r(k)\right] - \Pi_{\ell}(k)\right)}{\left(\sum_{r=1}^{m^{\mathfrak{n}}}\left[\Pi_r(k)\right]\right)^2}.2\Pi_{\ell}(k) \\ &.\frac{\left(x_i(k) + O_i^f\left(x_i(k-1)\right)\theta_i^f(k) - \mathfrak{m}_i^f(k)\right)}{\left(\sigma_i^f(k)\right)^2}\Bigg).\end{aligned} \tag{36}$$

4. CONVERGENCE ANALYSIS OF THE VGL(λ) AND SNVGL(λ) APPROACHES

Because the VGL(λ) and SNVGL(λ) approaches use an n-step critic network, and the n-step optimal control equation is spacial greedy case for the general optimal control form, the following proofs perform with both approaches (VGL(λ) and SNVGL(λ)) as will be demonstrated in Definition 2.

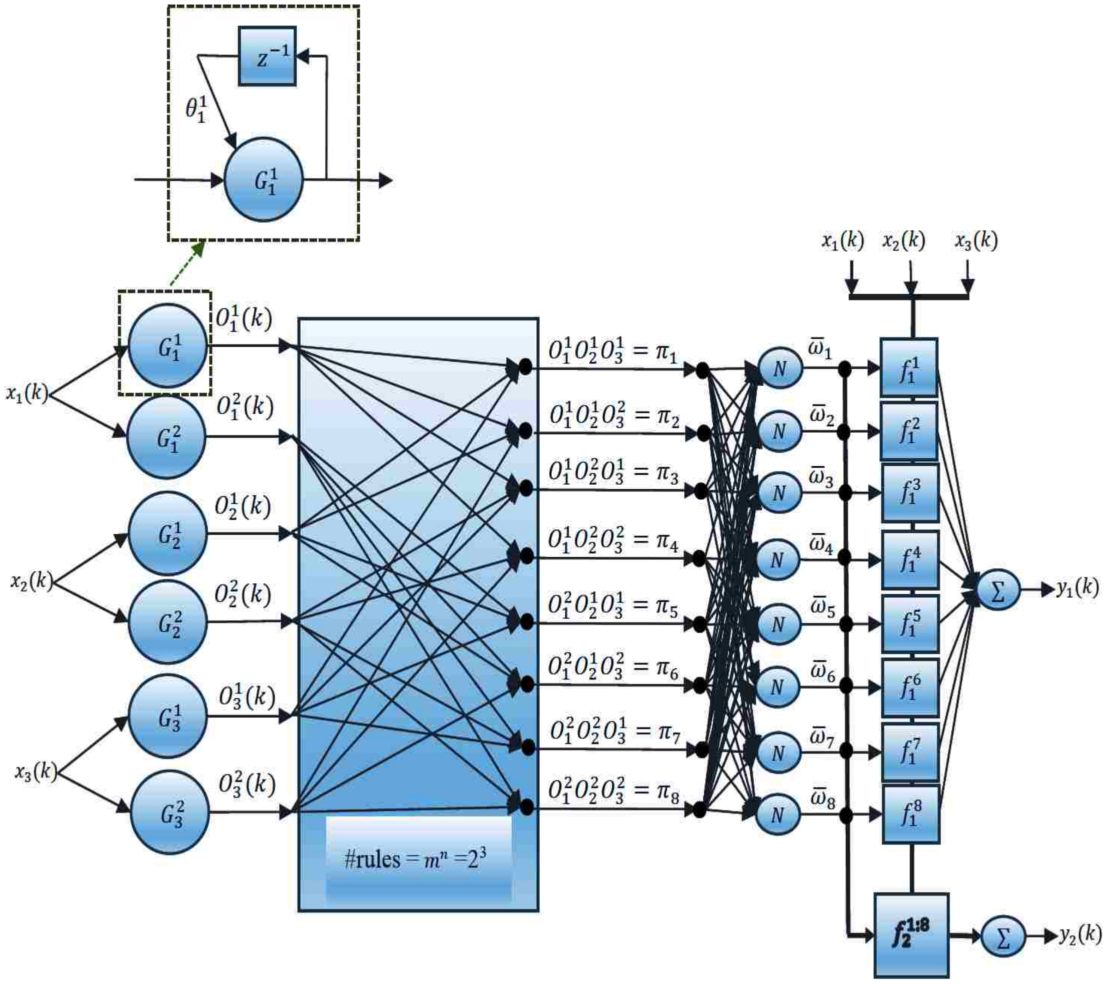
4.1. DT-HJB Equation for $R^{\lambda}(k)$. The n-step TD leaning with $R^{\lambda}(k)$ is given as

$$R^{\lambda}(k) =U\big(u(k), x(k)\big) + \gamma\lambda R^{\lambda}(k+1) + \gamma(1-\lambda)J(k+1), \tag{37}$$

where $J(k)$ is a function approximated for the value function at k (k is a discrete time index), and consider an affine nonlinear discrete-time (DT) dynamic system described by

$$x(k+1) =f\big(x(k)\big) + g\big(x(k)\big)u\big(x(k)\big), \tag{38}$$

$x \in \mathbb{R}^{\mathfrak{p}}$ is the state vector, $u\big(x(k)\big) \in \mathbb{R}^{\mathfrak{q}}$ is the control vector, and both $f\big(x(k)\big)$ and $g\big(x(k)\big)$ are differential functions with an equilibrium state at $x = 0$ (e.g., $f(0) = g(0) = 0$). It should be assumed that (38) is Lipschitz continuous and stable on a compact set $\Omega_0 \in \mathbb{R}^{\mathfrak{p}}$.



z^{-1}
θ_1^1
G_1^1
$x_1(k)$ $x_2(k)$ $x_3(k)$
$O_1^1(k)$
$O_1^2(k)$
$O_2^1(k)$
$O_2^2(k)$
$O_3^1(k)$
$O_3^2(k)$
G_1^1 G_1^2 G_2^1 G_2^2 G_3^1 G_3^2
$O_1^1O_2^1O_3^1 = \pi_1$
$O_1^1O_2^1O_3^2 = \pi_2$
$O_1^1O_2^2O_3^1 = \pi_3$
$O_1^1O_2^2O_3^2 = \pi_4$
$O_1^2O_2^1O_3^1 = \pi_5$
$O_1^2O_2^1O_3^2 = \pi_6$
$O_1^2O_2^2O_3^1 = \pi_7$
$O_1^2O_2^2O_3^2 = \pi_8$
#rules = $m^n = 2^3$
N
$\bar{\omega}_1$ … $\bar{\omega}_8$
f_1^1 … f_1^8
$f_2^{1:8}$
Σ
$y_1(k)$
$y_2(k)$

Figure 4. A structure for TSRNF that uses VGL(λ) and NSVGL(λ). TSRNF consists of five layers. The first layer represents the premise layer, which has premise parameters ($\mathfrak{m}_{\mathfrak{n}\times m}, \boldsymbol{\sigma}_{\mathfrak{n}\times m}, \boldsymbol{\theta}_{\mathfrak{n}\times m}$). The second layer is the rule layer, which has $m^{\mathfrak{n}}$ rule nodes. The third layer is the normalization layer to normalize the output values coming from the rule layer. The fourth layer is the consequent layer, which has consequent parameters ($\boldsymbol{a}^{m^{\mathfrak{n}}}_{p\times(\mathfrak{n}+1)}$). The fifth layer is the output layer to obtain the final output signal. The premise and consequent parameters are trained by using a backpropagation gradient algorithm. The number of inputs, MFs and outputs are represented by $\mathfrak{n} = 3$, $m = 2$ and $p = 2$, respectively.

Definition 1 cf [43] (stabilization system): A nonlinear system is stable on a compact set $\Omega_0 \in \mathbb{R}^{\mathrm{p}}$, if a control input $u\big(x(k)\big) \in \mathbb{R}^{\mathrm{q}}$ exists for all initial conditions $x(0) \in \mathbb{R}^{\mathrm{p}}$ such that the state $x(k) \to 0$ as $k \to \infty$.

The infinite-horizon cost function is given as

$$J(k) = \sum_{i=k}^{\infty} \gamma^{i-k} U\Big(x(i), u\big(x(i)\big)\Big), \tag{39}$$

where the utility function is chosen as the quadratic form, such as $U\big(x(i), u\big(x(i)\big)\big) = x^T(i)Qx(i) + u^T\big(x(i)\big)Ru\big(x(i)\big)$, where $Q = Q^T > 0 \in \mathbb{R}^{\mathrm{p}\times\mathrm{p}}$ and $R = R^T > 0 \in \mathbb{R}^{\mathrm{q}\times\mathrm{q}}$. Then, the objective of the function to minimize (39) is to find the admissible control action, which is given in the following definition.

Definition 2: If $u\big(x(k)\big)$ is continuous on a compact set $\Omega_u \in \mathbb{R}^{\mathrm{q}}$ with $u(0) = 0$, and $J(k)$ is finite for $\forall x(0) \in \Omega_0$, then $u\big(x(k)\big)$ is defined to be admissible with respect to (39). The definition of the infinite horizon optimal cost function is

$$J^*(k) = \inf_{u\left(x(k)\right)\in\Theta} \Big\{x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big) + \gamma J^*(k+1)\Big\}, \tag{40}$$

where Θ is a set of all infinite horizon admissible control lows. According to Bellman's optimality principle, the optimal cost function in a finite horizon that satisfies the DT-HJB is

$$J^*(k) = \min_{u\left(x(k)\right)} \Big\{x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big) + \gamma J^*(k+1))\Big\}. \tag{41}$$

Then, the optimal value function for n-step TD learning is

$$\begin{aligned} R^{\lambda^*}(k) = \min_{u\left(x(k)\right)} \Big\{&x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big) + \gamma\Big(\lambda R^{\lambda^*}(k+1) + \\ &(1-\lambda)J^*(k+1)\Big)\Big\}. \end{aligned} \tag{42}$$

The gradient on the right-hand side of (42) with respect to $u\big(x(k)\big)$ gives the optimal control u^*, which satisfies the first-order necessary condition

$$\begin{aligned}0 =&\frac{\partial}{\partial u\big(x(k)\big)}\big(x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big)\big) + \gamma\Big(\frac{\partial x(k+1)}{\partial u\big(x(k)\big)}\Big)^T \\ &\frac{\partial}{\partial x(k+1)}\Big(\lambda R^{\lambda^*}(k+1) + (1-\lambda)J^*(k+1)\Big).\end{aligned} \tag{43}$$

Thus, $u^*(.)$ for the n-step is given as

$$u^*\big(x(k)\big) = -\frac{\gamma}{2}R^{-1}g^T\big(x(k)\big)\Big(\lambda\frac{\partial R^{\lambda^*}(k+1)}{\partial x(k+1)} + (1-\lambda)\frac{\partial J^*(k+1)}{\partial x(k+1)}\Big). \tag{44}$$

The optimal control policy with the n-step method is used to prove the convergence of the VGL(λ) iteration algorithm and the SNVGL(λ) iteration algorithm as well.

4.2. Derivation of Iteration VGL(λ). The initial values of n-step cost functions and their targets are $J_0(.) = 0$ and $R_0^\lambda(.) = 0$, respectively. These solve the initial policy μ_0 by

$$\begin{aligned}\mu_0\big(x(k)\big) =&arg \min_{u\left(x(k)\right)} \Big\{x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big)+ \\ &\gamma\Big(\lambda R_0^\lambda(k+1) + (1-\lambda)J_0(k+1)\Big)\Big\}.\end{aligned} \tag{45}$$

Then, we update the $R^\lambda(.)$ is updated according to the previous iteration of $R^\lambda(.)$ and the value function, $J(.)$, to give

$$\begin{aligned}R_1^\lambda(k) = &\min_{u\left(x(k)\right)} \Big\{x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big) + \gamma\Big(\lambda R_0^\lambda(k+1) + (1-\lambda) \\ &J_0(k+1)\Big)\Big\} = x^T(k)Qx(k) + \mu_0^T\big(x(k)\big)R\mu_0\big(x(k)\big) + \gamma\Big[\lambda R_0^\lambda\Big(f(k)+ \\ &g(k)\mu_0\big(x(k)\big)\Big) + (1-\lambda)J_0\Big(f(k) + g\big(x(k)\big)\big)\mu_0\big(x(k)\big)\Big)\Big].\end{aligned} \tag{46}$$

For $I = 1, 2, \ldots$, the iteration values for μ_I and R_I^λ are given as

$$\begin{aligned}\mu_I(\big(x(k)\big)) =&arg \min_{u\left(x(k)\right)} \Big\{x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big) + \gamma \\ &\Big(\lambda R_I^\lambda(k+1) + (1-\lambda)J_I^0(k+1)\Big)\Big\},\end{aligned} \tag{47}$$

and

$$\begin{aligned}R_{I+1}^\lambda(k) =& \min_{u\left(x(k)\right)} \Big\{x^T(k)Qx(k) + u^T\big(x(k)\big)Ru\big(x(k)\big) + \gamma\lambda R_I^\lambda(x(k+1)) + \\ &\gamma(1-\lambda)J_I(k+1)\Big\} \\ =&x^T(k)Qx(k) + \mu_I^T\big(x(k)\big)R\mu_I\big(x(k)\big) + \gamma\lambda R_I^\lambda(k+1) + \gamma(1-\lambda)J_I(k+1),\end{aligned} \tag{48}$$

respectively, where I is the iteration index of the action policy as well as the value function and its target, while k is the time index of the state and the control for system trajectories.

4.3. Convergence of Iterative VGL(λ) Algorithm. The proofs that show R_i^λ to the optimal valued function with $I \to \infty$ leads to $J_I \to J^*$ with $I \to \infty$ are presented. Moreover, This paper proves that $\mu_I \to u^*$ as $I \to \infty$.

Lemma 1: Let v_I be any arbitrary sequence of control policies, and $\mu_I(.)$ has already been defined in (47). Let Ψ_I^λ be

$$\begin{aligned}\Psi_{I+1}^\lambda(k) =&x^T(k)Qx(k) + v_I^T\big(x(k)\big)Rv_I\big(x(k)\big) + \gamma\lambda\Psi_I^\lambda\Big(f(k) + g\big(x(k)\big) \\ &v_I\big(x(k)\big)\Big) + \gamma(1-\lambda)J_I\Big(f(x_k) + g\big(x(k)\big)v_I\big(x(k)\big)\Big),\end{aligned} \tag{49}$$

R_I^λ is defined in (48). If $R_0^\lambda = \Psi_0^\lambda = 0$, then $R_{I+1}^\lambda(k) \le \Psi_{I+1}^\lambda(k)$, $\forall I$.

Proof: Because $R_{I+1}^\lambda(k)$ minimizes the right-hand side of (48) with respect to $\mu_I\big(x(k)\big)$, and because $R_0^\lambda = \Psi_0^\lambda = J_0 = 0$, then by induction it follows that $R_{I+1}^\lambda(k) \le \Psi_{I+1}^\lambda(k)$,$\forall I$.

Lemma 2: Let $R_I^\lambda(.)$ be defined as in (47); if the system is controllable, then there is an upper bound of Y such that $R_I^\lambda(.) \le Y$, $\forall I$.

Proof: Let $\xi_I\big(x(k)\big)$ be a sequence of admissible control policies, and $J_0(.) = R_0^\lambda(.) = J_0(.) = \Theta_0^\lambda(.) = 0$, where J_k^λ is defined and updated as in (48), and J_I is defined and updated by (To simplify, $U\Big(x(k), \xi_I\big(x(k)\big)\Big)$ is denoted as $U(k, \xi_I\big(k\big))$)

$$\Theta_{I+1}^\lambda(k) = U\Big(x(k), \xi_I\big(x(k)\big)\Big) + \gamma\Big(\lambda\Theta_i^\lambda(k+1) + (1-\lambda)J_I(k+1)\Big), \tag{50}$$

where $U\Big(x(k), \xi_I\big(x(k)\big)\Big) = x^T(k)Qx(k) + \xi_I^T\big(x(k)\big)R\xi_I\big(x(k)\big)$. Equation (50) expands to be

$$\begin{aligned}\Theta_{I+1}^\lambda(k) =& U(k, \xi_I\big(k\big)) + \gamma\Bigg[\lambda\Bigg(U(k+1, \xi_{I-1}\big(k+1\big)) + \gamma\Big[\lambda \\ & \Theta_{I-1}^\lambda(k+2) + (1-\lambda)J_{I-1}\big(k+2\big)\Big] + (1-\lambda)J_i(k+1)\Bigg)\Bigg].\end{aligned} \tag{51}$$

By expanding further, the following is obtained:

$$\begin{aligned}\Theta_{I+1}^\lambda(k) =& U(k, \xi_I\big(k\big)) + \gamma\Bigg[\lambda\Bigg(U(k+1, \xi_{I-1}(\big(k+1\big))+ \\ & \gamma\Big[U(k+2, \xi_{I-2}\big(k+2\big)) + \gamma\big[\lambda\Theta_{I-2}^\lambda(k+3) + (1-\lambda) \\ & J_{I-2}(k+3)\big] + (1-\lambda)J_{I-1}(k+2)\Big] + (1-\lambda)J_I(k+1)\Bigg)\Bigg], \\ =& U(k, \xi_I\big(k\big)) + (\gamma\lambda)U(k+1, \xi_{I-1}\big(k+1\big)) + (\gamma\lambda)^2 \\ & U(k+2, \xi_{I-2}\big(k+2\big)) + (\gamma\lambda)^3\Theta_{I-2}^\lambda(k+3) + (\gamma\lambda)^2\gamma \\ & (1-\lambda)J_{I-2}(k+3) + (\gamma\lambda)\gamma(1-\lambda)J_{I-1}(k+2) + \gamma \\ & (1-\lambda)J_I(k+1).\end{aligned} \tag{52}$$

With further expanding, we obtain

$$\begin{aligned}\Theta_{I+1}^{\lambda}(k) =& U(k, \xi_I(k)) + (\gamma\lambda)U(k+1, \xi_{I-1}(k+1)) + (\gamma\lambda)^2 \\ & U(k+2, \xi_{I-2}(k+2)) + \ldots + (\gamma\lambda)^I U(k+I, \xi_0(k+i)) + \\ & (\gamma\lambda)^{I+1}\Theta_0^{\lambda}(k+I+1) + (\gamma\lambda)^I \gamma(1-\lambda)J_0(k+I+1) + \\ & (\gamma\lambda)^{I-1}\gamma(1-\lambda)J_1(k+I) + \ldots + (\gamma\lambda)\gamma(1-\lambda) \\ & J_{I-1}(k+2) + \gamma(1-\lambda)J_I(k+1),\end{aligned} \tag{53}$$

where $\Theta_0^{\lambda}(k+I+1) = J_0(k+I+1) = 0$, because $\Theta_0^{\lambda}(.) = J_0(.) = 0$. Thus,

$$\begin{aligned}\Theta_{I+1}^{\lambda}(k) = & \sum_{j=0}^{I}\left[(\gamma\lambda)^j U(k+j, \xi_{I-j}(k+j))\right] + \gamma(1-\lambda) \\ & \sum_{j=0}^{I-1}\left[(\gamma\lambda)^j J_{I-j}(k+j+1)\right].\end{aligned} \tag{54}$$

From (4) that $J(\upsilon) = \sum_{l=0}^{n}\left[\gamma^l U(\upsilon+l, u(\upsilon+l))\right]$ with $\upsilon = k+j+1$ and $u(.) = \xi(.)$ at the same iteration trajectory number ($n = I - j$), the $\Theta_{I+1}^{\lambda}(k)$ is

$$\begin{aligned}\Theta_{I+1}^{\lambda}(k) = & \sum_{j=0}^{I}\left[(\gamma\lambda)^j U(k+j, \xi_{I-j}(k+j))\right] + \gamma(1-\lambda)\sum_{j=0}^{I-1}\left[(\gamma\lambda)^j\right. \\ & \left.\sum_{l=0}^{I-j}\left[\gamma^l U(k+j+1+l, \xi_l(k+j+1+l))\right]\right].\end{aligned} \tag{55}$$

Because Definition 1 shows that ξ_I is an admissible control policies and $x(k) \to 0$ as $k \to \infty$, there exists an upper bound of Y such that

$$\begin{aligned}\forall I:\Theta^{\lambda}_{I+1}(x_k) \leq \lim_{I\to\infty}&\Bigg(\sum_{j=0}^{I}\Bigg[(\gamma\lambda)^j\Big(x^T(k+j)Qx(k+j)+\\ &\xi^T_{I-j}\big(x(k+j)\big)R\xi_{I-j}\big(x(k+j)\big)\Big)\Bigg]+\gamma(1-\lambda)\\ &\sum_{j=0}^{I-1}\Bigg[(\gamma\lambda)^j\sum_{l=0}^{I-j}\Big[\gamma^l\Big(x^T(k+j+1+l)Q\\ &x(k+j+1+l)+\xi^T_l\big(x(k+j+1+l)\big)R\xi_l\big(x(k+j+1+l)\big)\Big)\Big]\Bigg]\Bigg).\\ &=Y.\end{aligned} \tag{56}$$

According to Lemma 1, $\mu_I(x(k)) = \xi_I\big(x(k)\big)$ and $\Psi^{\lambda}_{I+1}(k) = \Theta^{\lambda}_{I+1}(k)$. Therefore, $R^{\lambda}_{I+1}(k) \leq \Theta^{\lambda}_{I+1}(k) \leq Y$, $\forall I$ can be obtained.

Theorem 1: Consider μ_I and R^{λ}_I as defined in (47), and (48), respectively. Then, R^{λ}_I is non-decreasing such that $R^{\lambda}_I \leq R^{\lambda}_{I+1}$, $\forall I$.

Proof: Define μ_I and R^{λ}_I from (47) and (48), respectively. A new n-step value function Γ^{λ}_I is defined by

$$\Gamma^{\lambda}_{I+1}(k) = x^T(k)Qx(k) + \mu^T_I\big(x(k)\big)R\mu_I\big(x(k)\big) + \gamma\lambda\Gamma^{\lambda}_I(k+1) + \gamma(1-\lambda)J_I(k+1). \tag{57}$$

Mathematical induction illustrates that $\Gamma^{\lambda}_I \leq R^{\lambda}_{I+1}$ as follows: For $I = 0$, $R^{\lambda}_1(k) = x^T(k)Qx(k)+\mu^T_0\big(x(k)\big)R\mu_0\big(x(k)\big)+\gamma\Big(\lambda R^{\lambda}_0(k+1)+(1-\lambda)J_0(k+1)\Big)$; with $\Gamma^0_0 = J_0 = R^{\lambda}_0 = 0$. Then,

$$R^{\lambda}_1(k) - \Gamma^{\lambda}_0(k) = x^T(k)Qx(k) + \mu^T_0\big(x(k)\big)R\mu_0\big(x(k)\big) \geq 0. \tag{58}$$

Thus, $R_1^\lambda(k) \geq \Gamma_0^\lambda(k)$. A similar procedure is used, and assuming it reaches and holds for $I-1$, then $R_I^\lambda(k) \geq \Gamma_{I-1}^\lambda(k)$. For I, considering that

$$R_{I+1}^\lambda(k) = x^T(k)Qx(k) + \mu_I^T\big(x(k)\big)R\mu_I\big(x(k)\big) + \gamma\Big(\lambda R_I^\lambda(k+1) + (1-\lambda)J_I(k+1)\Big), \quad (59)$$

and

$$\Gamma_I^\lambda(k) = x^T(k)Qx(k) + \mu_I^T\big(x(k)\big)R\mu_I\big(x(k)\big) + \gamma\Big(\lambda\Gamma_{I-1}^\lambda(k+1) + (1-\lambda)J_{I-1}(k+1)\Big), \quad (60)$$

the following is obtained:

$$\begin{aligned} R_{I+1}^\lambda(k) - \Gamma_I^\lambda(k) =& \gamma\Big(\lambda R_I^\lambda(k+1) + (1-\lambda)J_I(k+1)\Big) - \\ & \gamma\Big(\lambda\Gamma_{I-1}^\lambda(k+1) + (1-\lambda)J_{I-1}(k+1)\Big). \end{aligned} \quad (61)$$

With further rearranging and applying (4) so that $J(\upsilon) = \sum_{l=0}^{n}\Big[\gamma^l U(\upsilon+l, u\big(\upsilon+l\big))\Big]$ with $\upsilon = k+1$ and $u(.) = \mu(.)$ at similar iteration trajectories, it can be determined that

$$\begin{aligned} R_{I+1}^\lambda(k) - \Gamma_I^\lambda(k) =& \gamma\Bigg(\lambda\Big(R_I^\lambda(k+1) - \Gamma_{I-1}^\lambda(k+1)\Big) + \\ & (1-\lambda)\Big(J_I(k+1) - J_{I-1}(k+1)\Big)\Bigg), \\ =& \gamma\Bigg(\lambda\Big(R_I^\lambda(k+1) - \Gamma_{I-1}^\lambda(k+1)\Big) + \\ & (1-\lambda)\Big(\sum_{l=0}^{I}\Big[\gamma^l U(k+1+l, \mu_l\big(k+1+l\big))\Big] - \\ & \sum_{l=0}^{I-1}\Big[\gamma^l U(k+1+l, \mu_l\big(k+1+l\big))\Big]\Big)\Bigg). \end{aligned} \quad (62)$$

Because

1. $R_I^\lambda(k) \geq \Gamma_{I-1}^\lambda(k)$,

2. $\sum_{l=0}^{I}\left[\gamma^l U(k+1+l,\mu_l(k+1+l))\right] = \gamma^I U(k+1+l,\mu_I(k+1+l)) + \sum_{l=0}^{I-1}\left[\gamma^l U(k+1+l,\mu_l(k+1+l))\right]$,

3. $U(k+1+l,\mu_l(k+1+l)) = x^T(k+1+l)Qx(k+1+l) + \mu_l^T(k+1+l)R\mu_l(k+1+l)$,

then $R_{I+1}^{\lambda}(k) - \Gamma_I^{\lambda}(k) \geq 0$. Because $R_{I+1}^{\lambda}(k) \geq \Gamma_I^{\lambda}(k)$ and $\Gamma_I^{\lambda}(k) \geq R_I^{\lambda}(k)$ (Lemma 1), then $R_{I+1}^{\lambda}(k) \geq R_I^{\lambda}(k)$.

Theorem 2: Define $\lim_{I\to\infty} R_I^{\lambda}(k) = R_{\infty}^{\lambda}(k)$ as the infinite limits of the n-step value functions, where R_I^{λ} is defined as in (48). With controllable system states and $R_0^{\lambda}(.) = J_0(.) = 0$, R_I^{λ} is limited by $J^*(k)$, where J^* is described in (40). That is, $R_{\infty}^{\lambda}(k) = J^*$.

Proof: let $\zeta_I(x(k))$ be an arbitrary sequence of admissible control policies, and let $v_0^{\lambda}(.) = J_0(.) = \Lambda_0^{\lambda}(.) = 0$, where R_I^{λ} is defined and updated as in (48). Λ_{I+1}^{λ} is updated by:

$$\begin{aligned}\Lambda_{I+1}^{\lambda}(k) =& x^T(k)Qx(k) + \zeta_I^T(x(k))R\zeta_I(x(k)) + \gamma\Big(\lambda \\ &\Lambda_I^{\lambda}(k+1) + (1-\lambda)J_I(k+1)\Big).\end{aligned} \tag{63}$$

From Lemma 1 and Lemma 2, it can be concluded that $R_{I+1}^{\lambda}(k) \leq \Lambda_{I+1}^{\lambda}(k) \leq Y, \forall i$. By defining $\lim_{I\to\infty}\Lambda_I^{\lambda}(k) = \Lambda_{\infty}^{\lambda}(k)$, and it can be obtained that $R_{\infty}^{\lambda}(k) \leq \Lambda_{\infty}^{\lambda}(k) \leq Y$ for all admissible control policy sequences. If $\zeta(x(k)) = u^*(x(k))$ with the results of Lemma 2, then

$$\begin{aligned}Y = \lim_{I\to\infty}\Bigg(&\sum_{j=0}^{I}\Bigg[(\gamma\lambda)^j\Big(x^T(k+j)Qx(k+j) + \xi_{I-j}^T(x(k+j)) \\ &R\xi_{I-j}(x(k+j))\Big)\Bigg] + \gamma(1-\lambda)\sum_{j=0}^{I-1}\Bigg[(\gamma\lambda)^j\sum_{l=0}^{I-j}\Big[\gamma^l\Big(x^T(k \\ &+j+1+l)Qx(k+j+1+l) + \xi_l^T(x(k+j+1+l)) \\ &R\xi_l(x(k+j+1+l))\Big)\Big]\Bigg]\Bigg),\end{aligned} \tag{64}$$

such that $Y \geq J^{*}(k)$, where J^{*} is described in (40). Therefore, from Lemma 2 with $R_{\infty}^{\lambda}(k) \leq \Lambda_{\infty}^{\lambda}(k) \leq Y$, it is obtained that $R_{\infty}^{\lambda}(k) \leq J^{*}(k)$. Because $J^{*}(k)$ is the infimum of the cost function that derives from all other admissible control sequences, this paper concludes that $J^{*}(k) \leq R_{\infty}^{\lambda}(k)$. This implies that $J^{*}(k) \leq R_{\infty}^{\lambda}(k) \leq J^{*}(k)$, and hence, $R_{\infty}^{\lambda}(k) = J^{*}(k)$. Because $R_{\infty}^{\lambda}(k)$ is the target value that is used to train the value function to yield $J_{\infty}(k)$, then $J_{I}(k) \rightarrow J^{*}(k)$ as $I \rightarrow \infty$.
According to Theorem 2, $R_{I}^{\lambda}(k) \rightarrow J^{*}(k)$ as $I \rightarrow \infty$. Because $g_{I}^{\lambda}(k) = \dfrac{\partial R_{I}^{\lambda}(k)}{\partial x(k)}$ and $g_{I}(k) = \dfrac{\partial J_{I}(k)}{\partial x(k)}$, it can be concluded that $g_{I}^{\lambda}(k)$ and $g_{I}(k) \rightarrow g^{*}(k)$ as $I \rightarrow \infty$, where g^{*} is the gradient of the optimum value function.

Corollary 1: $\mu_{I}\big(x(k)\big)$ and R_{I}^{λ} are defined as in (47) and (48), respectively. If $x(k)$ is controllable, then the R_{I}^{λ} forces the controller (neuro-fuzzy actor network) to converge to $u^{*}\big(x(k)\big)$ as $I \rightarrow \infty$ (i.e., $\lim_{I\rightarrow\infty} \mu_{I}\big(x(k)\big) = u^{*}\big(x(k)\big)$). Similar conclusions can apply to gradients of $R_{I}^{\lambda}(k)$ and $J_{I}(k)$ with respect to $x(k)$ to yield $g_{I}^{\lambda}(k)$ and $g_{I}(k)$ with $I \rightarrow \infty$, respectively.

5. COMPACTING THE NF-VGL(λ) AND NF-SNVGL(λ) ITERATIVE ALGORITHM WITH RNF AND TSRNF STRUCTURES

In this section, the value iterative ADP algorithm based on NF-VGL(λ) and NF-SNVGL(λ) is performed. Let $\hat{\omega}_c$ in $\hat{G}\big(x(k), \hat{\omega}_c\big)$ (see Fig. (1) and Fig. (2)) represent a combination of premise and consequent parameters, which is denoted as $\hat{\omega}_c^{\{p\}}$ and $\hat{\omega}_c^{\{c\}}$, respectively. Because the inputs of the critic network in the VGL (or DHP) approach has the same number of outputs, which is equal to the number of states ($\mathfrak{p}$), the inputs and outputs for the RNF and TSRNF structures are equal to $\mathfrak{p}$ ($\mathfrak{p} = \mathfrak{n} = p$). In other words, $\hat{\omega}_c^{\{p\}} = \Big[\mathfrak{m}, \boldsymbol{\sigma}, \boldsymbol{\theta}\Big]_{\mathfrak{p}\times 3m}$ for both RNF and TSRNF structures. $\hat{\omega}_c^{\{c\}} = \Big[\boldsymbol{\omega}\Big]_{\mathfrak{p}\times m^{\mathfrak{p}}}$ for RNF, while $\hat{\omega}_c^{\{c\}} = \Big[\boldsymbol{a}\Big]_{\mathfrak{p}\times m^{\mathfrak{p}}\times(\mathfrak{p}+1)}$ for TSRNF. Also, let $\hat{\omega}_a$ in $\hat{A}\big(x(k), \hat{\omega}_a\big)$ (see Fig. (1)) represents a combination of $\hat{\omega}_a^{\{p\}}$ and $\hat{\omega}_a^{\{c\}}$ for premise and consequent parameters, respectively. Because the inputs of the actor network are equaled to $\mathfrak{p}$ and the outputs are $\mathfrak{q}$ (the number of control signals), the inputs and outputs for the RNF and TSRNF structures are equal to $\mathfrak{p}$ and $\mathfrak{q}$, respectively. In other words, $\hat{\omega}_a^{\{p\}} = \Big[\mathfrak{m}, \boldsymbol{\sigma}, \boldsymbol{\theta}\Big]_{\mathfrak{p}\times 3m}$ for both RNF and TSRNF structures ($\mathfrak{n} = \mathfrak{p}$). $\hat{\omega}_c^{\{c\}} = \Big[\boldsymbol{\omega}\Big]_{\mathfrak{q}\times m^{\mathfrak{p}}}$ for RNF ($p = \mathfrak{p}$), while $\hat{\omega}_c^{\{c\}} = \Big[\boldsymbol{a}\Big]_{\mathfrak{q}\times m^{\mathfrak{p}}\times(\mathfrak{p}+1)}$ for TSRNF ($p = \mathfrak{p}$).

5.1. The Value-Iteration-based NF-VGL(λ).

5.1.1. The *n*-step Critic Network. The forward propagation output signal for the critic network is expressed for RNF as:

$$y_j(k) = \sum_{\ell=1}^{m^{\mathfrak{p}}} \left[\omega_{j\ell} \prod_{i=1}^{\mathfrak{p}} \left[O_i^{\alpha_i^\ell}\big(x_i(k)\big) \right] \right], j = 1, \ldots, \mathfrak{p}, \tag{65}$$

and for TSRNF as:

$$\begin{aligned} y_j(k) =& \frac{1}{\sum_{r=1}^{m^{\mathfrak{p}}} \left[\prod_{i=1}^{\mathfrak{p}} \left[O_i^{\alpha_i^r}\big(x_i(k)\big) \right] \right]} \sum_{\ell=1}^{m^{\mathfrak{p}}} \left[\sum_{i=1}^{\mathfrak{p}} \left[a_{ji}^\ell x_i(k) \right] + a_{j(\mathfrak{p}+1)}^\ell \times \right. \\ & \left. \prod_{i=1}^{\mathfrak{p}} \left[O_i^{\alpha_i^\ell}\big(x_i(k)\big) \right] \right], \\ & j = 1 \ldots, \mathfrak{p}, \end{aligned} \tag{66}$$

Thus, $\hat{g}_I(k) = [y_1(k), y_2(k), \ldots, y_{\mathfrak{p}}(k)]_I^T$, where I is the iteration index and k is the time index. With the quadratic form of the utility function, $U\Big(x(k), \mu_I\big(x(k)\big)\Big) = x^T(x)Qx(k) + \mu_I^T\big(x(k)\big)R\mu_I\big(x(k)\big)$ and (38) for the system state equation, the target value, which is defined in (10) with the general $U(.)$, is given by

$$\begin{aligned} g_I^\lambda(k) =& 2Qx(k) + 2\left[\frac{\partial \mu_{I-1}\big(x(k)\big)}{\partial x(k)}\right]^T R\mu_{I-1}\big(x(k)\big) \\ & + \left[\frac{\partial f\Big(x(k), \hat{\mu}_{I-1}\big(x(k)\big)\Big)}{\partial x(k)} + \frac{\partial f\Big(x(k), \hat{\mu}_{I-1}\big(x(k)\big)\Big)}{\partial \hat{\mu}_{I-1}\big(x(k)\big)} \right. \\ & \left. \frac{\partial \hat{A}_{I-1}\Big(x(k), \hat{\omega}_a\Big)}{\partial x_k} \right]^T \Big(\gamma\lambda g_{I-1}^\lambda(k+1) + \gamma(1-\lambda)\hat{g}_{I-1}(k+1) \Big), \end{aligned} \tag{67}$$

where (25) is used to obtain $\dfrac{\partial \hat{A}_{I-1}\Big(x(k), \hat{\omega}_a\Big)}{\partial x_k}$ for RNF, while (36) is used for TSRNF. The premise weight matrix ($\hat{\omega}_c^{\{p\}}$) and consequent weight matrix ($\hat{\omega}_c^{\{c\}}$) are tuned by backpropagating the prediction error of the critic network, which is given as:

$$e_{cI}(k) = g_I^{\lambda}(k) - \hat{g}_I(k). \tag{68}$$

The objective function for the $\hat{G}_{cI}\big(x(k), \hat{\omega}_c\big)$ network is to minimize $E_{cI}(k) = 0.5\big(e_{cI}(k)\big)^2$ by updating the value for the weights ($\hat{\omega}_c$) according to the gradient descent algorithm inside the local inner-loop, which is given by:

$$\begin{aligned}
\hat{\omega}_{cI}^{\{p\}} &= \hat{\omega}_{cI}^{\{p\}} + \ell_c \frac{\partial E_{cI}(k)}{\partial \hat{\omega}_{cI}^{\{p\}}} = \hat{\omega}_{cI}^{\{p\}} - \ell_c \frac{\partial \hat{G}_{cI}\big(x(k), \hat{\omega}_c\big)}{\partial \hat{\omega}_{cI}^{\{p\}}} e_{cI}(k),\\
\hat{\omega}_{cI}^{\{c\}} &= \hat{\omega}_{cI}^{\{c\}} + \ell_c \frac{\partial E_{cI}(k)}{\partial \hat{\omega}_{cI}^{\{c\}}} = \hat{\omega}_{cI}^{\{c\}} - \ell_c \frac{\partial \hat{G}_{cI}\big(x(k), \hat{\omega}_c\big)}{\partial \hat{\omega}_{cI}^{\{c\}}} e_{cI}(k),
\end{aligned} \tag{69}$$

where ℓ_c is the critic learning rate.

5.1.2. The Actor Network. The forward propagation output signal for the RNF actor network is represented by (65), while (66) is the forward propagation equation for the TSRNF actor network with $j = 1, \ldots, \mathfrak{q}$. Therefore, $\hat{\mu}_{I-1}\big(x(k)\big) = [y_1(k), y_2(k), \ldots, y_{\mathfrak{q}}(k)]_{I-1}^T$. A target action vector is calculated with an n-step critic network as

$$\begin{aligned}
e_{a(I-1)}(k) =& 2R\hat{\mu}_{I-1}\big(x(k)\big) + \gamma \left[\frac{\partial f\Big(x(k), \hat{\mu}_{I-1}\big(x(k)\big)\Big)}{\partial \hat{\mu}_{I-1}\big(x(k)\big)}\right]^T \times \Big(\lambda g_{I-1}^{\lambda}(k+1) + \\
& (1-\lambda)\lambda \hat{g}_{I-1}(k+1)\Big).
\end{aligned} \tag{70}$$

The objective function for this network is minimizing the actor error, which is given by using the gradient descent algorithm, which is obtained by

$$\begin{aligned}\hat{\omega}_{a(I-1)}^{\{p\}} =&\hat{\omega}_{a(I-1)}^{\{p\}} - \ell_a \frac{\partial \hat{A}_{(I-1)}\big(x(k),\hat{\omega}_a\big)}{\partial \hat{\omega}_{a(I-1)}^{\{p\}}} e_{a(I-1)}(x_k),\\ \hat{\omega}_{a(I-1)}^{\{c\}} =&\hat{\omega}_{a(I-1)}^{\{c\}} - \ell_a \frac{\partial \hat{A}_{(I-1)}\big(x(k),\hat{\omega}_a\big)}{\partial \hat{\omega}_{a(I-1)}^{\{c\}}} e_{a(I-1)}(x_k),\end{aligned} \tag{71}$$

where ℓ_a is the actor learning rate.

5.2. The Value-Iteration-based NF-SNVGL(λ).

5.2.1. The n-step Critic Network. The estimated output of the gradient value function ($\hat{g}_I(k) = [y_1(k), y_2(k), \dots, y_\text{p}(k)]_I^T$) is obtained in a manner that is similar to (65) and (66) by using RNF and TSRNF, respectively. The target value, which is defined in (16) is represented with the quadratic form of the utility function by

$$g_I^\lambda(k) =2Qx(k) + \left[\frac{\partial f\Big(x(k),\mu_{I-1}\big(x(k)\big)\Big)}{\partial x(k)}\right]^T \Big(\gamma\lambda g_{I-1}^\lambda(k+1) + \gamma(1-\lambda)\hat{g}_{I-1}(k+1)\Big). \tag{72}$$

The premise weight matrix ($\hat{\omega}_c^{\{p\}}$) and consequent weight matrix ($\hat{\omega}_c^{\{c\}}$) are tuned by back-propagating the prediction error of the critic network as in (68) and (69).

5.2.2. The n-step Optimal Control. The optimal control equation, which is defined in (14) with the quadratic form of the utility function and the affine type of state equation, is given by

$$\mu_{I-1}\big(x(k)\big) = -\frac{\gamma}{2}R^{-1}g^T\big(x(k)\big)\Big(\lambda g_{I-1}^\lambda(k+1) + (1-\lambda)\hat{g}_{I-1}(k+1)\Big). \tag{73}$$

Then, the actor error is calculated by

$$e_{a(I-1)}(k) = \mu_{I-1}\big(x(k)\big) - \hat{\mu}_{I-1}\big(x(k)\big). \tag{74}$$

The objective function for this network is minimizing the square of the actor error, which is given by $E_{a(I-1)}(k) = 0.5\Big(e_{a(I-1)}(k)\Big)^2$. The actor weights are updated by minimizing $E_{a(I-1)}(k)$ according to the gradient descent algorithm as in (71).

6. SIMULATION STUDY

To examine this paper' approaches, a complex nonlinear model is used, which is a two-wheeled dynamic nonholonomic mobile robot model. The performance results of VGL(λ) are compared with DHP. Combining RNF and TSRNF structures with VGL(λ) is another test to show the robust structure appropriate for VGL(λ). Furthermore, VGL(λ) and SNVGL(λ) are evaluated with various ambient noises (unmodeled bounded disturbances and left/right wheel friction). To demonstrate the effect of the recurrent parameter (θ) in both RNF and TSRNF structures, two tables are presented including mean-squared-errors (MSEs). In this paper, MSE is calculated by taking the average of the utility function ($U(.)$) for two states during the iterations.

6.1. The Nonlinear Dynamic Model of Mobile Robot. A differential-drive mobile robot contains two independently driven wheels mounted on the left and right of its chassis at the same axis, and a castor wheel (free rotating wheel) mounted at the front for balancing the mobile robot. An inertial Cartesian frame represents the position of the mobile robot, while $q = [x_c, y_c, \theta]^T$ is the set of coordinates for the center of mass of the robot and the robot's orientation with respect to the Cartesian frame. The two independent driving wheels are provided with the necessary torque for generating a left angular velocity (w_L) and a right angular velocity (w_R), which in turn generate a linear velocity (v_1) and angular velocity (v_2) for the mobile robot as follows:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.5\bar{r} & -0.5\bar{r} \\ \dfrac{\bar{r}}{2b} & -\dfrac{\bar{r}}{2b} \end{bmatrix} \begin{bmatrix} w_R \\ w_L \end{bmatrix}, \tag{75}$$

where $\bar{r}$ is the wheel radius and b is half of the robot width. The different forces for the mobile robot mechanical motion are considered in the literature for the dynamic model but not the kinematic model. The kinematic model is only considered for the motion. As stated in [58] and [77] - [79], the dynamic model of the mobile robot has $\bar{n}$ dimensional configuration space subjected to r constraints as described by

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)u + A^T(q)\Psi, \tag{76}$$

with $A(q)\dot{q} = 0$ as a constrained kinematic wheel, where $q \in \mathbb{R}^{\bar{n}}$ is a coordinate vector, $M(q) \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is a is a symmetric positive definite inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{\bar{n}\times\bar{n}}$ is the centripetal and Coriolis matrix, $F(\dot{q}) \in \mathbb{R}^{\bar{n}}$ is a surface friction force vector, $G(q) \in \mathbb{R}^{\bar{n}}$ is a gravity vector, $\tau_d \in \mathbb{R}^{\bar{n}}$ is a bounded unknown disturbance, $B(q) \in \mathbb{R}^{\bar{n}\times\mathfrak{q}}$ is an input transformation matrix, $u \in \mathbb{R}^{\mathfrak{q}}$ is the input torque vector, $A(q) \in \mathbb{R}^{r\times\bar{n}}$ is the full rank matrix associated with the constraints, and $\Psi \in \mathbb{R}^{r}$ is the Lagrange multiplier (constraint forces) vector. In this case study, there are two control inputs, which are a left torque (τ_L) and a right torque (τ_R). Since the system does not change in vertical position and has a constant value for the potential energy, $G(q)$ is set to zero. Using Lagrange multipliers to reduce the dynamic model from $\bar{n}$ to $\mathfrak{p} = \bar{n} - r$, (76) is pre-multiplied by spanning the linear independent null space of the $A(q)\dot{q}$ matrix, (which is denoted as the Jacobian matrix of $S_c(q) \in \mathbb{R}^{\bar{n}\times\mathfrak{p}}$). In this case, a kinematic equation is given as follows:

$$\dot{q} = S_c(q)v, \tag{77}$$

where

$$\dot{q} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix}, \quad S_c(q) = \begin{bmatrix} cos(\theta) & -dsin(\theta) \\ sin(\theta) & dcos(\theta) \\ 0 & 1 \end{bmatrix}, \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},$$

and d is the center of gravity. The final affine dynamic model is obtained from the kinematic equation (77) as follows: By taking the derivative of (77)

$$\ddot{q} = \dot{S}_c(q)v + S_c(q)\dot{v}. \tag{78}$$

Substitute (75), (77) and (78) into (76) to obtain

$$\dot{v} = -\bar{M}^{-1}(q)\Big(\bar{V}(q,\dot{q})v + \bar{F}(\dot{q}) + \bar{\tau}_d\Big) + \bar{M}^{-1}(q)\bar{\tau}. \tag{79}$$

$\bar{M}(q)$ is an invertible matrix:

$$\bar{M}(q) = \begin{bmatrix} m_T + 2\dfrac{I^b_{YY}}{\bar{r}^2} & 0 \\ 0 & m_T d^2 + I_T + 2I^b_{YY}\dfrac{b^2}{\bar{r}^2} - 4m_w d^2 \end{bmatrix},$$

$$S_c(q) = \begin{bmatrix} 0 & -dv_2(m_T - 2m_w) \\ dv_2(m_T - 2m_w) & 0 \end{bmatrix}, \tag{80}$$

$$\bar{\tau}_d = \begin{bmatrix} \bar{\tau}_R \\ \bar{\tau}_L \end{bmatrix}, \bar{F}(\dot{q}) = \frac{1}{\bar{r}} \begin{bmatrix} f_v(w_R + w_L) + f_c(\Delta(w_R) + \Delta(w_L)) \\ bf_v(w_R - w_L) + bf_c(\Delta(w_R) - \Delta(w_L)) \end{bmatrix},$$

and

$$\bar{\tau} = \bar{B}\tau = \begin{bmatrix} 0.5\bar{r} & -0.5\bar{r} \\ \dfrac{\bar{r}}{2b} & -\dfrac{\bar{r}}{2b} \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}, (\Delta(.) \text{ is a sigmoid function}). \tag{81}$$

All mobile robot dynamic parameters are defined in Table 2. A noisy environment was assumed and therefore unknown bounded frictions (f_v and f_c) and unstructured disturbances for the left wheel ($\bar{\tau}_L$) and the right wheel ($\bar{\tau}_R$) were added.

Table 2. Parameters of the dynamic mobile robot.
ϱ is various random values to test the dynamic model performance in different frictions

Symbol	Description	Value
m_T	Mass of the chassis	$10[kg]$
m_w	The mass of each wheel	$2[kg]$
$\bar{r}$	The wheel radius	$0.05[m]$
b	Half of the robot width	$0.1[m]$
d	The center of gravity offset form the rear axle	$0.1[m]$
I^b_{YY}	The wheel moment of inertial	$1[kg.m^2]$
I_T	The platform total moment of inertia	$5[kg.m^2]$
f_v	The viscous friction coefficient	$5 \times \varrho[N.m.s]$
f_c	The Coulomb friction coefficient	$8 \times \varrho[N.m.s]$

6.2. Simulation Results. In this case study, the first state ($x_1(k)$) and the second state ($x_2(k)$) are the linear velocity (v_1) and the angular velocity (v_2), respectively, whereas $x(k) = v(k) = [x_1(k) \quad x_2(k)]^T = [v_1(k) \quad v_2(k)]^T \in \mathbb{R}^2$ is the state vector ($\mathfrak{p} = 2$). The $\mu\big(x(k)\big) = [\tau_R \quad \tau_L]^T \in \mathbb{R}^2$ is the control action ($\mathfrak{q} = 2$). The external instantaneous cost function, which is also used for calculating the MSE for evaluation, is

$$U\Big(x(k), \mu\big(x(k)\big)\Big) = \Big[x(k) - x_d(k)\Big]^T Q \Big[x(k) - x_d(k)\Big] + \mu^T\big(x(k)\big) R \mu\big(x(k)\big), \tag{82}$$

where $x_d(k) = [x_{d1}(k) \quad x_{d2}(k)]^T$ is a desired velocity state vector, which takes constant values for all finite horizon time; x_{d1} is a desired value of the linear velocity state for the mobile robot, while x_{d2} is the desired angular velocity state; Q and R are 2-D identity matrices. In this work, $x_d(k) = [3 \quad 6.5]^T$ for all time steps in order to make a circle trajectory after applying the kinematic equation (77). The discount factor (γ) is 0.95. The number of MFs for each input for both critic and actor networks, which is used in both RNF and TSRNF structures, is 12 ($m = 12$); therefore, the number of rules is $m^{\mathfrak{p}} = 12^2 = 144$. The initial learning rate parameters are set to $\ell_c = 0.001$ for the critic networks and

$\ell_a = 0.001$ for the actor network. The training for each iteration of either network will be terminated if the error drops under 10^{-6} or if the number of iterations meets the stopping threshold for the internal cycle (30 iterations for actor and critic networks). The initial state is $x(0) = [5 \quad 5]^T$.

The first evaluation is the effectiveness of λ value to show the comparison between VGL(0) (or DHP) and VGL(λ). The RNF structure is selected to perform the critic and actor networks for five independent runs. Each run has 12000 iterations to train the ADP algorithm for 100 time steps each. The friction coefficients are set to 5 for f_v and 8 for f_c. Both f_v and f_c multiply by ϱ. ϱ is a parameter that multiplies by a random one digit value, which is applied during each time step over the iterations. The unstructured disturbances are set to 8 for $\bar{\tau}_R$ and $\bar{\tau}_L$ multiplying by ϱ. In this study case, ϱ is set to 0.001. Fig. 5 illustrates the average MSEs for the two velocity states during the iterations for five runs. RNF-VGL($\lambda = 0.98$) is more efficient and faster to learn than RNF-VGL($\lambda = 0$). [59], [58], [55] illustrated how λ value performs better for VGL with lambda than DHP when using the feed feedforward neural network. The input universes of discourse for the two velocity states are presented in Fig. 6 and Fig. 7 for the critic network and the actor network, respectively. Twelve GMFs ($m = 12$) are distributed between -20 and 30 for the input universe of discourse. The initial recurrent parameters ($\boldsymbol{\theta}$) for critic and actor networks are randomly chosen within [-1.2, 1.2]. The initial output parameters, $\boldsymbol{\omega}$ for RNF and $\boldsymbol{a}$ for TSRNF are randomly selected within [0.3 -0.3]. For one selected run, Fig. 6 - Fig. 13 show the initial and the finial learning distribution of GMFs, and they also show the learning θ during training. The initial distribution for GMFs for the two input velocity states to the critic network in RNF-VGL(0) is illustrated in Fig. 6, and it also shows the GMFs for the final shape after learning in the last iteration (12000th). Fig. 7 illustrates the initial and final distribution for GMFs for the two input velocity states using the actor network in RNF-VGL(0). Fig. 8 demonstrates the learning of twelve θ recurrent parameters during all

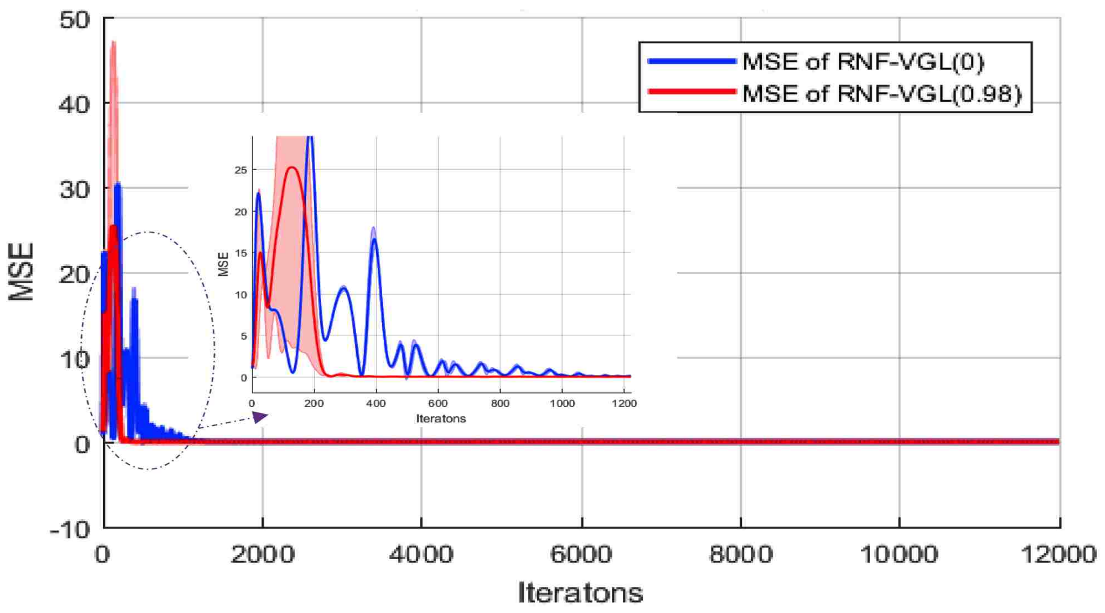


Figure 5. Average of MSEs for the two states for comparisons RNF-VGL(λ = 0) and RNF-VGL(λ = 0.98) with an impact on disturbances and frictions. Five independent runs are taken. The shaded region represents the runs, while the solid line represents the mean of runs. RNF-VGL(λ = 0.98) allows for faster learning than RNF-VGL(λ = 0).

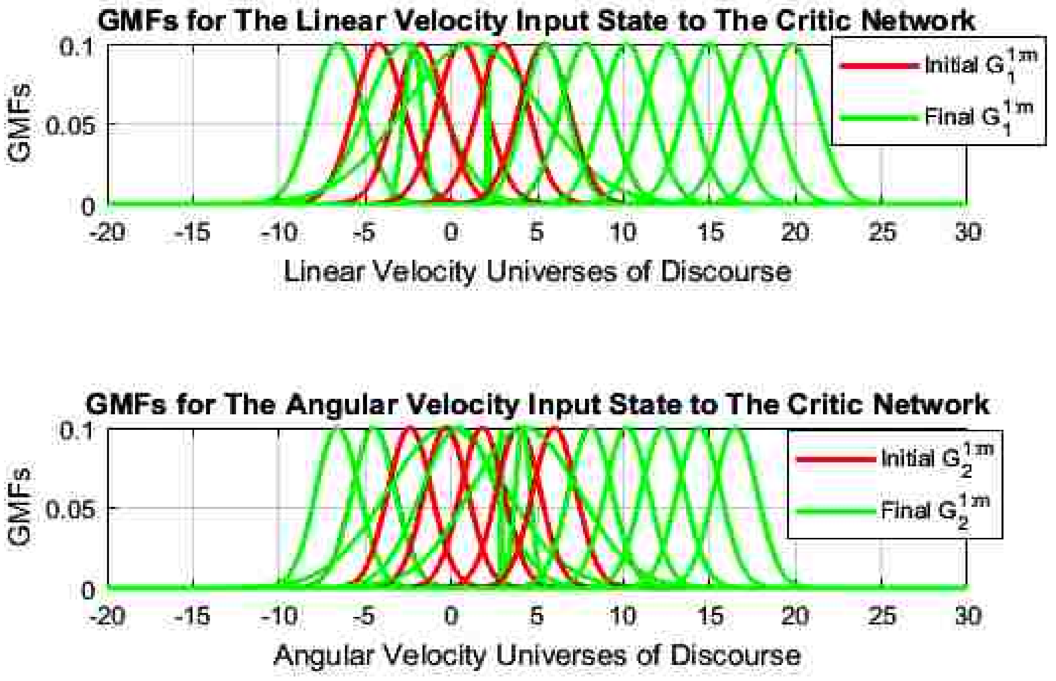


Figure 6. Initial and final learned Gaussian membership functions (GMFs) for both input states (linear and angular velocities) to the critic network in RNF-VGL(0).

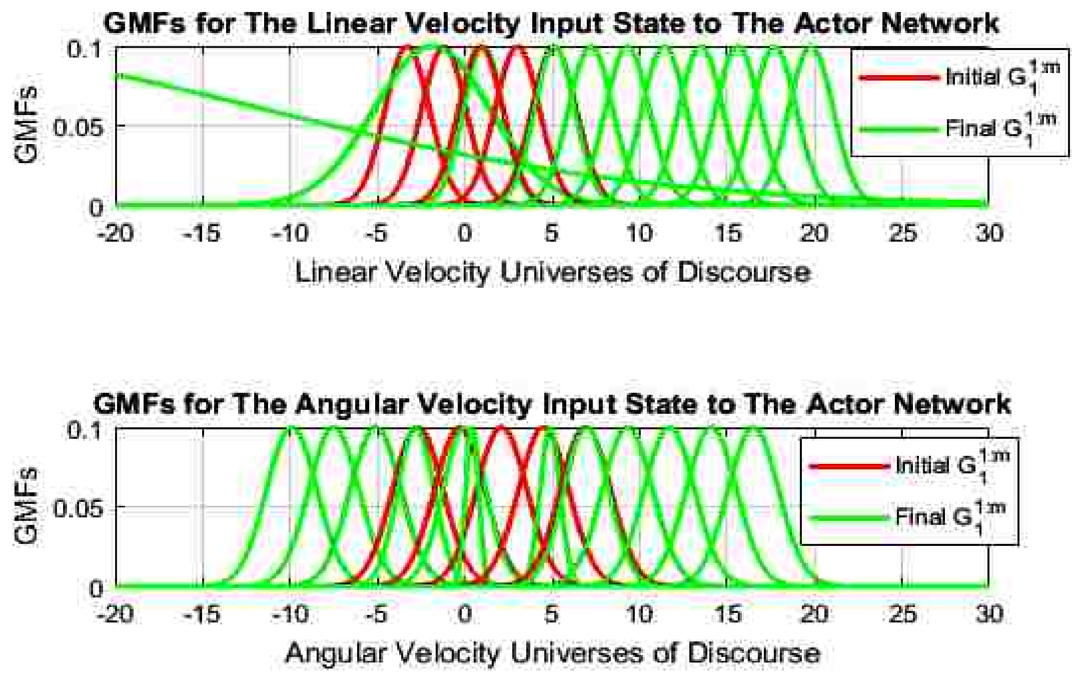
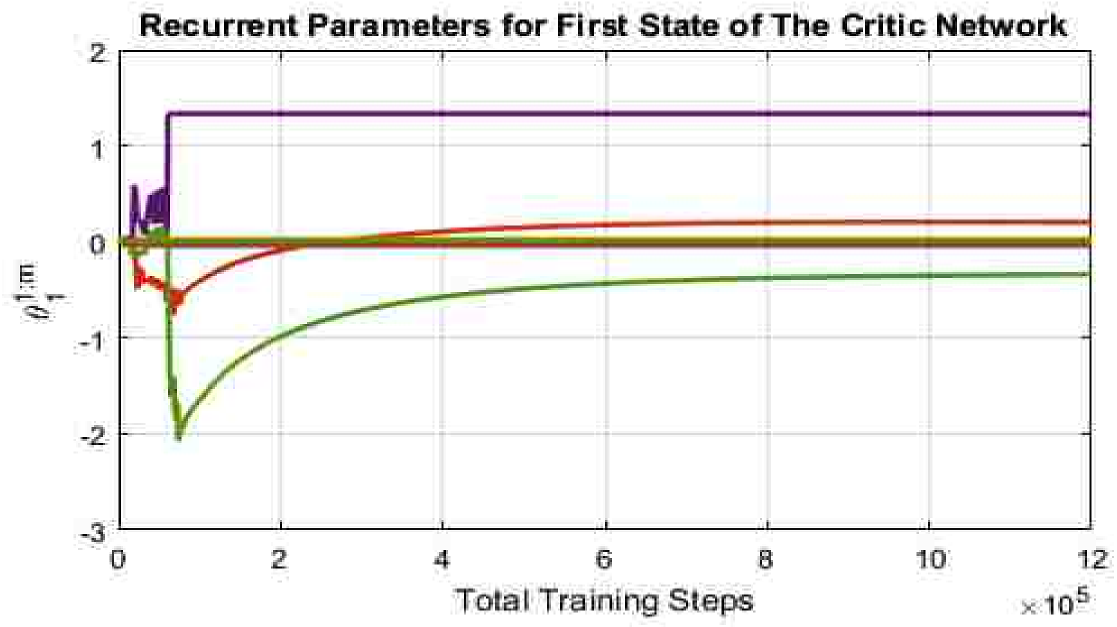


Figure 7. Initial and final learned GMFs for both linear and angular velocity input states to the actor network in RNF-VGL(0).

of the total training steps (12000 iteration with 100 time steps) in the critic network for both of the input states in RNF-VGL(0). Fig. 9 demonstrates learning θ during the total training steps for the actor network in RNF-VGL(0).

Fig. 10 - Fig. 13 is similar to Fig. 6 - Fig. 9 except for RNF-VGL(λ). Obviously, most GMFs in both critic and actor networks train within [-10, 10] of the universe of discourse for both input states for RNF-VGL(0) and RNF-VGL(λ) approaches. Furthermore, most θ parameters in RNF-VGL(λ) reach stable values faster than they do in RNF-VGL(0)

The second evaluation is a comparison between the VGL(λ) and SNVGL(λ) approaches. The RNF structure is used for both approaches. The initial parameters for the $\boldsymbol{\theta}$, $\boldsymbol{\omega}$, $\boldsymbol{a}$, the number of rules, impact of disturbances and frictions with ϱ value, and GNFs distribution are set similar to the previous test. Fig. 14 illustrates the average MSEs for the two velocity states during the iterations for five runs using RNF-VGL(λ) and



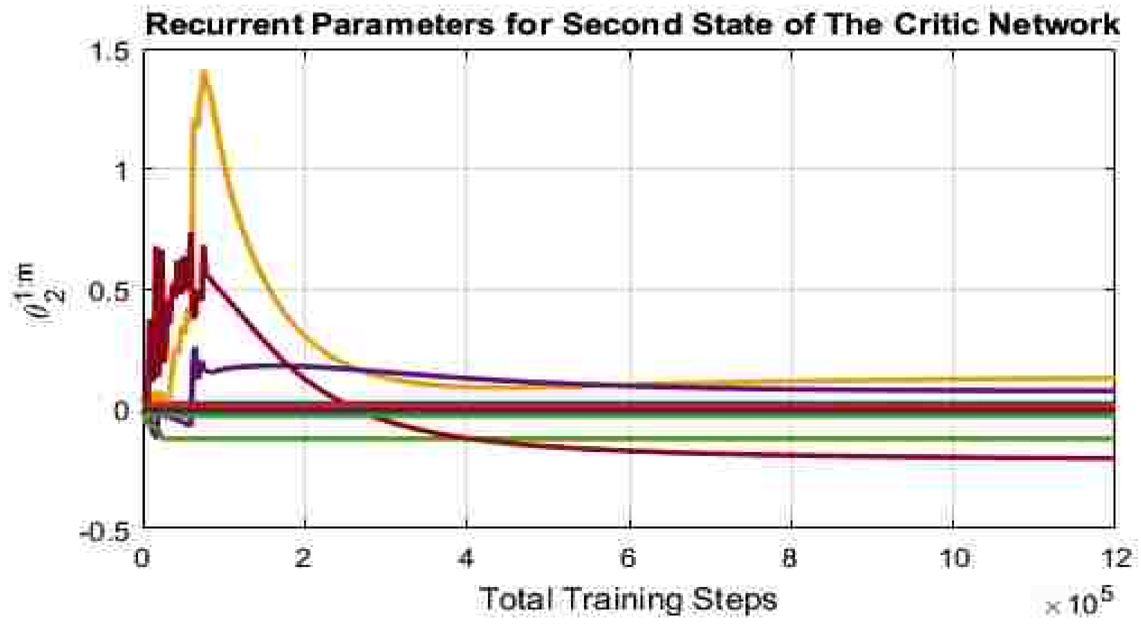
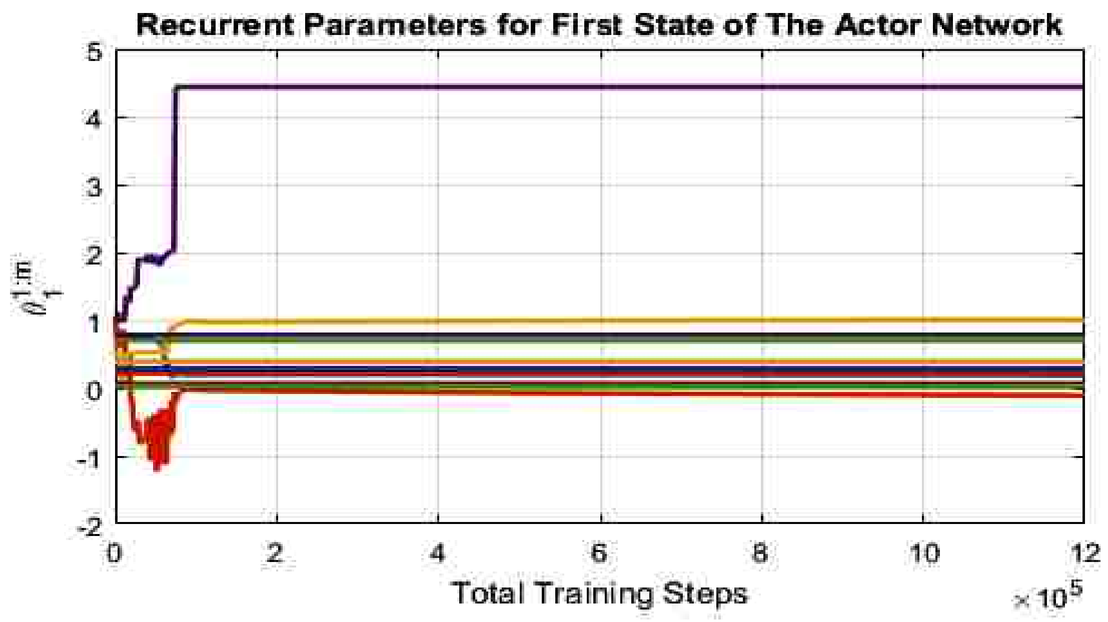


Figure 8. Deviation of the recurrent parameters ($\boldsymbol{\theta}$) for both input velocity states from the initialized values of the critic network of RNF-VGL(0) during all of the total training steps, which is 12000 iteration with 100 time steps each.



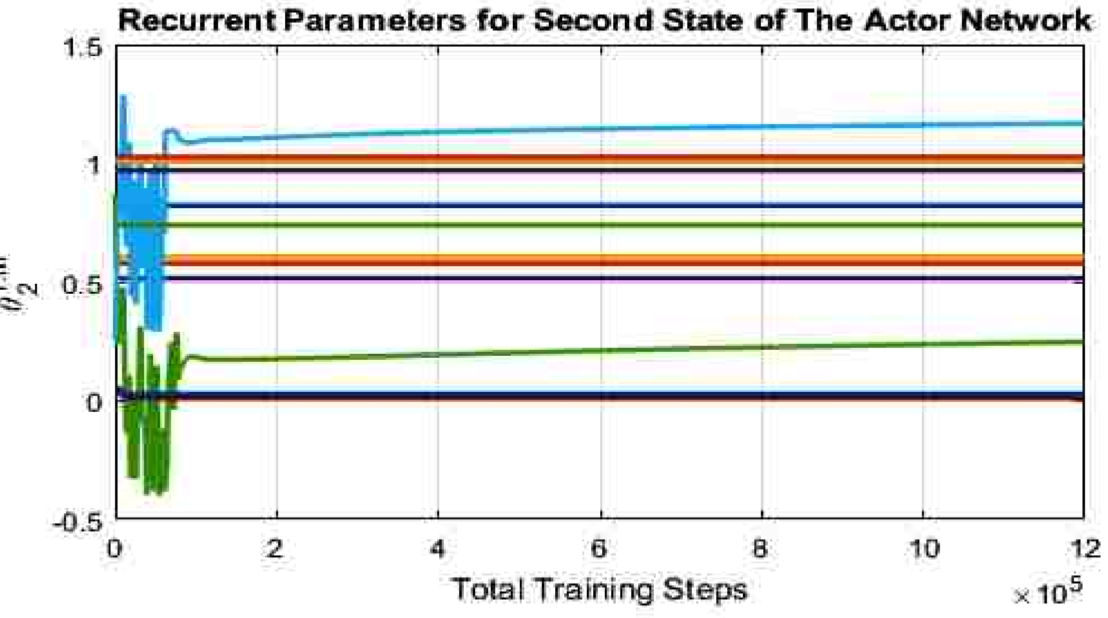
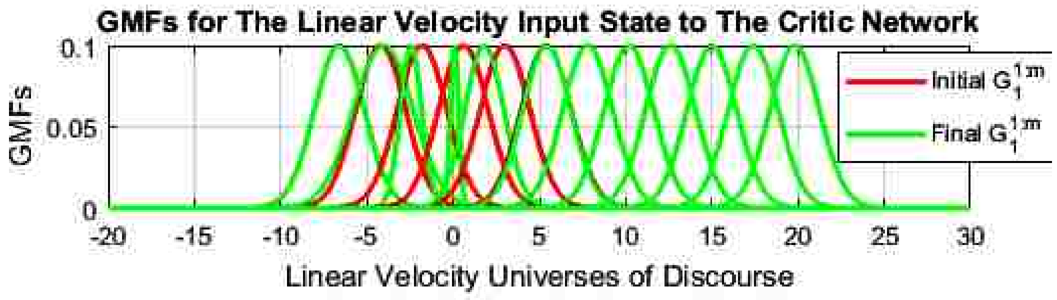


Figure 9. Deviation of the recurrent parameters ($\boldsymbol{\theta}$) for both input velocity states from the initialized values of the actor network of RNF-VGL(0) during the total training steps.



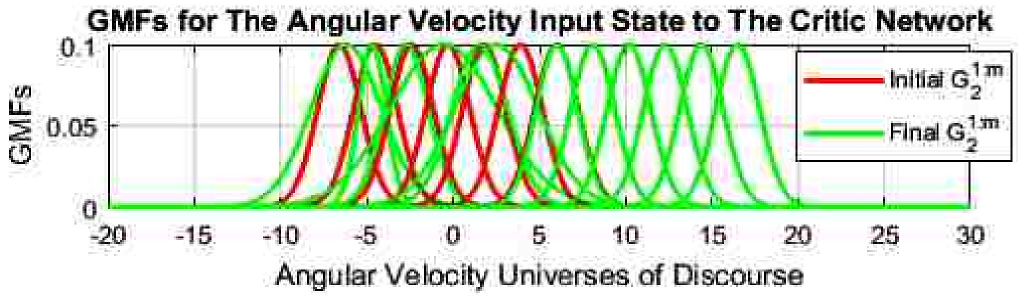
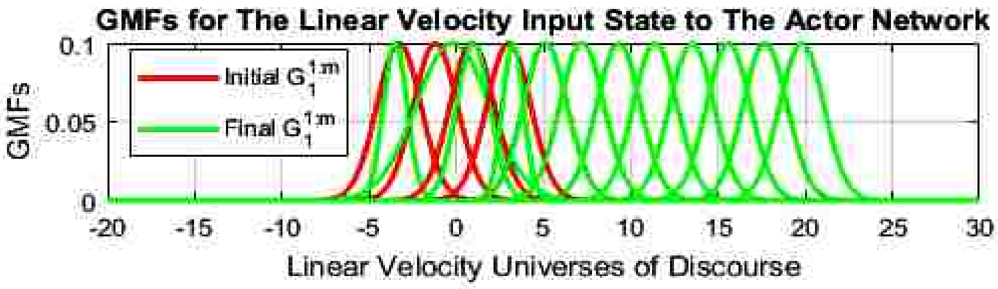


Figure 10. Initial and final learned GMFs for both linear and angular velocity input states to the actor network in RNF-VGL(λ).



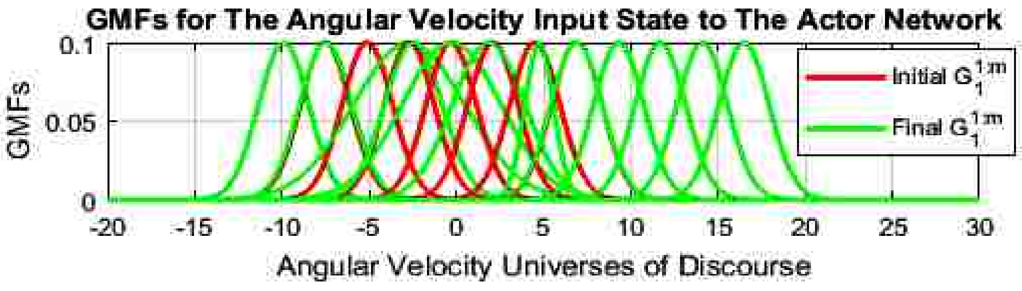
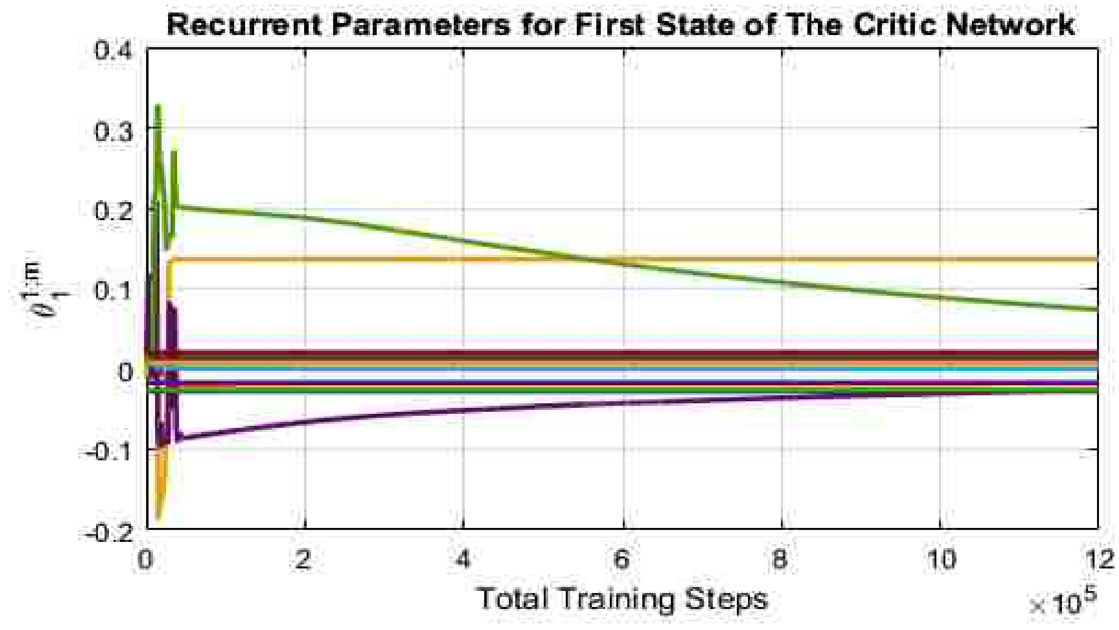


Figure 11. Initial and final learned GMFs for both input states to the actor network in RNF-VGL(λ).



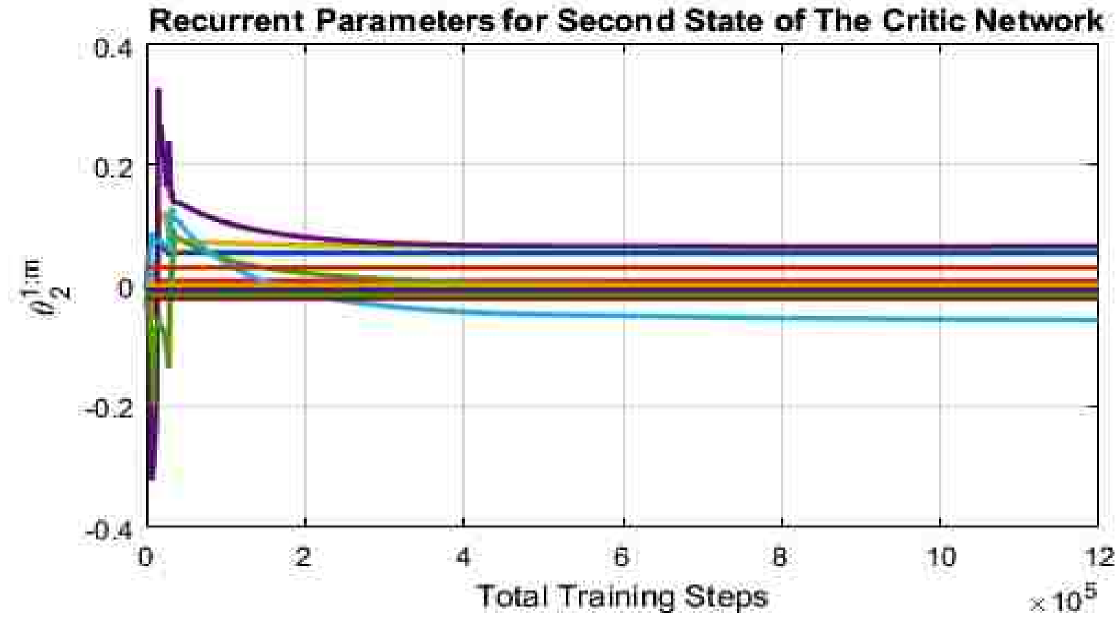
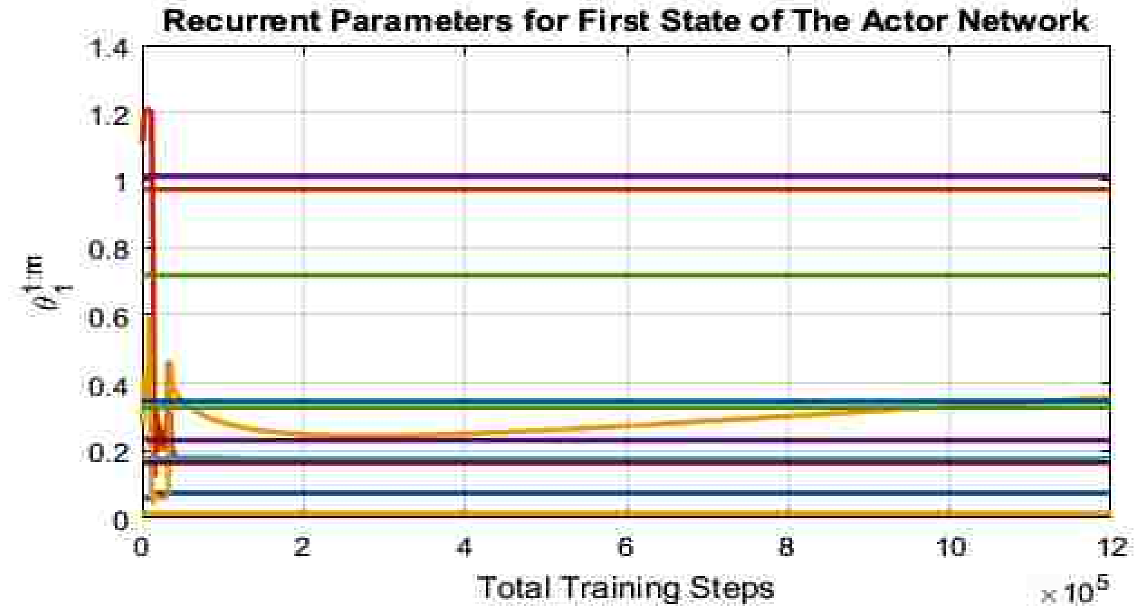


Figure 12. Deviation of θ for both input velocity states from the initialized values of the critic network of RNF-VGL(λ) during all the total training steps.



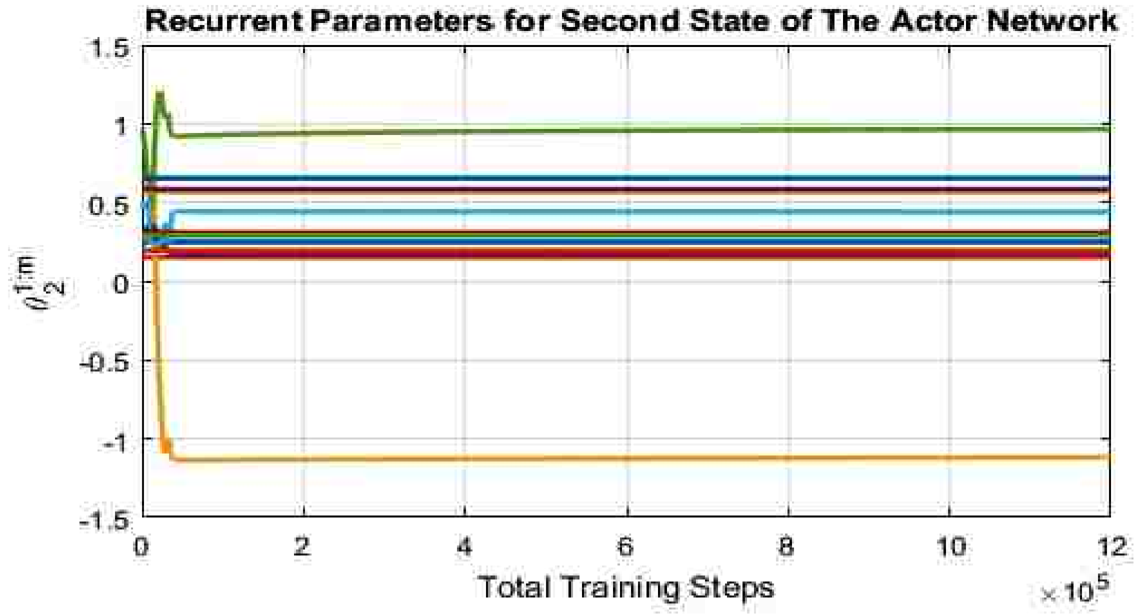


Figure 13. Deviation of θ for both input velocity states from the initialized values of the actor network of RNF-VGL(λ) during the total training steps.

RNF-SNVGL(λ). RNF-SNVGL(λ) is more efficient and faster to learn than RNF-VGL(λ). However, RNF-SNVGL(λ) is faster than RNF-VGL(λ) at reaching a stable performance. The stable MSE value starts at iteration number 45 for RNF-SNVGL(λ), while for RNF-SNVGL(λ), the stable MSE ranging from iteration number 280 as demonstrated in Fig. 15. The MSE value starting at iteration number 280 to 12000 using RNF-VGL(λ) is smaller than RNF-SNVGL(λ) (as discussed in Table 3). Fig. 15 shows the circular trajectory in the second, twentieth, three-hundredth, and 12000th iterations. Fig. 16 presents the torques for the right and left mobile robot wheels during the iterations for RNF-VGL(λ) and RNF-SNVGL(λ) for five independent runs. As shown in the enlarged figures for both torques, RNF-SNVGL(λ) reaches stability faster than RNF-VGL(λ). Fig. 17 demonstrates the convergence of the gradient of the cost functions during the iterations for the states of RNF-VGL(λ) and RNF-SNVGL(λ) in one selected run. The convergence of the gradient of the cost functions ($g_I^\lambda(k)$ and $g_I(k)$) to the optimal cost function indicates the effectiveness of the iterative RNF-VGL(λ) and RNF-SNVGL(λ).

For further comparison, the RNF-VGL(λ) and RNF-SNVGL(λ) are examined under different noise levels as shown in Table 3. Table 3 summarizes the simulation results of average MSEs through 12000 iterations with 100 time steps. Two types of MSEs are presented in Table 3, which are all-iteration (All Iter) and final-iteration (Final Iter). The all-iteration is the average of 12000 MSEs, while the final-iteration is the MSE at the last iteration (12000th). MSEs for the final iteration values demonstrate that:

1. RNF-SNVGL(λ) has a better performance than the RNF-VGL(λ) in free or small amounts of noise impact ($\varrho = 0.0001$) with and without θ parameters, but the performance becomes worse when the noise is gradually increased.

2. RNF-VGL(λ) with θ parameters is more efficiency than without θ parameters, with an improvement of 21.178% in a decreasing percentage format of MSEs over all ϱ values.

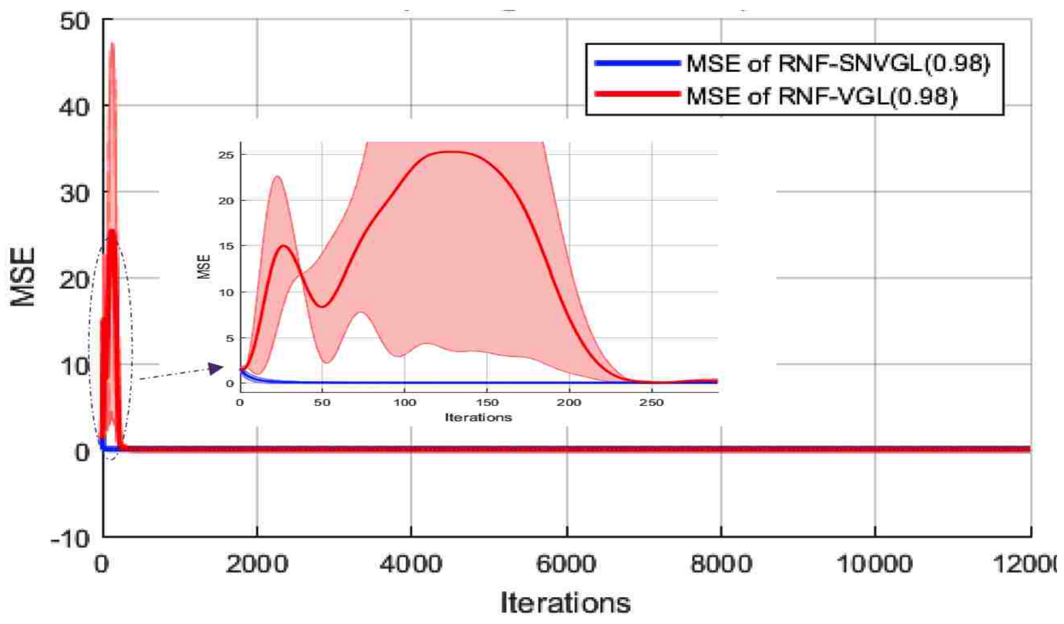


Figure 14. Average of MSEs for the two states for comparisons: RNF-VGL(λ) and RNF-SNVGL(λ) with ϱ = 0.001. Five independent runs are taken. The shaded region represents the runs, while the solid line represents the mean of the runs. RNF-SNVGL(λ) learns faster a faster learning than RNF-VGL(λ).

3. RNF-SNVGL(λ) with θ parameters is more efficient than without θ parameters, with an improvement of 11.778% in a decreasing percentage format of MSEs over all ϱ values.

4. The overall improvement with θ parameters for both RNF-VGL(λ) and RNF-SNVGL(λ) is 15.421% in a decreasing percentage format of MSEs over all ϱ values.

The MSEs for the all-iteration values show that RNF-SNVGL(λ) requires fewer iterations than RNF-SNVGL(λ) to reach stable training when considering the first pointof the previous point. The third evaluation is a comparison between the VGL(λ) and SNVGL(λ) approaches with a TSRNF structure. The initial parameters for the $\boldsymbol{\theta}$, $\boldsymbol{\omega}$, $\boldsymbol{a}$, the number of rules, impact of disturbances and frictions with ϱ value, and the GNFs distribution set is similar to the previous test. Fig. 18 illustrates the average MSEs for the two velocity states during the iterations for five runs for TSRNF-VGL(λ) and TSRNF-SNVGL(λ). TSRNF-SNVGL(λ)

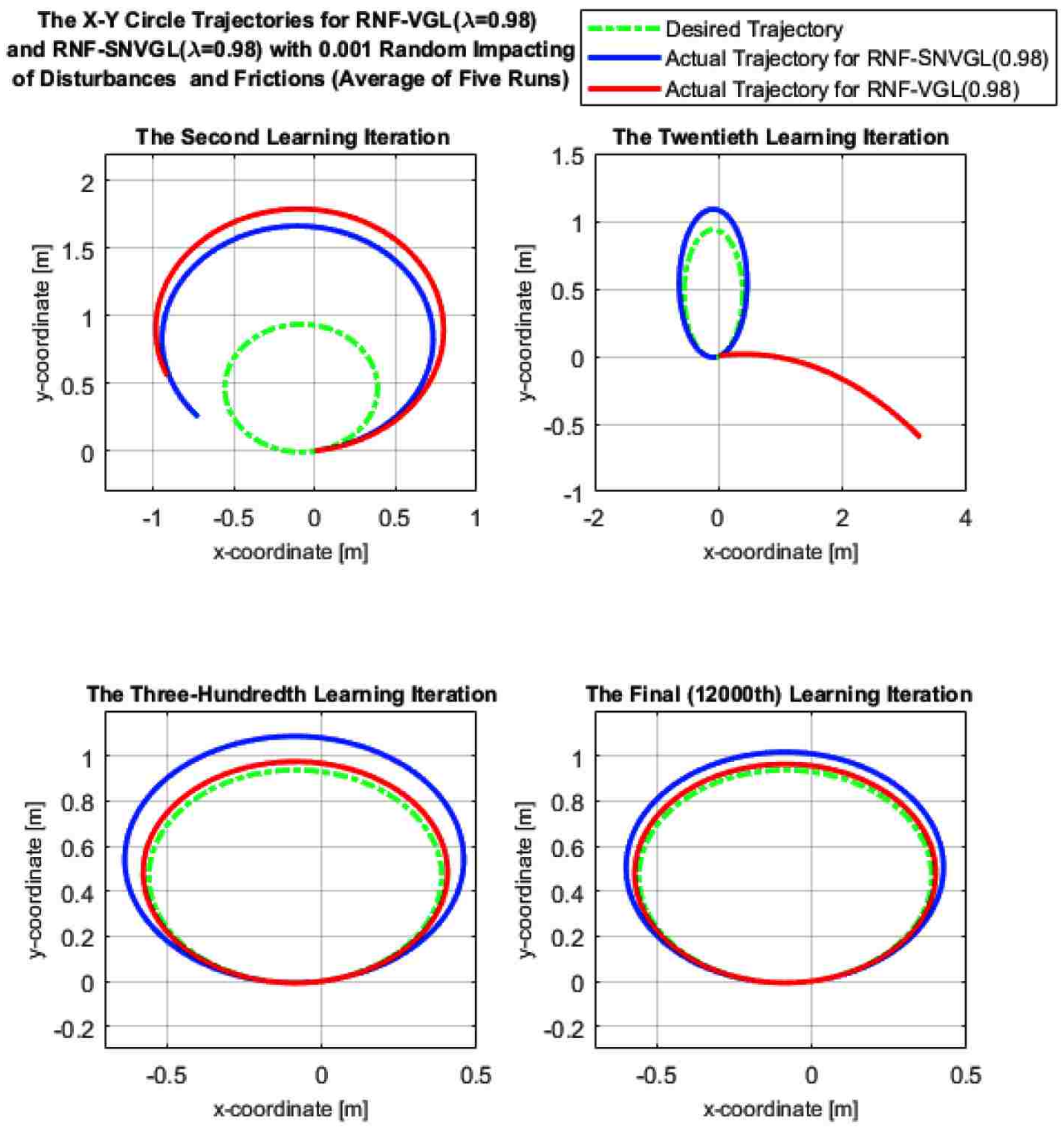


Figure 15. The X-Y circle trajectories for RNF-VGL(λ) and RNF-SNVGL(λ) with ϱ = 0.001. The mean of five independent runs is shown. RNF-SNVGL(λ) is faster, but RNF-VGL(λ) performs better and improves with long training iterations.

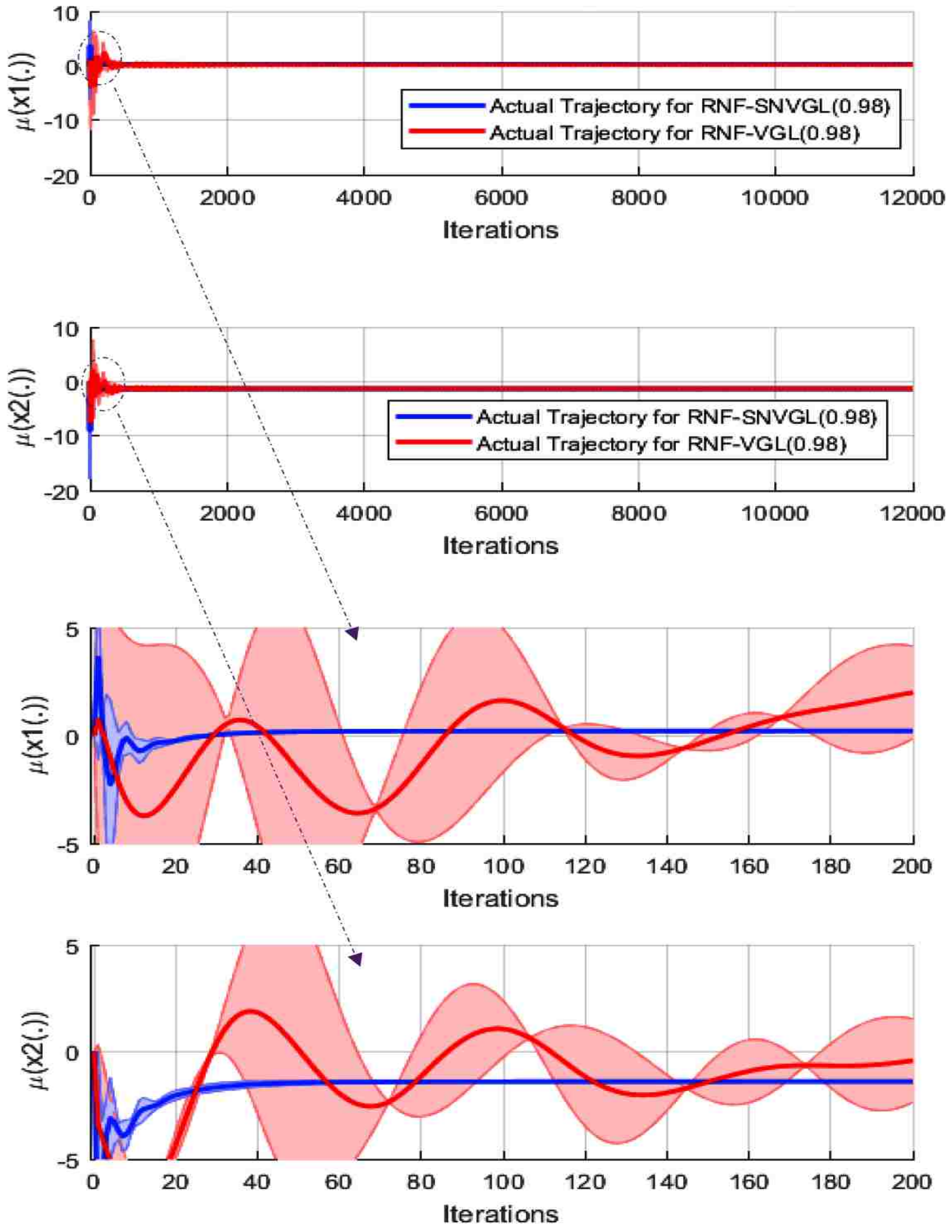


Figure 16. The average of the right and left input torques for the dynamic mobile robot for RNF-VGL(λ) and RNF-SNVGL(λ) with $\varrho = 0.001$. The shaded region represents the runs, while the solid line represents the mean of the runs.

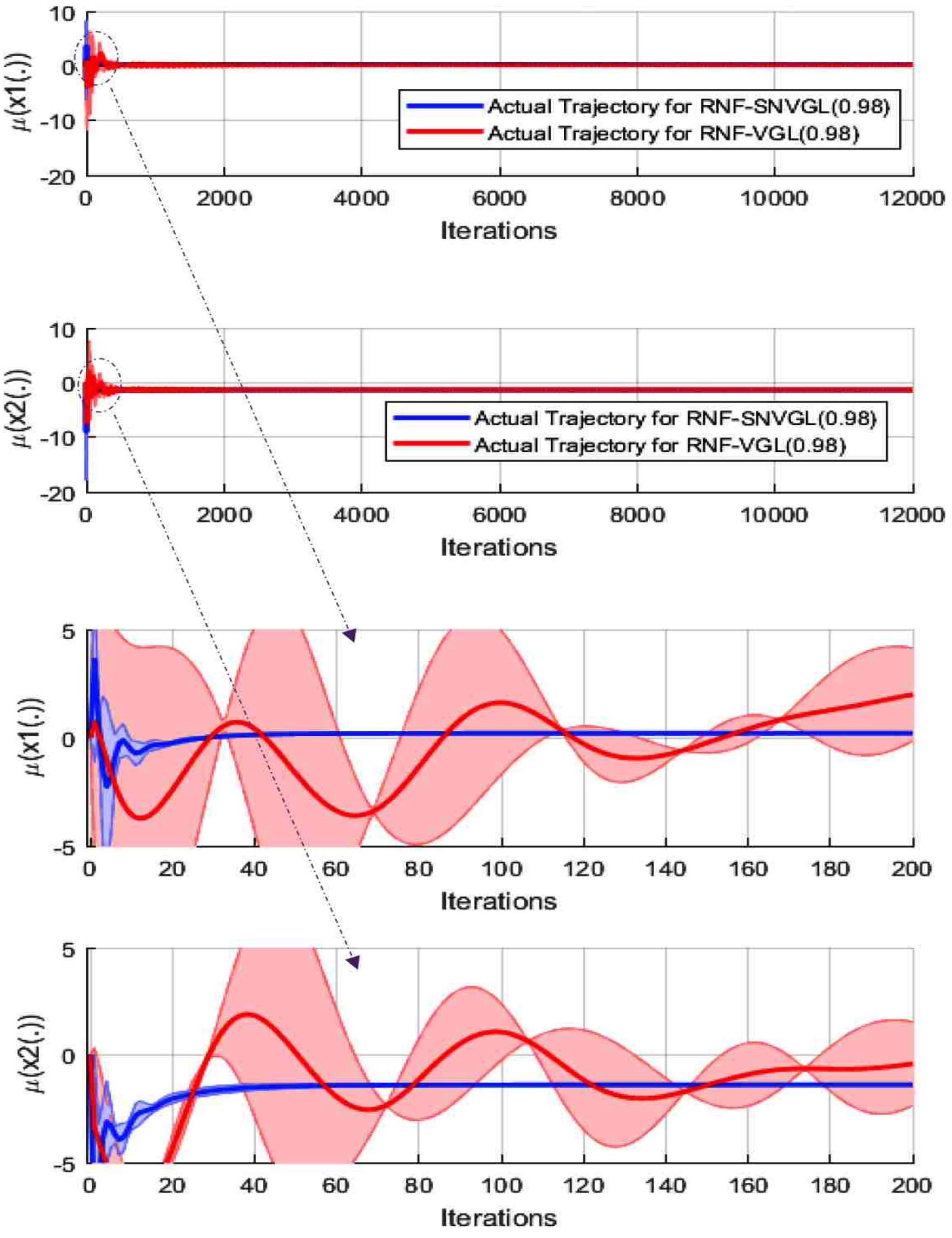


Figure 17. The average gradient of the two input velocity states of the value function trajectories for the critic network in both RNF-VGL(λ) and RNF-SNVGL(λ).

Table 3. MSE values for RNF-VGL(λ) and RNF-SNVGL(λ) with and without θ recurrent parameters at various noise levels (different ϱ values). All-iteration (All Iter) is the average of 12000 MSE, while final-iteration (Final Iter) is the MSE at the last iteration (12000th).

MSE		RNF Structure for Critic and Actor Networks							
		θ				$\theta = 0$			
		RNF-VGL(λ)		RNF-SNVGL(λ)		RNF-VGL(λ)		RNF-SNVGL(λ)	
		All Iter	Final Iter	All Iter	Final Iter	All Iter	Final Iter	All Iter	Final Iter
ϱ	0.0001	2.2758	0.0032	0.0056	0.0022	0.3578	0.0039	0.0051	0.0027
	0.001	1.6735	0.0058	0.0193	0.0185	0.4052	0.0062	0.0196	0.0197
	0.01	6.5847	0.0065	0.0220	0.0216	0.4206	0.0069	0.0216	0.0218
	0.1	1.0404	0.0071	0.0270	0.0271	0.2484	0.0096	0.0213	0.0237
	1	0.3749	0.0403	0.0468	0.0421	0.1685	0.0532	0.0562	0.0585

is more efficient and faster to learn than RNF-VGL(λ). However, TSRNF-SNVGL(λ) is faster than TSRNF-VGL(λ) at reaching a stable performance. The stable MSE value starts at iteration number 25 for TSRNF-SNVGL(λ), while for TSRNF-SNVGL(λ), the stable MSE starts at iteration number 45 as demonstrated in Fig. 19. Fig. 19 shows the circular trajectory in the second, twentieth, three-hundredth, and 12000th iterations. Fig. 20 presents the torques for the right and left mobile robot wheels during iterations for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) for five independent runs. As shown in enlarged figures for both torques, the TSRNF-SNVGL(λ) is faster than TSRNF-VGL(λ) to reach stable. Fig. 21 demonstrates the convergence of the gradient of the cost functions during iterations for the two states of TSRNF-VGL(λ) and TSRNF-SNVGL(λ) in one selected run. The convergence of the gradient of the cost functions ($g_I^{\lambda}(k)$ and $g_I(k)$) to the optimal cost function indicates the effectiveness of the iterative TSRNF-VGL(λ) and TSRNF-SNVGL(λ).

TSRNF-VGL(λ) and TSRNF-SNVGL(λ) are examined under different noise levels as shown in Table 4 for the two types of MSEs (all-iteration and final-iteration). MSEs for the final iteration values demonstrate that:

1. TSRNF-SNVGL(λ)performs better than TSRNF-VGL(λ) in a free or small amount of noise impact ($\varrho = 0.0001$) with and without θ parameters, but the performance become worse when the ϱ value is increased gradually.

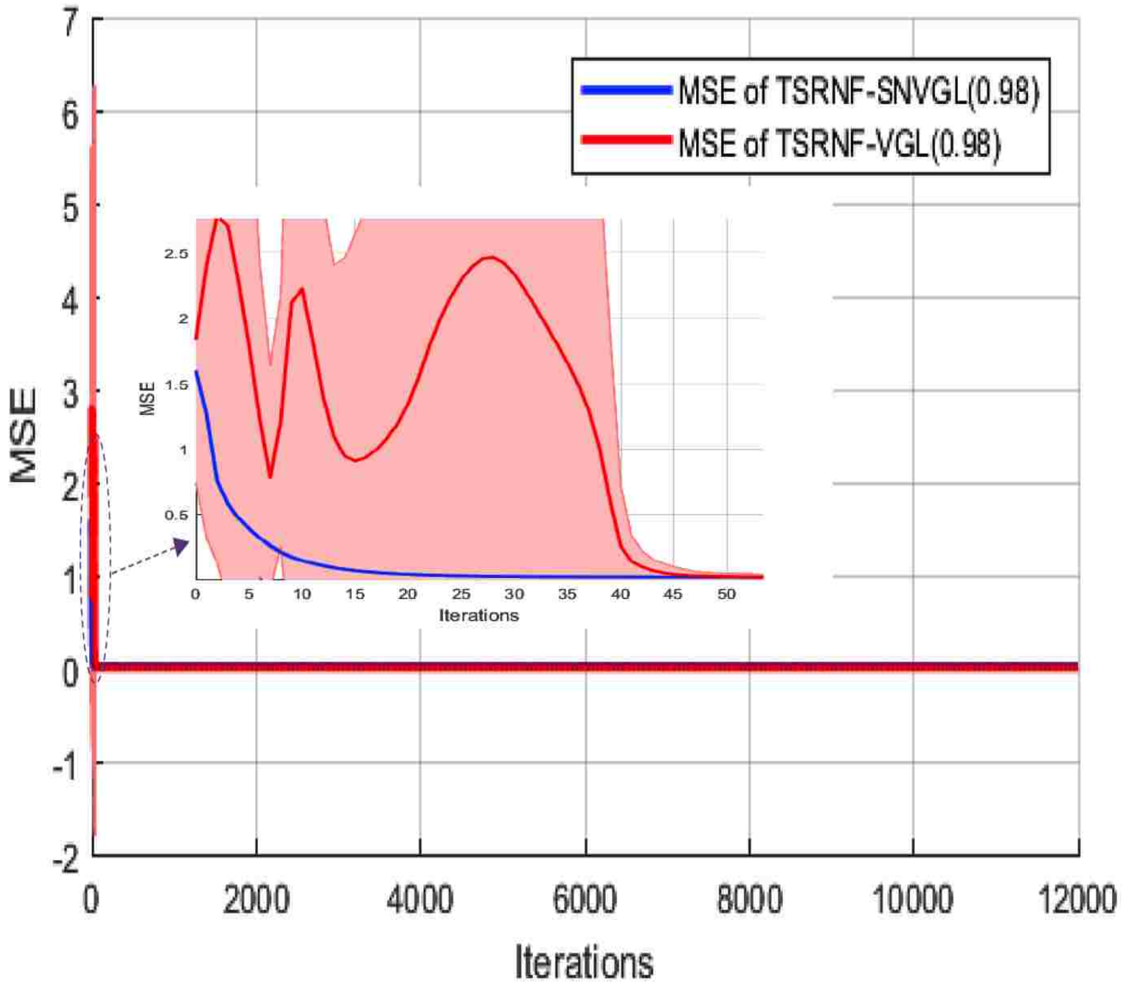


Figure 18. The average of the MSEs for comparing the two states of TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with $\varrho = 0.001$. Five independent runs are taken. The shaded region represents the runs, while the solid line represents the mean of the runs. TSRNF-SNVGL(λ) learns faster than TSRNF-VGL(λ).

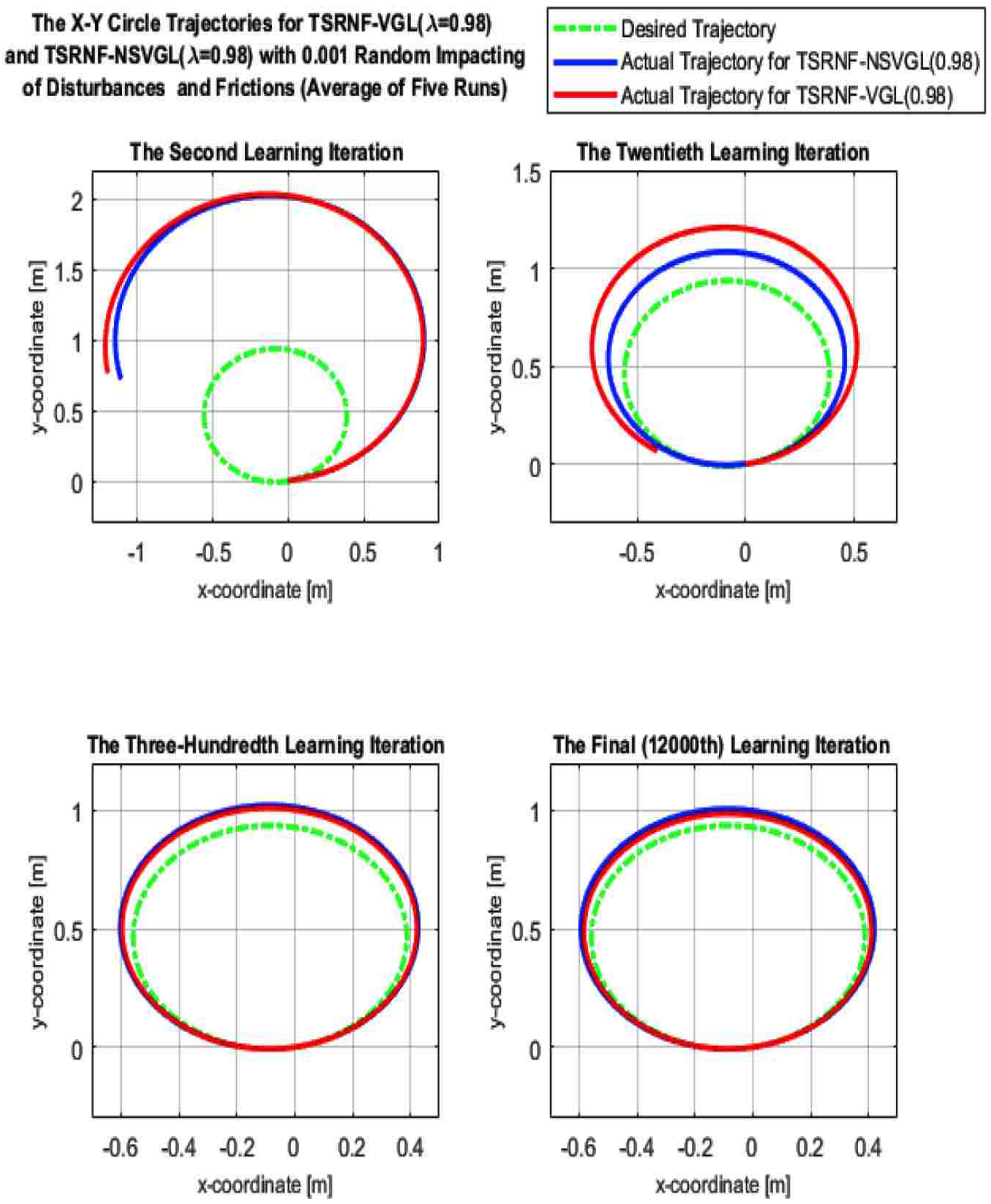


Figure 19. The X-Y circle trajectories for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with ϱ = 0.001. The mean of five independent runs is shown. TSRNF-SNVGL(λ) learns faster, but TSRNF-VGL(λ) perfoms better ad improves with long training iterations.

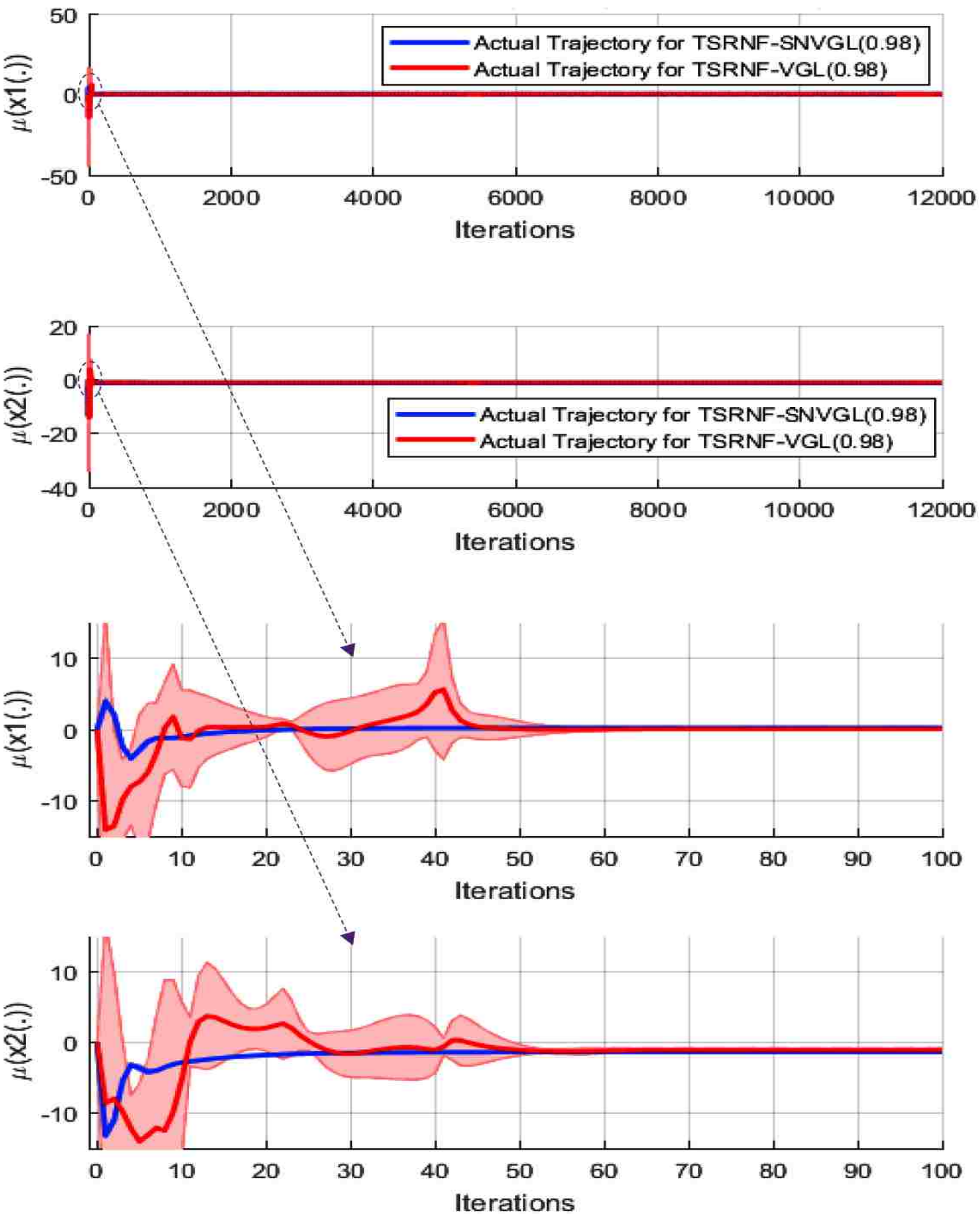


Figure 20. The average of the right and left input torques to the dynamic mobile robot for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with ϱ = 0.001. The shaded region represents the runs, while the solid line represents the mean of the runs.

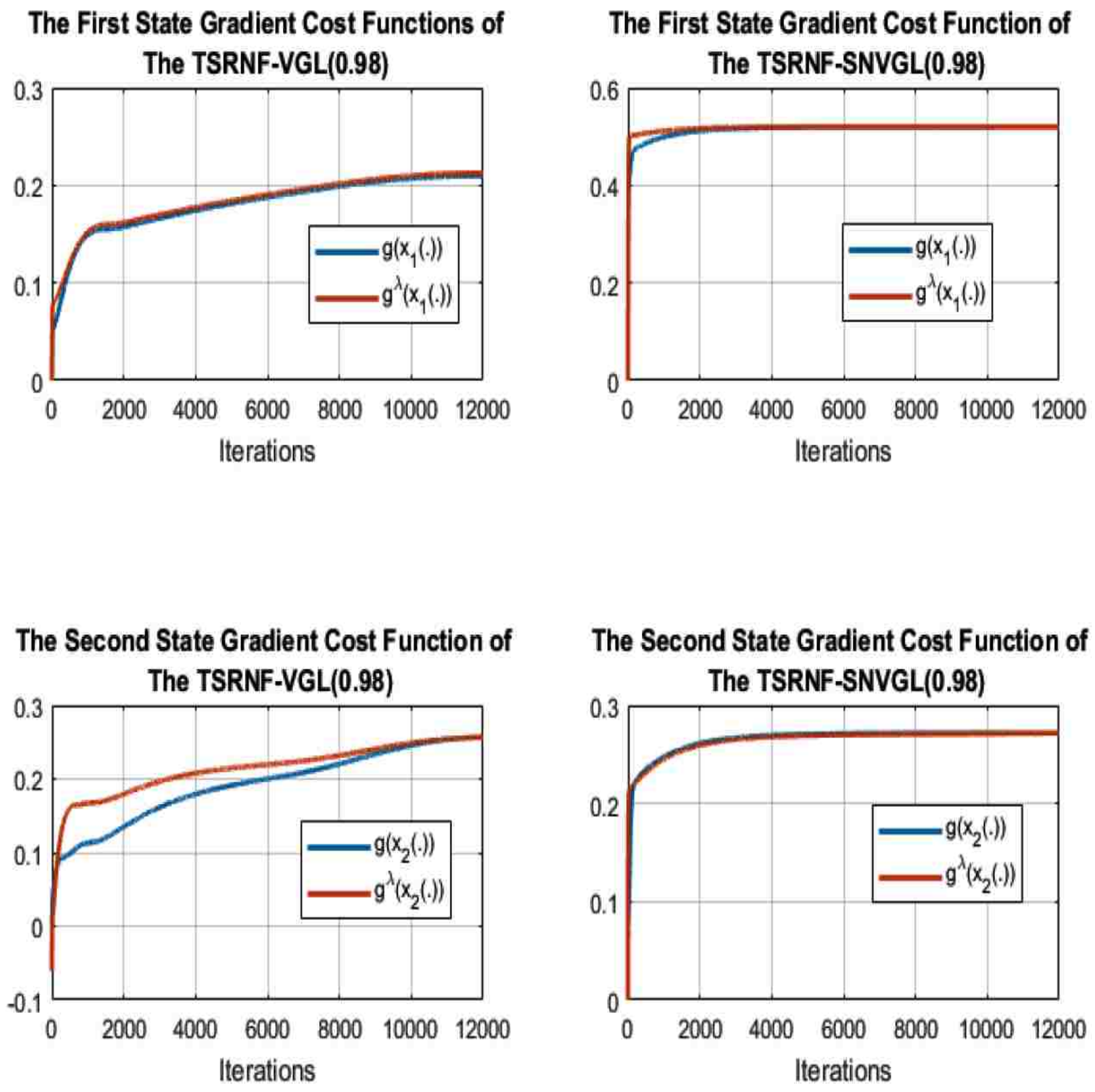


Figure 21. The average gradient of the two input velocity states of the value function trajectories for the critic network in both TSRNF-VGL(λ) and TSRNF-SNVGL(λ).

Table 4. MSE values for TSRNF-VGL(λ) and TSRNF-SNVGL(λ) with and without θ recurrent parameters at various noise levels (different ϱ values). All-iteration (All Iter) is the average of 12000 MSE, while final-iteration (Final Iter) is the MSE at the last iteration (12000th).

MSE		TSRNF Structure for Critic and Actor Networks							
		θ				$\theta = 0$			
		NF-VGL(λ)		TSRNF-SNVGL(λ)		TSRNF-VGL(λ)		TSRNF-SNVGL(λ)	
		All Iter	Final Iter	All Iter	Final Iter	All Iter	Final Iter	All Iter	Final Iter
ϱ	0.0001	0.0105	0.0025	0.0059	0.0012	0.0113	0.0028	0.0064	0.0015
	0.001	0.0417	0.0051	0.0190	0.0184	0.0549	0.0064	0.0189	0.0184
	0.01	0.0239	0.0053	0.0132	0.0126	0.0079	0.0072	0.0189	0.0183
	0.1	0.0215	0.0073	0.0182	0.0177	0.0120	0.0073	0.0182	0.0178
	1	0.0226	0.0093	0.0128	0.0124	0.0682	0.0093	0.0128	0.0125

2. TSRNF-VGL(λ) with θ parameters is more efficient than it is without θ parameters, with an improvement of 10.606% and a decreasing percentage of MSEs over all ϱ values.

3. TSRNF-SNVGL(λ) with θ parameters is more efficient than it is without θ parameters, with an improvement of 9.051% and a decreasing percentage of MSEs over all ϱ values.

4. The overall improvement with θ parameters for both TSRNF-VGL(λ) and TSRNF-SNVGL(λ) is 9.556% with decreasing percentage of MSEs over all ϱ values.

MSEs for the all-iteration values present with TSRNF-SNVGL(λ) require fewer iterations than those present with TSRNF-SNVGL(λ) to reach stable training. Overall, the comparison between RNF- for VGL(λ) and SNVGL(λ) and TSRNF- VGL(λ) and SNVGL(λ) are

1. With θ, TSRNF with both VGL(λ) and SNVGL(λ) has a better performance than RNF with 47.362% and decreasing percentage of MSEs for the final-iteration with all ϱ values.

2. Without θ, TSRNF with both VGL(λ) and SNVGL(λ) has a better performance than RNF with 50.775% and a decreasing percentage of MSEs for the final-iteration with all ϱ values.
3. With and without θ, TSRNF with both VGL(λ) and SNVGL(λ) has a better performance improvement than RNF with 96.965% in a decreasing percentage format of all-iteration MSEs for all ϱ values.

7. CONCLUSION

VGL(λ) is the high-performance algorithm in ADP that is used in this work, which is inspired by a gradient of TD(λ). VGL(λ) is implemented by using two networks (critic and actor). This work derives and performs the VGL(λ) architecture with a single adaptive critic network (SNVGL(λ)). The SNVGL(λ) is compared with regular VGL(λ) to track a reference trajectory of a simulation of a nonlinear dynamic model of a nonholonomic mobile robot. Because a noisy environment is more realistic in the real world, the controller (actor network in VGL(λ) or optimal control equation in SNVGL(λ)) is tuned under uncertainties to compute the optimal right and left torques. Therefore, two-hybrid neuro-fuzzy structures were used in both actor and critic networks (RNF and TSRNF). Moreover, convergence proofs were proved to demonstrate the application of an ADP iteration algorithm with the VGL(λ) and SNVGL(λ) approaches.

BIBLIOGRAPHY

[1] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310 - 335, Aug. 2011.

[2] F. L. Lewis, and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32 - 50, Aug. 2009.

[3] R. Padhi, N. Unnikrishnan, X. Wang, and S. N. Balakrishnan, “A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems,” *Neural Netw.*, vol. 19, no. 10, pp. 1648 - 1660, Dec. 2006.

[4] B. Luo, D. Liu, T. Huang, and D.Wang, Member, “Model-Free Optimal Tracking Control via Critic-Only Q-Learning,” *Neural Netw.*, vol. 27, no. 10, pp. 2134 - 2144, Oct. 2016.

[5] J. Fu, H. He, and X. Zhou, “Adaptive learning and control for MIMO system based on adaptive dynamic programming,” *IEEE Trans. Neural Netw.*, vol. 22, no. 7, pp. 1133 - 1148, Jul. 2011.

[6] Z. Ni, H. He, and J. Wen, “Adaptive learning in tracking control based on the dual critic network design,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 6, pp. 913 - 928, Jun. 2013.

[7] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, Eds., *Handbook of Learning and Approximate Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2004.

[8] F. L. Lewis, and D. Liu, Eds., *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. Hoboken, NJ, USA: Wiley, 2013.

[9] Q. Wei, and D. Liu, “Adaptive dynamic programming for optimal tracking control of unknown nonlinear systems with application to coal gasification,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 4, pp. 1020 - 1036, Oct. 2014.

[10] R. Bellman, *Dynamic Programming*. Princeton Univ. Press, NJ, USA, 1957.

[11] D. Prokhorov, and D. C. Wunsch, “Adaptive critic designs,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 8, no. 8, pp. 997 - 1007, Sep. 1997.

[12] P. J. Werbos, “Approximate dynamic programming for real-time control and neural modeling,” Handbook of Intelligent Control: *Neural, Fuzzy, and Adaptive Approaches (Chapter 13)*, Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[13] F.-Y. Wang, H. Zhang, and D. Liu, “Adaptive dynamic programming: An introduction,” *IEEE Comput. Intell. Mag.*, vol. 4, no. 2, pp. 39 - 47, May 2009.

[14] F. L. Lewis, and D. Vrabie, “Reinforcement learning and adaptive dynamic programming for feedback control,” *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32 - 50, Sep. 2009.

[15] S. Ray, G. K. Venayagamoorthy, B. Chaudhuri, and R. Majumder, “Comparison of adaptive critic-based and classical wide-area controllers for power systems,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 4, pp. 1002 - 1007, Aug. 2008.

[16] P. J. Werbos, “Backpropagation through time: What it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550 - 1560, Oct. 1990.

[17] H. Zhang, Y. Luo, and D. Liu, “Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints,” *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490 - 1503, Sep. 2009.

[18] J. Si, and Y. Wang, “Online learning control by association and reinforcement,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 12, no. 2, pp. 264 - 276, Mar. 2001.

[19] Z. Ni, H. He, X. Zhong, and D. Prokhorov, “Model-Free dual heuristic dynamic programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 8, pp. 1834 - 1839, Aug. 2015.

[20] H. He, Z. Ni, and J. Fu, “A three-network architecture for on-line learning and optimization based on adaptive dynamic programming,” Neurocomputing, vol. 78, no. 1, pp. 3 - 13, Feb. 2012.

[21] X. Fanga, D. Zhenga, H. He, and Z. Nib, “Data-driven heuristic dynamic programming with virtual reality,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 166, no. 6, pp. 244 - 255, Oct. 2015.

[22] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, “GrDHP: A general utility function representation for dual heuristic dynamic programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 3, pp 614 - 626, Mar. 2015.

[23] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Dual heuristic programming excitation neurocontrol for generators in a multimachine power system,” *IEEE Trans. Indus. Applying*, vol. 39, no. 2, pp. 382 - 394, Mar. 2003.

[24] N. Zhang, and D. C. Wunsch, “A Comparison of Dual Heuristic Programming (DHP) and neural network based stochastic optimization approach on collective robotic search problem,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 1, pp. 248 - 253, Jul. 2003.

[25] C. Lian, and X. Xu, “Motion planning of wheeled mobile robots based on heuristic dynamic programming,” *IEEE Proceeding of the World Congress on Intelligent Control and Automation Shenyang*, pp 576 - 580, Jul. 2014.

[26] P. J. Werbos, and X. Pang, “Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot,” *IEEE Proc. Conf. Systems, Man and Cybernetics (SMC)* , pp. 1764-1769, Oct. 1996.

[27] D. Wunsch, “The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 79-82, Jul. 2000.

[28] R. Ilin, R. Kozma, and P. J. Werbos,“Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator,” *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 929-937, Jun. 2008.

[29] S. Al-Dabooni, and D. Wunsch, “Online Model-Free N-Step HDP with Stability Analysis,” Under Preparing.

[30] Z. Ni, H. He, J. Wen, and X. Xu, “Goal representation heuristic dynamic programming on maze navigation,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 12, pp. 2038 - 2050, Dec. 2013.

[31] N. Zheng, and P. Mazumder, “Hardware-Friendly Actor-Critic Reinforcement Learning Through Modulation of Spike-Timing-Dependent Plasticity,” *IEEE Trans. on Computers*, vol. 66, no. 2, pp. 299-311, Feb. 2017.

[32] S. Al-Dabooni, and D. Wunsch, “The Boundedness Conditions for Model-Free HDP(λ),” Under reviewing for *IEEE Trans. Neural Netw. Learn. Syst.*

[33] S. Al-Dabooni, and D. Wunsch, “Heuristic dynamic programming for mobile robot path planning based on Dyna approach,” *IEEE, International Joint Conference on Neural Networks (IJCNN)*, pp. 3723 - 3730, Jul. 2016.

[34] B. Xu, C. Yang, and Z. Shi, “Reinforcement learning output feedback NN control using deterministic learning technique,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 635 - 641, Mar. 2014.

[35] Q. Wei, and D. Liu, “Data-driven neuro-optimal temperature control of waterâĂŞgas shift reaction using stable iterative adaptive dynamic programming,” *IEEE Trans. Ind. Electron.*, vol. 61, no. 11, pp. 6399 - 6408, Nov. 2014.

[36] D. Liu, D. Wang, F.-Y. Wang, H. Li, and X. Yang, “Neural-networkbased online HJB solution for optimal robust guaranteed cost control of continuous-time uncertain nonlinear systems,” *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2834 - 2847, Dec. 2014.

[37] H. He, Self-Adaptive Systems for *Machine Intelligence*. New York, NY, USA: Wiley, 2011.

[38] D. Liu, and Q. Wei, “Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 621 - 634, Mar. 2014.

[39] Y. Sokolov, R. Kozma, L. D. Werbos, and P. J. Werbos, “Complete stability analysis of a heuristic approximate dynamic programming control design,” *Automatica*, vol. 59, pp. 9 - 18, Sep. 2015.

[40] F. Liu, J. Sun, J. Si, W. Guo, and S. Mei, “A boundedness result for the direct heuristic dynamic programming,” *Neural Networks*, vol. 32, pp. 229-235, Aug. 2012.

[41] Y. Tang, H. He, Z. Ni, X. Zhong, D. Zhao, and X. Xu, “Fuzzy-Based Goal Representation Adaptive Dynamic Programming,” *IEEE Trans. on Fuzzy Sys.*, vol. 24, no. 5, pp. 1156 - 1176, Oct. 2016.

[42] X. Zhong, Z. Ni, and H. He, “A Theoretical Foundation of Goal Representation Heuristic Dynamic Programming,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[43] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear hjb solution using approximate dynamic programming: convergence proof," *IEEE Trans. on Sys., Man, and Cyb., Part B*, vol. 38, no. 4, pp. 943 - 949, Aug. 2008.

[44] D. Liu, and Q. Wei, "Finite-approximation-error-based optimal control approach for discrete-time nonlinear systems," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 779 - 789, Apr. 2013.

[45] D. Liu, and Q. Wei, "Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 621 - 634, Mar. 2014.

[46] H. Zhang, Y. Luo, and D. Liu, "Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490 - 1503, Sep. 2009.

[47] Y. Yang, D. Wunsch, and Y. Yin, "Hamiltonian-driven adaptive dynamic programming for continuous nonlinear dynamical systems," *IEEE Trans. Neural Netw. Learn. Syst.* vol. PP, no. 12, pp. 1 - 12, Feb. 2017 (to be published).

[48] D. Liu, D. Wang, D. Zhao, Q. Wei, and N. Jin, "Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 3, pp. 628 - 634, Jul. 2012.

[49] D. Liu, and D. Wang, "Optimal control of unknown nonlinear discretetime systems using the iterative globalized dual heuristic programming algorithm," in *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. New York, NY, USA: Wiley, pp. 52 - 74, Jan. 2013,

[50] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9 - 44, Aug. 1988.

[51] H. Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, "True Online Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 145, no. 17, pp. 1 - 40, Jan. 2016.

[52] R. S. Sutton, A. R. Mahmood, and M. White, "An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 73, no. 17 pp. 1 - 29, Jan. 2016.

[53] S. P. Singh, and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1, pp. 123 - 158, Mar. 1996.

[54] H. Seijen, and R. S. Sutton, "True Online TD(λ)," *Proceedings International Conference on Machine Learning*, pp. 692 - 700, Jan. 2014.

[55] M. Fairbank, and E. Alonso, "Value-Gradient Learning," *IEEE, International Joint Conference on Neural Networks (IJCNN)*, pp. 1 - 8, Jun. 2012.

[56] M. Fairbank, E. Alonso, and D. Prokhorov, “An Equivalence Between Adaptive Dynamic Programming With a Critic and Backpropagation Through Time,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 12, pp. 2088 - 2100, Dec. 2013.

[57] F. L. Lewis, and D. Liu, Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, Chapter 7, John Wiley and Sons, Jan. 2013.

[58] S. Al-Dabooni, and D. Wunsch, “Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning,” *IEEE, International Joint Conference on Neural Networks (IJCNN)*, pp. 2820 - 2827, May 2017.

[59] S. Al-Dabooni, and D. Wunsch, “An Improved N-Step Value Gradient Learning Adaptive Dynamic Programming Algorithm for Online Learning, with Convergence Proof and Case Studies”, Under review for *IEEE Trans. Neural Netw. Learn. Syst.*

[60] D. Zhao, Y. Zhu, and H. He, “Neural and fuzzy dynamic programming for under-actuated systems,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1 - 7, Jun. 2012.

[61] S. Mohagheghi, G. K. Venayagamoorthy, and R. G. Harley, “Fully evolvable optimal neurofuzzy controller using adaptive critic designs,” *IEEE Trans. on fuzzy systems, vol. 16, no. 6,* pp 1450 - 1461, Dec. 2008.

[62] S. Mohagheghi, G. K. Venayagamoorthy, and R. G. Harley, “Adaptive critic design based neuro-fuzzy controller for a static compensator in a multimachine power system,” *IEEE Trans. on Power Systems,* vol. 21, no. 4, pp 1450 - 1461, Nov. 2006.

[63] Y. Zhu, D. Zhao, and H. He, “Integration of fuzzy controller with adaptive dynamic programming,” *IEEE conference World Congress In Intelligent Control and Automation (WCICA),* pp. 310 - 315, Jul.2012.

[64] Y. Tang, C. Mu, and H. He, “SMES-based damping controller design using fuzzy-GrHDP considering transmission delay,” *IEEE Trans. on Applied Superconductivity,* vol. 26, no. 7, pp 1 - 6, Oct. 2016.

[65] H. Zhang, J. Zhang, G. H. Yang, and Y. Luo, “Leader-based optimal coordination control for the consensus problem of multiagent differential games via fuzzy adaptive dynamic programming,” *IEEE Trans. on Fuzzy Systems,* vol. 23, no. 1, pp 152 - 163, Feb. 2015.

[66] J. L. Zhang, H G. zhang, L. U. Yan-Hong, and H. J. liang, “Nearly optimal control scheme using adaptive dynamic programming based on generalized fuzzy hyperbolic model,” *Acta Automatica Sinica,* vol. 39, no. 2, pp 142 - 1148 Feb. 2013.

[67] X. Bai X, D. Zhao D, and J. Yi, “Coordinated multiple ramps metering based on neuro-fuzzy adaptive dynamic programming,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN),* pp. 241 - 248, Jun. 2009.

[68] X. Luo, Y. Lv, R. Li, and Y. Chen, “Web service QoS prediction based on adaptive dynamic programming using fuzzy neural networks for cloud services,” *IEEE Access,* vol. 3, pp 2260 - 2269, 2015.

[69] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. NewYork, NY, USA: Wiley, 2012.

[70] R. S. Sutton, and A. Barto,*Reinforcement Learning: An Introduction*, Cambridge, U.K.: Cambridge Univ. Press, 1998.

[71] C. H. Lee, and C. C. Teng, “Identification and control of dynamic systems using recurrent fuzzy neural networks,” *IEEE Trans. on fuzzy systems*, vol. 8, no. 4, pp. 349 - 366, Aug. 2000.

[72] J.-S. Jang, C.-T. Sun, and E. Mizutani, *Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence* Upper Saddle River, NJ: Prentice-Hall, 1997.

[73] M. Sugeno, G. Kang, “Structure identification of fuzzy model,” *Fuzzy Sets Syst*, vol. 28, pp.15âĂŞ33, 1988.

[74] J.-S. Jang, “ANFIS: adaptive-network-based fuzzy inference system,” *IEEE trans. on systems, man, and cybernetics,* vol. 23, no. 3, pp. 665-685, May 1993.

[75] J. W. Yeh, and S. F. Su, “Efficient approach for RLS type learning in TSK neural fuzzy systems,” *IEEE trans. on cybernetics,* vol. 47, no. 9, pp. 2343-2352, Sep. 2017.

[76] J.-S. Wang, and C.-G. Lee, “Self-adaptive neuro-fuzzy inference systems for classification applications” *IEEE Trans. on Fuzzy Sys,* vol. 10, no. 2, pp. 790-802, Dec. 2002.

[77] R. Fierro, and F. L. Lewis, “Control of a Nonholonomic Mobile Robot Using Neural Networks,” *IEEE Trans. Neural Networks*, vol. 9, no. 4, pp. 589 - 600, July 1998.

[78] W. S. Lin, L. H. Chang, and P. C. Yang, “Adaptive critic anti-slip control of wheeled autonomous robot,” *IET Control Theory and Applications*, vol. 1, no. 1, pp. 51 - 57, Jan. 2007.

[79] T. Dierks, and S. Jagannathan, “Neural Network Output Feedback Control of Robot Formations,” *IEEE Trans. on Sys, Man, and Cyb.*, vol. 40, no.2, pp. 383 - 399, Apr. 2010.

SECTION

2. SUMMARY AND CONCLUSIONS

2.1. MODEL ORDER REDUCTION BY CLUSTERING SYSTEM POLES

Two main subjects are investigated in this dissertation. First, the model order reduction algorithms are used to make systems less complexity for addressing with retaining the original system properties. Second, adaptive dynamic programming algorithm is used in optimal control field to obtain an approximating solution for Hamilton-Jacobi-Bellman equation by interacting with its environment to obtain an optimal control policy. In the model order reduction part that is illustrated as the first paper in this dissertation, we create a new novel algorithm that uses agglomeration hierarchical clustering based on performance evaluation. Many advantages are demonstrated in this work that can be used with single-input single-output large order models and multi-input multi-outputs large order models. This method gives a major advantage for reducing error to produce a robust reduced simple model that has response(s) similar to the original model. Optimizing the pole clusters is achieved by taking the minimum MSE among all pole clusters at an appropriate level in the hierarchical dendrogram. The agglomeration hierarchical clustering based on performance evaluation algorithm is considered a lower level in the hierarchy as the base model. Our reduction algorithm procedure are continued processing level by level until reaching to the second order of reduced model (top level). We demonstrate the simplicity and robustness of the reduced model after applying various cases.

2.2. ADAPTIVE DYNAMIC PROGRAMMING WITH N-STEP PREDICTION PARAMETER

In adaptive dynamic programming part, heuristic dynamic programming is combined with Dyna algorithm method, which is the second paper in this dissertation. This combination gives a powerful tool for system responses with the robust results. This novel technique is used for path planing of mobile robot in unknown environment fill with obstacles. We obtain excellent performance by comparing with one step Q-learning, Sarsa (λ) and Dyna-Q-learning algorithms. The multi-robots cooperative navigation are performed that has a significant advantage to enhance the efficiency of the virtual common environment model. A modern adaptive dynamic programming algorithm, which is the extension of dual heuristic dynamic programming is used as a third paper in this dissertation. This paper is used for racking a reference trajectory for a mobile robot with the impacts of unmodeled bounded disturbances with various friction parameter values. Therefore, we use a hybrid fuzzy neural network to deal with these noise affections in both citric and actor networks. The combination structure address the effects of most unstructured disturbance / friction signals. Model-free action dependence heuristic dynamic programming with n-step for prediction cost values is illustrated as the fourth paper in this dissertation. The uniformly ultimately bounded stability proofs with n-step are provided as a pioneer project in adaptive dynamic programming algorithm with testing the performance with three simulation studies. The fifth paper is addressed what lamination in the fourth paper. Whereas the forth paper should provided the value functions for first iteration (similar to policy iteration algorithm by providing the initial policy). In the fifth paper presents the on-line learning for model-free action dependence heuristic dynamic programming with n-step for prediction cost by adding extra citric network. Good performance is demonstrated in the fifth paper for this dissertation by examining a simulation analysis on a nonlinear system and a inverted pendulum benchmark problem in various circumstances, as well as solving a 2-D maze problem. The uniformly ultimately bounded stability proofs with on-line learning of n-step prediction cost values are provided in the fifth paper. The sixth and seventh papers deal with the gradient of TD(λ) for adaptive dynamic programming, which called value-gradient learning. The batch-mode and direct on-line implementation is provided for the sixth and seventh papers, respectively. The on-line design is more memory efficient by overcoming the drawback of using eligibility-trace storage for system states in online-mode implementation. Both papers (sixth and seventh in the dissertation) are provided the convergence proofs are provided for both gradients of one- and n-step value functions with respect to system states. We apply neural networks, the recurrent hybrid neuro-fuzzy and a first-order Takagi-Sugeno recurrent hybrid neuro-fuzzy to implement our approaches in for last two paper in the dissertation to verify the theoretical analyses.

BIBLIOGRAPHY

[1] W. Schilders, “Introduction to model order reduction,” *Theory, Research Aspects and Applications*, vol. 13, pp. 3-32, 2008.

[2] P. Trnka, C. Sturk, H. Sandberg, V. Havlena, and J. Rehor, “Structured model order reduction of parallel models in feedback,” *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 3, pp. 739-752, May 2013.

[3] N. A. Gershenfeld, *The Nature of Mathematical Modeling*, Cambridge University Press, Mathematics, 1999.

[4] D. Xue, Y. chen, and D. P. Atherton, *Linear feedback control analysis and design with Matlab*, Society for Industrial and Applied Mathematics, Advances in Design and Control, Jan. 2007.

[5] K. J. Astrom and R. M. Murray. *Feedback systems: an introduction for scientists and engineers*, Princeton University Press, Apr. 2010.

[6] I. Kale, J. Gryka, G. D. Cain, and B. Beliczynski, “FIR filter order reduction: balanced model truncation and Hankel-norm optimal approximation,” *IEEE Proc. Vision, Image and Signal Processing*, vol. 141, no. 3, pp. 168-174, Jun. 1994.

[7] W. Wang, G. N. Paraschos, and M. N. Vouvakis, “Fast frequency sweep of FEM models via the balanced truncation proper orthogonal decomposition,” *IEEE trans. Antennas and Propagation*, vol. 59, no. 11, pp. 4142-4154, Oct. 2011.

[8] A. K. Sinha and J. Pal, “Simulation based reduced order modelling using a clustering technique,” *Computers and Electrical Engineering*, vol. 16, no. 3, pp. 159-169, Jan. 1990.

[9] J. Pal, “Improved Pade approximants using stability equation method,” *IEEE Proc. Electronics Letters*, vol. 19, no. 11, pp. 426-427, May 1983.

[10] N. Gupta and A. Narain, “Reduction of discrete interval systems through fuzzy-C means clustering with dominant pole retention,” *Australian Control Conference (AUCC)*, pp. 348-353, Nov. 2015.

[11] C. B. Vishwakarma and R. Prasad, “Time domain model order reduction using Hankel matrix approach,” *Journal Franklin Institute*, vol. 351, no. 6, pp. 3445-3456, Jun. 2014.

[12] V. P. Singh, P. Chaubey, and D. Chandra, “Model order reduction of continuous time systems using pole clustering and chebyshev polynomials,” *IEEE Proc., Engineering and Systems Conference*, pp. 1-4, Mar. 2012.

[13] W. T. Beyene, “Pole-Clustering and Rational-Interpolation Techniques for Simplifying Distributed Systems,” *IEEE trans. Circuits and syst. Fund. Theory and App.*, vol. 46, no. 12, pp. 1468-1472, Dec. 1999.

[14] R. Xu and D. Wunsch, “Survey of clustering algorithms,” *IEEE Trans Neural Netw.*, vol. 16, no. 3, pp. 645-678, May 2005.

[15] R. Xu and D. Wunsch, *Clustering*, Wiley-IEEE Press, Oct. 2008.

[16] A. Mirzaei, and M. Rahmati, “A novel hierarchical-clustering-combination scheme based on fuzzy-similarity relations,” *IEEE Trans. Fuzzy Syst.*, vol. 18, no.1, pp. 27-39, Feb. 2010.

[17] A. Proietti, L. Liparulo, and M. Panella, “2D hierarchical fuzzy clustering using kernel-based membership functions,” *IEEE Electronics Letters*, vol. 52, no.3, pp. 193-195, Feb. 2016.

[18] L. Zheng and T. Li, “Semi-supervised Hierarchical Clustering,” *IEEE Proc. International Conference on Data Mining*, pp. 982-991, Dec. 2011.

[19] J. Singh, C. B. Vishwakarma, and K. Chattterjee, “Biased reduction method by combining improved modified pole clustering and improved Pade approximations,” *Applied Mathematical Modeling*, vol. 40, pp. 1418-1426, Jan. 2016.

[20] P. Kalaiselvi and V. G. Pratheep, “Analysis of interval system using model order reduction,” *IEEE Proc. International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1-6, Mar. 2015.

[21] G. Parmar, S. Mukher, and R. Prasad, “System reduction using factor division algorithm and eigen spectrum analysis,” *Applied Mathematical Modeling*, vol. 31, pp. 2542-2552, 2007.

[22] A. C Antoulas, D. Sorensen, and S. Gugercin, “A survey of model reduction methods for large-scale systems,” *Contemporary Mathematics*, vol. 280, pp. 193-219, Oct. 2001.

[23] Y. Chahlaoui and P. V. Dooren, “Benchmark Examples for Model Reduction of Linear Time-Invariant Dynamical Systems,” *Springer Proceedings of a Workshop Dimension Reduction of Large-Scale Systems*, vol. 45, pp. 379-392, Oct. 2003.

[24] M. Ghanavati, V. J. Majd, and M. Ghanavati, “Control of inverted pendulum system by using a new robust model predictive Control Strategy,” *IEEE Conference on Control and Communications Systems*, vol. 12, pp. 33-38, Sep. 2011.

[25] S. Jung, H. Cho, and T. C. Hsia, “Neural network control for position tracking of a two-axis inverted pendulum system: experimental studies,” *IEEE Trans Neural Netw.*, vol. 18, no. 4, pp. 1042-1048, Jul. 2007.

[26] L. B. Prasad, B. Tyagi, and H. O. Gupta, "Optimal control of nonlinear inverted pendulum dynamical system with disturbance Input using PID controller and LQR," *IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, vol. 12, pp. 540-545, Nov. 2011.

[27] M. Juneja and S. K. Nagar, "Comparative study of model order reduction using combination of PSO with conventional reduction techniques," *IEEE International Conference on Industrial Instrumentation and Control (ICIC)*, pp. 406-411, May 2015.

[28] S. Sehgal and S. Tiwari, "LQR control for stabilizing triple link Inverted pendulum system," *IEEE International Conference on Power, Control and Embedded Systems*, pp. 1-5, Dec. 2012.

[29] Alok Sinha, *Linear Systems: Optimal and Robust Control*, CRC. Press, 2007.

[30] A. K. Mittal, R. Prasad, and S. P. Sharma, "Reduction of linear dynamic systems using an error minimization technique," *Journal of Institution of Engineers (IE)*, vol. 84, pp. 201-206, Mar. 2004.

[31] S. Mukherjee, Satakshi, and R.C. Mittal, "Model order reduction using response matching technique," *Journal Franklin Institute*, vol. 342, pp. 503-519, Aug. 2005.

[32] R. Prasad and J. Pal, "Stable reduction of linear systems by continued fractions" *Journal of Institution of Engineers (IE)*, vol. 72, pp. 113-116, Oct. 1991.

[33] I. D. Smith and T. N. Lucas, "Least-squares moment matching reduction methods" *Electronics Letters*, vol. 31, no. 11, pp. 929-930, May 1995.

[34] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol.43, no.5, pp.1141 - 1153, Sept. 2013.

[35] H. Xiao, L. Liao, and F. Zhou, "Mobile robot path planning based on Q-ANN," *Proceedings of the IEEE International Conference on Automation and Logistics,* pp. 2650 - 2654, Aug. 2007.

[36] G. Yang, E. Chen, and C. An, "Mobile robot navigation using neural Q-learning," *Proceedings of the IEEE International Conference on Machine Learning and Cybernetics,* vol.1, pp. 48 - 52, Aug. 2004.

[37] R. S. Sutton, and A. Barto, *Reinforcement Learning: An Introduction, Cambridge,* U.K.: Cambridge Univ. Press, 1998.

[38] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[39] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control.* NewYork, NY, USA: Wiley, 2012.

[40] R. S. Sutton, “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming,” *Proc. 7th Int. Conf. Mach. Learn,* pp. 216 - 224, 1990.

[41] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Mach. Learn*, vol. 3, no. 1, pp. 9 - 44, 1988.

[42] D. Liu, and H. Zhang, “A Neural Dynamic Programming approach For Learning Control Of Failure Avoidance Problems,” *International Journal Of Intelligent Control And Systems,* vol. 10, No. 1, 21 - 32, Mar. 2005.

[43] D. Prokhorov, and D. C. Wunsch, “Adaptive critic designs,” *IEEE Transactions on Neural Networks,* vol. 8, pp. 997 - 1007, Sept. 1997.

[44] J. Si, A. G. Barto, and W. B. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming,* New York, NY, USA: Wiley, 2004.

[45] P. J. Werbos, “Approximate dynamic programming for real-time control and neural modeling,” *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches (Chapter 13)*, Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[46] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, “Adaptive critic designs: A case study for neurocontrol,” *Neural Networks*, vol. 8, pp. 1367 - 1372, 1995.

[47] J. Si, and Y. Wang, “Online learning control by association and reinforcement,” *IEEE Trans. Neural Netw.,* vol. 12, no. 2, pp. 264 - 276, Mar. 2001.

[48] H. He, Z. Ni, and J. Fu, “A three-network architecture for on-line learning and optimization based on adaptive dynamic programming,” *Neurocomputing*, vol. 78, no. 1, pp. 3 - 13, Feb. 2012.

[49] X. Fanga, D. Zhenga, H. He, and Z. Nib, “Data-driven heuristic dynamic programming with virtual reality”, vol. 166, pp. 244 - 255, Oct. 2015.

[50] Z. Ni, H. He, D. Zhao, X. Xu, and D. V. Prokhorov, “GrDHP: A General Utility Function Representation for Dual Heuristic Dynamic Programming,” *IEEE transactions on neural networks and learning systems,* vol. 26, no. 3, pp. 614 - 627, mar. 2015.

[51] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Dual heuristic programming excitation neurocontrol for generators in a multimachine power system,” *IEEE Trans. Ind. Appl.,* vol. 39, no. 2, pp. 382 - 394, Mar. 2003.

[52] N. Zhang, and D. C. Wunsch, “A Comparison of Dual Heuristic Programming (DHP) and Neural Network Based Stochastic Optimization Approach on Collective Robotic Search Problem,” *Proceedings Neural Networks of the IEEE*, vol.1, pp. 248 - 253, Jul. 2003.

[53] C. Lian, and X. Xu, “Motion Planning of Wheeled Mobile Robots Based on Heuristic Dynamic Programming,” *IEEE Proceeding of the 11th World Congress on Intelligent Control and Automation Shenyang*, pp. 576 - 580, Jul. 2014.

[54] X. Yang, M. Moallem, and R. V. Patel, “A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation,” *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 35, no. 6, pp. 1214 - 1224, Dec. 2005.

[55] S. Pawlikowski, “Development of a Fuzzy Logic Speed and Steering Control System For an Autonomous Vehicle,” *Master thesis, University of Cincinnati, Department of Mechanical Engineering*, Jan. 1999.

[56] O. Caelen, and G. Bontempi, “Improving the exploration strategy in bandit algorithms,” *International Conference on Learning and Intelligent Optimization,* pp 56 - 68, 2008.

[57] T. Tateyama, S. Kawata, and Y. Shimomura, “Parallel Reinforcement Learning Systems Using Exploration Agents and Dyna-Q Algorithm,” *SICE Annual Conference*, pp. 2774 - 2778, Sep. 2007.

[58] K. Miyazaki, M. Yamamura, and S. Kobayashi, “K certainty exploration method: an action selector to identify the environment in reinforcement learning,” *Artificial Intelligence*, vol. 91, no.l, pp. 155 - 171, l997.

[59] Y. Zhang, and M. Feng, “Application of reinforcement learning based on artificial neural network to robot soccer,” *Journal of Harbin Institute of Technology*, vol.36 ,no.7, pp. 859 - 861, Jul. 2004.

[60] K. Ito, and Y. Imoto, “A study of reinforcement learning with knowledge sharing for distributed autonomous system,” *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 3, pp. 1120 - 1125, Jul. 2003.

[61] C. Samson, “Velocity and torque feedback control of a nonholonomic cart,” *Advanced Robot Control*, pp. 125 - 151, Jan. 1991.

[62] S. Mitra, and Y. Hayashi, “Neuro-fuzzy rule generation: survey in soft computing framework,” *IEEE Trans. Neural Networks*, vol. 11, no. 3, pp. 748 - 768, Aug. 2002.

[63] Z. Ni, H. He, X. Zhong, and D. Prokhorov, “Model-Free Dual Heuristic Dynamic Programming,” *IEEE Trans. Neural Networks*, vol. 26, no. 8, pp. 1834 - 1839, Aug. 2015.

[64] S. Al-Dabooni, and D. Wunsch, “Heuristic Dynamic Programming for Mobile Robot Path Planning Based on Dyna Approach,” *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 3723 - 3730, Jul. 2016.

[65] M. Fairbank, and E. Alonso, “Value-Gradient Learning,” *IEEE International Joint Conference on Neural Networks (IJCNN)*), pp. 1 - 8, Jun. 2012.

[66] R. Fierro, and F. L. Lewis, "Control of a Nonholonomic Mobile Robot Using Neural Networks," *IEEE Trans. Neural Networks*, vol. 9, no. 4, pp. 589 - 600, Jul. 1998.

[67] W. S. Lin, L. H. Chang, and P. C. Yang, "Adaptive critic anti-slip control of wheeled autonomous robot," *IET Control Theory and Applications*, vol. 1 , no. 1, pp. 51 - 57, Jan. 2007.

[68] T. Dierks, and S. Jagannathan, "Neural Network Output Feedback Control of Robot Formations," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 40, no.2, pp. 383 - 399, Apr. 2010.

[69] R. Kelly, J. Llamas, and R. Campa, "A measurement procedure for viscous and coulomb friction," *IEEE Trans. on Instrumentation and Measurement*, vol.49, no. 4, pp. 857 - 861, Aug. 2000.

[70] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550 - 1560, 1990

[71] Y. Liu, Y. Lin, S. Wu, C. Chuang, and C. Lin, "Brain Dynamics in Predicting Driving Fatigue Using a Recurrent Self-Evolving Fuzzy Neural Network," *IEEE Trans. Neural Network,* vol. 27, no. 2, pp. 347 - 360, Feb. 2016.

[72] X. Luo, Y. Lv, R. Li, and Y. Chen, "Web Service QoS Prediction Based on Adaptive Dynamic Programming Using Fuzzy Neural Networks for Cloud Services," *IEEE Access, Special Section on Challenges for Smart Worlds,* vol. 3, pp. 2260 - 2269, Nov. 2015.

[73] F. Lin, P. Chou, C. Chen, and Y. Lin, "DSP-Based Cross-Coupled Synchronous Control for Dual Linear Motors via Intelligent Complementary Sliding Mode Control," *IEEE Trans.Industrial Electronics,* vol. 59, no. 2, pp. 1061 - 1073, Feb. 2012.

[74] A. K. Palit, G. Doeding, W. Anheier, and D. Popovic, "Backpropagation based training algorithm for Takagi-Sugeno type MIMO neuro-fuzzy network to forecast electrical load time series," *Proc. of Int. Conf. on FUZZ-IEEE*, vol. 1, pp. 86 - 91, May 2002.

[75] T. Dierks, and S. Jagannathan, "Online Optimal Control of Affine Nonlinear Discrete-Time Systems With Unknown Internal Dynamics by Using Time-Based Policy Update," *IEEE Trans. Neural Network*, vol. 23 , no. 7 , pp. 1118 - 1129, June 2012.

[76] M. Fairbank, D. Prokhorov, and E. Alonso, "Approximating Optimal Control with Value Gradient Learning," *Chapter 7 in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, New York, NY, USA: John Wiley and Sons, pp. 142-161, Jan. 2013.

[77] S. Al-Dabooni, and D. Wunsch, "Mobile Robot Control Based on Hybrid Neuro-Fuzzy Value Gradient Reinforcement Learning," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 2820-2827, May 2017.

[78] X. Bai, D. Zhao, and J. Yi. "ADHDP (λ) strategies based coordinated ramps metering with queuing consideration," *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 22-27, May 2009.

[79] T. Li, D. Zhao, and J. Yi. "Heuristic dynamic programming strategy with eligibility traces," *IEEE American Control Conference*, pp. 4535-4540, Jun. 2008.

[80] X. Bai, D. Zhao, and J. Yi, "Ramp Metering Based on on-line ADHDP(λ) controller," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1847-1852, Jul. 2008.

[81] X. Bai, D. Zhao, and J. Yi, "Coordinated multiple ramps metering based on neuro-fuzzy adaptive dynamic programming," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 241-248, Jul. 2009.

[82] H. Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, "True On-line Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 145, no. 17, pp. 1-40, Jan. 2016.

[83] R. S. Sutton, A. R. Mahmood, and M. White, "An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 73, no. 17, pp. 1-29, Jan. 2016.

[84] H. Seijen, and R. S. Sutton, "True Online TD(λ)," *Proceedings of the* 31^{st} *International Conference on Machine Learning*, pp. 692-700, Jan. 2014.

[85] Y. Sokolov, R. Kozma, L. D. Werbos, and P. J. Werbos, "Complete stability analysis of a heuristic approximate dynamic programming control design," *Automatica*, vol. 59, pp. 9-18, Sep. 2015.

[86] F. Liu, J. Sun, J. Si, W. Guo, and S. Mei, "A boundedness result for the direct heuristic dynamic programming," *Neural Networks*, vol. 32, pp. 229-235, Aug. 2012.

[87] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219-245, Jan. 2000.

[88] M. Fairbank, "Reinforcement learning by value gradients," *eprintarXiv:0803.3539*, Mar. 2008.

[89] Y. Zhu, D. Zhao , and D. Liu, "Convergence analysis and application of fuzzy-HDP for nonlinear discrete-time HJB systems," *Neurocomputing*, vol. 149, pp. 124-131, Feb. 2015.

[90] C. Yang, Y. Jiang, Z. Li, W. He, and C. Su, "Neural control of bimanual robots with guaranteed global stability and motion precision," *IEEE Trans. Industrial Informatics*, vol. 13, no. 3, pp. 1162-1171, Jun. 2017.

[91] C. Yang, X. Wang, L. Cheng, and H. Ma, "Neural-learning-based telerobot control with guaranteed performance," *IEEE Trans. Cybernetics*, vol. PP, no. 99, pp. 1-12, Jun. 2016.

[92] G. Zhang, M. Y. Hu, B. E. Patuwo, and D. C. Indro, "Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis," *European Journal of Operational Research*, vol. 116, no. 1, pp. 16-32, Jul. 1999.

[93] D. Liu, and D. Wang, "Optimal Control of Unknown Nonlinear Discrete-Time Systems Using the Iterative Globalized Dual Heuristic Programming Algorithm," *Chapter 3 in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, New York, NY, USA: John Wiley and Sons, pp. 52-77, Jan. 2013.

[94] C. Chen, D. Dong, H. Li, J. Chu, and T. Tarn, "Fidelity-Based Probabilistic Q-Learning for Control of Quantum Systems," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 5, no. 10, pp. 920-933, May 2014.

[95] P. J. Werbos, and X. Pang, "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot," *IEEE Proc. Conf. Systems, Man and Cybernetics (SMC)* , pp. 1764-1769, Oct. 1996.

[96] D. Wunsch, "The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution," *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 79-82, Jul. 2000.

[97] R. Ilin, R. Kozma, and P. J. Werbos,"Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 929-937, Jun. 2008.

[98] N. Zheng, and P. Mazumder, "Hardware-Friendly Actor-Critic Reinforcement Learning Through Modulation of Spike-Timing-Dependent Plasticity," *IEEE Trans. on Computers*, vol. 66, no. 2, pp. 299-311, Feb. 2017.

[99] B. Luo, D. Liu, T Huang, and D. Wang, "Model-Free Optimal Tracking Control via Critic-Only Q-Learning," *IEEE Trans. Neural Netw. and Learn Syst.*, vol. 27, no. 5, pp. 2134-2144, Oct. 2016.

[100] The National Council of Teachers of Mathematics (NCTM): https://illuminations.nctm.org

[101] X. Zhong, Z. Ni, and H. He, "A Theoretical Foundation of Goal Representation Heuristic Dynamic Programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2513 - 2525, Dec. 2016.

[102] V. Mnih, A. P. Badia , M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International Conference on Machine Learning,* pp. 1928-1937, Jul. 2016.

[103] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. KÃijttler, J. Agapiou, J. Schrittwieser, and J. Quan, "StarCraft II: A New Challenge for Reinforcement Learning," arXiv preprint arXiv:1708.04782. Aug. 2017.

[104] S. Al-Dabooni, and D. Wunsch, “The Boundedness Conditions for Model-Free HDP(λ),” Under reviewing for *IEEE Trans. Neural Netw. and Learn Syst.*

[105] Y. Tang, H. He, Z. Ni, X. Zhong, D. Zhao, and X. Xu, “Fuzzy-Based Goal Representation Adaptive Dynamic Programming,” *IEEE Trans. on Fuzzy Sys.*, vol. 24, no. 5, pp. 1156 - 1176, Oct. 2016.

[106] F. Y. Wang, H. Zhang, and D. Liu, “Adaptive dynamic programming: An introduction,” IEEE Computational Intelligence Magazine, vol. 4, no. 2, pp. 39-47, May 2009.

[107] H. He, Self-Adaptive Systems for *Machine Intelligence*. New York, NY, USA: Wiley, 2011.

[108] S. P. Singh, and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” *Machine Learning*, vol. 22, no. 1, pp. 123 - 158, Mar. 1996.

[109] Y. Xiao, M. Wasei, P. Hu, P. Wieringa, and F. Dexter, “Dynamic Management of Perioperative Processes: A Modeling and Visualization Paradigm,” *International Federation of Automatic Control (IFAC)*, vol. 39, no. 3, pp. 647 - 652, Jan. 2006.

[110] K. L. Keller, and R. Staelin, “Effects of Quality and Quantity of Information on Decision Effectiveness,” *Journal of Consumer Research*, vol. 14, no. 2, pp. 200 - 213, Sep. 1987.

[111] D. P. Bertsekas, “Approximate policy iteration: A survey and some new methods,” *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310 - 335, Aug. 2011.

[112] F. L. Lewis, and D. Vrabie, “Reinforcement learning and adaptive dynamic programming for feedback control,” *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32 - 50, Aug. 2009.

[113] R. Padhi, N. Unnikrishnan, X. Wang, and S. N. Balakrishnan, “A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems,” *Neural Netw.*, vol. 19, no. 10, pp. 1648 - 1660, Dec. 2006.

[114] J. Fu, H. He, and X. Zhou, “Adaptive learning and control for MIMO system based on adaptive dynamic programming,” *IEEE Trans. Neural Netw.*, vol. 22, no. 7, pp. 1133 - 1148, Jul. 2011.

[115] Z. Ni, H. He, and J. Wen, “Adaptive learning in tracking control based on the dual critic network design,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 6, pp. 913 - 928, Jun. 2013.

[116] F. L. Lewis, and D. Liu, Eds., *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. Hoboken, NJ, USA: Wiley, 2013.

[117] Q. Wei, and D. Liu, “Adaptive dynamic programming for optimal tracking control of unknown nonlinear systems with application to coal gasification,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 4, pp. 1020 - 1036, Oct. 2014.

[118] S. Ray, G. K. Venayagamoorthy, B. Chaudhuri, and R. Majumder, “Comparison of adaptive critic-based and classical wide-area controllers for power systems,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 4, pp. 1002 - 1007, Aug. 2008.

[119] H. Zhang, Y. Luo, and D. Liu, “Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints,” *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490 - 1503, Sep. 2009.

[120] B. Xu, C. Yang, and Z. Shi, “Reinforcement learning output feedback NN control using deterministic learning technique,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 635 - 641, Mar. 2014.

[121] Q. Wei, and D. Liu, “Data-driven neuro-optimal temperature control of waterâĂŞgas shift reaction using stable iterative adaptive dynamic programming,” *IEEE Trans. Ind. Electron.*, vol. 61, no. 11, pp. 6399 - 6408, Nov. 2014.

[122] D. Liu, D. Wang, F.-Y. Wang, H. Li, and X. Yang, “Neural-network based online HJB solution for optimal robust guaranteed cost control of continuous-time uncertain nonlinear systems,” *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2834 - 2847, Dec. 2014.

[123] D. Liu, and Q. Wei, “Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 3, pp. 621 - 634, Mar. 2014.

[124] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, “Discrete-time nonlinear hjb solution using approximate dynamic programming: convergence proof,” *IEEE Trans. on Sys., Man, and Cyb., Part B*, vol. 38, no. 4, pp. 943 - 949, Aug. 2008.

[125] D. Liu, and Q. Wei, “Finite-approximation-error-based optimal control approach for discrete-time nonlinear systems,” *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 779 - 789, Apr. 2013.

[126] D. Liu, D. Wang, D. Zhao, Q. Wei, and N. Jin, “Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming,” *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 3, pp. 628 - 634, Jul. 2012.

[127] F. L. Lewis, and D. Liu, Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, Chapter 7, John Wiley and Sons, Jan. 2013.

[128] S. Al-Dabooni, and D. Wunsch, “Online Model-Free N-Step HDP with Stability Analysis,” Under Preparing.

[129] D. P. Bertsekas, “Approximate policy iteration: A survey and some new methods,” *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310 - 335, Aug. 2011.

[130] Y. Yang, D. Wunsch, and Y. Yin, “Hamiltonian-driven adaptive dynamic programming for continuous nonlinear dynamical systems,” *IEEE Trans. Neural Netw. Learn. Syst.* vol. PP, no. 12, pp. 1 - 12, Feb. 2017 (to be published).

[131] M. Fairbank, E. Alonso, and D. Prokhorov, “An Equivalence Between Adaptive Dynamic Programming With a Critic and Backpropagation Through Time,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 12, pp. 2088 - 2100, Dec. 2013.

[132] S. Al-Dabooni, and D. Wunsch, “An Improved N-Step Value Gradient Learning Adaptive Dynamic Programming Algorithm for Online Learning, with Convergence Proof and Case Studies”, Under review for *IEEE Trans. Neural Netw. Learn. Syst.*

[133] D. Zhao, Y. Zhu, and H. He, “Neural and fuzzy dynamic programming for under-actuated systems,” *IEEE/INNS, International Joint Conference on Neural Networks (IJCNN)*, pp. 1 - 7, Jun. 2012.

[134] S. Mohagheghi, G. K. Venayagamoorthy, and R. G. Harley, “Fully evolvable optimal neurofuzzy controller using adaptive critic designs,” *IEEE Trans. on fuzzy systems, vol. 16, no. 6,* pp 1450 - 1461, Dec. 2008.

[135] S. Mohagheghi, G. K. Venayagamoorthy, and R. G. Harley, “Adaptive critic design based neuro-fuzzy controller for a static compensator in a multimachine power system,” *IEEE Trans. on Power Systems,* vol. 21, no. 4, pp 1450 - 1461, Nov. 2006.

[136] Y. Zhu, D. Zhao, and H. He, “Integration of fuzzy controller with adaptive dynamic programming,” *IEEE conference World Congress In Intelligent Control and Automation (WCICA),* pp. 310 - 315, Jul.2012.

[137] Y. Tang, C. Mu, and H. He, “SMES-based damping controller design using fuzzy-GrHDP considering transmission delay,” *IEEE Trans. on Applied Superconductivity,* vol. 26, no. 7, pp 1 - 6, Oct. 2016.

[138] H. Zhang, J. Zhang, G. H. Yang, and Y. Luo, “Leader-based optimal coordination control for the consensus problem of multiagent differential games via fuzzy adaptive dynamic programming,” *IEEE Trans. on Fuzzy Systems,* vol. 23, no. 1, pp 152 - 163, Feb. 2015.

[139] J. L. Zhang, H G. zhang, L. U. Yan-Hong, and H. J. liang, “Nearly optimal control scheme using adaptive dynamic programming based on generalized fuzzy hyperbolic model,” *Acta Automatica Sinica,* vol. 39, no. 2, pp 142 - 1148 Feb. 2013.

[140] C. H. Lee, and C. C. Teng, “Identification and control of dynamic systems using recurrent fuzzy neural networks,” *IEEE Trans. on fuzzy systems*, vol. 8, no. 4, pp. 349 - 366, Aug. 2000.

[141] J.-S. Jang, C.-T. Sun, and E. Mizutani, *Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence* Upper Saddle River, NJ: Prentice-Hall, 1997.

[142] M. Sugeno, G. Kang, “Structure identification of fuzzy model,” *Fuzzy Sets Syst*, vol. 28, pp.15âĂŞ33, 1988.

[143] J.-S. Jang, “ANFIS: adaptive-network-based fuzzy inference system,” *IEEE trans. on systems, man, and cybernetics,* vol. 23, no. 3, pp. 665-685, May 1993.

[144] J. W. Yeh, and S. F. Su, “Efficient approach for RLS type learning in TSK neural fuzzy systems,” *IEEE trans. on cybernetics,* vol. 47, no. 9, pp. 2343-2352, Sep. 2017.

[145] J.-S. Wang, and C.-G. Lee, “Self-adaptive neuro-fuzzy inference systems for classification applications” *IEEE Trans. on Fuzzy Sys,* vol. 10, no. 2, pp. 790-802, Dec. 2002.

VITA

Seaar Al-Dabooni received his B.S. degrees in Computer Engineering from Basrah University, Basrah, Iraq, in 2004 with rank (1) out of (79) students. He received his M.S. degrees in Computer Engineering from Basrah University, Basrah, Iraq, in 2009 with rank (1) out of (9) students. He is a member of the Applied Computational Intelligence Laboratory (ACIL) directed by Dr. Wunsch. During his Ph.D., he completed several project that related with Model Order Reduction Reinforcement Learning, Adaptive Dynamic Control and Robotic Systems. In May 2018, he received his Ph.D. in Computer Engineering from the Missouri University of Science and Technology, Rolla, Missouri, USA.