JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 2, April 2005

A Reinforcement Learning - Adaptive Control Architecture for Morphing

John Valasek,[*] Monish D. Tandale,[†] and Jie Rong[‡]
Texas A&M University, College Station, Texas 77843-3141

This paper develops a control methodology for morphing, which combines Machine Learning and Adaptive Dynamic Inversion Control. The morphing control function, which uses Reinforcement Learning, is integrated with the trajectory tracking function, which uses Structured Adaptive Model Inversion Control. Optimality is addressed by cost functions representing optimal shapes corresponding to specified operating conditions, and an episodic Reinforcement Learning simulation is developed to learn the optimal shape change policy. The methodology is demonstrated by a numerical example of a 3-D morphing air vehicle, which simultaneously tracks a specified trajectory and autonomously morphs over a set of shapes corresponding to flight conditions along the trajectory. Results presented in the paper show that this methodology is capable of learning the required shape and morphing into it, and accurately tracking the reference trajectory in the presence of parametric uncertainties and initial error conditions.

Nomenclature

α	learning rate
β	positive step-size parameter
δ	temporal difference error
$\boldsymbol{\varepsilon}$	tracking error for linear states
$\boldsymbol{\epsilon}$	tracking error for angular states
γ	discount factor
π	reinforcement learning policy
$\Phi_j(s)$	basis function
ρ	density of air
$\boldsymbol{\sigma} = [\phi \quad \theta \quad \psi]^T$	3-2-1 Euler angles that describe the orientation of the body axes relative to the inertial axes
$\boldsymbol{\sigma}_r$	Euler angles for the reference trajectory
θ_{pa}	parameter vector for approximating the preference function
θ_v	parameter vector for approximating the value function
$\boldsymbol{\Theta}$	true inertia parameter vector
$\hat{\boldsymbol{\Theta}}$	adaptive inertia parameter vector

Received 18 March 2001; revision received 30 July 2003; accepted for publication 26 August 2003.

[*] Associate Professor & Director, Flight Simulation Laboratory, Aerospace Engineering Department. Associate Fellow, AIAA. valasek@tamu.edu, http://jungfrau.tamu.edu/valasek/

[†] Graduate Research Assistant, Flight Simulation Laboratory, Aerospace Engineering Department. Student Member, AIAA. monish@neo.tamu.edu

[‡] Graduate Research Assistant, Flight Simulation Laboratory, Aerospace Engineering Department. Student Member, AIAA. jierong@tamu.edu

$\tilde{\boldsymbol{\Theta}}$	error between adaptive and true inertia parameter vector
$\boldsymbol{\tau}$	6×6 symmetric positive definite matrix
Γ	positive scalar constant
$\boldsymbol{\omega} = [p \quad q \quad r]^T$	angular velocity of the smart block along the body axes
$\tilde{\boldsymbol{\omega}}$	matrix representation of the cross product between $\boldsymbol{\omega}$ and a compatible vector
$A(s_t)$	set of actions available in state s_t
c_j	preselected center states vector
C_ϕ	$\cos(\phi)$
C_{da}, K_{da}	design matrices
C_y, C_z	center state vectors
f	flight condition
F	control force
F_d	drag force
H	basis vector for function approximation
I	mass moments of inertia of the smart block in the body axes
J_l	nonlinear transformation relating $\dot{\mathbf{p}}_c$ and $\mathbf{v}_c$
J_a	nonlinear transformation relating $\dot{\boldsymbol{\sigma}}$ and $\boldsymbol{\omega}$
J	cost function
J_y	cost function component corresponding to the y dimension
J_z	cost function component corresponding to the z dimension
m	mass of the smart block
$\hat{m}$	adaptive parameter that learns the mass
M	control moment
M_d	drag moment
$\mathbf{p}_c = [X \quad Y \quad Z]^T$	position of the center of mass of the smart block along the inertial axes
$p(s, a)$	preference function: tendency to select action a at state s
R	reward
$\mathcal{R}$	weighting matrix
S	set of possible states for reinforcement learning
S_θ	$\sin(\theta)$
$S_y(f)$	optimal y dimension at flight condition f
$S_z(f)$	optimal z dimension at flight condition f
$\mathbf{v}_c = [u \quad v \quad w]^T$	linear velocity of the smart block along the body axes
V	Lyapunov function
$V^\pi(s)$	state value function for policy π
V_y	voltage control for the y dimension
V_z	voltage control for the z dimension
$[X_b Y_b Z_b]^T$	body axes
$[X_N Y_N Z_N]^T$	inertial axes
$[x \quad y \quad z]^T$	dimension of the smart block along the body axes
$Y_a(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}})$	regression matrix

I. Introduction

CURRENT interest in morphing vehicles has been fuelled by advances in smart technologies, including materials, sensors, actuators, and their associated support hardware and microelectronics. Morphing research has led to a series of breakthroughs in a wide variety of disciplines that, when fully realized for aircraft applications, have the potential to produce large increments in aircraft system safety, affordability, and environmental compatibility.[1] Although there are several definitions and interpretations of the term morphing, it is generally agreed that the concept refers to large scale shape changes or transfigurations. Various organizations which are researching morphing technologies for both air and space vehicles have adopted their own definitions according to their needs. The National Aeronautics and Space Administration's (NASA) Morphing Project defines morphing as an efficient, multi-point adaptability that includes macro, micro, structural and/or fluidic approaches.[2] The Defense Advanced Research Projects Agency (DARPA) uses the definition of a platform that is able to change its state substantially (on the order of 50%) to adapt to changing mission environments, thereby providing a superior system capability that is not possible without reconfiguration. Such a design integrates innovative combinations of advanced materials, actuators, flow controllers, and mechanisms to achieve the state change.[3]

In spite of their relevance, these definitions do not adequately address or describe the supervisory and control aspects of morphing. Reference 4 attempts to do so by making a distinction between Morphing for Mission Adaptation, and Morphing for Control. In the context of flight vehicles, Morphing for Mission Adaptation is a large scale, relatively slow, in-flight shape change to enable a single vehicle to perform multiple diverse mission profiles. Conversely, Morphing for Control is an in-flight physical or virtual shape change to achieve multiple control objectives, such as maneuvering, flutter suppression, load alleviation and active separation control. In this paper, the authors consider the problem of Morphing for Mission Adaptation.

In the context of intelligent systems, three essential functionalities of a practical Morphing for Mission Adaptation capability are:

1. When to reconfigure
2. How to reconfigure
3. Learning to reconfigure

When to reconfigure is driven by mission priorities/tasks, and leads to optimal shape being a system parameter. In the context of a reconfigurable vehicle such as an aircraft, each shape results in performance values (speed, range, endurance, etc.) at specific flight conditions (Mach number, altitude, angle-of-attack, and sideslip angle). It is a major issue, as the inability for a given aircraft to perform multiple missions successfully can directly be attributed to shape, at least if aerodynamic performance is the primary consideration. This is because for a given task or mission, there is usually an ideal or optimal vehicle shape, e.g. configuration.[4] However, this optimality criteria may not be known over the entire flight envelope in actual practice, and the mission may be modified or completely changed during operation. *How* to reconfigure is a problem of sensing, actuation, and control.[5] They are important and challenging since large shape changes produce time-varying vehicle properties, and especially, time-varying moments and products of inertia. The controller must therefore be sufficiently robust to handle these potentially wide variations. *Learning* to reconfigure is perhaps the most challenging of the three functionalities, and the one which has received the least attention. Even if optimal shapes are known, the actuation scheme(s) to produce them may be only poorly understood, or not understood at all; Reinforcement Learning is therefore a candidate approach. It is important that learning how to reconfigure is also life-long learning. This will enable the vehicle to be more survivable, operate more safely, and be multi-role.

This paper proposes and develops a conceptual control architecture which addresses these essential functionalities of Morphing for Mission Adaptation. Called Adaptive-Reinforcement Learning Control (A-RLC), it is a marriage of Adaptive Dynamic Inversion Control and Machine Learning. By comparison to morphing research reported in the literature which focus on structures and actuation, A-RLC addresses the optimal shape changing of an entire vehicle. A-RLC learns the commands which produce the optimal shape, defined as a function of operating condition, while maintaining accurate trajectory tracking. A-RLC uses Structured Adaptive Model Inversion (SAMI) as the trajectory tracking controller for handling time-varying properties, parametric uncertainties, and disturbances. For learning the optimality relations between the operating conditions and the shape, and learning how to produce the optimal shape at every operating condition over the life of the vehicle, A-RLC uses Reinforcement Learning. Optimality is addressed with cost functions which penalize deviations from the optimal shape. The paper first defines the shape changing dynamics of a hypothetical three-dimensional Smart Block air vehicle of constant volume, which can morph in all the three spatial dimensions. The Reinforcement Learning module of A-RLC is developed next, and uses an actor-critic method to learn how to morph into specified shapes. This is followed by development of the SAMI control module, which handles Smart Block parametric uncertainties, and initial error conditions, while tracking a trajectory. The A-RLC methodology is demonstrated with a numerical example of the Smart Block air vehicle autonomously morphing over a set of optimal shapes, corresponding to specified flight conditions, while tracking a specified trajectory.

II. Morphing Smart Block Simulation

A smart block in the form of a rectangular parallelopiped as shown in Fig. 1, represents the morphing air vehicle. The morphing used in this research involves a change in the dimensions of the rectangular parallelopiped, while maintaining a total volume of 32 units. The Reinforcement Learning module specifies the y and z dimensions, corresponding to the current flight condition and the x dimension is calculated by enforcing the constant volume condition, $x = \frac{32}{yz}$. It is assumed that the smart block is composed of a smart material (Piezoelectric, Shape Memory Alloy (SMA) or Carbon Nanotubes) whose shape can be modulated by applying voltage. For simulation purposes, we assume a second order spring mass damper kind of a model with a nonlinear spring function. The model for relating the y dimension to the applied V_y voltage is given in Eq. 1

$$\ddot{y} + 2.5\dot{y} + 2.5y + 0.4sin(\pi(y-2)) - 5 = V_y \tag{1}$$

and similarly for the z dimension

$$\ddot{z} + 1.8\dot{z} + 2z + 0.6(z-2)(z-4) - 4 = V_z \tag{2}$$

The above dynamic models render a nonlinear relationship between the steady-state dimension and applied voltage, as shown in Fig. 2. Also, Fig. 3 shows how the y-dimension and the z-dimension evolve for various constant inputs, with varied initial conditions. Note that the morphing dynamics are highly nonlinear.The Reinforcement Learning module records the cost incurred by applying the various possible voltages. Before measuring the cost associated with a particular action, the new commanded shape should be allowed to reach steady-state. For the present system the response is over damped for some cases, and overshoots for others, but all trajectories reach steady-state within ten seconds. Thus the time interval between commanded morphing should be at least ten seconds. Note that the coefficients in Eq. 1 and Eq. 2 are selected arbitrarily to form a conceptual model for the morphing dynamics and do not represent a specific material.

The Cost Function is selected to be a function of the current shape and the current flight condition. The current shape can be identified uniquely from the y and z dimensions of the smart block. The current flight condition is identified by discrete values from 0 to 5, so the Cost Function $J\colon [2,4] \times [2,4] \times [0,5] \to [0,\infty]$

The total cost function J can be written as a sum $J = J_y + J_z$. The optimal shapes are arbitrarily selected to be nonlinear functions of the flight condition and are shown graphically in Fig. 4.

$$S_y = 3 + \cos\left(\frac{\pi}{5} f\right) \tag{3}$$

$$S_z = 2 + 2e^{-0.5f} \tag{4}$$

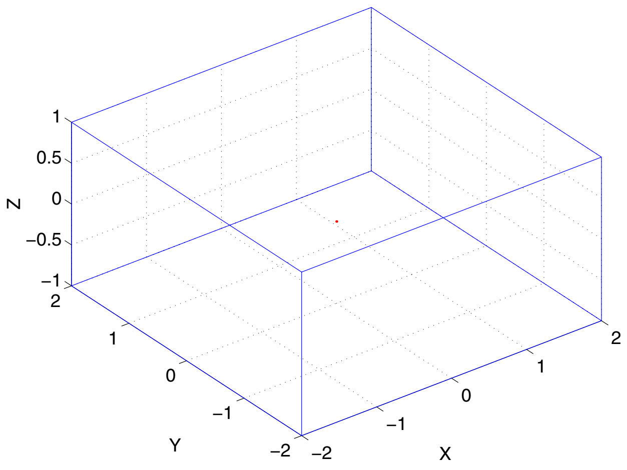

Fig. 1 Shape of the morphing smart block.

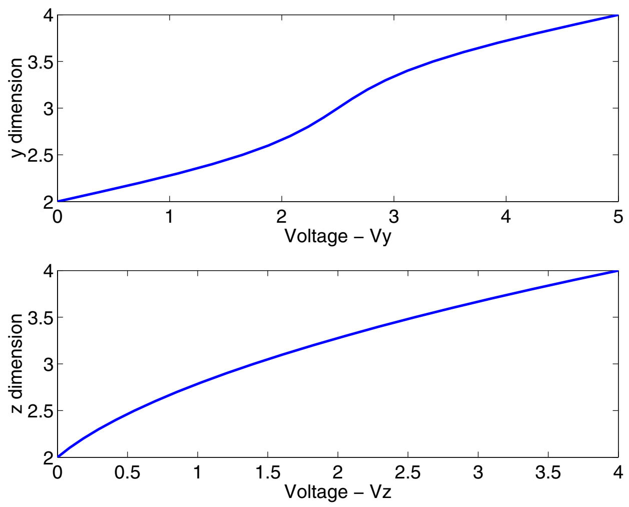

Fig. 2 The Steady State Dimensions corresponding to the Applied Voltage.

With the optimal y dimension given by Eq. 3, the cost function J_y is defined as

$$J_y = (y - S_y(f))^2 \tag{5}$$

The surface plot of J_y is shown in Fig. 5. Similarly, the cost function J_z is defined as

$$J_z = (z - S_z(f))^2 \tag{6}$$

The surface plot of J_z is shown in Fig. 5. For the simulation we specify the flight conditions that the smart block should fly at various locations along a pre-designated flight path (Fig. 6). For this preliminary example the optimal shapes are not correlated to the flight path, but depend only on the flight condition.

The objective of the Reinforcement Learning Module is to learn the control policy that, for a specific flight condition, commands the voltage which produces the optimal shape S_y and S_z. Therefore, the Reinforcement Learning module seeks to minimize the total cost J over the entire flight trajectory.

A. Mathematical Model for the Dynamic Behavior of the Smart Block

The dynamic behavior of the smart block is like a hovering air vehicle. The simulation assumes absence of gravity. The vehicle has thrusters along all three axis which act as the control effectors. Note that the total velocity vector of the vehicle can point in any arbitrary direction.

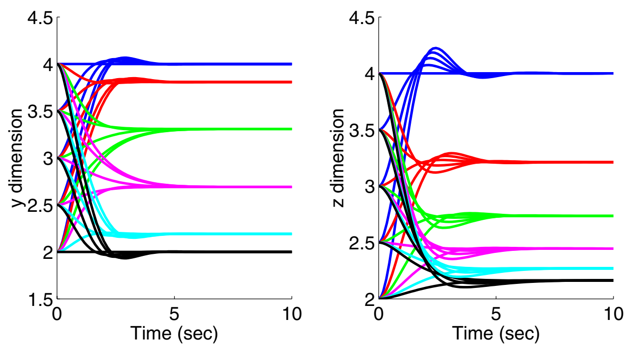

Fig. 3 The Morphing Dynamics for y and z dimensions when subjected to various input voltages Vy and Vz.

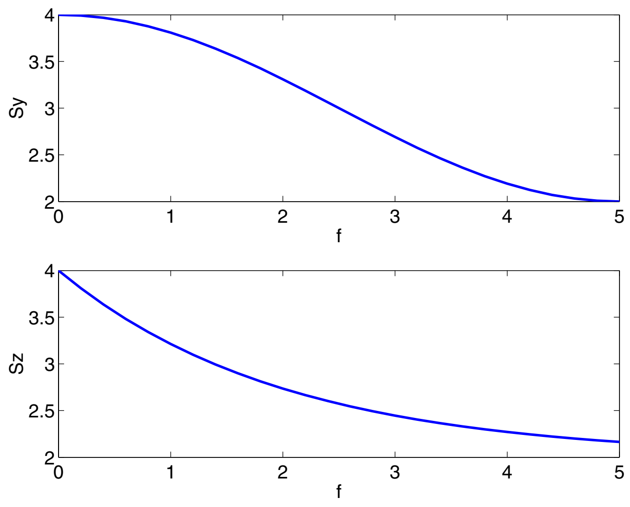

Fig. 4 Optimal Shapes at the different Flight Conditions.

The dynamic behavior of the smart block is modeled by nonlinear six degree-of-freedom equations.[6] The equations are written in two coordinate systems: an inertial axis system, and a body fixed axis with origin at the center of mass of the smart block.

The structured model approach is followed and the dynamic equations are partitioned into 'kinematic level' and 'acceleration level' equations. The kinematic states and the acceleration states are related by the differential equations

$$\dot{\mathbf{p}}_c = J_l \mathbf{v}_c \tag{7}$$

$$\dot{\boldsymbol{\sigma}} = J_a \boldsymbol{\omega} \tag{8}$$

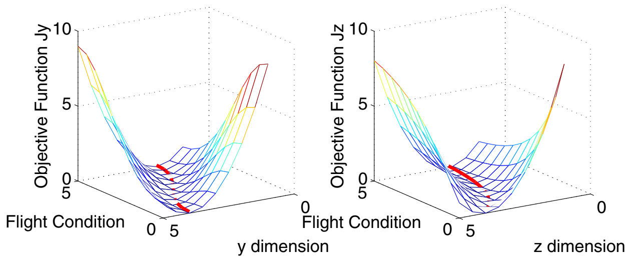

Fig. 5 Cost Function Components J_y & J_z.

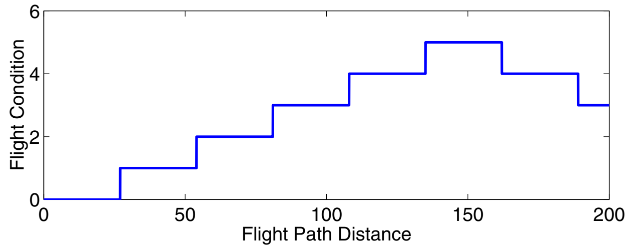

Fig. 6 Flight Condition at various Flight Path Locations.

where

$$J_l = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix}$$

$$J_a = \begin{bmatrix} 1 & S_\phi tan(\theta) & C_\phi tan(\theta) \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi sec(\theta) & C_\phi sec(\theta) \end{bmatrix} \tag{9}$$

The acceleration level differential equations are

$$m\dot{\mathbf{v}}_c + \tilde{\boldsymbol{\omega}} m\mathbf{v}_c = \mathbf{F} + \mathbf{F}_d \tag{10}$$

$$I\dot{\boldsymbol{\omega}} + \dot{I}\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}} I\boldsymbol{\omega} = \mathbf{M} + \mathbf{M}_d \tag{11}$$

$$\tilde{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \tag{12}$$

Note that for this study the motion of the smart block is simulated in the absence of gravity. Also Eq. 11 has an additional term $\dot{I}\boldsymbol{\omega}$, when compared to rigid body equations of motion. This term is a consequence of the shape change and is responsible for speeding up or slowing down the rotation of the block due to the time rate of change in the moment of inertia about a particular axis.

The drag force $\mathbf{F}_d$ is modeled as a function of the air density ρ, the square of the velocity along the axis and the area of the smart block perpendicular to the axis.

$$\mathbf{F}_d = \frac{-\rho}{2} \begin{bmatrix} u^2 sgn(u) yz \\ v^2 sgn(v) xz \\ w^2 sgn(w) xy \end{bmatrix} \tag{13}$$

Similarly, the drag moment $\mathbf{M}_d$ is modeled as a function of the air density ρ, the square of the angular velocity along the axis and the area of the surfaces of the smart block parallel to the axis.

$$\mathbf{M}_d = \frac{-\rho}{2} \begin{bmatrix} p^2 sgn(p) x(y+z) \\ q^2 sgn(q) y(x+z) \\ r^2 sgn(r) z(x+y) \end{bmatrix} \tag{14}$$

B. Trajectory Generation

The reference trajectory is arbitrary and is generated as a combination of straight lines and sinusoidal curves. Fig. 7 shows a sample reference trajectory that the smart block is required to track. The total flight path is divided into an odd number of segments of variable lengths. During every odd numbered segment the Y and Z locations remain

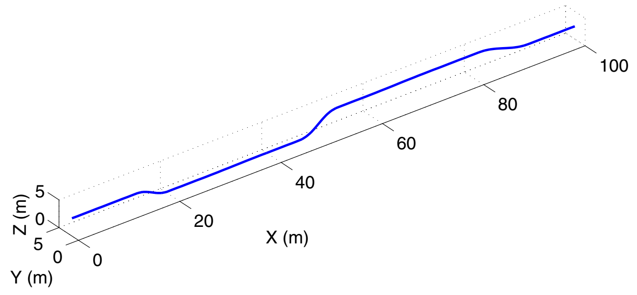

Fig. 7 Reference Trajectory for Positions along Inertial Axis.

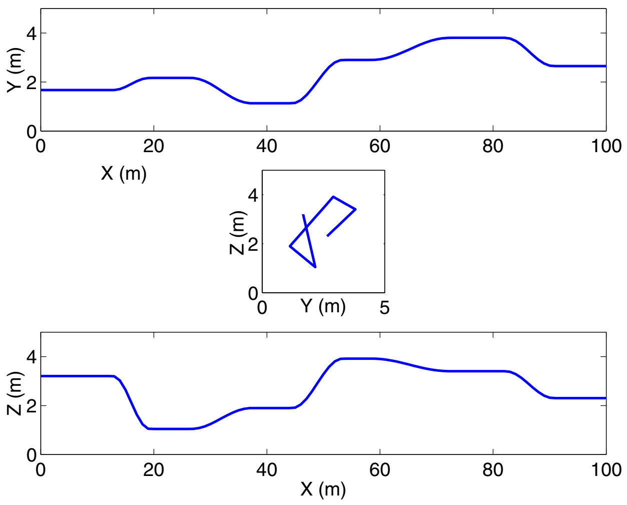

Fig. 8 Projections of the Reference Trajectory in Y-X, Z-Y and Z-X planes.

constant while the values of Y and Z are generated by a random function. The even numbered segments connect the trajectories in two adjacent sections with smooth sinusoidal curves. The smart block is supposed to move along the flight trajectory with a constant inertial velocity along the X_N inertial axis. For the attitude reference, it is supposed to trace prescribed sinusoidal oscillations along the X_N inertial axis. Fig. 8 shows the projections of the reference trajectory in Y-X, Z-Y and Z-X planes. Fig. 8, subplot 1, shows the top view and subplot 3 shows the side view with motion from left to right. Subplot 2 shows the front view with the smart block moving towards the observer.

III. Reinforcement Learning Module

A. Overview of Reinforcement Learning

Reinforcement Learning is a method of learning from interaction to achieve a goal.[7] Reinforcement Learning methodology makes use of agents and environments. An agent is defined as a learner and decision-maker, and the environment defined as everything outside the agent. The agent interacts with its environment, and the objective is for the agent to learn a mapping or policy, $\pi : S \to A$ from the state-space S to the action space A that maximizes some scalar reinforcement signal $r : S \times A \to R$ over a specified period of time[8]. Reinforcement Learning methods specify how the agent changes its policy as a result of its experiences, and generally speaking the goal is to maximize the total amount of reward over the long run. They are particularly suited for problems that either have no model, or very complex models, and for this reason are model-free control methods. Reinforcement Learning has been applied to a wide variety of physical control tasks, both real and simulated. For example, an acrobat system is a two-link, under-actuated robot roughly analogous to a gymnast swinging on a highbar. Controlling such a system by Reinforcement Learning has been studied by many researchers.[9,10,11] Besides Reinforcement Learning, other model-free control approaches include artificial Neural Networks and Fuzzy Logic[12].

In Reinforcement Learning, the agent interacts with its environment at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives some representation of the environment's state, $s_t \in S$, and on that basis it selects an action, $a_t \in A(s_t)$. One time step later, partly as a consequence of its action, the agent receives a numerical reward, $r_{t+1} = R$, and finds itself in a new state, s_{t+1}. The mapping from states to probabilities of selecting each possible action at each time step, denoted by π, is called the agent's policy. Here $\pi_t(s, a)$ indicates the probability that $a_t = a$ given $s_t = s$ at time t.

The value of a state s under a policy π, denoted by $V^\pi(s)$, is defined as the expected return starting from s and thereafter, following policy π. The objective of Reinforcement Learning is to find the optimal policy π^* that has the optimal state-value function $V^*(s)$, defined as $V^*(s) = \max_\pi V^\pi(s)$. This method of finding an optimal policy is called policy iteration, and the process of computing the current $V^\pi(s)$ is called policy evaluation. Using $V^\pi(s)$, the policy π can then be improved to policy π', and this process is called policy improvement. Finally, $V^{\pi'}(s)$ can then be used in successive iterations to compute a yet more improved policy, π''.

There are three major methods for policy iteration: Dynamic Programming, Monte Carlo methods, and Temporal-Difference learning. Dynamic Programming refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). The key idea is the use of value functions to organize and structure the search for good policies. Classical Dynamic Programming algorithms [13,14,15] are of limited utility in Reinforcement Learning, both because of their assumption of a perfect model and their great computational expense. However, they are very important theoretically. Monte Carlo methods are employed to estimate functions using an iterative, incremental procedure. The term "Monte Carlo" is sometimes used more broadly for any estimation method whose operation involves a significant random component. For the present context it represents methods which solve the Reinforcement Learning problem based on averaging sample returns. To ensure that well-defined returns are available, they are defined only for episodic tasks, and it is only upon the completion of an episode that value estimates and policies are changed. By comparison with Dynamic Programming, Monte Carlo methods can be used to learn optimal behavior directly from interaction with the environment, with no model of the environment's dynamics. They can be used with simulation, and it is easy and efficient to focus Monte Carlo methods on a small subset of the states. All Monte Carlo methods for Reinforcement Learning have been developed only recently, and their convergence properties are not well understood. Temporal-Difference methods can be viewed as an attempt to achieve much the same effect as Dynamic Programming, but with less computation and without assuming a perfect model of the environment. Sutton's method of Temporal-Differences is a form of the policy evaluation method in Dynamic Programming in which a control policy π_0 is to be chosen[16]. The prediction problem becomes that of learning the expected discounted rewards, $V^\pi(i)$, for each state i in S using π_0. With the learned expected discounted rewards, a new policy π_1 can be determined that improves upon π_0. The algorithm may eventually converge to some policy under this iterative improvement procedure, as in Howard's algorithm.[17] Q-Learning is a form of the successive approximations technique of Dynamic Programming, first proposed and developed by Watkins.[18] Q-learning learns the optimal value functions directly, as opposed to fixing a policy and determining the corresponding value functions, like Temporal-Differences. It automatically focuses attention to where it is needed, thereby avoiding the need to sweep over the state-action space. Additionally, it is the first provably convergent direct adaptive optimal control algorithm.

In many applications of Reinforcement Learning to control tasks, the state space is too large to enumerate the value function so function approximators must be used to compactly represent it. For example, tile coding has been used in many Reinforcement Learning systems.[19,20,21,22] Other commonly used approaches include neural networks, clustering, nearest-neighbor methods and cerebellar model articulator controller.

B. Implementation of Reinforcement Learning Module

For the present research, the agent in the smart block problem is its Reinforcement Learning module, which attempts to minimize the total amount of cost over the entire flight trajectory. To reach this goal, it endeavors to learn, from its interaction with the environment, the optimal policy that, given the specific flight condition, commands the voltage that changes the smart block's shape to the optimal one. The environment is the flight conditions which the smart block is flying in, along with its shape. We assume that the Reinforcement Learning module has no prior knowledge of the relationship between voltages and the dimensions of the block, as defined by morphing control functions in Eq. 1 and Eq. 2. Also it does not know the relationship between the flight conditions, costs and the optimal shapes as defined in Eq. 4 to Eq. 6. However, the Reinforcement Learning module does know all possible voltages that can be applied. It has accurate, real-time information of the smart block's shape, the present flight condition, and the current cost provided by a variety of sensors.

The Reinforcement Learning module learns the state value function using Actor-Critic methods.[7] These are classical Temporal-Difference learning methods that utilize two parametric structures, one called the actor, and the other called the critic. The actor consists of a parameterized control law or policy function, which is used to select actions. The critic approximates a value function that is used to critique the actions made by the actor, thereby capturing the effect that the policy function will have on the future cost. At any given time, the critic provides guidance on how to improve the policy function. In return, the actor can be used to update the critic. An algorithm that successively iterates between these two operations will converge to the optimal solution over time. An Actor-Critic method is selected for the current research primarily for convenience of implementation and evaluation, due to it's separate memory structure to explicitly represent the policy, independent of the value function. Since two separate modules are used to represent value functions and policies, modifications to one module (different types of action sets, state-spaces, and reward functions) can be made without affecting the other module. Additionally, Actor-Critic methods are able to handle problems with a continuous state-space. One way to do this (shown below) is to use linear function approximation methods. A parameterized functional form with a parameter vector is used to represent the value functions in the Critic, and the action preference functions in the Actor. Central to the operation of the Actor-Critic method is the Temporal-Difference error defined in Eq. 15, which is used to determine whether an action is better or worse than expected.

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{15}$$

At the same time, the critic updates its current estimated value function using the Temporal-Difference error

$$V(s_t) \leftarrow V(s_t) + \delta_t \tag{16}$$

In Eq. 15, V is the current estimated value function used by the critic. If the Temporal-Difference error is positive, it suggests that the tendency to select a_t when in state s_t should be strengthened for the future. If the Temporal-Difference error is negative, it suggests that the tendency should be weakened. The policy implemented by the actor is based on the preference functions, $p(s, a)$ which indicate the tendency to select each a when in each state s. Example policies are, the greedy policy

$$\pi_t(s, a) = arg \max_a p(s, a) \tag{17}$$

and the Gibbs softmax policy[7]

$$\pi_t(s, a) = Pr\{a_t = a | s_t = s\} = \frac{e^{p(s,a)}}{\sum_a e^{p(s,a)}} \tag{18}$$

The strengthening or weakening tendency described above is implemented by increasing or decreasing $p(s_t, a_t)$, for instance, using

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t \tag{19}$$

As shown in the Morphing Smart Block Simulation section, for this research the dimensions of the morphing block are continuous. Therefore, the morphing block has a continuous state-space, and its state-value function $V^\pi(s)$ and action-preference function $P_{at}(s)$ are both defined on continuous domains. One approach to solving this type of Reinforcement Learning problem with a continuous state-space is function approximation methods. In the present example, the approximate state-value function $V^\pi(s)$ is represented using a parameterized functional form with parameter vector θ_v

$$V^\pi(s) = \sum_{j=1}^{N} \theta_{vj}\Phi_j(s), \qquad \forall s \in S \tag{20}$$

where $\Phi_j(s)$ is predetermined and satisfies

$$\Phi_i(s) \cdot \Phi_j(s) = \begin{Bmatrix} 1 & if & i = j \\ 0 & if & i \neq j \end{Bmatrix} \tag{21}$$

Thus the basis vector

$$H = \begin{bmatrix} \Phi_i(s) & \Phi_2(s) & \dots & \Phi_N(s) \end{bmatrix}^T \tag{22}$$

is an orthogonal vector that satisfies

$$HH^T = I_{N\times N} \tag{23}$$

Using Eq. 22, Eq. 20 becomes

$$V^{\pi}(s) = H^T \theta_v, \qquad \forall s \in S \tag{24}$$

As H^T is fixed, $V^{\pi}(s)$ depends totally on θ_v, and varies from time step to time step only when θ_v changes. Updating θ_v is related to the Temporal-Difference error δ_t as defined in Eq. 15. The process of deriving the updating formula for θ_v is shown as follows: First, we selected the gradient descent method for updating $V(s_t)$, upon which Eq. 16 becomes

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t \tag{25}$$

Using Eq. 24, Eq. 25 becomes

$$H_t^T \theta_v \leftarrow H_t^T \theta_v + \alpha\delta_t \tag{26}$$

where $H_t = \begin{bmatrix} \Phi_1(s_t) & \Phi_2(s_t) & \ldots & \Phi_N(s_t) \end{bmatrix}$, and H_t is the known coefficient matrix. Using Eq. 23, the formula for updating θ_v becomes

$$\theta_v \leftarrow \theta_v + \alpha\delta_t H_t \tag{27}$$

Note that the purpose of the Reinforcement Learning module is not to compute the exact $V^{\pi}(s)$, but to learn the optimal policy π^* with the optimal state-value function $V^*(s)$. In the process of policy iteration, an intermediate policy π can be improved to a better policy π' even if its state-value function $V^{\pi}(s)$ is not exactly computed.

Similarly for the actor, the preference function for each action is represented using a parameterized functional form with parameter vector θ_{pa}.

$$p_a(s) = H^T \theta_{pa} \tag{28}$$

Since $V(s)$ and $p_a(s)$ are both defined on the same state-space, the basis vector H can also be used for $V(s)$. Using Eq. 19, and following a similar process from Eq. 25 to Eq. 27, the formula for updating θ_{pa} is

$$\theta_{pa} \leftarrow \theta_{pa} + \alpha\beta\delta_t H_t^T \tag{29}$$

Actor-critic methods are on-policy learning methods since the critic must learn about and critique whatever policy is currently being followed by the actor. The Temporal-Difference error is the only output of the critic and drives all learning in both actor and critic, as suggested in Eq. 27 and Eq. 29. The Reinforcement Learning module is summarized in Fig. 9.

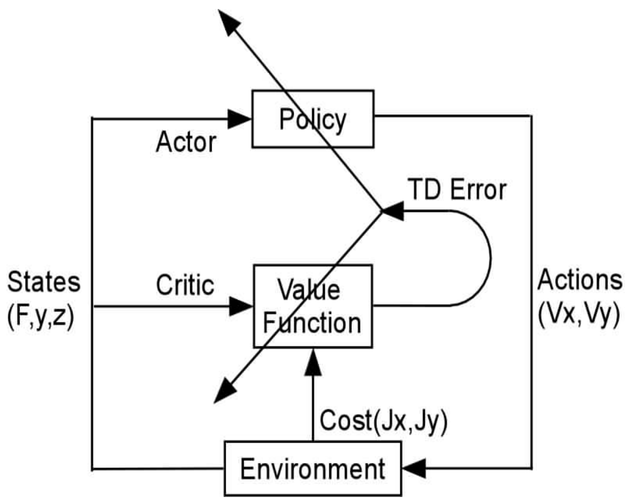

Fig. 9 Reinforcement Learning Module.[7]

The Reinforcement Learning agent always encounters an exploration-exploitation dilemma when it selects the next action given its current state[7]. If it selects a greedy action that has the highest preference for the current state, then it is exploiting its knowledge obtained so far, about the values of the actions. If instead it selects one of the non-greedy actions, then it is exploring to improve its estimate of the non greedy actions' values. In the early stage of the learning, the Reinforcement Learning agent employs a uniform probability policy under which each action has the same probability to be selected. In this way it can explore as many states and actions as possible. In the middle stage, it changes to a Gibbs Softmax Policy as described in Eq. 18, under which the actions with a higher preference have higher probability to be selected. Using this policy allows the module to partially explore and partially exploit at the same time. In the final stage, it uses a greedy policy, as described in Eq. 17, that allows it to totally exploit its previous experience.

IV. Structured Adaptive Model Inversion

A. A Brief Introduction to Structured Adaptive Model Inversion (SAMI)

The goal of the SAMI controller is to track the reference trajectories, even when the dynamic properties of the smart block change, due to the morphing. Structured Adaptive Model Inversion (SAMI)[23] is based on the concepts of Feedback Linearization[24], Dynamic Inversion, and Structured Model Reference Adaptive Control (SMRAC).[25,26,27] In SAMI, dynamic inversion is used to solve for the control. The dynamic inversion is approximate, as the system parameters are not modeled accurately. An adaptive control structure is wrapped around the dynamic inverter to account for the uncertainties in the system parameters.[28,29,30] This controller is designed to drive the error between the output of the actual plant and that the reference trajectories to zero, with prescribed error dynamics. Most dynamic systems can be represented as two sets of differential equations, an exactly known kinematic level part, and a momentum level part with uncertain system parameters. The adaptation included in this framework can be limited to only the uncertain momentum level equations, thus restricting the adaptation only to a subset of the state-space, enabling efficient adaptation. SAMI has been shown to be effective for tracking spacecraft[31] and aggressive aircraft maneuvers[32]. The SAMI approach has been extended to handle actuator failures and to facilitate correct adaptation in presence of actuator saturation.[33,34,35]

B. Mathematical Formulation of the SAMI Controller for Attitude Control

The attitude equations of motion for the smart block are given by Eq. 8 and Eq. 11. Without the angular drag and the $\dot{I}\boldsymbol{\omega}$ term, Eq. 8 and Eq. 11 can be manipulated to obtain the following form[23]

$$I_a^*(\boldsymbol{\sigma})\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}} = P_a^T(\boldsymbol{\sigma})\mathbf{M} \tag{30}$$

where the matrices $I_a^*(\sigma)$, $C_a^*(\sigma,\dot{\sigma})$ and $P(\sigma)$ are defined as

$$P_a(\sigma) \triangleq J_a^{-1}(\sigma) \tag{31}$$

$$I_a^*(\sigma) \triangleq P_a^T I P_a \tag{32}$$

$$C_a^*(\sigma,\dot{\sigma}) \triangleq -I_a^*\dot{J}_a P_a + P_a^T[\widetilde{P_a\dot{\sigma}}]I P_a \tag{33}$$

The left hand side of Eq. 30 can be linearly parameterized as follows

$$I_a^*(\boldsymbol{\sigma})\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}} = Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})\boldsymbol{\Theta} \tag{34}$$

where and $\boldsymbol{\Theta}$ is defined as $\boldsymbol{\Theta} \triangleq \begin{bmatrix} I_{11} & I_{22} & I_{33} & I_{12} & I_{13} & I_{23}\end{bmatrix}^T$. It can be seen that the product of the inertia matrix and a vector can be written as

$$I\boldsymbol{\nu} = \Lambda(\boldsymbol{\nu})\boldsymbol{\Theta}, \qquad \forall \boldsymbol{\nu} \in \mathbb{R}^3 \tag{35}$$

where $\Lambda \in \mathbb{R}^{3\times 6}$ is defined as

$$\Lambda(\boldsymbol{\nu}) \triangleq \begin{bmatrix} \nu_1 & 0 & 0 & \nu_2 & \nu_3 & 0 \\ 0 & \nu_2 & 0 & \nu_1 & 0 & \nu_3 \\ 0 & 0 & \nu_3 & 0 & \nu_1 & \nu_2 \end{bmatrix} \tag{36}$$

The terms on the left hand side of Eq. 30 can be written as

$$\begin{aligned} I_a^*\ddot{\boldsymbol{\sigma}} &= P_a^T I P_a \ddot{\boldsymbol{\sigma}} \\ &= P_a^T \Lambda(P_a\ddot{\boldsymbol{\sigma}})\boldsymbol{\Theta} \qquad (37) \\ C_a^*\dot{\boldsymbol{\sigma}} &= -P_a^T I P_a \dot{J}_a P_a \dot{\boldsymbol{\sigma}} + P_a^T[\widetilde{P_a\dot{\boldsymbol{\sigma}}}]I P_a \dot{\boldsymbol{\sigma}} \\ &= P_a^T\{-\Lambda(P_a \dot{J}_a P_a\dot{\boldsymbol{\sigma}}) + [\widetilde{P_a\dot{\boldsymbol{\sigma}}}]\Lambda(P_a\dot{\boldsymbol{\sigma}})\}\boldsymbol{\Theta} \qquad (38) \end{aligned}$$

Combining Eqs. 37 and 38 we have the linear minimal parametrization for the inertia matrix[36].

$$\begin{aligned} & I_a^*(\boldsymbol{\sigma})\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}} \\ &= P_a^T\{\Lambda(P_a\ddot{\boldsymbol{\sigma}}) - \Lambda(P_a\dot{J}_aP_a\dot{\boldsymbol{\sigma}}) + [\widetilde{P_a\dot{\boldsymbol{\sigma}}}]\Lambda(P_a\dot{\boldsymbol{\sigma}})\}\boldsymbol{\Theta} \\ &= Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})\boldsymbol{\Theta} \qquad (39) \end{aligned}$$

The attitude tracking problem can be formulated as follows. The control objective is to track an attitude trajectory in terms of the 3-2-1 Euler angles. The desired reference trajectory is assumed to be twice differentiable with respect to time. Let $\boldsymbol{\epsilon} \triangleq \boldsymbol{\sigma}\text{-}\boldsymbol{\sigma_r}$ be the tracking error. Differentiating twice and multiplying by I_a^* throughout

$$I_a^*\ddot{\boldsymbol{\epsilon}} = I_a^*\ddot{\boldsymbol{\sigma}} - I_a^*\ddot{\boldsymbol{\sigma}}_{\boldsymbol{r}} \qquad (40)$$

Adding $(C_{da} + C^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\,\boldsymbol{\epsilon}$ on both sides,

$$I_a^*\ddot{\boldsymbol{\epsilon}} + (C_{da} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} = I_a^*\ddot{\boldsymbol{\sigma}} - I_a^*\ddot{\boldsymbol{\sigma}}_{\boldsymbol{r}} + (C_{da} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} \qquad (41)$$

The RHS of Eq. 41 can be written as

$$(I_a^*\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}}) - (I_a^*\ddot{\boldsymbol{\sigma}}_{\boldsymbol{r}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}}) + C_{da}\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} \qquad (42)$$

From Eq. 30 and the construction of Y similar to Eq. 39, the RHS of Eq. 41 can be further written as

$$P_a^T\mathbf{M} - Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)\boldsymbol{\Theta} + C_{da}\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} \qquad (43)$$

So the control law can be now chosen as

$$\mathbf{M} = P_a^{-T}\{Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)\boldsymbol{\Theta} - C_{da}\dot{\boldsymbol{\epsilon}} - K_{da}\boldsymbol{\epsilon}\} \qquad (44)$$

The above control law requires that the inertia parameters $\boldsymbol{\Theta}$ be known accurately, but they may not be known accurately in actual practice. So by using the certainty equivalence principle[30], adaptive estimates for the inertia parameters $\hat{\boldsymbol{\Theta}}$ will be used for calculating the control.

$$\mathbf{M} = P_a^{-T}\{Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)\hat{\boldsymbol{\Theta}} - C_{da}\dot{\boldsymbol{\epsilon}} - K_{da}\boldsymbol{\epsilon}\} \qquad (45)$$

With the control law given in Eq. 45 the closed loop dynamics takes the following form

$$I_a^*\ddot{\boldsymbol{\epsilon}} + (C_{da} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} = Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})\widetilde{\boldsymbol{\Theta}} \qquad (46)$$

where $\widetilde{\boldsymbol{\Theta}} = \widehat{\boldsymbol{\Theta}} - \boldsymbol{\Theta}$. The update for the parameter $\hat{\boldsymbol{\Theta}}$ and the stability proof can be obtained by doing a Lyapunov analysis.

Consider the candidate Lyapunov function

$$V = \frac{1}{2}\dot{\boldsymbol{\epsilon}}^T I_a^* \dot{\boldsymbol{\epsilon}} + \frac{1}{2}\boldsymbol{\epsilon}^T K_{da} \boldsymbol{\epsilon} + \frac{1}{2}\tilde{\boldsymbol{\Theta}}^T \tau^{-1} \tilde{\boldsymbol{\Theta}} \tag{47}$$

Taking the derivative of the Lyapunov function along the closed loop trajectories given by Eq. 46

$$\dot{V} = \frac{1}{2}\dot{\boldsymbol{\epsilon}}^T \dot{I}_a^* \dot{\boldsymbol{\epsilon}} + \dot{\boldsymbol{\epsilon}}^T I_a^* \ddot{\boldsymbol{\epsilon}} + \dot{\boldsymbol{\epsilon}}^T K_{da} \boldsymbol{\epsilon} + \frac{1}{2}\dot{\tilde{\boldsymbol{\Theta}}}^T \tau^{-1} \tilde{\boldsymbol{\Theta}} \tag{48}$$

which can be simplified as

$$\dot{V} = \dot{\boldsymbol{\epsilon}}^T [\frac{1}{2}\dot{I}_a^* - C_a^*]\dot{\boldsymbol{\epsilon}} - \dot{\boldsymbol{\epsilon}}^T C_{da} \dot{\boldsymbol{\epsilon}} + (\dot{\boldsymbol{\epsilon}}^T Y_a(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \dot{\boldsymbol{\sigma}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\sigma}}_r) + \dot{\tilde{\boldsymbol{\Theta}}}^T \tau^{-1})\tilde{\boldsymbol{\Theta}} \tag{49}$$

The first term vanishes because $[\frac{1}{2}\dot{I}_a^* - C_a^*]$ is skew symmetric. Setting the coefficient of $\tilde{\boldsymbol{\Theta}}$ to 0, to obtain the adaptive laws.

$$\dot{\tilde{\boldsymbol{\Theta}}} = -\tau Y_a(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \dot{\boldsymbol{\sigma}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\sigma}}_r)^T \dot{\boldsymbol{\epsilon}} \tag{50}$$

From the definition of $\tilde{\boldsymbol{\Theta}}$ and assuming that the true parameter $\boldsymbol{\Theta}$ remains constant,

$$\dot{\hat{\boldsymbol{\Theta}}} = -\tau Y_a(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \dot{\boldsymbol{\sigma}}_{\boldsymbol{r}}, \ddot{\boldsymbol{\sigma}}_r)^T \dot{\boldsymbol{\epsilon}} \tag{51}$$

This update law renders the derivative of the Lyapunov function

$$\dot{V} \quad = \quad -\dot{\boldsymbol{\epsilon}}^T C_{da} \dot{\boldsymbol{\epsilon}} \tag{52}$$

Thus the derivative is negative semi definite. From Eq. 47 and Eq. 52 it can be concluded that $\boldsymbol{\epsilon}$,$\dot{\boldsymbol{\epsilon}}$ and $\boldsymbol{\Theta}$ are bounded. Using the standard procedure of application of Barbalat's lemma[30,37], asymptotic stability of the tracking error dynamics can be concluded for bounded reference trajectories.

C. Mathematical Formulation of the SAMI Controller for Control of Linear States

Following similar lines as the attitude controller, Eq. 7 and Eq. 10, without the drag term can be cast in the following form

$$I_l^* \ddot{\mathbf{p}}_c + C_l^* \dot{\mathbf{p}}_c = P_l^T \mathbf{F} \tag{53}$$

where the matrices I_l^*, C_l^* and P_l are defined as

$$P_l \triangleq J_l^{-1} \tag{54}$$

$$I_l^* \triangleq P_l^T m P_l \tag{55}$$

$$C_l^* \triangleq -I_l^* \dot{J}_l P_l + P_l^T \tilde{\boldsymbol{\omega}} m P_l \tag{56}$$

The control law with the adaptive parameter $\hat{m}$ is

$$\mathbf{F} = P_l^{-T}\{Y_l m - C_{dl}\dot{\boldsymbol{\varepsilon}} - K_{dl}\boldsymbol{\varepsilon}\} \tag{57}$$

where

$$Y_l \quad = P_l^T P_l \ddot{\mathbf{p}}_{c(ref)} - P_l^T P_l \dot{J}_l P_l \dot{\mathbf{p}}_{c(ref)} + P_l^T \tilde{\boldsymbol{\omega}} P_l \dot{\mathbf{p}}_{c(ref)} \tag{58}$$

Subscript $_{(ref)}$ indicates the respective reference state and $\boldsymbol{\varepsilon} \triangleq \mathbf{p}_c - \mathbf{p}_{c(ref)}$ is the tracking error for the linear position. The update law for the adaptive parameter $\hat{m}$ is

$$\dot{\hat{m}} = -\Gamma Y_l^T \dot{\boldsymbol{\varepsilon}} \tag{59}$$

V. A-RLC Architecture Functionality

The Adaptive-Reinforcement Learning Control Architecture is composed of two sub-systems: Reinforcement Learning and Structured Adaptive Model Inversion (SAMI) (Fig. 10). The two sub-systems interact significantly during both the episodic learning stage, when the optimal shape change policy is learned, and the operational stage, when the plant morphs and tracks a trajectory. For either type of stage, the system functions as follows.

Considering the Reinforcement Learning sub-system at the top of Fig. 10 and moving counterclockwise, the Reinforcement Learning module initially commands an arbitrary action from the set of admissible actions. This action is sent to the plant, which produces a shape change. The cost associated with the resultant shape change in terms of system states, parameters, and user defined performance measures, is evaluated with the cost function and then passed to the Critic. The Critic calculates the Temporal-Difference error using its current estimated state value function, which is sent to the Actor. The Critic modifies it's state value function according to the Temporal-Difference error, and likewise the Actor updates it's action preference function. For the next episode, the Actor generates a new action based on the current policy and its updated action preference function, and the sequence repeats itself.

Considering the SAMI sub-system at the bottom of Fig. 10, shape changes in the plant due to actions generated by the Actor cause the plant dynamics to change. The SAMI controller maintains trajectory tracking irrespective of the changing dynamics of the plant due to these shape changes.

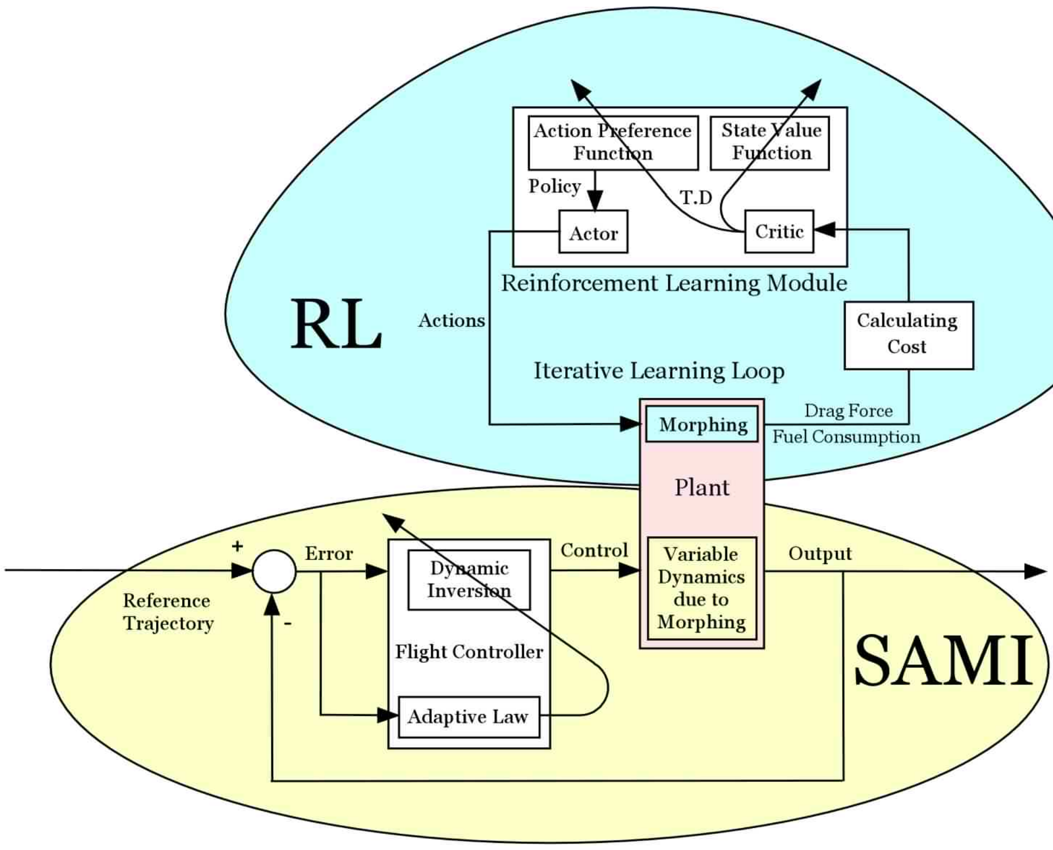

Fig. 10 Adaptive-Reinforcement Learning Control Architecture.

VI. Numerical Example

A. Purpose and Scope

The purpose of the numerical example is to demonstrate the learning performance and the trajectory tracking performance of the A-RLC architecture. To learn the optimal shape change policy a total of 200 learning episodes are used, each consisting of a single 200 second transit through a 100 meter long path. The reference trajectory to be tracked in each episode is generated arbitrarily. The sequence of the flight conditions is random, and changes twice during each episode. At a particular flight condition the Reinforcement Learning Module commands arbitrary voltages at ten second intervals, that result in the corresponding shape and records the cost expended due to that shape. Thus it builds up its knowledge base and learns the optimal shape change policy.

After the learning episodes are complete, the learned knowledge of the Reinforcement Learning Module and the trajectory tracking capability of the SAMI Controller have to be evaluated. These are evaluated with a single pass through the path, with a randomly generated reference trajectory and an arbitrary flight condition change at approximately 50 second intervals. Fig. 6 shows a typical flight condition sequence.

B. Learning the Optimal Policy for Shape Change

For the definitions of the morphing control functions and the cost functions defined by Eq. 1 to Eq. 5, this Reinforcement Learning problem can be decoupled into two independent problems that correspond to the y and z dimensions respectively. For the y dimension Reinforcement Learning problem, the state has two elements: the value of y dimension and the flight condition $s = \{y, f\}$. The state set consisting of all possible states is a 2-Dimensional continuous Cartesian space $S = [2, 4] \times [0, 5]$. For each state, the action set consists all possible voltages that can be applied $A(s) = \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$. Similarly for the z dimension, the state set is $[2, 4] \times [0, 5]$, and the action set is $\{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$. Note that the control can take only the discrete values shown above.

For simplicity, the training is conducted only at six discrete flight conditions$\{0, 1, 2, 3, 4, 5\}$. For the function approximation for $V_t^\pi(s)$ and $p_{at}(s)$, a relatively simple linear approximation method was used: tile coding. The basis function $\Phi_j(s)$is defined as follows

$$\Phi_j(s) = \begin{cases} 1 & \text{if } (s - c_j)^T \mathcal{R}^{-1}(s - c_j) \leq 1 \\ 0 & if (s - c_j)^T \mathcal{R}^{-1}(s - c_j) > 1 \end{cases} \tag{60}$$

For the y dimension, the set of all center state vectors is 2-dimensional, $C_y = y \times f$, where $f = \{0, 1, 2, 3, 4, 5\}$, and C_y has 102 total elements. The weighting matrix $\mathcal{R}^{-1}$ is defined as

$$\mathcal{R}^{-1} = \begin{bmatrix} r_1^2 & 0 \\ 0 & r_2^2 \end{bmatrix}^{-1} \qquad r_1 = 0.1, \quad r_2 = 0.0625 \tag{61}$$

In tile coding, the basis function Φ_j is called a binary feature or a tile. One advantage of tile coding is the capability of controlling the overall features which are activated at one time. In the present case, only one feature is present at one time. For the z dimension, the tile $\Phi_j(s)$, the center state vector set $C_z = z \times f$, and the weighting matrix $\mathcal{R}^{-1}$ are defined in the same fashion.

Fig. 11 is a graphical representation of the errors in the preferred voltages of the y-dimension selected by the Actor, when compared with the true values used for simulation. The preferred voltages are calculated based on the policy and the estimated action preference functions currently used. As the learning progresses from 20 to 200 episodes, the error plot becomes flatter and the errors at more points converge to zero. This indicates that the action preference functions are being learned accurately. The decrease in error is less significant from 100 to 200 episodes than from 20 to 60 episodes, since the focus shifts from exploration to exploitation, and the action preference functions asymptotically converge to those of the optimal policy. Note that the error at $y = 3.5$ appears to increase with successive episodes. This point in the state-space has only been visited at flight condition 1 within 200 episodes, and not at any other flight condition. Since a function approximation is being used, extrapolation is occurring and may give rise to an incorrect voltage, and therefore nonzero error. With a finer resolution of discrete voltages in the action set, this would not occur. However, here it does not affect the results since that point is never visited in this example (see Fig. 3).

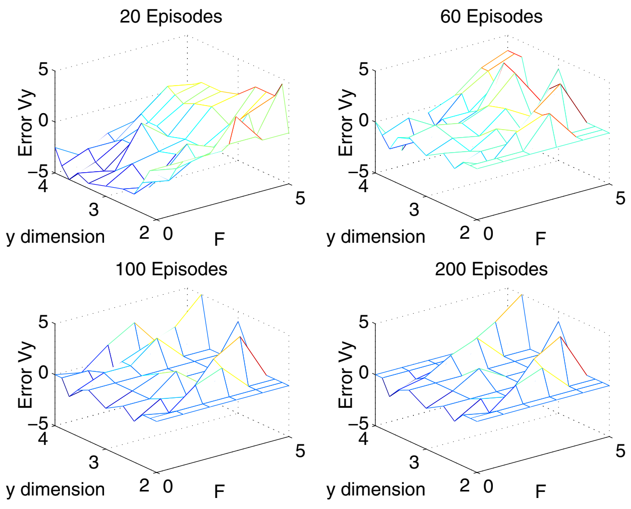

Fig. 11 Error in the Action Preference Function after a) 20 episodes, b) 60 episodes, c) 100 episodes , d) 200 episodes.

Figure 12 compares the time histories of the achieved shapes with those of the optimal shapes for a typical single learning episode of the later stages, as a result of the voltages commanded by the Reinforcement Learning agent. Values of the achieved shape are seen to be close, but not exactly equal to, the optimal values. The Reinforcement Learning agent cannot learn the optimal policy exactly because the applied control can take only discrete values (which are specified in the action set). The voltage that leads to the optimal shape may lie in between the values in the voltage action set. This problem can be rectified by using a continuous action set, so that any value in between the control limits can be commanded. Use of a continuous action set in the A-RLC formulation will be addressed in future research.

C. Trajectory Tracking

The control objective for the SAMI controller is to track the reference trajectories, irrespective of initial condition errors, parametric uncertainties, and changes in the dynamic behavior of the smart block due to morphing.

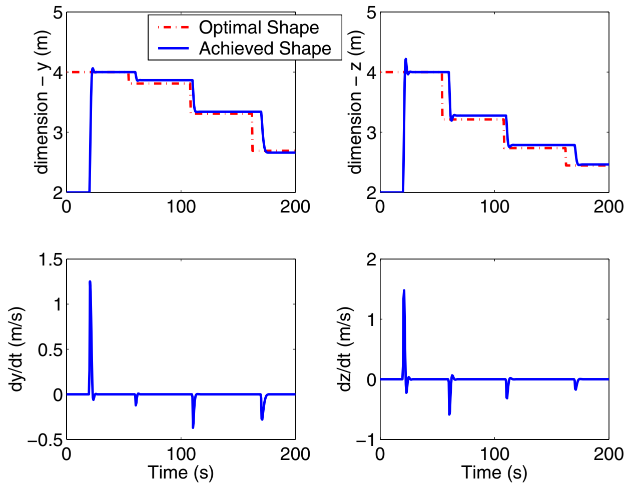

Fig. 12 Comparison between True Optimal Shape and Shape Learned by the Reinforcement Learning Agent, for a Sequence of Flight Conditions.

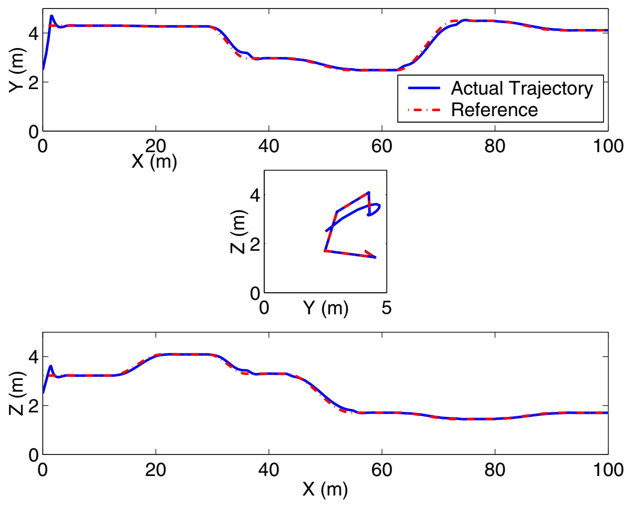

Fig. 13 Projections of the Trajectory in Y-X, Z-Y and Z-X planes.

Figure 13 shows that the adaptive control maintains close tracking of the reference trajectories. Figure 14 and Fig. 15 show the time histories of the linear and angular states respectively. The deviations from the reference trajectory for the linear velocities v and w seen in Fig. 14, are due to rapid changes in the smart block's dimensions. Also note that the vehicle has displacements along y and z inertial axes with commanded pitch and yaw angles zero. Since the conceptual air vehicle has thrusters on all three axes and is capable of hover, total velocity can be in any arbitrary direction.

Figure 16 shows the variation in the true parameters and the update of the adaptive parameters. The adaptive parameters do not converge to the true parameters, which is commonly seen in adaptive control systems. The parameters only converge to the true values if the reference trajectory is persistently exciting.[30] Most importantly, Fig.16 shows that the adaptive parameters are bounded, and do not diverge. Figure 17 shows that the control forces and moments are well behaved.

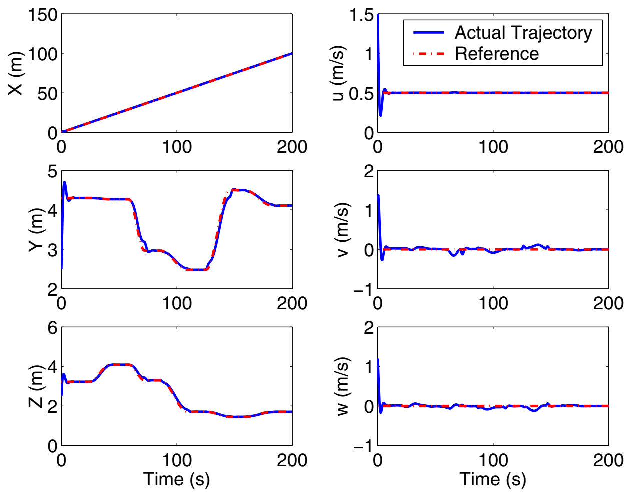

Fig. 14 Time Histories of the Linear States.

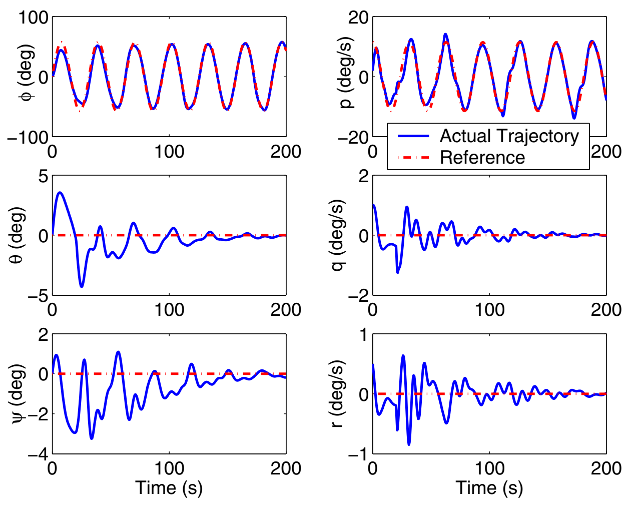

Fig. 15 Time Histories of the Angular States.

The stability proof for SAMI presented earlier guarantees asymptotic stability of the tracking error only for constant true parameters. Here it has been applied it for a morphing smart block in which the parameters are varying. SAMI can still retain these stability properties if there is a large timescale separation between the morphing dynamics and the update of the adaptive parameters. If the morphing takes place very slowly relative to the update of the adaptive parameters, $\dot{\Theta}$ is negligible as compared to $\dot{\hat{\Theta}}$ and the update of the adaptive parameter is not affected. As per the definition that was cited for Morphing for Mission Adaptation, it is a large scale, relatively slow, in-flight shape change. So we conclude that this control architecture is promising candidate for Morphing for Mission Adaptation. In contrast to this, Morphing for Control, addresses rapid shape changing needed for maneuvering, etc. The present trajectory tracking controller cannot handle rapid shape changes and hence is not applicable to Morphing for Control.

In the current simulation, the true parameters which are learned are the inertia and mass. Mass remains constant, but the inertia changes as the smart block morphs into different shapes. The SAMI controller does not implement the

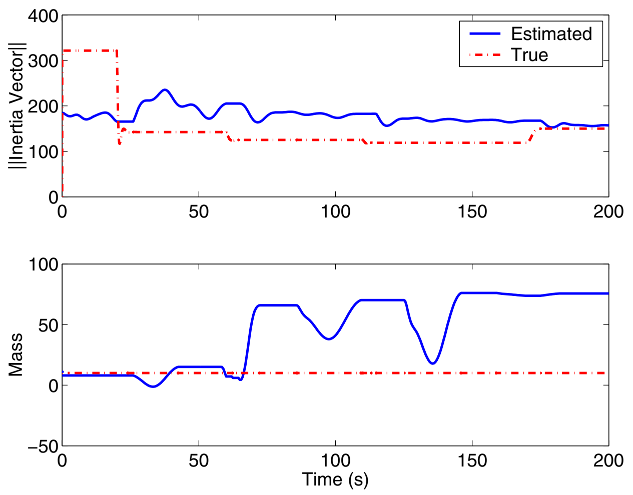

Fig. 16 Time Histories of the Adaptive Parameters.

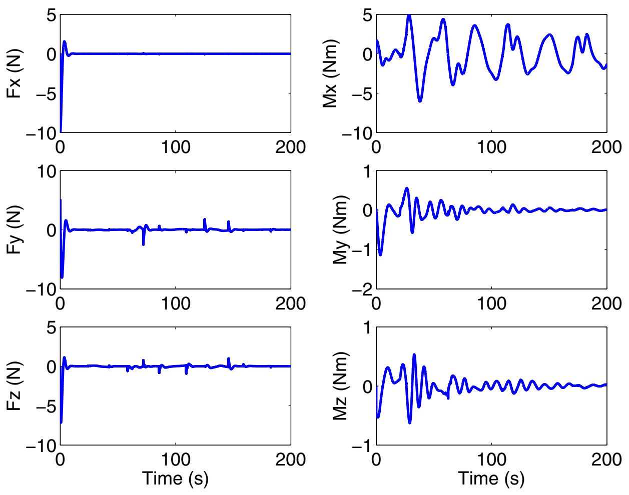

Fig. 17 Time Histories of the Control Forces and Moments.

term $\dot{I}\boldsymbol{\omega}$ in the Eq. 11, so $\dot{I}\boldsymbol{\omega}$ acts as a disturbance that perturbs the tracking. However, the SAMI controller is able to maintain adequate tracking performance. Note that the design of the controller does not explicitly account the drag force and drag moment which act as external disturbances. However, the simulation results show that SAMI is able to handle external disturbances due to the drag forces thereby ensuring that A-RLC controller performs well.

VII. Conclusions and Future Research

This paper proposed and developed a control methodology for morphing vehicles, combining machine learning and adaptive dynamic inversion control. For optimal shapes of an entire vehicle, defined as a function of operating condition, this Adaptive-Reinforcement Learning Control learns the commands which produce the optimal shape, while maintaining accurate trajectory tracking. Selection of cost functions, derivations of the dynamical model and trajectory of the air vehicle, the Reinforcement Learning controller, and the Structured Adaptive Model Inversion controller were presented. The methodology was demonstrated by a numerical example of a three-dimensional air vehicle autonomously morphing while tracking a specified trajectory over a finite set of flight conditions.

Based on the results presented in this paper, it is concluded that:

1. For the numerical example presented, the Adaptive-Reinforcement Learning Controller maintains asymptotic tracking in the presence of parametric uncertainties and initial condition errors.
2. The control architecture retains the stability properties of the SAMI controller if the morphing is slow as compared to the rate of update of adaptive parameters. Morphing for Mission Adaptation is a slow shape change, hence Adaptive-Reinforcement Learning Control is a promising candidate methodology for the control of Morphing for Mission Adaptation.
3. The Reinforcement Learning Module can successfully learn the control policy that results in the optimal shape at every flight condition. In addition, the Reinforcement Learning module can function in realtime, which may lead to better performance as time progresses since learning continues as the system operates.

Several aspects will be addressed in future research. First, the present trajectory tracking controller cannot handle rapid shape changes and hence is not applicable to Morphing for Control. A trajectory tracking controller which uses the shape parameter as a state in the system is being investigated, to handle fast morphing. The current action set in the Reinforcement Learning Module is composed of only discrete values, and efforts are underway to accommodate continuous action sets in the A-RLC methodology. Finally, a more realistic air vehicle model than the conceptual hovering air vehicle used here will be used, and the current smart material model is also being refined to incorporate the hysteresis behavior commonly observed in Shape Memory Alloys.

Acknowledgments

The authors wish to acknowledge the support of the Texas Institute for Intelligent Bio-Nano Materials and Structures for Aerospace Vehicles. The material is based upon work supported by NASA under award no. NCC-1-02038. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration. The authors thank the Associate Editor and reviewers for their many insightful comments and suggestions, which improved the paper.

References

[1]Wlezien, R., Horner, G., McGowan, A., Padula, A., Scott, M., Silcox, R., and Simpson, J., "The Aircraft Morphing Program," No. AIAA-98-1927.

[2]McGowan, A.-M. R., Washburn, A. E., Horta, L. G., Bryant, R. G., Cox, D. E., Siochi, E. J., Padula, S. L., and Holloway, N. M., "Recent Results from NASA's Morphing Project," *Proceedings of the 9th Annual International Symposium on Structures and Materials*, No. SPIE Paper Number 4698-11, San Diego, CA, 17-21 March 2002.

[3]Wilson, J. R., "Morphing UAVs change the shape of warfare," *Aerospace America*, February 2004, pp. 23–24.

[4]Bowman, J., Weisshaar, T., and Sanders, B., "Evaluating The Impact Of Morphing Technologies On Aircraft Performance," *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, No. AIAA-2002-1631, Denver, CO, 22-25 April 2002.

[5]Scott, M. A., Montgomery, R. C., and Weston, R. P., "Subsonic Maneuvering Effectiveness of High Performance Aircraft Which Employ Quasi-Static Shape Change Devices," *Proceedings of the SPIE 5th Annual International Symposium on Structures and Materials*, San Diego, CA, March 1-6 1998.

[6]Nelson, R. C., *Flight Stability and Automatic Control*, chap. 3, McGraw-Hill, 1998, pp. 96–105.

[7]Sutton, R. and Barto, A., *Reinforcement Learning - An Introduction*, The MIT Press, Cambridge, Massachusetts, 1998.

[8]Si, J., Barto, A. G., Powell, W. B., and Wunsch, D., editors, *Handbook of Learning and Approximate Dynamic Programming*, chap. 16, IEEE Press Series on Computational Intelligence, Wiley-IEEE Press, 2004, p. 411.

[9]DeJong, G. and Spong, M. W., "Swinging up the acrobot: An example of intelligent control," *Proceedings of the American Control Conference*, 1994, pp. 2158–2162.

[10]Boone, G., "Minimum-time control of the acrobot," *International Conference on Robotics and Automation*, Albuquerque, NM, 1997.

[11]Sutton, R. S., "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, edited by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, MIT Press, Cambridge MA, pp. 1038–1044.

[12]Kartalopoulos, S. V., *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications*, IEEE Press Understanding Science & Technology Series, Wiley-IEEE Press, August 29 1995.

[13]Bellman, R. E., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[14]Bellman, R. E. and Dreyfus, S. E., *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ, 1962.

[15]Bellman, R. E. and Kalaba, R. E., *Dynamic Programming and Modern Control Theory*, Academic Press, New York, 1965.

[16]Sutton, R. S., "Learning to Predict by the method of Temporal Differences," *Machine Learning*, Vol. 3, 1998, pp. 9–44.

[17]Williams, R. J. and Baird, L. C., "Analysis of some incremental variants of policy iteration: First Steps toward Understanding Actor-Critic Learning Systems," Tech. Rep. NU-CCS-93-11, Boston, 1993.

[18]Watkins, C. J. C. H. and Dayan, P., *Learning from Delayed Rewards*, Ph.D. thesis, Cambridge Unversity, Cambridge, UK, 1989.

[19]Moore, A. W., *Efficient Memory-Based Learning for Robot Control*, Ph.D. thesis, University of Cambridge, Cambridge, UK, 1990.

[20]Shewchuk, J. and Dean, T., "Towards learning time-varying functions with high input dimensionality," *Proceedings of the Fifth IEEE International Symposium on Intelligent Control*, 1990, pp. 383–388.

[21]Lin, C. S. and Kim, H., *IEEE Transactions on Neural Networks*, No. 530-533, 1991.

[22]Miller, W. T., Scalera, S. M., and Kim, A., "A Neural Network control of dynamic balance for a biped walking robot," *Proceedings on the Eighth Yale Workshop on Adaptive and Learning Systems*, Dunham Laboratory, Yale University, Center for Systems Science, 1994, pp. 156–161.

[23]Subbarao, K., *Sructured Adaptive Model Inversion: Theory and Applications to Trajectory Tracking for Non-Linear Dynamical Systems*, Ph.D. thesis, Aerospace Engineering Department, Texas A&M University, College Station, TX, 2001.

[24]Slotine, J. and Li, W., *Applied Nonlinear Control*, Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 1991, pp. 207–271.

[25]Akella, M. R., *Structured Adaptive Control: Theory and Applications to Trajectory Tracking in Aerospace Systems*, Ph.D. thesis, Aerospace Engineering Department, Texas A&M University, College Station, TX, 1999.

[26]Schaub, H., Akella, M. R., and Junkins, J. L., "Adaptive Realization of Linear Closed-Loop Tracking Dynamics in the Presence of Large System Model Errors," *The Journal of Astronautical Sciences*, Vol. 48, October-December 2000, pp. 537–551.

[27]Akella, M. R. and Junkins, J. L., "Structured Model Reference Adaptive Control in the Presence of Bounded Disturbances," *AAS/AIAA Space Flight Mechanics Meeting*, Monterey, CA, Feb 9-11 1998, pp. 98–121.

[28]Narendra, K. and Annaswamy, A., *Stable Adaptive Systems*, Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 1989.

[29]Sastry, S. and Bodson, M., *Adaptive Control: Stability, Convergence, and Robustness*, Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 1989, pp. 14–156.

[30]Ioannou, P. A. and Sun, J., *Robust Adaptive Control*, chap. 1, Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 1996, pp. 10–11.

[31]Subbarao, K., Verma, A., and Junkins, J. L., "Structured Adaptive Model Inversion Applied to Tracking Spacecraft Maneuvers," *Proceedings of the AAS/AIAA Spaceflight Mechanics Meeting*, No. AAS-00-202, Clearwater, FL, 23-26 January 2000.

[32]Subbarao, K., Steinberg, M., and Junkins, J. L., "Structured Adaptive Model Inversion Applied to Tracking Aggressive Aircraft Maneuvers," *Proceedings of the AIAA Guidance, Navigation and Control Conference*, No. AAS-00-202, Montreal Canada, 6-9 August 2001.

[33]Tandale, M. D. and Valasek, J., "Structured Adaptive Model Inversion Control to Simultaneously Handle Actuator Failure and Actuator Saturation," *AIAA Guidance, Navigation, and Control Conference*, No. AIAA-2003-5325, Austin, TX, 11-14 August 2003.

[34]Tandale, M. D., Subbarao, K., Valasek, J., and Akella, M. R., "Structured Adaptive Model Inversion Control with Actuator Saturation Constraints Applied to Tracking Spacecraft Maneuvers," *Proceedings of the American Control Conference*, Boston, MA, 2 July 2004.

[35]Tandale, M. D. and Valasek, J., "Adaptive Dynamic Inversion Control with Actuator Saturation Constraints Applied to Tracking Spacecraft Maneuvers," *Proceedings of the 6th International Conference on Dynamics and Control of Systems and Structures in Space*, Riomaggiore, Italy, 18-22 July 2004.

[36]Ahmed, J., Coppola, V. T., and Bernstein, D. S., "Adaptive Asymptotic Tracking of Spacecraft Attitude Motion with Inertia Matrix Identification," *AIAA Journal of Guidance Control and Dynamics*, Vol. 21, Sept-Oct 1998, pp. 684–691.

[37]Khalil, H. K., *Nonlinear Systems*, Prentice Hall, Upper Saddle River, NJ, 3rd ed., 2001.