Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Reinforcement-Learning-based Magneto-hydrodynamic Control of Hypersonic Flows

Nilesh V. Kulkarni
QSS Group, Inc., NASA Ames Research Center, Moffett Field, CA-94035

Minh Q. Phan
Thayer School of Engineering, Dartmouth College, Hanover, NH – 03755

Abstract*-In this work, we design a policy-iteration-based Q-learning approach for on-line optimal control of ionized hypersonic flow at the inlet of a scramjet engine. Magneto-hydrodynamics (MHD) has been recently proposed as a means for flow control in various aerospace problems. This mechanism corresponds to applying external magnetic fields to ionized flows towards achieving desired flow behavior. The applications range from external flow control for producing forces and moments on the air-vehicle to internal flow control designs, which compress and extract electrical energy from the flow. The current work looks at the later problem of internal flow control. The baseline controller and Q-function parameterizations are derived from an off-line mixed predictive-control and dynamic-programming-based design. The nominal optimal neural network Q-function and controller are updated on-line to handle modeling errors in the off-line design. The on-line implementation investigates key concerns regarding the conservativeness of the update methods. Value-iteration-based update methods have been shown to converge in a probabilistic sense. However, simulations results illustrate that realistic implementations of these methods face significant training difficulties, often failing in learning the optimal controller on-line. The present approach, therefore, uses a policy-iteration-based update, which has time-based convergence guarantees. Given the special finite-horizon nature of the problem, three novel on-line update algorithms are proposed. These algorithms incorporate different mix of concepts, which include bootstrapping, and forward and backward dynamic programming update rules. Simulation results illustrate success of the proposed update algorithms in re-optimizing the performance of the MHD generator during system operation.

I. Introduction

In recent years, possible application of magneto-hydrodynamics (MHD) in high hypersonic systems has generated a lot of excitement [1-6]. Using MHD as an integral part of these systems has been suggested for various applications. Magneto-hydrodynamics refers to the study of ionized flows [7]. The dynamics of these flows is, therefore, governed by fluid equations coupled with the electromagnetic equations. For high Mach numbers, the system has enough kinetic energy so that relatively small electromagnetic effects can lead to big overall effects that can be used for engineering benefits. Using external electromagnetic actuators provides control authority over MHD systems, and their implementation can be treated using control theory.

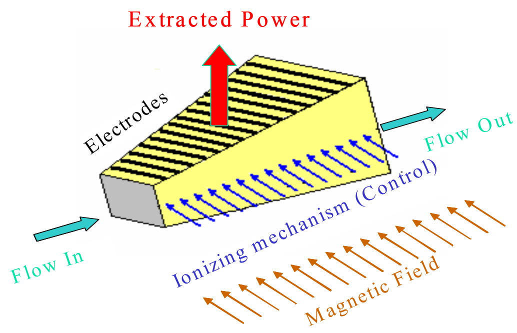

Fig. 1. MHD channel at the inlet of a scramjet engine (After Ref. [8])

Figure 1 illustrates an MHD channel at the inlet of a scramjet engine of a hypersonic vehicle. The current air-breathing propulsion technology becomes inapplicable beyond a certain Mach number. This device introduces an innovative concept that can make hypersonic flight possible with an air-breathing scramjet engine. As the air enters the inlet of the engine, it is ionized using a chosen ionizing mechanism. An external magnetic field is also applied perpendicular to the flow direction. The ionized air flowing across the magnetic field gives rise to an electromotive force (*e.m.f*) in a direction perpendicular to both the air flow and the direction of the applied magnetic field. By placing electrodes across the channel, electrical energy can be collected. The MHD device, thus, acts as a power generator. Part of the kinetic energy of the flow is converted into electrical energy. As a result, the flow slows down to low supersonic Mach numbers at which scramjet combustion is possible. Due to the low static temperature of these hypersonic flows, the applications rely on an external source of ionization. High-energy electron beams have been shown to be the most efficient ionizers [8-9] The electron beam current profile can be varied along the hypersonic channel to optimize the performance of the MHD device. Electron beam current can, therefore, be treated as a control input variable for the MHD system, and the resultant optimization problem can be handled with optimal control theory.

Controlling the flow through the inlet in an optimal manner is a critical element of the design for optimizing the performance of the engine and, thereby, of the overall vehicle. From an optimal control standpoint, this represents a challenging application for several reasons. The dynamics of the flow with the electro-magnetic interaction are highly non-linear in nature. Further, there exist significant uncertainties in the empirical modeling of these flows. A conservative design approach, therefore, corresponds to using all the available knowledge of the system in designing the off-line optimal controller. This controller can then be updated on-line using the observed system behavior for re-optimizing the system performance. Reference [10] describes the off-line optimal control design of this MHD device, which uses a mixed predictive control and dynamic programming approach. Various formulations of the on-line update algorithm are possible depending on whether value functions or Q-functions are used, and whether the derivative forms of these update rules are used. These have been categorized as heuristic dynamic programming (HDP), action-dependent heuristic dynamic programming (ADHDP), dual heuristic programming (DHP), and action-dependent dual heuristic programming (ADDHP) in the adaptive critic literature [11-12]. Three forms of these, viz. HDP, DHP, and ADDHP, need the system model in their update laws. References [13-16} present different implementations of these architectures. Given that the motivation of the on-line update algorithm is in enhancing the performance for modeling errors in the off-line design, the model-free ADHDP or the Q-function-based formulation is used in this application.

The second critical issue in the choice of the update method is the use of policy iteration or value iteration. Reference [17] has shown the convergence of value-iteration-based updates for all 4 updates discussed above. However, these convergence results are probabilistic in nature. In practice, value-iteration-based updates are often seen to fail in learning the optimal functional form of the controller. Contrarily, policy-iteration-based updates, though being slower, represent a much conservative design. References [18-19] have illustrated time-based convergence of the policy-iteration-based Q-learning update to the true optimal gains of the linear quadratic regulator. Given that conservativeness of the update is more important than speed in a critical engineering system, this application uses the policy-iteration-based Q-learning paradigm.

The rest of the paper is organized as follows. Section II discusses the details of the modeling of this MHD system. Section III defines the performance measure for the optimal control problem. Section IV outlines the off-line optimal control design, and extraction of the functional forms necessary for the on-line design. Section V presents the on-line optimal control design. Given the finite horizon nature of the problem, the on-line implementation illustrates three novel update algorithms. Section VI presents the simulation results with the different update algorithms. Finally section VII provides the conclusions of this study.

II. Modeling the MHD Power Generator

The detailed analysis of the MHD system would typically consist of solving 3-dimensional, time-dependent MHD equations with the electron beam current as an input to this system. For the present work, we focus on steady state behavior with dependence on the *x*-coordinate alone, along the length of the channel. The system of partial differential equations thereby reduces to a system of ordinary differential equations with the position along the length of the channel as the independent coordinate. In a supersonic flow perturbations or inputs given to the flow are only felt downstream of the flow. The *x*-coordinate along the flow therefore behaves like the time-coordinate in the sense that any event occurring at time *t* affects the system only for time greater than *t*. The *x*-coordinate can therefore be thought of as being equivalent to the time coordinate when we look at the simplified ordinary differential equation system corresponding to the steady state one-dimensional flow. In terms of this *x*-*t* equivalence, we can now look at the performance optimization of the MHD generator as a standard optimal control problem with the independent variable *x*.

The system dynamics can be describe as:

$$\frac{d\mathbf{w}}{dx} = \mathbf{f}[\mathbf{w}(x), \mathbf{u}(x), x] \tag{1}$$

Here $\mathbf{u}(x)$ corresponds to the control which in this case is the electron beam current, and x corresponds to the position variable. $\mathbf{w}(x)$ corresponds to the state vector for the system given as:

$$\mathbf{w}(x) = \begin{bmatrix} \rho_f & v_f & P_f & n_{ef} \end{bmatrix}^T \tag{2}$$

ρ_f (kg/m^3) is the density of the flow, v_f (m/s) is the velocity of the fluid, P_f (N/m^2) is the static pressure of the fluid, and n_{ef} (1/m^3) is the electron number density along the channel. Reference [20] describes in detail the geometry and the governing dynamics of the flow along the MHD channel.

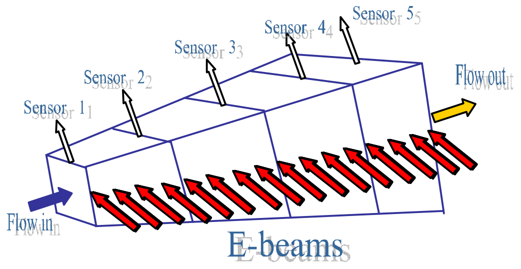

Fig. 2. Sensor and actuator placement of the MHD channel

Figure 2 illustrates the implementation outline for the MHD channel. It assumes 5 equally spaced sensors along the channel, with the first one at the inlet, and the last at the channel exit. These sensors divide the MHD generator into 4 sections. The generator has 4 controllers corresponding to the first 4 sensors. Each controller uses the state information at the corresponding sensor location to compute the e-beam profile from that sensor location to the next sensor location. The sensor at the exit is used to provide the exit flow variables for evaluating the exit cost function terms such as exit Mach number, exit pressure, exit temperature etc.

III. Performance measure definition

Before discussing the details of the control architecture it is important to outline the performance measure for this system, which optimizes the e-beam current along the channel. Some of the considerations for this definition are:

1) To minimize deviations from prescribed flow conditions at the end of the channel. After going through the MHD channel, the flow enters the combustion chamber of the engine. There are therefore certain prescribed values of flow variables such as temperature, Mach number that need to be attained at the end of the channel.
2) To maximize the net energy extracted from the system that corresponds to the difference between the energy extracted from the flow and the energy spent on the e-beam ionization.
3) To minimize the net usage of the e-beam current.

There are several other beneficial flow characteristics that can be incorporated in the cost function such as penalizing adverse pressure profile or minimizing the net entropy rise in the channel. These are given in more detail in Ref. [21]. Posing the optimization as a minimization problem, one candidate cost function that implements the requirements enumerated above is given as:

$$J = p_1\left[M_f(x_f) - M_{fe}\right]^2 + \int_0^{x_f}\left\{\frac{q_1}{\rho_f v_f A_f}\left[Q_{\beta f}A_f - k_f(1-k_f)\sigma_f v_f^2 B_f^2 A_f\right] + r_1 j_{bf}^2\right\}dx \tag{3}$$

$M_f(x_f)$ is the flow Mach number at the channel exit and M_{fe} is the prescribed exit Mach number. $Q_{\beta f}$ is the energy deposited by the electron beams (J/m^3). A_f is the cross-sectional area of the channel (m^2). k_f is the load factor, which is a measure of the extracted energy converted to electrical energy versus heat. σ_f represents the conductivity of the fluid [1/(Ohm*m)]). B_f is the externally applied magnetic field (Tesla). j_{bf} is the electron beam current (Amperes). p_1, q_1, and r_1 are the weighting elements of the individual terms of the cost function. The terms on the first line correspond to the end position cost, minimizing which ensures the exit Mach number to be close to its prescribed value. The term on the second line corresponds to the incremental cost function. The first element of the integrand maximizes the net energy extracted from the system. The second term in the integrand penalizes excessive use of the electron beam current. An appropriate choice of the weighting elements sets the relative importance of the different terms in the cost function.

IV. Q-FUNCTION AND CONTROLLER INITIALIZATION FROM THE OFF-LINE DESIGN

The structure of the sensor and actuator placement, as outlined in Figure 2, motivated a mixed predictive control and dynamic-programming-based off-line optimal control design. Reference [22] describes the general nature of this parameterized predictive control design. The system states between sensor locations are predicted using trained neural network models. These are used in a dynamic-programming-based architecture, which designs the 4 controllers, starting with the last one first and moving upstream till the inlet controller. The control design uses two groups of neural networks. The first group is used to model the controller, and the second group is used to model the cost-to-go function.

A. Neural Network Controller

The optimal controller corresponds to a feedback function of the sensed state, $\mathbf{w}(x)$. There are only a discrete number of sensors in the channel. Based on the sensed state at each sensor location, the optimal controller, therefore, provides the control inputs from that sensor location to the next sensor location. The controller design is correspondingly broken down into the design of 4 controllers, one each for the first 4 sensors. Each of the controllers is parameterized using neural networks.

The actuators for the MHD channel correspond to individual electron beam windows that are assumed to be placed continuously along the length of the channel. Each of these windows can generate electron beams with different current setting. The total number of outputs of each neural network controller can, therefore, equal the number of electron beam windows between two sensors. For the current geometry, the width of the individual e-beam window is 0.5 cm. For the assumed channel length of 3 meters, this corresponds to 600 electron beams. To simplify the control approach, the windows are grouped so that each group of 6 windows produces e-beams with the same current setting. The control inputs to the flow, therefore, correspond to 100 e-beam current values. The 4 neural network controllers are, therefore, responsible for producing 25 values of electron beam current each between two sensor locations.

The geometry of the MHD channel in the actual implementation can differ from the one assumed in the current study. If the channel is longer, then more electron beam windows would need to be grouped together that produce the same output current. This, however, can reduce the resolution of the control action on the flow. One way to make the controller design independent of the channel geometry is through the use of basis functions. The control inputs between two sensor locations can be given as:

$$\mathbf{u}(\mathbf{w}_{sensor}, i) = \sum_{j=1}^{N}\alpha_j(\mathbf{w}_{sensor})\phi_j(i),\ \ i = 1, \dots, r \tag{4}$$

Equation (4) parameterizes the control, **u**, in terms of basis functions ϕ_j, j = 1,…, N. r is the total number of resulting control values. The coefficients of the basis functions, α_j, are now expressed as functions of the sensed flow variables at the corresponding sensor location. The choice and the number of basis functions that need to be used is dependant on how nonlinear the true optimal control trajectory is, and what functional form it takes. However, this is not known apriori. The standard approach is to use trial and error to choose the functional form, as well as the number of basis functions. Several basis functions were analyzed for the current MHD implementation. Gaussian functions were chosen as they gave us the best results.

In terms of a neural network, this controller structure can be implemented by designing the neural network controller with 3 layers. For a sensed state, $\mathbf{w}_{\text{sensor}}$, the first two layers, a sigmoid and a linear layer now output the values of the coefficients α_j. The third layer is chosen as a linear layer, with the connection weights between the second and third layer given by,

$$w_{ij} = \phi_j(i) \quad , \ i = 1 , \ldots , r; \ j \ = \ 1 , \ldots , N \tag{5}$$

The size of the third layer therefore equals the total number of control values given by the controller, *r*. While training the neural network controller, these weights are kept fixed. Only the weights corresponding to the first two layers are updated.

B. Neural Network Cost-to-go Function Estimator

The cost-to-go function, $V\left[\mathbf{w}(x_c), \bar{\mathbf{u}}(x_c, x_f), x_c\right]$, is defined as:

$$V\left[\mathbf{w}(x_c), \bar{\mathbf{u}}(x_c, x_f), x_c\right] = p_1\left[M_f(x_f) - M_{fe}\right]^2 + \int_{x_c}^{x_f}\left[\frac{q_1}{\rho_f v_f A_f}\left[Q_{\beta f}A_f - k_f(1-k_f)\sigma_f {v_f}^2 {B_f}^2 A_f\right] + r_1 {j_{bf}}^2\right]dx \tag{6}$$

$\bar{\mathbf{u}}(x_c, x_f)$ denotes the control profile from the position x_c to the end-position x_f. Contrary to the cost function, *J* [Eq. (18-19)], $V\left[\mathbf{w}(x_c), \bar{\mathbf{u}}(x_c, x_f), x_c\right]$ is defined for every state $\mathbf{w}(x)$, at all positions. Minimizing the cost-to-go function, therefore, provides a feedback controller, which gives the optimal control profile as a function of the system state $\mathbf{w}(x)$. Given that the access to the state is available only at the five sensor locations, the cost-to-go function networks are designed for the first four sensor locations.

Optimizing the controller in the off-line design provides the optimal cost-to-go function from each sensor location to the end of the channel, $V^*\left[\mathbf{w}_{sensor}(i)\right]$. This is now used to compute the optimal off-line Q-function. This optimal off-line Q-function, $Q^*\left[\mathbf{w}_{sensor}(i), \boldsymbol{\alpha}(\mathbf{w}_{sensor}, i)\right]$, for any given state-control pair, $\left[\mathbf{w}_{sensor}(i), \boldsymbol{\alpha}(\mathbf{w}_{sensor}, i)\right]$, is given as:

$$Q^*\left[\mathbf{w}_{sensor}(i), \boldsymbol{\alpha}(\mathbf{w}_{sensor}, i)\right] = U\left[\mathbf{w}_{sensor}(i), \boldsymbol{\alpha}(\mathbf{w}_{sensor}, i)\right] + V^*\left[\mathbf{w}_{sensor}(i+1)\right] \tag{7}$$

Here the control variable corresponds to the vector of coefficients, which defines the control between the i^{th} sensor and (i+1) $^{\text{th}}$ sensor location. A two-layer network is now trained to be the optimal off-line Q-function network for each sensor location. The inputs of this network are the state-control pair $\left[\mathbf{w}_{sensor}(i), \boldsymbol{\alpha}(\mathbf{w}_{sensor}, i)\right]$. For supervising the training, the desired value is computed using Eq. (7). The utility function, $U\left[\mathbf{w}_{sensor}(i), \boldsymbol{\alpha}(\mathbf{w}_{sensor}, i)\right]$, is computed using the system models. Figure 3 illustrates the Q-function network for a given sensor location *i*.

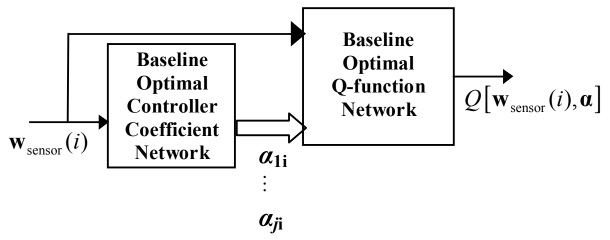
Baseline Optimal Controller Coefficient Network
Baseline Optimal Q-function Network
$\mathbf{w}_{\text{sensor}}(i)$
α_{1i}
$\vdots$
α_{ji}
$Q\left[\mathbf{w}_{\text{sensor}}(i), \boldsymbol{\alpha}\right]$

Fig. 3. Q-function and controller parameterization for a sensor *i*.

The on-line optimal control design corresponds to providing update equations for the 4 Q-functions and the 4 controllers. Given the nature of the problem, the design of update equations for these Q-functions and controllers presents interesting possibilities. The early literature treated reinforcement learning problems as episodic in nature. The solution procedure starts at a randomly picked state in the state-space, and takes decisions till it reaches the stipulated end of the decision making process. This corresponds to an episode, where the solution procedure learns costs associated with decisions taken in that episode. The knowledge gathered during the episode is used to update the estimates of the value or the Q-functions, and the solution procedure is initialized with a new starting state. Episodic updates are, therefore, off-line in nature since the experiment can be repeated. The on-line forward dynamic programming algorithms with infinite decision steps are, instead, inherently non-episodic in nature. The finite decision-step on-line MHD performance optimization problem can, however, be cast as episodic in nature.

The actual physical system is governed by equations that have both time and space dependence. At hypersonic Mach numbers, the response time for the flow to go through the channel is of the order of milliseconds, while, depending on the flight trajectory, the channel inlet conditions vary on the order of seconds. It is, therefore, assumed that the flow reaches steady-state for a given inlet condition and the applied control, before the inlet condition changes. With this assumption, the problem of optimization becomes episodic in nature. Two different time indices are considered: k_T, on the order of seconds, and k_t, on the order of milliseconds. At every k_T, the channel sees a starting state based on the flow conditions at the channel inlet.Correspondingly, at k_t = 1, the controller 1 gives the control profile from the channel-inlet to the location of sensor-2. At k_t = 2, sensor-2 observes the flow steady-state conditions at its location, and gives the control profile upto the location of sensor-3. Similarly at time indices, k_t = 3 and k_t = 4, controllers 3 and 4 give the control inputs in the subsequent sections of the channel. At k_t = 5, the final sensor senses the state variables at the channel exit. These control profiles are held constant till the next time index k_T, when the inlet sees new flow conditions, and the index k_t is re-initialized to 1. The time index k_T, therefore, corresponds to an episode. Consequently, the current problem, as an exception, is on-line episodic in nature due to these inner and the outer loops defined by the time indices k_T and k_t.

V. Policy-Iteration-Based Q-Learning Update Equations for the MHD Channel

In the following discussion, 3 different implementations of the Q-learning algorithm are presented. The first implementation evaluates all 4 Q-functions and controllers at every time index k_T. In this case, after sensing the flow state and giving control inputs, each of the 4 Q-functions are updated using the observed performance. This policy evaluation is carried out for a stipulated number of episodes, after which, the policy is improved by updating all the 4 controllers. This policy evaluation and improvement is implemented till all 4 sensor Q-functions and controllers get optimized. The second implementation presents a one-sensor-at-a-time update algorithm. In this case, the update follows the classical backward dynamic programming method where the Q-function for sensor-4 is first evaluated and then the controller-4 for sensor-4 is optimized. This is followed by the evaluation and improvement of the sensor-3 Q-function and controller, then the sensor-2 Q-function and controller, and finally the sensor-1 Q-function and controller. The third implementation presents a mixed Monte Carlo-bootstrapped update algorithm for evaluating and improving the Q-functions and the controllers. The backward one-sensor-at-a-time implementation in the second case uses only the observed performance to evaluate the Q-functions. This is a Monte Carlo update method. The bootstrapped update method, on the other hand, uses an estimate of the value function from the next sensor location to the end of the channel in updating the Q-function for the present sensor location. In the second implementation, for example, once the Q-function for sensor-4, for example, has been optimized, the sensor-3 Q-function update equations can use this optimal sensor-4 Q-function in its update. So the third implementation uses the observed performance along with the existing downstream optimal Q-functions to design a mixed update method.

A. Simultaneous all-sensor Q-learning update for the MHD channel

The update equations for the Q-functions and controllers are given as follows:

At every time index k_T,

• One of the four sections of the channel is given control input that is exploratory in nature. To choose the section that gets the exploratory input, a set, Δ^{π}, is defined as:

$$\Delta^{\pi} = \{\delta(1), \delta(2), \delta(3), \delta(4)\} \tag{8}$$

One of the elements of the set is set to 1, and the rest of the elements set to zero. Each of the elements has equal probability to be set to 1.

• For $k_t = i_{\text{sensor}} = 1$ through 4

o The state $\mathbf{w}_{\text{sensor}}$ is sensed at the sensor location at the corresponding time index k_t.

o Based on this sensed value, the electron beam current control profile from that sensor location to the next sensor location is given as:

$$\mathbf{u}(\mathbf{w}_{\text{sensor}}, i_{\text{sensor}}) = \boldsymbol{\Phi}^{b} \boldsymbol{\alpha}(\mathbf{w}_{\text{sensor}}, i_{\text{sensor}})$$
$$\boldsymbol{\alpha}(\mathbf{w}_{\text{sensor}}, i_{\text{sensor}}) = \begin{bmatrix} \mathbf{f}^{\boldsymbol{\alpha}}_{\text{NN}}\left[\mathbf{w}_{\text{sensor}}, \mathbf{W}^{\boldsymbol{\alpha}}_{\text{NN}}\left(i_{\text{PI}}, i_{\text{sensor}}\right)\right] \\ +\delta\left(i_{\text{sensor}}\right) * diag\left(\varepsilon_1 \quad \varepsilon_2 \quad \ldots \quad \varepsilon_p\right) * ones\left(p,1\right) \end{bmatrix} \tag{9}$$

ε_1 through ε_p are random numbers chosen from a prescribed range. The term $ones(p,1)$ refers to a column vector with p elements, all set to the value one. Eq. (9) provides the capability of exploring the $\boldsymbol{\alpha}$-space for each sensor. Depending on the element of the set Δ^{π} that equals 1, the corresponding section is given an exploratory control input, while the remaining sections are given control input based on the current policy.

• The Q-function, by definition, is specific to a particular policy. This implies that the control parameters, in evaluating the Q-function at the given sensor location, can assume any arbitrary value, but the control input in all the following sections has to be based on the current policy. Thus, the Q-function at the given sensor location cannot be computed if an exploratory input, non-conformal with the policy, is given at any proceeding section. The Q-function update is correspondingly skipped for all sections preceding the section with the exploratory control input. For the sections thus selected, the Q-function error is added to the existing Q-function composite error, to get the new composite error.

$$E^{Q}_{\text{NN}}(i_{\text{sensor}}) =$$
$$\frac{1}{p_{i_{\text{sensor}}}} \sum_{i=1}^{p_{i_{\text{sensor}}}} \left\langle \begin{matrix} \mathbf{f}^{Q}_{\text{NN}}\left[\mathbf{w}_{\text{sensor}}, \boldsymbol{\alpha}, \mathbf{W}^{Q}_{\text{NN}}\left(i_{\text{sensor}}\right)\right] - U\left[\mathbf{w}_{\text{sensor}}(i_{\text{sensor}}), \boldsymbol{\alpha}\right] \\ + \sum\limits_{i=i_{\text{sensor}}+1}^{4} U\left\{\mathbf{w}_{\text{sensor}}(i), f^{\boldsymbol{\alpha}}_{\text{NN}}\left[\mathbf{w}_{\text{sensor}}, \mathbf{W}^{\boldsymbol{\alpha}}_{\text{NN}}\left(i_{\text{PI}}, i_{\text{sensor}}\right)\right]\right\} \end{matrix} \right\rangle^{2} \tag{10}$$

• The derivative of this composite error for each sensor with respect to its Q-function parameter vector, $\mathbf{W}^{Q}_{\text{NN}}\left(i_{\text{sensor}}\right)$, is computed. The parameter vector is subsequently updated using a gradient-based update method to reduce the Q-function composite error. This outlines the policy evaluation step.

The policy is improved after this evaluation step as:

$$\mathbf{W}^{\boldsymbol{\alpha}}_{\text{NN}}\left(i_{\text{PI}}+1, i_{\text{sensor}}\right) = \underset{\mathbf{W}^{\mathbf{u}}_{\text{NN}}\left(i_{\text{PI}}, i_{\text{sensor}}\right)}{\arg\min}\, Q\left\{\mathbf{w}_{\text{sensor}}, \mathbf{f}^{\boldsymbol{\alpha}}_{\text{NN}}\left[\mathbf{w}_{\text{sensor}}, \mathbf{W}^{\boldsymbol{\alpha}}_{\text{NN}}\left(i_{\text{PI}}, i_{\text{sensor}}\right)\right]\right\}$$
$$\text{for all } \mathbf{w}_{\text{sensor}}, i_{\text{sensor}} \tag{11}$$

The steepest descent approach for improving the policy is given as:

$$\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}}+1,i_{\text{sensor}}\right)\leftarrow\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},i_{\text{sensor}}\right)-\alpha_{\text{controller}}\sum_{i=1}^{p}\frac{\partial Q\left\{\mathbf{w}_{\text{sensor}},\mathbf{f}_{\text{NN}}^{\boldsymbol{\alpha}}\left[\mathbf{w}_{\text{sensor}},\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},i_{\text{sensor}}\right)\right]\right\}}{\partial\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},i_{\text{sensor}}\right)} \tag{12}$$

where $\alpha_{\text{controller}}$ represents the controller learning rate. These policy evaluation and policy improvement steps are iterated to enhance the performance of the MHD generator.

B. One-sensor-at-a-time backward Q-learning implementation for the MHD channel

The policy that defines the Q-function at a given sensor location is given by the control parameters from the next sensor location to the last sensor location. These dependencies are given as:

- $Q^{\boldsymbol{\pi}^{\mathbf{W}}}\left(i_{\text{sensor}}\right)$ for $i_{\text{sensor}}=1$ is defined for

$$\boldsymbol{\pi}^{\mathbf{W}}(1)=\left[\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},2\right)\quad\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},3\right)\quad\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},4\right)\right] \tag{13}$$

- $Q^{\boldsymbol{\pi}^{\mathbf{W}}}\left(i_{\text{sensor}}\right)$ for $i_{\text{sensor}}=2$ is defined for

$$\boldsymbol{\pi}^{\mathbf{W}}(2)=\left[\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},3\right)\quad\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},4\right)\right] \tag{14}$$

- $Q^{\boldsymbol{\pi}^{\mathbf{W}}}\left(i_{\text{sensor}}\right)$ for $i_{\text{sensor}}=3$ is defined for

$$\boldsymbol{\pi}^{\mathbf{W}}(3)=\left[\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},4\right)\right] \tag{15}$$

- Finally $Q^{\boldsymbol{\pi}^{\mathbf{W}}}\left(i_{\text{sensor}}\right)$ for $i_{\text{sensor}}=4$ is a policy free Q-function

The dependencies outlined by Eqs. (13-15) suggest a sequential Q-function update method consistent with dynamic programming. The Q-function for sensor-4 can be fully evaluated before any other preceding sections. After evaluating and optimizing the sensor-4 Q-function, the sensor-3 Q-function can be evaluated and optimized. This backward procedure is inherent in the dynamic programming based update method outlined in Ref. [10]. The current implementation corresponds to the on-line sequential version as compared to the off-line batch method of Ref. [10].

C. .Mixed Monte Carlo-Bootstrapped Q-learning implementation for the MHD channel

The evaluation of the Q-function for both the simultaneous-all-sensor, and one-sensor-at-a-time implementations are based on observed performance values corresponding to a Monte Carlo update. For example, the desired value of the Q-function for a state-control pair at the sensor-1 location is computed by giving the policy specific control inputs from section 2 to the end of the channel, and summing all the observed individual sectional performances. A bootstrapped update rule, instead, uses a function approximator to estimate the cost-to-go function. The Monte Carlo update is more accurate than the bootstrapped update as it uses the actual value rather than the estimated value of the Q-function for computing the desired signal. This update, however, has to wait for the system to go through the episode to provide data for using in its update rule. If a Q-function approximator can provide accurate estimates of the cost-to-go function, then, using this information along with actual data from the system can substantially accelerate the learning process. This hints to a mixed Monte Carlo-bootstrapped update algorithm that can be designed for the MHD generator. This procedure is formalized as follows.

At every time index k_T,

- For $k_t=i_{\text{sensor}}=1$ through 4

The control parameters are given by Eq. (9). If the Q-function is being evaluated for the sensor location i_{sensor}, then $\delta\left(i_{\text{sensor}}\right)=1$, else $\delta\left(i_{\text{sensor}}\right)=0$

- The control parameters, $\boldsymbol{\alpha}\left(\mathbf{w}_{\text{sensor}},i_{\text{sensor}}\right)$, and the sensed states $\mathbf{w}_{\text{sensor}}\left(i_{\text{sensor}}\right)$ are stored in the existing tuple data-set

$$T\left(i_{\text{sensor}}\right)=\left\{\begin{array}{l}\left[\mathbf{w}_{\text{sensor}}\left(i_{\text{sensor}},i\right),\boldsymbol{\alpha}\left(\mathbf{w}_{\text{sensor}},i_{\text{sensor}},i\right),\mathbf{w}_{\text{sensor}}\left(i_{\text{sensor}}+1,i\right)\right]\\ ,i=1\ldots k_T\end{array}\right\} \tag{16}$$

- The Q-function error is computed only for the section whose Q-function is being evaluated. This error is added to the existing Q-function composite error for the section, to get the new composite error.

$$E_{\text{NN}}^{Q}(i_{\text{sensor}})=\frac{1}{p_{i_{\text{sensor}}}}\sum_{i=1}^{p_{i_{\text{sensor}}}}\left\langle\begin{array}{l}\mathbf{f}_{\text{NN}}^{Q}\left[\mathbf{w}_{\text{sensor}}(i_{\text{sensor}},i),\boldsymbol{\alpha},\mathbf{W}_{\text{NN}}^{Q}\left(i_{\text{sensor}}\right)\right]-U\left[\mathbf{w}_{\text{sensor}}(i_{\text{sensor}},i),\boldsymbol{\alpha}(i)\right]\\ +\displaystyle\sum_{j=i_{\text{sensor}}+1}^{4}U\left\{\mathbf{w}_{\text{sensor}}(j,i),f_{\text{NN}}^{\boldsymbol{\alpha}}\left[\mathbf{w}_{\text{sensor}}(j,i),\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(i_{\text{PI}},i_{\text{sensor}}\right)\right]\right\}\end{array}\right\rangle^{2}$$
$$+\frac{1}{q_{i_{\text{sensor}}}}\sum_{i=1}^{q_{i_{\text{sensor}}}}\left\langle\begin{array}{l}\mathbf{f}_{\text{NN}}^{Q}\left[\mathbf{w}_{\text{sensor}},\boldsymbol{\alpha},\mathbf{W}_{\text{NN}}^{Q}\left(i_{\text{sensor}}\right)\right]-U\left[\mathbf{w}_{\text{sensor}}(i_{\text{sensor}},i),\boldsymbol{\alpha}(i)\right]\\ +\mathbf{f}_{\text{NN}}^{Q}\left\{\mathbf{w}_{\text{sensor}}(i_{\text{sensor}}+1,i),f_{\text{NN}}^{\boldsymbol{\alpha}}\begin{bmatrix}\mathbf{w}_{\text{sensor}}(i_{\text{sensor}}+1,i),\\ \mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}}\left(*,i_{\text{sensor}}+1\right)\end{bmatrix},\mathbf{W}_{\text{NN}}^{Q}\left(i_{\text{sensor}}\right)\right\}\end{array}\right\rangle^{2} \tag{17}$$

The first summation computes the part of the composite error given by the observed system data. The second summation corresponds to the Q-function error that uses the elements of the tuple data-set, $T\left(i_{\text{sensor}}\right)$, to provide additional data points. For example, when the sensor-4 Q-function is being evaluated and improved, the tuple data-set, $T(3)$, for the third sensor location gets populated. The sensor-4 Q-function and controller, once optimized, can be used to provide the estimated optimal cost-to-go function value for the sensor-4 states stored in $T(3)$. The error computation for the case of $T(3)$ is given as:

$$T(3)=\left\{\left[\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}\left(\mathbf{w}_{\text{sensor}},3,i\right),\mathbf{w}_{\text{sensor}}(4,i)\right],i=1\ldots p_3\right\}$$

$$Q\left\{\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}\left(\mathbf{w}_{\text{sensor}},3,i\right)\right\}=\mathbf{f}_{\text{NN}}^{Q}\left[\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}\left(\mathbf{w}_{\text{sensor}},3,i\right)\right],\quad i=1\ldots p_3$$

$$Q\{\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}(\mathbf{w}_{\text{sensor}},3,i)\}_{\text{desired}}$$
$$=U\left[\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}(\mathbf{w}_{\text{sensor}},3,i)\right]$$
$$+\mathbf{f}_{\text{NN}}^{Q^*}\left\{\mathbf{w}_{\text{sensor}}(4,i),f_{\text{NN}}^{\boldsymbol{\alpha}^*}\left[\mathbf{w}_{\text{sensor}}(4,i),\mathbf{W}_{\text{NN}}^{\boldsymbol{\alpha}^*}(4)\right],\mathbf{W}_{\text{NN}}^{Q^*}(4)\right\},i=1\ldots p_3$$
$$e_{T(3)}(i)=\left\langle \begin{array}{l} Q\{\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}(\mathbf{w}_{\text{sensor}},3,i)\} \\ -Q\{\mathbf{w}_{\text{sensor}}(3,i),\boldsymbol{\alpha}(\mathbf{w}_{\text{sensor}},3,i)\}_{\text{desired}} \end{array}\right\rangle^2,i=1\ldots p_3 \tag{18}$$

This bootstrapped error, summed for all the $p_{i_{\text{sensor}}=3}$ data points of $T(3)$, represents the second summation in Eq. (18). Since this error requires the optimal sensor-4 Q-function and controller, its computation is carried out immediately after the sensor-4 Q-function and controller are optimized. In a similar manner, the bootstrapped errors for sensor-2 and sensor-1 are computed immediately after sensor-3 Q-function and controller, and sensor-2 Q-function and controller, are optimized respectively.

- The derivative of this composite error with respect to its Q-function parameter vector, $\mathbf{W}_{\text{NN}}^{Q}(i_{\text{sensor}})$, is computed, and the parameter vector is updated using a chosen gradient update method.

A big advantage of this update rule is the substantial acceleration in the update rate. For example, the sensor-3 Q-function evaluation using the Monte Carlo alone one-section-at-a-time update rule given in the previous subsection initiates after the sensor-4 Q-function and controller are optimized and subsequent system data starts becoming available. For the mixed Monte-Carlo bootstrapped update, after the sensor-4 Q-function and controller are optimized and before the subsequent data starts getting collected, the evaluation of the sensor-3 Q-function can be initiated with the bootstrapped error computed using $T(3)$. This advantage increases for the sensor-2 and sensor-1 Q-functions as the data-sets, $T(2)$ and $T(1)$, have a lot more data points available, which can be used to start updating the Q-function using their respective bootstrapped errors.

VI. Simulation results

For an on-line scenario, the data distribution is attached to the particular flight trajectory, which is given by free-stream altitude and Mach number. These free-stream conditions translate to flow variables at the channel inlet. For the current implementation, the chosen flight trajectory represents an arbitrary flight altitude within the prescribed limits, with the vehicle accelerating from Mach 7.2 to Mach 8.8. This is followed by another flight altitude, and the vehicle decelerating from Mach 8.8 to 7.2. This sequence is then repeated for the length of the test flight.

A. Verification criteria for MHD generator performance improvement:

An important aspect of the proposed on-line design is defining an appropriate performance metric for measuring the success of the update methods. Improvement of the performance of the MHD generator at any single altitude-Mach number pair using the proposed algorithms, while important, does not necessarily suggest the success of these algorithms in general. The success of both the policy evaluation and policy improvement steps needs to be monitored. An average Q-function is, therefore, defined that considers the averaged Q-function function value at each of the sensor locations computed over a range of free-stream altitudes and Mach numbers.

$$\bar{Q}(i_{\text{sensor}})=\frac{1}{p}\sum_{i=1}^{p}\sum_{k=i_{\text{sensor}}}^{4}U\left[\mathbf{w}_{\text{sensor}}(k,h_i,M_i),\boldsymbol{\alpha}(k,h_i,M_i)\right] \quad \forall \text{ all } (h_i,M_i) \tag{19}$$

$\mathbf{w}_{\text{sensor}}(1,h_i,M_i)$ refers to the inlet system state for the corresponding free stream Mach number, and $\mathbf{w}_{\text{sensor}}(2,h_i,M_i)$, $\mathbf{w}_{\text{sensor}}(3,h_i,M_i)$, and $\mathbf{w}_{\text{sensor}}(4,h_i,M_i)$ correspond to the system states resulting with policy specific control parameters. The altitudes and Mach numbers, (h_i,M_i), are chosen as:

$$\begin{array}{lll} (h_1,M_1)=(27\text{km},7.2); & (h_4,M_4)=(30\text{km},7.2); & (h_7,M_7)=(33\text{km},7.2) \\ (h_2,M_2)=(27\text{km},8.0); & (h_5,M_5)=(30\text{km},8.0); & (h_8,M_8)=(33\text{km},8.0) \\ (h_3,M_3)=(27\text{km},8.8); & (h_6,M_6)=(30\text{km},8.8); & (h_9,M_9)=(33\text{km},8.8) \end{array} \tag{20}$$

These numbers represent altitudes and Mach numbers that are evenly spread across the space of free stream conditions. For each of these inlet conditions, the performance of the MHD generator is computed using the 4 controllers updated till that point in the simulation. It should be noted that such verification can only be carried out in a simulation mode, where the actual behavior of the system is assumed to be known so that the simulation is simultaneously carried out for these chosen altitudes and free stream Mach numbers for computing this average Q-function, while the system is assumed to be operating at other conditions. In a real experiment, the performance at these altitudes and Mach numbers cannot be observed till the system operates at all of them. One of the challenges of reinforcement learning algorithms, as applied to this problem, lies in verification of their performance in an on-line setting. For the present discussion, however, the prescribed method is used in the analysis of the performance. Table I lists the weighting parameters chosen in the cost function

Table I
Weighting parameters chosen in Eq. (3)

p_1	q_1	r_1	r_2
20	0.00001	0.005	25

B. Case 1: Results using the all-sensor simultaneous update method

This case illustrates the application of the policy iteration algorithm, where the policy for each of the sensor location is simultaneously evaluated and then improved. Fig. 4a through 4d monitor the performance of the update algorithm by observing the behavior of the average Q-function for each sensor location. The solid line represents the average Q-function computed using Eq. (19) by running the simulation at the operating conditions given in Eq. (20). The dashed line represents the averaged output of the Q-function network for the system states corresponding to these operating conditions and the control input given by the current controller network. The policies for each of the sensor location are evaluated by training the Q-function network for 150 time steps and then improved by updating the controller network. Levenberg-Marquardt algorithm is used as the training algorithm in the evaluation step, and an adaptive learning rate gradient descent algorithm used in the improvement step.

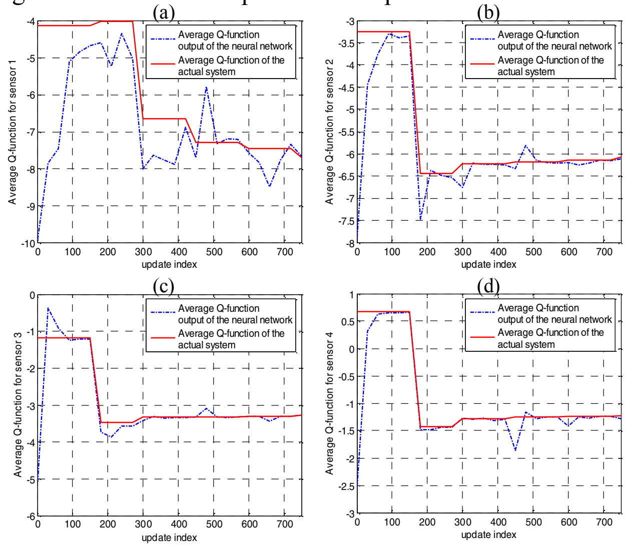

Fig. 4. Comparison of the Actual Averaged Q-function with the Averaged-function Network Output with the all-sensor simultaneous update

The performance of both the policy evaluation and the policy improvement steps can be assessed from these figures. Successful policy evaluation corresponds to the dashed line converging to the solid line during the 150 time-step period. Successful policy improvement corresponds to the subsequent lowering in value of the average Q-function of the actual system denoted by the solid line. Fig. 4a illustrates the behavior of the average Q-function for sensor-1, which also corresponds to the overall performance of the MHD generator. It can be noted that during every policy evaluation period of 150 time steps, the average Q-function output of the neural network gets closer to the average Q-function output of the actual system. This indicates success of the policy evaluation step carried out using the semi-batch training given by Eqs. (10-12). The discontinuous nature of the graph results from the fact that these observations are taken every 30 time steps. It is also important to note that it takes around 150 time-step updates for learning the policy. If the policy were to be updated without complete evaluation in fewer updates, then there is no guarantee that this incompletely evaluated policy will provide the correct gradients for improving the policy. This brings out the conservativeness of the policy iteration based update versus the value iteration based update that asks the policy to be updated at every time step. Regarding the policy improvement results, it is observed that the actual average Q-function increases slightly at the first policy improvement step before decreasing for the subsequent 4 improvement steps. This was attributed to a big learning rate for the controller at the first sensor location at the first improvement step after which the rate gets adjusted appropriately for the subsequent improvement steps, resulting in a consistent decrease in value of the Q-function.

The policy evaluation and improvement results for the other sensor locations (2 through 4) are portrayed in Figs. 4b through 4d. The semi-batch Levenberg-Marquardt algorithm is successful in correctly training the Q-function network after every policy improvement step (every 150 time steps). Regarding the effectiveness of the policy improvement, at the first policy improvement step, the average Q-function for each of these sensor locations decreases as expected for successful policy improvement. For all the subsequent policy improvement steps, however, the average Q-function shows a slight increase. This was initially attributed to an incorrectly tuned learning rate for the neural network controller. However, any amount of tuning did not alter this general behavior. Further investigation revealed that this is a natural consequence of the dependence of the Q-function upon the previous section controller. The average sensor-1 Q-function is calculated for the chosen free stream conditions given by Eq. (20). However, the average Q-function for sensor-2 is computed for system states resulting based on the free stream inlet conditions as well as the control applied in the first section. Similarly the average sensor-3 Q-function is computed for system states resulting from the free stream inlet conditions and the control applied in the first two sections. Now consider a particular operating point corresponding to a particular free stream altitude and Mach number. After the first policy improvement step, the electron beam current applied in the first quarter section of the channel increases from its previous value, as seen in subplot 2 of Fig. 5. This reduces the Mach number at the sensor-2 location. A lower Mach number at the beginning of section 2 implies that the power extracted while going from this Mach number to the prescribed exit Mach number decreases. Correspondingly, the Q-function for these flow variables, which is defined with a negative sign on the net power extracted, is higher than the Q-function for the flow variables that were seen at this sensor location before the policy improvement step. This observation applies to the further sensor locations 3 and 4 as well. The insight brings out the richness and complexity of this reinforcement learning implementation for this finite horizon problem that has multiple Q-functions and controllers with dependencies versus infinite time optimal control problems that have a single Q-function and controller.

Fig. 5 illustrates the Mach number and electron beam current input profile along the channel, before and after the update for the free stream inlet conditions: $(h, M) = (30\text{km}, 8.0)$. After update, the Mach number gets closer to 1.5. The actual values of the exit Mach number before and after the update are 1.81 and 1.47 respectively.

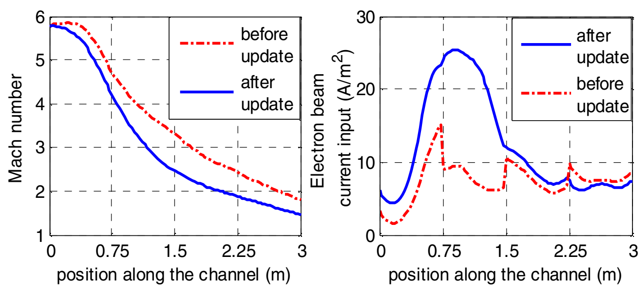

Fig. 5. Mach number and electron beam current input for $(h, M) = (30\text{km}, 8.0)$, before and after update.

C. Case 2: Results using sequential one-sensor-at-a-time backward update

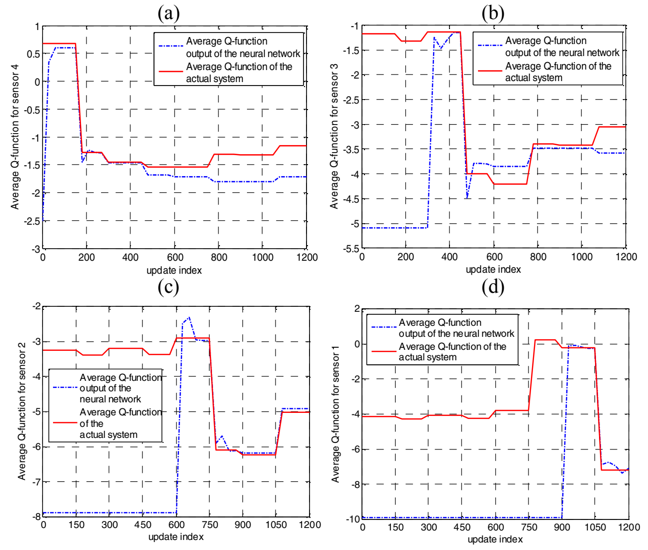

Fig. 6. Comparison of the Actual Averaged Q-function with the Averaged-function Network Output with the one-sensor-at-a-time backward update

In this case, the Q-function and the controller for section 4 are optimized first, followed by the section 3, section 2 and section 1 Q-function and controller respectively. The policy evaluation, as in the previous case, is carried out for 150 time steps and two policy improvements are carried out for each sensor location. This case, therefore, requires a much longer update time (1200 time steps) compared to the previous case (750 time steps). Fig. 6a illustrates the policy evaluation and improvement for sensor-4 location. The semi-batch based Levenberg-Marquardt algorithm is successful in learning the policy. After two policy improvements, the average Q-function reaches a value that is comparable to the one achieved in the previous case. The Q-function and the controller network are thereafter frozen, and the policy evaluation and improvement is initiated for the sensor-3 location. The sensor-3 Q-function and controller neural networks also go through two policy evaluations and improvements. In both the sensor-3 and sensor-4 policy improvements, the Q-function decreases in the second policy improvement step as compared to the previous case shown in Figs. 4c and 4d. This is again due to the fact that when the sensor-4 policy is being evaluated and improved, the policies of all the previous sensor locations are fixed. So the Q-function average is computed for the same flow variables. However, after two policy improvement steps, when the sensor-4 policy is frozen, the average sensor-4 Q-function for the actual system starts increasing slightly as the average is now computed for flow variables that are influenced by the sensor-3 controller. At the same time, the output of the sensor-4 Q-function network prediction starts deteriorating as it has never seen these flow variables as its input during its training phase. While the same effect occurs in case 1, the sensor-4 Q-function is getting re-evaluated and improved, and so the Q-function network continues to predict the actual averaged sensor-4 Q-function. Figs. 6c and 6d illustrate a similar behavior for sensors 2 and 1 respectively.

The policy evaluation, as in the previous case, is carried out for 150 time steps and two policy improvements are carried out for each sensor location. This case, therefore, requires a much longer update time (1200 time steps) compared to the previous case (750 time steps). Fig. 6a illustrates the policy evaluation and improvement for sensor-4 location. The semi-batch based Levenberg-Marquardt algorithm is successful in learning the policy. After two policy improvements, the average Q-function reaches a value that is comparable to the one achieved in the previous case. The Q-function and the controller network are thereafter frozen and the policy evaluation and improvement is initiated for the sensor-3 location. The sensor-3 Q-function and controller neural networks also go through two policy evaluations and improvements. In both the sensor-3 and sensor-4 policy improvements, the Q-function decreases in the second policy improvement step as compared to the previous case shown in Figs. 4c and 4d. This is again due to the fact that when the sensor-4 policy is being evaluated and improved, the policies of all the previous sensor locations are fixed. So the Q-function average is computed for the same flow variables. Similarly when the sensor-3 policy is being evaluated and improved, the policies of sensor-1 and sensor-2 are fixed, and the Q-function average for the sensor-3 location is computed for the same flow variables. However, after two policy improvement steps, when the sensor-4 policy is frozen, the average sensor-4 Q-function for the actual system starts increasing slightly as the average is now computed for flow variables that are influenced by the sensor-3 controller. At the same time, the output of the sensor-4 Q-function network prediction starts deteriorating as it has never seen these flow variables as its input during its training phase. While the same effect occurs in case 1, the sensor-4 Q-function is getting re-evaluated and improved, and so the Q-function network continues to predict the actual averaged sensor-4 Q-function. Figs. 6c and 6d illustrate a similar behavior for sensors 2 and 1 respectively.

The sequential sensor-by-sensor update based on the dynamic programming principle, therefore, suffers from two major disadvantages when applied in an on-line setting. Firstly the update is slower as each sensor is evaluated and improved individually. Secondly when the previous sensor locations start getting evaluated and improved, the frozen downstream sensor Q-function and controller see new flow variables that they have not been exposed to, and consequently their prediction quality is poor.

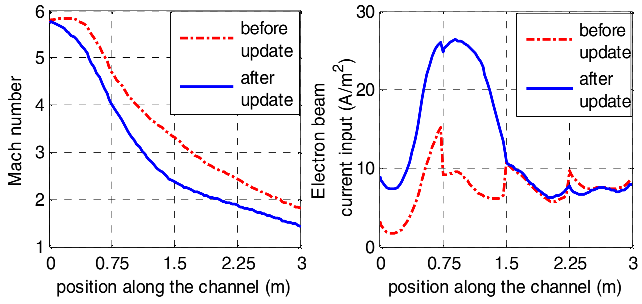

Fig. 7. Mach number and electron beam current input for $(h, M) = (30\text{km}, 8.0)$, before and after update.

Fig. 7 illustrates the Mach number and electron beam current input profile along the channel before and after the update for the free stream inlet conditions: $(h, M) = (30\text{km}, 8.0)$. Before the update the exit Mach number is 1.81. After the update it is 1.44. It is interesting to note the similarities in the general features of the improved Mach number profiles and the electron beam current input profiles obtained with the simultaneous all-sensor-at-at-time and the current implementation (Fig. 5 and Fig. 7). This is particularly encouraging because the resulting optimal solutions are similar regardless of the method used.

D. Case 3: Results using one-sensor-at-a-time update with bootstrapping

This case is similar to the sequential backward dynamic programming case, but it removes the various inefficiencies inherent in that implementation by using bootstrapping. Specifically, there are three main differences between the two implementations (Case 3 vs. Case 2). The sensor-4 Q-function network and controller training remains exactly the same in both the implementations. For the bootstrapped update, however, while the sensor-4 policy is being evaluated and improved, the states and the control inputs at the previous sensor locations are stored. After two policy improvements of the sensor-4 networks, the sensor-3 policy evaluation and improvement is initiated. This update uses the previously stored sensor-3 states and control inputs and the resulting sensor-4 state to compute the desired values of the sensor-3 Q-function using the improved sensor-4 Q-function and controller network as outlined by Eq. (18). These training points along with the training data, which are collected after the sensor-4 improvement, are used to train the sensor-3 Q-function network. This represents the first main difference corresponding to the use of this novel mixed Monte Carlo and bootstrapped update.

The second difference with the previous sequential backward implementation corresponds to the evaluation of the downstream Q-functions even after their policies are frozen after two policy improvement steps. In the previous implementation when the sensor-4 Q-function and controller networks are trained, they remain frozen while the previous sensor policies get improved. This leads to the separation of the averaged Q-function evaluated for the actual system and that computed by the neural network. This is the result of the network seeing new states as its inputs, after the previous sensor controller networks get trained. This issue, while not having much impact on the sequential update, is critical for the bootstrapped update since the bootstrapped update uses the downstream Q-functions for updating the upstream Q-functions. Evaluating the frozen policy for the new states seen by the downstream Q-functions provides an effective fix for this problem. Figs. 8a through 8d illustrate all these effects. For example, in Figs. 8a and 8b, the output of the neural network keeps tracking the Q-function for the actual system while in Figs. 6a and 6b, it separates. The semi-batch based Levenberg-Marquardt algorithm is successful every time in evaluating the new policy after the old policy is improved for all sensors.

The third difference with the previous sequential update algorithm can be noted by looking at the total update indices of the two cases. One of the issues with the previous sequential update algorithm is that the final update takes too long. For the bootstrapped update case, the two policy evaluations for the sensor-4 update require 150 training points each. However, the sensor-3 update has these 150 training points available as soon it starts its training. So the two policy evaluations for the sensor-3 Q-function are carried out using 125 and 100 updates respectively. Similarly after the sensor-3 policy is frozen, and the sensor-2 Q-function starts getting evaluated, it already has 525 training points available. The sensor-2 policy evaluations are therefore carried out only for 90 and 60 updates respectively. Finally the two sensor-1 policy evaluations are carried out for 50 updates each. The total updates therefore equal 775 versus 1200 for the previous case.

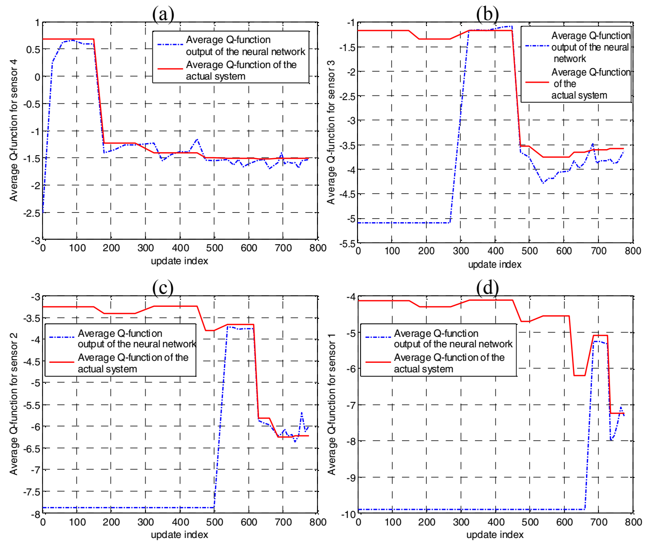

Fig. 8. Comparison of the Actual Averaged Q-function with the Averaged-function Network Output with the one-sensor-at-a-time backward update with bootstrapping.

Fig. 9 shows the Mach number and electron beam current input profile along the channel before and after the update for the free stream inlet conditions $(h, M) = (30\text{km}, 8.0)$. Before the update the exit Mach number is 1.81, while after update it reaches 1.51.

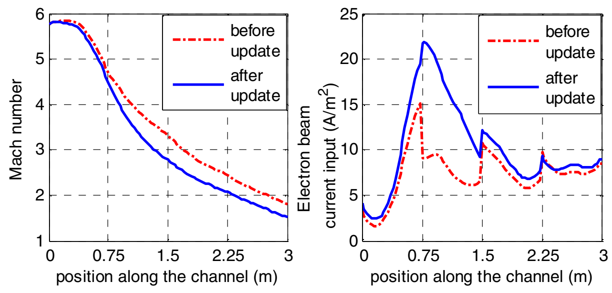

Fig. 9. Mach number and electron beam current input for $(h, M) = (30\text{km}, 8.0)$, before and after update.

VII. Conclusions

This paper illustrates how the off-line optimal controllers for the hypersonic MHD channel, designed in Ref. [10], can be updated on-line with the conservative policy-iteration-based Q-learning as the selected choice of the reinforcement learning algorithm. Due to the finite horizon nature of this optimal control problem, there is an interesting range of possible implementations of the basic policy-iteration-based Q-learning algorithm. Consequently, the paper proposes three different update rules involving a combination of concepts from Monte Carlo update to bootstrapped update, as well as sequential backward to simultaneous update of all the controllers in this MHD system. A semi-batch Levenberg-Marquardt algorithm is used in the policy evaluation step of learning the Q-function for all the update rules. The adaptive-learning-rate gradient descent algorithm is used to train the controllers in the policy improvement step. The MHD application provides rich and interesting insights into the dynamics of these update rules. The simulation results gauge the efficacy of each of the proposed update algorithms by examining the behavior of the Q-function network output and the actual Q-function averaged over a range of operating conditions. The results stress the importance of using the conservative policy-iteration-based learning versus its value iteration counterpart.

Acknowledgments

This work was funded by ANSER Corporation through a grant from the National Science Foundation. The authors wish to thank Ray Chase and Paul Werbos for this funding support. Prof. Richard Miles at Princeton University and his applied physics group, especially Mikhail Shneyder, provided the MHD modeling support. The authors also wish to thank Prof. Robert Stengel at Princeton University and Kalmanje Krishnakumar at NASA Ames Research Center for their support.

References:

[1] Chase, R.L., Mehta, U.B., Bogdanoff, D.W., Park, C., Lawrence, S., Aftosmis, M., Macheret, S.O., and Schneider, M.N., "Comments on a MHD-Bypass Spaceliner Performance," *AIAA Paper* 99-4965, Nov. 1999.

[2] Fraishtadt, V.L., Kuranov, A.L., and Sheikin, E.G., "Use of MHD Systems in Hypersonic Aircraft," *Technical Physics*, Vol. 43, No.11, 1998, p.1309.

[3] Gurijanov, E. P., and Harsha, P. T., "AJAX: New Directions in Hypersonic Technology," *7th AIAA International Spaceplanes and Hypersonic Technologies Conference*, Norfolk, VA, Nov. 1996; also AIAA Paper 96-4609, 1996.

[4] Bityurin, V.A., Lineberry, J.T., Potebnia, V.G., Alferov, V.I., Kuranov, A.L., and Sheikin, E.G., "Assessment of Hypersonic MHD Concepts," *AIAA Paper* 97-2323, 1997.

[5] Bityurin, V. A., Lineberry, J. T., Potebnia, V. G., Alferov, V. I., Kuranov, A. L., and Sheikin, E. G., "Assessment of HypersonicMHDConcepts," *28th AIAA Plasmadynamics and Lasers Conference*, Atlanta, 23–25 June 1997; also AIAA Paper 97-2393, 1997.

[6] Brichkin, D. I., Kuranov, A. L., and Sheikin, E. G., "MHD Technology for Scramjet Control," *8th AIAA International Spaceplanes and Hypersonic Technologies Conference*, Norfolk, VA, 27–30 April 1998; also AIAA Paper 98-1642, 1998.

[7] Rosa, R. J., "Magnetohydrodynamic Energy Conversion," Hemisphere Publications, Washington, 1967.

[8] Macheret, S. O., Shneider, M. N., and Miles, R. B., "MHD Power Extraction from Cold Hypersonic Air Flows with External Ionizers," *Journal of Propulsion and Power*, Vol. 18, No. 2, March-April 2002, pp. 424-431

[9] Macheret, S.O., Schneider, M.N., Miles, R.B., and Lipinski, R. J., "Electron Beam Generated Plasmas in Hypersonic Magnetohydrodynamic Channels," *AIAA Journal*, Vol. 39, No. 6, 2001, pp. 1127-1138.

[10] Kulkarni, N.V. and Phan, M.Q., "Optimal Feedback Control of the Magneto-Hydrodynamic Generator for a Hypersonic Vehicle," *AIAA Guidance, Navigation and Control Conference*, Austin, TX, August 2003, AIAA Paper 2003-5497.

[11] Werbos, P. J., "Approximate Dynamic Programming for Real Time Control and Neuro-modeling," *Handbook of Intelligent Control*, D.A. White and D.A. Sofge, (editors), Van Nostrand Reinhold, New York, 1992, pp 493 - 525

[12] Prokhorov, D. and Wunsch, D.C., "Adaptive Critic Designs," *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, 1997, pp. 997-1007.

[13] Balakrishnan, S.N. and Biega, V., "Adaptive Critic Based Neural Networks for Aircraft Optimal Control," *Journal of Guidance, Control and Dynamics*, Vol. 19, No. 4, August 1996.

[14] Neidhoefer, J. and Krishnakumar, K., "Immunized Adaptive Critic for an Autonomous Aircraft Control Application, *Artificial Immune System and Their Applications*, 1999.

[15] Enns, R. and Si, J., "Neuro-dynamic Programming Applied to Helicopter Flight Control," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Denver, CO, 2000, *AIAA* Paper 2000-4280.

[16] Ferrari, S. and Stengel, R. F. "An Adaptive Critic Global Controller," *Proceedings of the American Control Conference*, Anchorage, AK, May 2002.

[17] Landelius, T., and Knutsson, H., "Greedy Adaptive Critics for LQR Problems: Convergence Proofs," (Tech. Rep. No. LiTH-ISY-R-1896). Linkoping, Sweden: Computer Vision Laboratory (available at http://www.isy.liu.se/cvl/ScOut/TechRep/PaperInfo/lk96b.html)

[18] Bradtke, S. J., "Reinforcement learning applied to linear quadratic regulation," *Advances in Neural Information Processing Systems*, volume 5, pages 295-302. Morgan Kaufmann, San Mateo, CA, 1993

[19] Bradtke, S. J., Ydstie, B. E., Barto, A. G., "Adaptive linear quadratic control using policy iteration," *Proceedings of the American. Control Conference,* Baltimore, MD, June 1994, pp. 3475-3479.

[20] Kulkarni, N. V., and Phan, M. Q., "Data-Based Adaptive Predictive Control with Application to In-Flight MHD Power Generation," AIAA-2004-6221, *Proceedings of the 1st AIAA Intelligent Systems Technical Conference*, Chicago, Illinois, September 2004.

[21] Kulkarni, N.V. and Phan, M.Q., "Performance Optimization of a Magnetohydrodynamic Generator at the Scramjet Inlet," Journal of Propulsion and Power, Vol. 21, No. 3, 2005, pp. 822-830.

[22] Kulkarni, N.V. and Phan, M.Q., "Neural Networks Based Design of Optimal Controllers for Nonlinear Systems," *Journal of Guidance, Control, and Dynamics,* vol. 27 no. 5, September 2004.