Neuro Model-Reference Adaptive Control of an Inverted Double-Pendulum

Brian Cairl

May 20, 2014

Contents

List of Figures

1 | Introduction

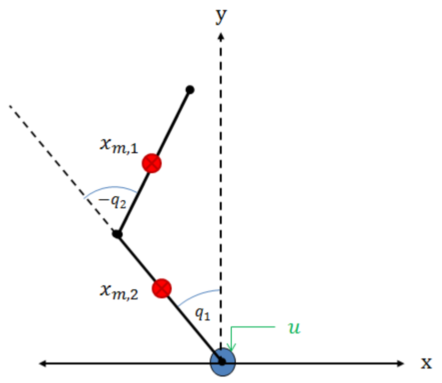


Figure 1.1: Double Inverted Pendulum

This report will detail an implementation of Neuro-Model Reference Adaptive Control (Neuro-MRAC) as applied to an under-actuated dynamical system: the inverted double pendulum. Neural networks, namely feed-forward neural networks, are ample candidates for incorporation in adaptive controllers due to their scalability and aptitude in modeling a variety of non-linear functions. Moreover, feed-forward networks can be trained an input-output mapping via the well-known Back-Propagation (B.P.) algorithm: an approximate gradient descent algorithm that can be utilized to adapt a feed-foward network of any size. In this study, a neural network will be utilized in concert with B.P. as an online compensator mechanism.

A neuro-compensator will be used to control the target plant by learning appropriate measures for approimately canceling unmodeled, dynamical variations between a linear reference model and non-linear target plant. Furthermore, the neuro-compensator will aid in stabilizing the plant by coercing it to behave like its stabilized, linear counterpart.

The control law derived in this report will be tested using a MATLAB-based simulation script. The script will simulate the double pendulum model, as well as implement the state-feedback and neuro-compensator control loops. Additionally the simulator will model sensor outputs with additive Gaussian white noise, so as to test controller performance law under practical noise conditions.

Simulation outputs from the Neuro-MRAC controlled plant will be compared with results from two other control additional control laws: one of which employs state feedback with the compensator turned off; and one of which utlizes a recursive least-squares estimator in place of the neural network.

2 | Target Plant Model

2.1 Inverted Double Pendulum Kinematics

For the purposes of this study, the kinematics of the double pendulum are described by $x_{m,1}$ and $x_{m,2}$, the centers of mass of the first and second links, respectively. The center of mass of each link is taken to be located at the mid-length, β_i, of each link.

$$x_{m,1} = \begin{bmatrix} -\beta_1 C_1 \\ \beta_1 S_1 \end{bmatrix} \qquad x_{m,2} = \begin{bmatrix} -a_1 C_1 - \beta_2 C_{12} \\ a_1 S_1 + \beta_2 S_{12} \end{bmatrix} \tag{2.1}$$

This kinematic description has been chosen such that the zero-state of the system occurs when the pendulum is in a fully upright position. This realization reduces the complexity of later formulations.

2.2 Inverted Double Pendulum Dynamics

Using the Euler-Lagrange equation, $\tau = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q}$, the driven dynamics of the double pendulum are formulated as follows:

$$\begin{aligned} \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = &\begin{bmatrix} m_1\beta_1^2 + m_2 a_1^2 + m_2\beta_2^2 + 2m_2 a_1 \beta_2 C_2 & m_2\beta_2^2 + m_2 a_1\beta_2 C_2 \\ m_2\beta_2^2 + m_2 a_1 \beta_2 C_2 & m_2\beta_2^2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \\ &+ \begin{bmatrix} -a_1\beta_2 m_2 S_2 \dot{q}_2 & -a_1\beta_2 m_2 S_2 \dot{q}_1 - a_1\beta_2 m_2 S_2 \dot{q}_2 \\ a_1\beta_2 m_2 S_2 \dot{q}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \\ &+ \begin{bmatrix} g(m_1\beta_1 + m_2 a_1) S_1 + g m_2 \beta_2 S_{12} \\ g m_2 \beta_2 S_{12} \end{bmatrix} \end{aligned} \tag{2.2}$$

The actuated joint at the base of the pendulum will be modeled as a first-order DC-motor. The coefficient k_t describes the a linear relationship between input current and output torque. This coefficient is extended to input voltage through the coefficient by $\rho_1 = k_t/R_m$, where R_m is the internal resistance of the motor. The coefficient k_{emf} describes the the back-emf of the motor. The coefficient ν_1 subsumes the effects of motor damping fricition and rotor inertia.

$$\tau_1 = \rho_1 u_1 - (\rho_1 k_{\text{emf}} + \nu_1)\dot{q}_1 = \rho_1 u_1 - M_1 \dot{q}_1 \tag{2.3}$$

The unactuated joint of the pendulum is modeled with a first-order viscuous friction approximation, parameterized by ν_2.

$$\tau_2 = -\nu_2 \dot{q}_2 \tag{2.4}$$

2.3 Linearized Inverted Double Pendulum Dynamics

As previously menditon, the target plant will be controlled around an unstable equilibrium point, $q = [0, 0]^T$. A linearization of the plant dynamics about this point yields the following equations:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} m_1\beta_1^2 + m_2a_1^2 + m_2\beta_2^2 + 2m_2a_1\beta_2 & m_2\beta_2^2 + m_2a_1\beta_2 \\ m_2a_1^2 + m_2a_1\beta_2 & m_2\beta_2^2 \end{bmatrix} \begin{bmatrix} \ddot{q_1} \\ \ddot{q_2} \end{bmatrix} + \begin{bmatrix} g(m_1\beta_1 + m_2a_1 + m_2\beta_2)q_1 + gm_2\beta_2q_2 \\ gm_2\beta_2q_1 + gm_2\beta_2q_2 \end{bmatrix} \tag{2.5}$$

For convenience, the linearized dynamics are rewritten like so:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \ddot{q_1} \\ \ddot{q_2} \end{bmatrix} + \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \tag{2.6}$$

2.4 State Space Formulation

The linearized plant dynamics, with state-vector z, take the following state-space form:

$$\dot{z} = A_z + B_z u \qquad q = C_z z \tag{2.7}$$

$$A_z = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -(G_{11}E_{11} + G_{12}E_{21}) & -(G_{11}E_{12} + G_{12}E_{22}) & -M_1E_{11} & -M_1E_{12} \\ -(G_{21}E_{11} + G_{22}E_{21}) & -(G_{21}E_{12} + G_{22}E_{22}) & -\nu_2E_{21} & -\nu_2E_{22} \end{bmatrix} \qquad B_z = \begin{bmatrix} 0 \\ 0 \\ \rho_1 \\ 0 \end{bmatrix} u$$

The transformation E, and corresponding block-matrix T, are used to transform the z state-space representation to a more convinient joint-space representation with state vector $\bar{x} = [q_1, q_2, \dot{q_1}, \dot{q_2}]^T$.

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \frac{1}{H_{11}H_{22} - H_{21}^2} \begin{bmatrix} H_{11} & -H_{21} \\ -H_{12} & H_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{2.8}$$

$$T = \begin{bmatrix} E & 0 \\ 0 & E \end{bmatrix} \tag{2.9}$$

The joint-space realization with with full-state output is achieved from *Equation 2.7* like so:

$$A = TA_zT^{-1} \qquad B = TB_z \quad C = I \tag{2.10}$$

3 Neuro-MRAC Controller

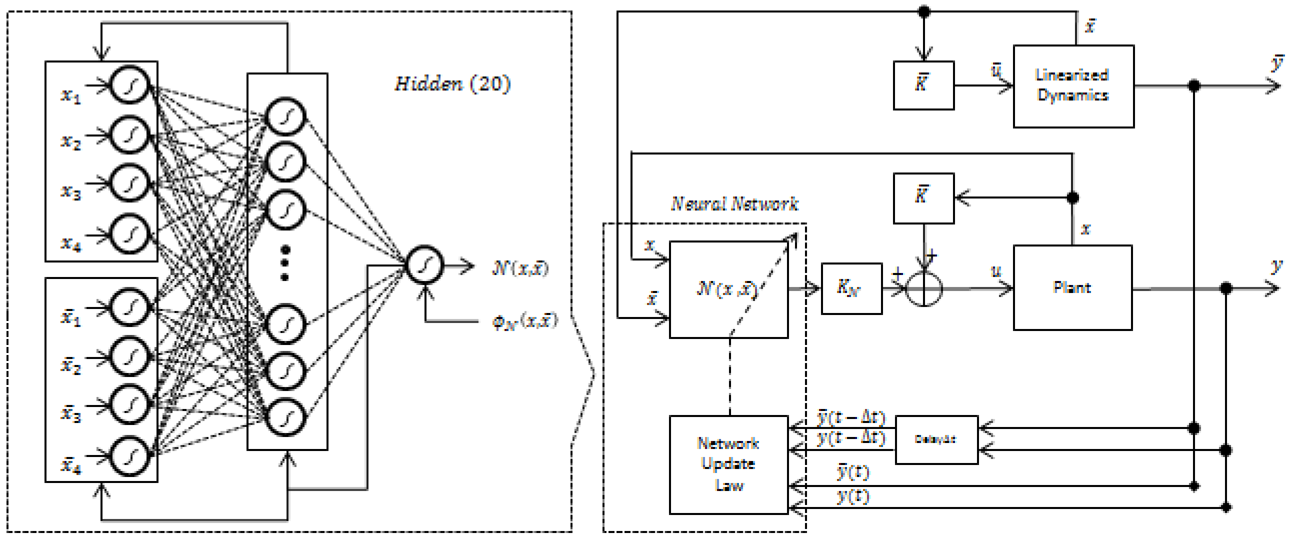


Figure 3.1: Compensator Network; Neuro-MRAC System Diagram

3.1 Control Law

The control law utilized in this study will feature a neuro-compensator mechanism, which will approximately linearize the state-dynamics of the nonlinear target plant. The basic control scheme utilized in this study is adapted from a scalar Neuro-MRAC configuration proposed by G. Irwin *et al* [4].

The reference-model utilized in this controller is a zero-state-linearized version of the target plant. This model, with state vector $\bar{x}$, is stabilized using a fixed gain state-feedback law $u = \bar{K}\bar{x}$. The controller gain $\bar{K}$ is generated as the solution to the well-known, infinite-time horizon Linear Quadratic Regulator problem. Utilizing an optimal state-feedback method allows for quick-adjustments to the model's convergence characteristics through the cost matrices Q and R. The stabilized reference model state dynamics take the following standard form:

$$\dot{\bar{x}} = (A\bar{x} + B\bar{K})\bar{x} \tag{3.1}$$

The main goal of employing the neuro-compensator mechanism is to extend the effectiveness of the L.T.I. state-feedback control law to the non-linear plant. Moreover, it is desired that the state feedback controller stabilizes the plant for system configurations well outside the small region about the selected equilibrium point where the reference-model dynamics are assumed to be a semi-accurate approximation for the actual plant dynamics. By observing state-output differences between the plant and the reference model,the neuro-compensator will attempt to cancel unmodeled nonlinear effects and coerce the plant to behave like its reference-model counterpart in an on-line, adaptive fashion.

In realizing the control-law at hand, the unmodeled plant dynamics (which may also include the effects of process and sensor noise) are subsumed in a parameter $\Delta A(t)$, which represents dynamical variations between the reference-model and the plant. Hence, the plant dynamics are assumed to take the following form:

$$\dot{x} = (A + \Delta A(t))x + Bu \qquad \Delta A \in \mathbb{R}^4 \tag{3.2}$$

The neuro-compensator mechanism, denoted $\mathcal{N}(x, \bar{x})$, is iteratively adapted using B.P. to cancel the variation $\Delta A(t)$, to yield an approximate linearization of the plant dynamics. The incorporation of the neuro-compensator into the assumed system dynamics with fixed-gain state-feedback can be realized like so:

$$\Delta A(t)x - B\hat{G}\mathcal{N}(x, \bar{x}) \approx 0 \qquad \mathcal{N}(x, \bar{x}) : \mathbb{R}^4 \to \mathbb{R}^1 \tag{3.3}$$

$$\dot{x} = (A + \Delta A(t) + B\bar{K})x - B\hat{G}\mathcal{N}(x, \bar{x}) \approx (A + B\bar{K})x \tag{3.4}$$

It can be seen in *Equation 3.4* that the full input to the plant will be a linear combination of neural-compensator output, $\mathcal{N}(x, \bar{x})$, and fixed-gain state-feedback:

$$u = -\hat{G}\mathcal{N}(x, \bar{x}) + \bar{K}x \tag{3.5}$$

3.2 Network Update Rule

As previously mentioned, the neuro-compensator should be trained such that for an observed, $\{x, \bar{x}\}$ pair, $\mathcal{N}(x, \bar{x})$ will produce a corresponding dynamical-compensation signal to cancel $\Delta A(t)$. This signal is learned by mapping the extended state-output, $[x, \bar{x}]^T$ to some, desired point-wise output $\phi_{\mathcal{N}}(x, \bar{x})$. The formulation of $\phi_{\mathcal{N}}(x, \bar{x})$ is based upon the output tracking error between the reference model and the plant, denoted ϵ. Because both the reference model and the actual plant yield full-state output, ϵ can be represented as follows:

$$\epsilon = y - \bar{y} = Cx - C\bar{x} = I(x - \bar{x}) \tag{3.6}$$

The tracking error, ϵ, and plant output, x, can be represented as an extended state $[x, \epsilon]^T$. The dynamics of this extended state can be realized in terms of x and $\bar{x}$ with the variation $\Delta A(t)$ replaced by the gain-multiplied, point-wise training signal $B\hat{G}\phi_{\mathcal{N}}(x, \bar{x})$. The following representation does not include $\mathcal{N}(x, \bar{x})$, as $\phi_{\mathcal{N}}(x, \bar{x})$ should be representative of the dynamical variations present when the compensator is not active (i.e. the compensation that $\mathcal{N}(x, \bar{x})$ should apply):

$$\begin{bmatrix} \dot{x} \\ \dot{\epsilon} \end{bmatrix} = \begin{bmatrix} I & 0 \\ I & -I \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{\bar{x}} \end{bmatrix} = \begin{bmatrix} A + B\bar{K} & 0 \\ A + B\bar{K} & -A - B\bar{K} \end{bmatrix} \begin{bmatrix} x \\ \bar{x} \end{bmatrix} + \begin{bmatrix} \hat{G}B \\ \hat{G}B \end{bmatrix} \phi_{\mathcal{N}}(x, \bar{x}) \tag{3.7}$$

The above equation is then re-arranged to form the desired network output equation:

$$\frac{\mathcal{B}^T}{\|\mathcal{B}\|_2^2 \hat{G}} \begin{bmatrix} I & 0 \\ I & -I \end{bmatrix} \left(\begin{bmatrix} \dot{x} \\ \dot{\bar{x}} \end{bmatrix} + \begin{bmatrix} \mathcal{D}_0 & 0 \\ 0 & \mathcal{D}_0 \end{bmatrix} \begin{bmatrix} x \\ \bar{x} \end{bmatrix} \right) = \phi_{\mathcal{N}}(x, \bar{x}) \tag{3.8}$$

$$\mathcal{B} = [B, B]^T \qquad \mathcal{D}_0 = -A - B\bar{K}$$

Since $\dot{x}$ and $\dot{\bar{x}}$ are not explicitly known for all states, a derivative approximation mechanism is necessary. For the controller at hand, a first order approximation for the state derivatives $\dot{x_3}, \dot{x_3}, \dot{\bar{x}_3}$ and $\dot{\bar{x}_4}$ will suffice, provided that sensor feedback is adequately low-passed. $\dot{x_1}, \dot{x_2}, \dot{\bar{x}_1}$, and $\dot{\bar{x}_2}$ are acquired directly from state feedback.

Approximations for the full state derivatives, $\dot{x_a}$ and $\dot{\bar{x}_a}$, are realized as follows:

$$\begin{bmatrix} \dot{x_a} \\ \dot{\bar{x}_a} \end{bmatrix} = \begin{bmatrix} \mathcal{D}_1 & 0 \\ 0 & \mathcal{D}_1 \end{bmatrix} \begin{bmatrix} x(t) \\ \bar{x}(t) \end{bmatrix} - \begin{bmatrix} \mathcal{D}_2 & 0 \\ 0 & \mathcal{D}_2 \end{bmatrix} \begin{bmatrix} x(t-\Delta t) \\ \bar{x}(t-\Delta t) \end{bmatrix} + E_{\Delta t} \tag{3.9}$$

Assuming that Δt is sufficiently small, the approximation error term, $E_{\Delta t}$, is neglected. Matrices $\mathcal{D}_1$ and $\mathcal{D}_2$ are defined by:

$$\mathcal{D}_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \Delta t^{-1} & 0 \\ 0 & 0 & 0 & \Delta t^{-1} \end{bmatrix} \quad \mathcal{D}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta t^{-1} & 0 \\ 0 & 0 & 0 & \Delta t^{-1} \end{bmatrix} \quad \Delta t > 0 \tag{3.10}$$

Utilizing the above derivative approximation and the network output equation, the network training-signal, $\phi_{\mathcal{N}}(x, \bar{x})$ becomes:

$$\frac{\mathcal{B}^T}{\|\mathcal{B}\|_2^2 \hat{G}} \left(\begin{bmatrix} \mathcal{D}_0 + \mathcal{D}_1 & 0 \\ \mathcal{D}_0 + \mathcal{D}_1 & -\mathcal{D}_0 - \mathcal{D}_1 \end{bmatrix} \begin{bmatrix} x(t) \\ \bar{x}(t) \end{bmatrix} - \begin{bmatrix} \mathcal{D}_2 & 0 \\ \mathcal{D}_2 & -\mathcal{D}_2 \end{bmatrix} \begin{bmatrix} x(t-\Delta t) \\ \bar{x}(t-\Delta t) \end{bmatrix} \right) = \phi_{\mathcal{N}}(x, \bar{x}) \tag{3.11}$$

The above training signal is utilized in concert with B.P. algorithm to update the compensator network.

3.3 The Neuro-compensator

3.3.1 Structure and Foward Activation

As illustrated in *Figure 3.1*, the neuro-compensator takes the form of a multistage system of summed non-linear basis functions. Connections between nodes are represented as weighted edges. Edge connections are represented by a set of connection matrices $\mathcal{W} = \{W^{1,2}, W^{2,3}\}$

The output of each neuron is generated as a weighted sum of inputs through a sigmoid-step activation function. For the controller at hand, a hyperbolic-tangent function is used as an activation for each neuron because of its bipolar output capabilities. The output of each q^{th} neuron on the i^{th} layer is written like so:

$$o_q^i = \tanh\left(\eta y_q^i\right) = \tanh\left(\eta \sum_{j=0}^{N} w_{j,q,k}^{i,i-1} o_j^{i-1}\right) \quad \eta > 0 \tag{3.12}$$

In the above equation, η represents the "activation temperature," which governs the output sensitivity of each neuron.

3.3.2 Back-Propagation Algorithm

The B.P. algorithm is an approximate gradient-descent algorithm used to update the weights of each feed-forward layer of neurons [6]. Since the network is updated on the basis of network output error with respect to some target value, errors must be translated backwards through the network, as target ouput values for hidden layers are not explicity known. The back-propagation algorithm used for feed-forward networks is scalable to a network of N many layers. This allows the network to be scalable to the complexity of the function-representation at hand without any extra formulation.

In the context of the control-law at hand, B.P. addresses the following quadratic minimization problem:

$$\min_{\mathcal{W}} \|\phi_{\mathcal{N}}(x,\bar{x}) - \mathcal{N}(x,\bar{x})\|_2^2 \quad \text{where} \quad \mathcal{W} = \left\{W^{1,2}, W^{2,3}\right\} \tag{3.13}$$

Following from the previously stated problem, the update factor for each edge-weighting element is formulated as follows:

$$\Delta w^i_{j,q,k} = -\gamma \frac{\partial o^i_j}{\partial w^i_{j,q,k}} e^i_j o^{i-1}_q + \mu \Delta w^i_{j,q,k-1} \qquad \gamma \in [0,1] \qquad \mu \in [0,1] \tag{3.14}$$

The coefficient $\mu \ll 1$ controls the amount of the previous reweighing factor $\Delta w^i_{j,q,k-1}$ incorporated in the current update step. This is known as BP-*momentum* and has been shown to speed up network convergence [2]. The coefficient γ represents a fixed network learning rate. $\frac{\partial o_q}{\partial w^n_{j,q}}$ is the gradient of the output signal with respect to each weighting element. Because a bi-polar activation function is being employed, the output gradient for each j^{th} output can be realized like so:

$$\nabla_y \mathcal{S}(y^i) = \text{diag}\left\{\mathcal{S}(y^i)\right\}(\vec{1} - \mathcal{S}(y^i)) \tag{3.15}$$

$$\delta^j = \frac{\partial o^i_j}{\partial w_{j,q,k}} = (\nabla_y \mathcal{S}(y^i))_j e^i_j \tag{3.16}$$

The simplicity of the gradient of the vectorized activation function, $\mathcal{S}(y^i)$, shown in *Equation 3.15*, makes it a favorable selection as the network's non-linear basis element.

The backward-translated error for each hidden layer is generated as follows:

$$e^n_j = \sum_{q=0}^{N} w^{n+1}_{j,q,k} \delta^{n+1}_j \tag{3.17}$$

3.4 Alternative Recursive Least-Squares Compensator

For the sake of comparison, a recursive least squares (R.L.S.) approach will be utilized as a drop-in replacement for the neuro-compensator. The least-squares estimator will perform the same task as the neural network in canceling $\Delta A(t)$, by generating an approximate approximate fit for $n-$samples of the network output training function $\phi_{\mathcal{N}}(x,\bar{x})$ over an associate sampling of the extended-states $[x,\bar{x}]^T$.

The estimator will be denoted: $\mathbb{L}(x,\bar{x})$ for the k^{th} aquired sample is defined as [1]:

$$\mathbb{L}(x,\bar{x}) = \hat{W} \begin{bmatrix} x \\ \bar{x} \end{bmatrix} \qquad \mathbb{L}(x,\bar{x}) : \mathbb{R}^4 \to \mathbb{R}^1 \qquad \hat{W} \in \mathbb{R}^{1\times 8} \tag{3.18}$$

The weights $\hat{W}$ are updated at each iteration by some $\lambda \Delta \hat{W}$, where λ is a step-size factor (dubbed the "forgetting factor"). In this approach, the controller will update itself over windows of samples. By setting $\lambda \in [0,1]$, the rate at which new sets of weights are incorporated and older weights are forgotten can be adjusted. The update law for $\hat{W}$ is formulated as a minimization of the cost function:

$$\min_{\mathcal{W}} \ \|\Phi_k - X_k \mathcal{W}_k\|^2 \tag{3.19}$$

In this formulation, Φ represents a vector of $n-$past samples of $\phi_{\mathcal{N}}$, namely: $\Phi = [\phi_{\mathcal{N},k-1}, \phi_{\mathcal{N},k-2}, ..., \phi_{\mathcal{N},k-n}]^T$. X is a $n \times 8$ vector of corresponding samples of $[x, \bar{x}]^T$. $\mathcal{W}_k$ is a set of update weights which are incoorperated in the update step as $\Delta \hat{W}_k = \hat{W}_k - \mathcal{W}_k$.

Minimzation over $\mathcal{W}_k$, leads to the following least-squares upate step:

$$\mathcal{W}_k = (X_k X_k^T + Iv)^{-1} X_k \Phi_k \tag{3.20}$$

The regularization term Iv is added to the inversion of $(X_k X_k^T)^{-1}$ to ensure that the matrix being inverted is always full rank. The iterated update of the weights $\hat{W}_k$ is formulated in-full as:

$$\hat{W}_{k+1} = \hat{W}_k + \lambda(\mathcal{W}_k - \hat{W}_k) \tag{3.21}$$

$$\hat{W}_{k+1} = \hat{W}_k + \lambda((X_k X_k^T + Iv)^{-1} X_k \Phi_k - \hat{W}_k) \tag{3.22}$$

The approximation of $\Delta A(t)$ utilized by the controller is therefore:

$$\Delta A(t) \approx \hat{G} B \mathbb{L}(x, \bar{x}) = \hat{G} B \hat{W} \begin{bmatrix} x \\ \bar{x} \end{bmatrix} \tag{3.23}$$

To clarify, $\hat{G}$ is the same fixed gain used in scaling the neuro-compensator output.

4 | Simulation and Results

4.1 Testing Conditions

For the set of results presented, LQR cost matrices $Q = 10I$ and $R = 1$ were selected to promote faster reference model state convergence. The neuro-compensator was outfitted with 80 hidden neurons and updated with a learning rate of $\gamma = 0.3$ For a unique set of initially randomized network weights, the controller is trained over a series of initial configurations, x_o, parameterized like so:

$$x_o = [q_1(0), 0, 0, 0]^T \qquad q_1(0) \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right] \tag{4.1}$$

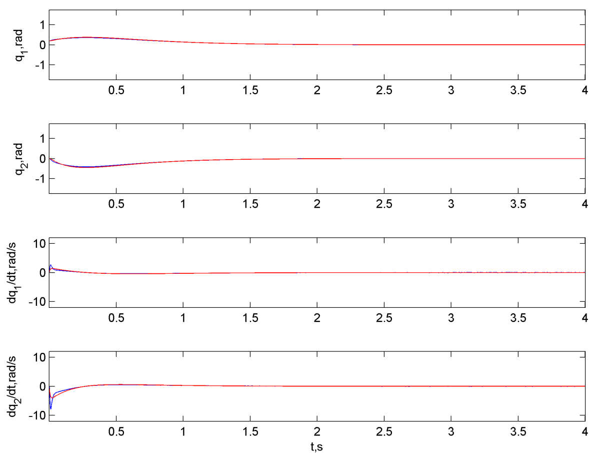
For each i^{th} successive training episode, $|q_1(0)^i| > |q_1(0)^{i-1}|$, so as to incrementally introduce the controller to state configurations further from the zero-state. The restriction of $q_1(0) \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ has been imposed as a result of inherent controller limitations revealed during testing. Repeated trials showed that for $|q_1(0)| > \frac{\pi}{4}$, the controller was insufficient for enforcing system stability.

For the set of simulations performed using the recursive-least squares appoach, a sampling window of $n = 100$ is utilized. The R.L.S. coefficients are set to $\lambda = 0.5$ and $\upsilon = 1 \times 10^{-4}$.

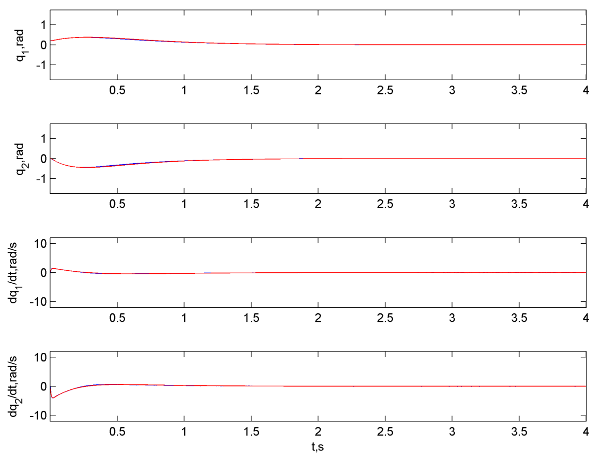
** A list of related notations and model settings has been included in* ***Appendix A***

4.2 Controller Performance as Compared to State Feedback

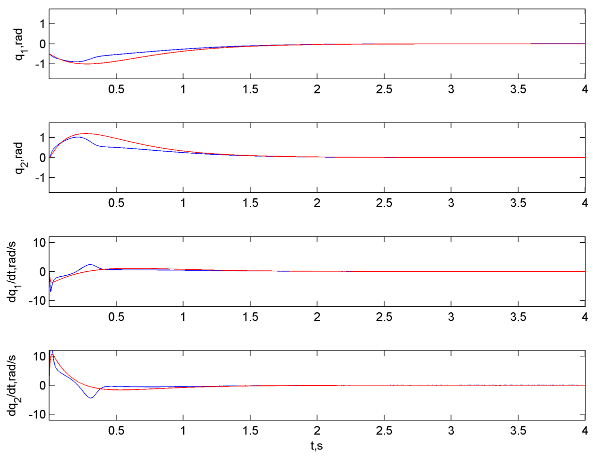
A simulation of the system for an initial condition relatively close to the zero-state yields similar results for the Neuro-MRAC controller and the pure state-feedback controller:



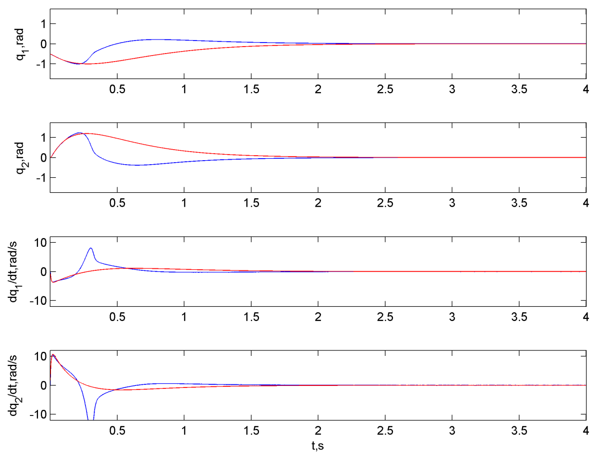
(a) Neuro-MRAC Controller Results; $q_1(0) = 11.25^o$



(b) State-Feedback Controller Results; $q_1(0) = 11.25^o$

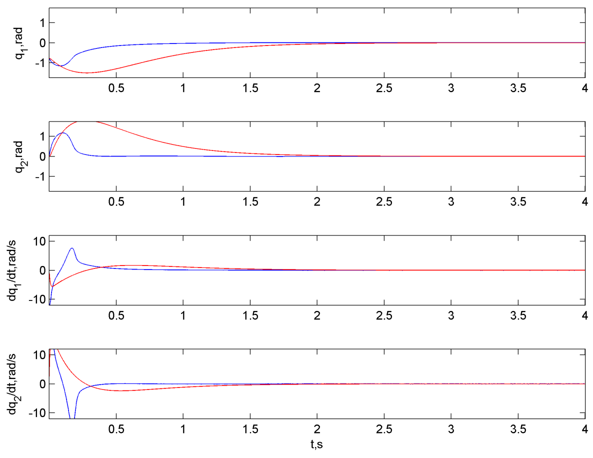


(a) Neuro-MRAC Controller Results; $q_1(0) = -30^o$

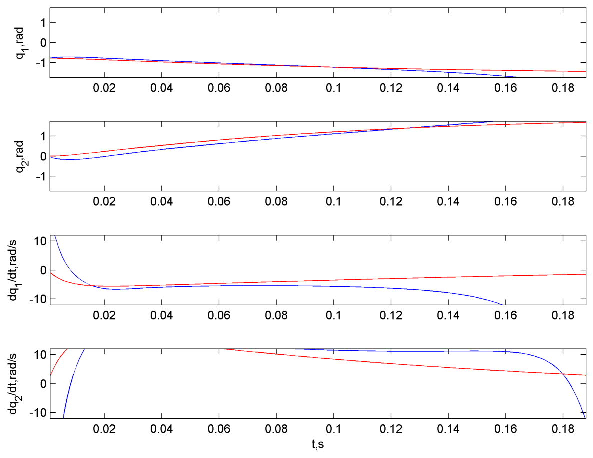


(b) State-Feedback Controller Results; $q_1(0) = -30^o$

Further trials clearly show that the Neuro-MRAC controller out-performs pure state-feeback, which was unsuccessful in stabilizing the system for the final set of initial conditions.



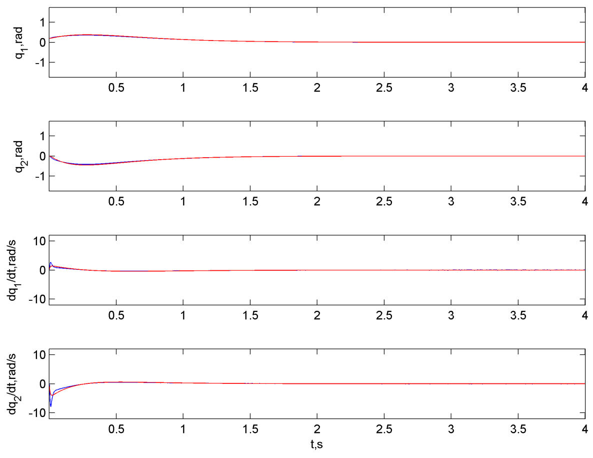
(a) Neuro-MRAC Controller Results; $q_1(0) = -45^o$



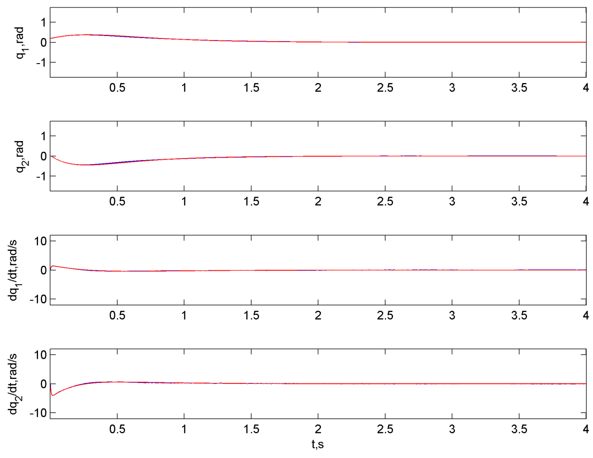
(b) State-Feedback Controller Results; $q_1(0) = -45^o$

4.3 Controller Performance as Compared to RLS-MRAC Approach

A simulation of the system for an initial condition relatively close to the zero-state yields similar results for the Neuro-MRAC controller and the RLS-MRAC controller:

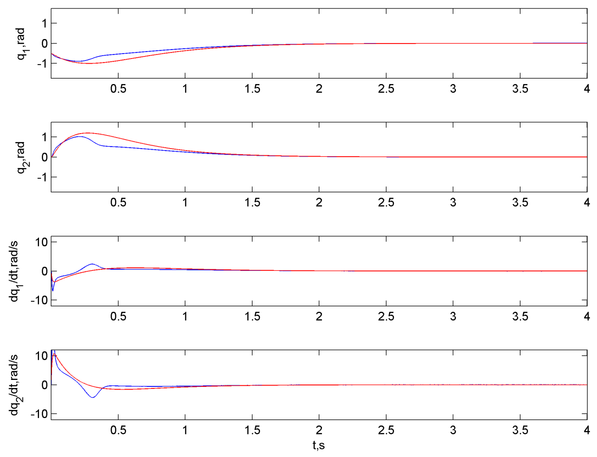


(a) Neuro-MRAC Controller Results; $q_1(0) = 11.25^o$

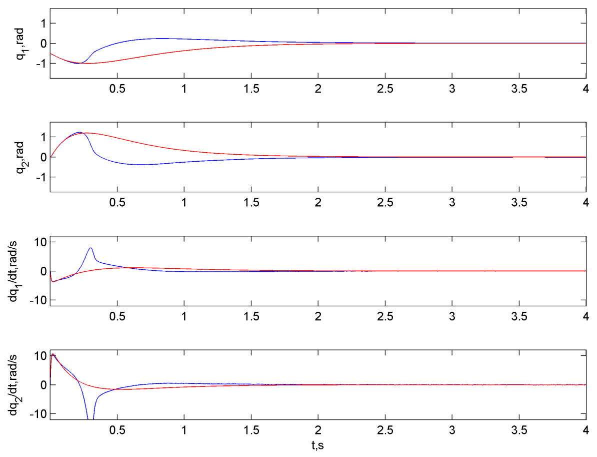


(b) RLS-MRAC Controller Results; $q_1(0) = 11.25^o$

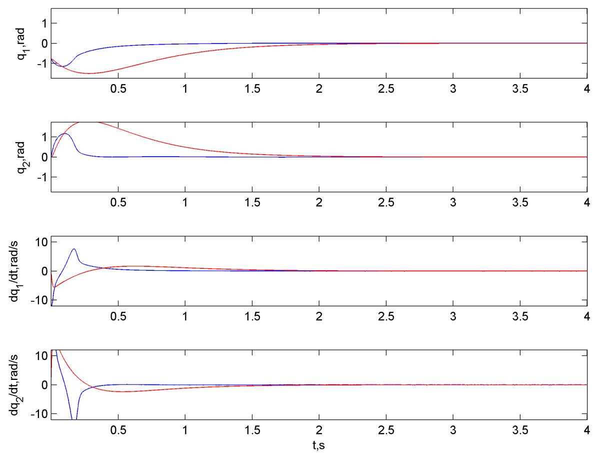
Further trials clearly show that the Neuro-MRAC controller out-performs RLS-MRAC, which was unsuccessful in stabilizing the system for the final set of initial conditions.



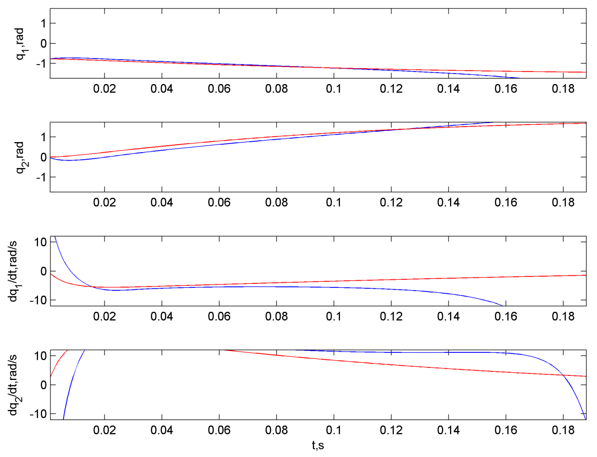
(a) Neuro-MRAC Controller Results; $q_1(0) = -30^o$



(b) RLS-MRAC Controller Results; $q_1(0) = -30^o$



(a) Neuro-MRAC Controller Results; $q_1(0) = -45^o$



(b) RLS-MRAC Controller Results; $q_1(0) = -45^o$

5 Conclusions

Based on empirical evidence generated in this study, the use of a feed-forward neuro-compensator has been shown to extend the apititude of a state-feedback controller when applied to the nonlinear system at hand. As originally surmised, a state feedback controller, on its own, could not stabilize the target system for substantial deviations from the zero-state.

It can also be concluded that a feed-foward neural network is the appropriate choice of function appoximation, as it can adequately model highly nonlinear functions. By comparison, a linear-weighted, R.L.S method of function approximation does not perform this task sufficiently.

Appendices

Appendix A

Summary of Model Notations and Simulation Settings

q_1	Angular position of base-joint (Joint-1)
q_2	Angular position of free-joint (Joint-2)
a_1	Length of first linkage
a_2	Length of second linkage
m_1	Mass of first linkage
m_2	Mass of second linkage
τ_1	Torque applied to Joint-1
τ_2	Torque applied to Joint-2
C_1	$\cos(q_1)$
S_1	$\sin(q_1)$
C_{12}	$\cos(q_1 + q_2)$
S_{12}	$\sin(q_1 + q_2)$
β_1	$a_1/2$
β_2	$a_2/2$
ν_1	Viscous friction of free joint
ν_2	Viscous friction of actuated joint
k_{emf}	Motor back-EMF gain
k_t	Motor current-to-torque gain
R_m	Internal motor resistance

Parameter	**Simulation Value**
a_1	$0.400m$
a_2	$0.400m$
m_1	$0.200kg$
m_2	$0.200kg$
ν_1	$0.0001\frac{N.m.s}{rad}$
ν_2	$0.0001\frac{N.m.s}{rad}$
k_{emf}	$0.0001\frac{V.s}{rad}$
k_t	$5.0\frac{N.m}{A}$
R_m	10Ω
Δt	$1ms$

References

[1] Girish Chowdhary and Eric N. Johnson. Least squares based modification for adaptive control. In *CDC*, pages 1767–1772. IEEE, 2010.

[2] G.P. Drago, M. Morando, and S. Ridella. An adaptive momentum back propagation (ambp). *Neural Computing and Applications*, 3(4):213–221, 1995.

[3] K. V. Lakshmi and Mashuq un Nabi. An adaptive neuro-fuzzy control approach for motion control of a robot arm. In *Informatics, Electronics Vision (ICIEV), 2012 International Conference on*, pages 832–836, May 2012.

[4] G. Lightbody and G.W. Irwin. Direct neural model reference adaptive control. *Control Theory and Applications, IEE Proceedings -*, 142(1):31–43, Jan 1995.

[5] Xiang-Jie Liu, Felipe Lara-Rosano, and C. W. Chan. Model-reference adaptive control based on neurofuzzy networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 34(3):302–309, 2004.

[6] Raul Rojas. The backpropagation algorithm. In *Neural Networks - A Systematic Introduction*, pages 151–184. Springer-Verlag, 1996.