AIAA 1st Intelligent Systems Technical Conference
20 - 22 September 2004, Chicago, Illinois

AIAA 2004-6220

A Reinforcement Learning - Adaptive Control Architecture for Morphing

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

This paper develops a control methodology for morphing, bringing together the traditionally disparate fields of feedback control and artificial intelligence. The morphing control function, which uses reinforcement learning, is integrated with the trajectory tracking function, which uses adaptive dynamic inversion control. Optimality is addressed by cost functions representing optimal shapes corresponding to specified operating conditions, and an episodic unsupervised 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, unmodeled dynamics and disturbances.

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 only 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 recon-

*Associate Professor & Director, Flight Simulation Laboratory, Aerospace Engineering Department. Associate Fellow, AIAA. valasek@aero.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

figure 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; unsupervised 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 two of the three essential functionalities of Morphing for Mission Adaptation: how to reconfigure, and learning to reconfigure. Called Adaptive-Reinforcement Learning Control (A-RLC), it is a marriage of traditional feedback control and Artificial Intelligence (AI). A-RLC uses Structured Adaptive Model Inversion (SAMI) as the controller for handling time-varying properties, parametric uncertainties, and disturbances. For 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 trajectory tracking module, which handles Smart Block parametric uncertainties, unmodeled dynamics, and disturbances, 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 in the presence of disturbances.

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

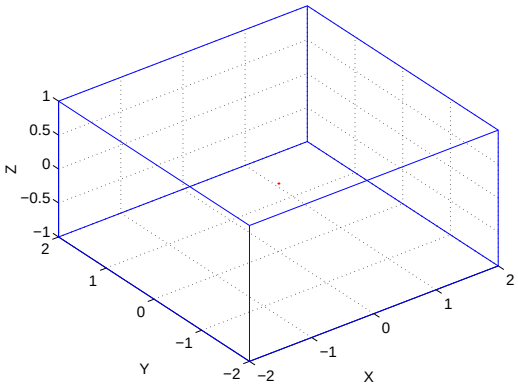


Fig. 1 Shape of the Morphing Smart Block.

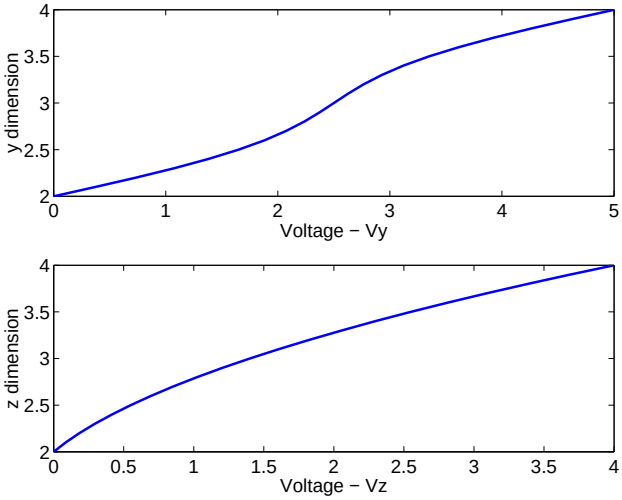


Fig. 2 The Steady State Dimensions corresponding to the Applied Voltage.

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 Eqn.1

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

and similarly for the z dimension

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

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 response is over damped for some cases, and overshoots for others and all the trajectories reach steady state within ten seconds. So a suitable time interval between successive morphings has to be greater than ten seconds.

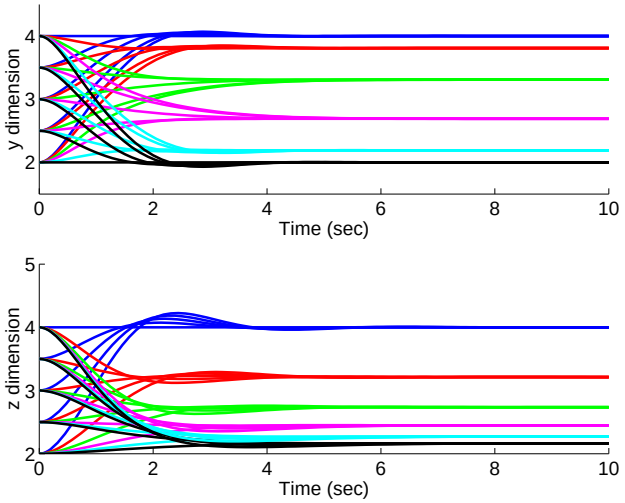


Fig. 3 The Morphing Dynamics for y and z dimensions when subjected to various input voltages - V_y and V_z.

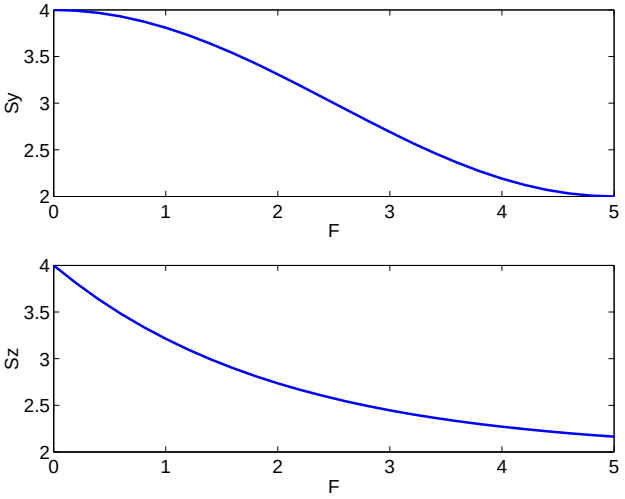


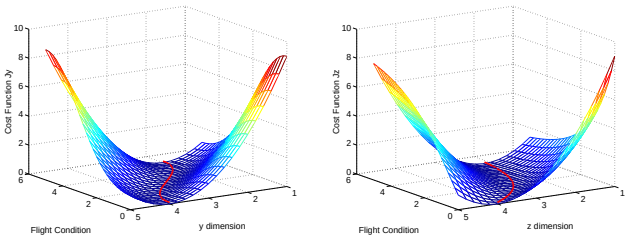
Fig. 4 Optimal Shapes at the different Flight Conditions.

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 $J : [2,4] \times [2,4] \times [0,5] \rightarrow [0,\infty]$

The total cost function J can be written as a sum $J = J_y + J_z$ where J_y and J_z are the costs associated with the y and the z dimension respectively for the different flight conditions. Let $S_y(F)$ be the optimal y dimension at flight condition F and $S_z(F)$ be the optimal z dimension at flight condition F. The optimal shapes are arbitrarily selected to be nonlinear functions of the flight condition and are shown graphically in Fig.4.

$$S_y = 3 + cos(\frac{\pi}{5}F) \tag{3}$$

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



a) J_y. **b)** J_z.

Fig. 5 Cost Function Components J_y and J_z

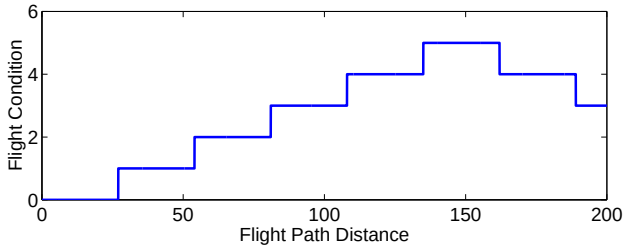


Fig. 6 Flight Condition at various Flight Path Locations.

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

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

where y is any arbitrary y dimension and F is the current flight condition. The surface plot of J_y is shown in Fig.5(a).

Similarly, the cost function J_z is defined as

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

where z is any arbitrary y dimension, F is the current flight condition and S_z is the optimal y dimension at flight condition F. The surface plot of J_z is shown in Fig.5(b). 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 (RL) Module is to learn the optimal control policy that, for a specific flight condition, commands the optimal voltage which produces the optimal shape S_y and S_z. Therefore, the RL module seeks to minimize the total cost J over the entire flight trajectory.

Mathematical Model for the Dynamic Behavior of the Smart Block

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 level variables are d_x, d_y, d_z, ϕ, θ and ψ. The states d_x, d_y, d_z are the positions of the center of mass of the smart block along the inertial X_N, Y_N and Z_N axes. ϕ, θ and ψ are the 3-2-1 Euler angles which give the relative orientation of the body axis with the inertial axis. The acceleration level states are the body-axis linear velocities u, v, w and the body-axis angular velocities p, q, r.

Let $\mathbf{p}_c = \begin{bmatrix} d_x & d_y & d_z \end{bmatrix}^T$, $\mathbf{v}_c = \begin{bmatrix} u & v & w \end{bmatrix}^T$, $\boldsymbol{\sigma} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ and $\boldsymbol{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^T$

The kinematic states and the acceleration states are related by the differential equations

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

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

where

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

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

and $C_\phi = cos(\phi)$, $S_\theta = sin(\theta)$ and so on. The acceleration level differential equations are

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

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

where m is the mass of the smart block, $\mathbf{F}$ is the control force, $\mathbf{F}_d$ is the drag force, I is the body axis moment of inertia, $\mathbf{M}$ is the control torque, $\mathbf{M}_d$ is the drag moment and $\widetilde{\boldsymbol{\omega}}\mathbf{V}$ is the matrix representation of the cross-product between vector $\boldsymbol{\omega}$ and vector $\mathbf{V}$.

Note that for this study the motion of the smart block is simulated in the absence of gravity. Also Eqn.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. The dimensions of the block along the body-axis are x, y, z respectively.

$$\mathbf{F}_d = \frac{-\rho}{2} \begin{bmatrix} u^2 sgn(u) yz \\ v^2 sgn(v) xz \\ w^2 sgn(w) xy \end{bmatrix} \tag{12}$$

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.

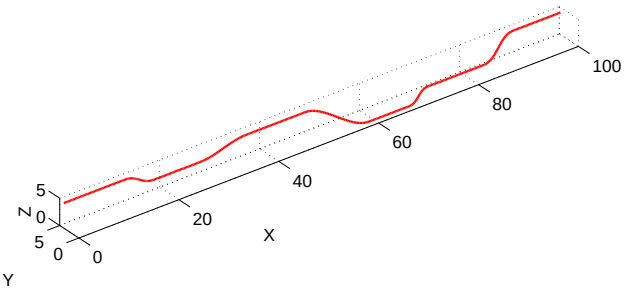


Fig. 7 Reference Trajectory for Positions along Inertial Axis.

$$\mathbf{M}_d = \frac{-\rho}{2} \begin{bmatrix} p^2 sgn(p) x(y+z) \\ q^2 sgn(q) y(x+z) \\ r^2 sgn(r) z(x+y) \end{bmatrix} \tag{13}$$

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 d_y and d_z locations remain constant while the values of d_y and d_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.

Reinforcement Learning Module

Overview of Reinforcement Learning

Reinforcement learning methods specify how the agent changes its policy as a result of its experiences. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over a long run. Reinforcement learning (RL) is a method of learning from interaction to achieve a goal.[7] The learner and decision-maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment and the agent interacts with its environment at each of a sequence of discrete time steps, $t = 0, 1, 2, 3....$ At each time step t, the agent receives some representation of the environment's state, $s_t \in S$, where S is a set of possible states, and on that basis it selects an action, $a_t \in A(s_t)$, where $A(s_t)$ is a set of actions available in state 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. $V^\pi(s)$ is called the state-value function for policy π. Almost all reinforcement learning algorithms are based on estimating $V^*(s)$ and the process of computing $V^\pi(s)$ is called policy evaluation. Using $V^\pi(s)$, π can be improved to a better π', and this process is called policy improvement. $V^{\pi'}(s)$ can then be computed to improve π' to an even better π''. The ultimate goal of RL is to find the optimal policy π^* that has the optimal state-value function, denoted by $V^*(s)$ and defined as $V^*(s) = \max_\pi V^\pi(s)$. This way of finding an optimal policy is called policy iteration. There exist three major methods for policy iteration: dynamic programming, Monte Carlo methods, and temporal-difference learning.

Reinforcement Learning has been applied to 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 RL has been studied by many researchers.[8,9,10] In many applications of RL to control tasks, the state space is too large to enumerate the value function. Some function approximators must be used to compactly represent the value function. For Example, tile coding has been used in many reinforcement learning systems.[11,12,13,14] Other commonly used approaches include neural networks, clustering, nearest-neighbor methods and cerebellar model articulator controller.

Implementation of Reinforcement Learning Module

For the present research, the agent in the smart block problem is its RL 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 optimal 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 RL module has no prior knowledge of the relationship between voltages and the dimensions of the block, as defined by morphing control functions MC_y and MC_z in Eqn.1 and Eqn.2. Also it does not know the relationship between the flight conditions, costs and the optimal shapes as defined in Eqn.4..Eqn.6. However, the RL 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 RL module learns the state value function using an Actor-critic method.[7] Actor-critic methods are classical Temporal-Difference (TD) learning methods that have a separate memory structure to explicitly represent the policy, independent of the value function. The actor has policy functions that are used to select actions and the critic has state-value functions that are used to criticize the actions made by the actor. After each action selection, the critic evaluates the new state by the TD error, as defined in Eqn.14, to determine whether the action is better or worse than expected.

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{14}$$

In Eqn.14, V is the current estimated value function used by the critic and γ is the discount factor. If the TD 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 TD 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

$$\pi_t(s, a) = arg\max_a p(s, a) \tag{15}$$

and the Gibbs softmax policy[7]

$$\pi_t(s, a) = Pr\{a_t = a|st = s\} = \frac{e^{p(s,a)}}{\sum_a e^{p(s,a)}} \tag{16}$$

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

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t \tag{17}$$

where β is a positive step-size parameter.

As the RL problem for this research, has states on continuous domains, its state-value function $V_t^\pi(s)$ at time t needs to be approximated using a parameterized functional form with a parameter vector $\boldsymbol{\theta}_{vt}$. Therefore $V_t^\pi(s)$ depends totally on $\boldsymbol{\theta}_{vt}$, and varies from time-step to time-step only when $\boldsymbol{\theta}_{vt}$ changes. Of the various function approximate function methods available, we represent $V_t^\pi(s)$ using tile coding:

$$V_t^\pi(s) = \sum_{j=1}^{N} \boldsymbol{\theta}_{vtj}\boldsymbol{\phi}_j(s) \tag{18}$$

$$\boldsymbol{\phi}_j(s) = \begin{Bmatrix} 1, if(s - c_j)^T\mathbf{R}^{-1}(s - c_j) \leq 1 \\ 0, if(s - c_j)^T\mathbf{R}^{-1}(s - c_j) > 1 \end{Bmatrix} \tag{19}$$

where c_y is the pre-selected center states vector and $\boldsymbol{\phi}_j(s)$ is called a binary feature. The estimation of $V_t^\pi(s)$ is simply the learning of the parameter vectors $\boldsymbol{\theta}_{vt}$ using some function approximation methods. The

training example used for the function approximation is the TD backup value

$$\tilde{y} = r_{t+1+\gamma V()s_{t+1}} \tag{20}$$

that can be gained after each interaction with the environment. In this paper we use the common gradient descent method to update $\boldsymbol{\theta}_{vt}$

$$\begin{aligned} \boldsymbol{\theta}_{vt} &= \boldsymbol{\theta}_{vt-1} + \alpha(\tilde{y} - \mathbf{H}_t\boldsymbol{\theta}_{vt-1})\mathbf{H}_t^T & (21)\\ &= \boldsymbol{\theta}_{vt-1} + \alpha\delta_t\mathbf{H_t^T} & (22)\\ \mathbf{H}_t &= [\boldsymbol{\phi}_1(s_t)\boldsymbol{\phi}_2(s_t)...\boldsymbol{\phi}_N(s_t)] & (23) \end{aligned}$$

where δ_t is the TD error as defined in Eqn.14; α is the learning rate; and $\mathbf{H}_t$ is the known coefficient matrix. Note that the purpose of the RL module is not to compute the exact $V^{\pi}(s)$, but to learn the optimal policy π^* with the optimal state-value function $V^*(s)$. Also in the process of the policy iteration, an intermediate policy π can be improved to a better policy π' even if its state-value function $V^*(s)$ is not exactly computed.

Similarly, the preference functions of the actor are all represented with tile coding. For each action a, its preference function at time t, $p_{at}(s)$ is defined as follows

$$p_{at}(s) = \sum_{j=1}^{N} \boldsymbol{\theta}_{p_{atj}}\boldsymbol{\phi}_j(s) \tag{24}$$

The training examples used here for the function estimation are the weakening or strengthening preference

$$\tilde{y} = p_{at}(s_t) + \beta\delta_t \tag{25}$$

$\boldsymbol{\theta}_{vt}$ is learned using the gradient descent update $\boldsymbol{\theta}_{p_{atj}}$

$$\begin{aligned} & \boldsymbol{\theta}_{p_{at}}\boldsymbol{\theta}_{p_{a(t-1)}} + \alpha[p_{at}(s_t) + \beta\delta_t - \mathbf{H}_t\boldsymbol{\theta}_{p_{a(t-1)}}]\mathbf{H}_t^T \\ = \; & \boldsymbol{\theta}_{p_{a(t-1)}} + \alpha\beta\delta_t\mathbf{H_t^T} \end{aligned} \tag{26}$$

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 TD error is the only output of the critic and drives all learning in both actor and critic, as suggested in Eqn.22 and Eqn.26. The RL module is summarized in Fig.8.[7]

The RL 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 RL 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 Eqn.16, 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 Eqn.15, that allows it to totally exploit its previous experience.

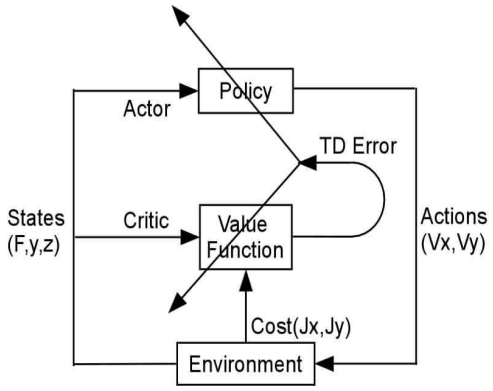


Fig. 8 Reinforcement Learning Module.

Structured Adaptive Model Inversion

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)[15] is based on the concepts of Feedback Linearization,[16] Dynamic Inversion, and Structured Model Reference Adaptive Control (SMRAC).[17,18] 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.[19–21] 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. The SAMI approach has been extended to handle actuator failures and to facilitate correct adaptation in presence of actuator saturation.[22–24]

Mathematical Formulation of the SAMI Controller for Attitude Control

The attitude equations of motion for the smart block are given by Eqn.8 and Eqn.11. Without the angular

drag and the $\dot{I}\boldsymbol{\omega}$ term, Eqn.8 and Eqn.11 can be manipulated to obtain the following form[15]

$$I_a^*(\boldsymbol{\sigma})\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}} = P_a^T(\boldsymbol{\sigma})\mathbf{M} \tag{27}$$

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

$$\begin{aligned} P_a(\boldsymbol{\sigma}) &\triangleq J_a^{-1}(\boldsymbol{\sigma}) && (28)\\ I_a^*(\boldsymbol{\sigma}) &\triangleq P_a^T I P_a && (29)\\ C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}) &\triangleq -I_a^*\dot{J}_a P_a + P_a^T[\widetilde{P_a\dot{\boldsymbol{\sigma}}}]IP_a && (30)\end{aligned}$$

The left hand side of Eqn.27 can be linearly parameterized as follows

$$I_a^*(\boldsymbol{\sigma})\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}} = Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})\boldsymbol{\theta} \tag{31}$$

where $Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})$ is a regression matrix and $\boldsymbol{\theta}$ is the constant inertia parameter vector 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

$$I\boldsymbol{\nu} = \Lambda(\boldsymbol{\nu})\boldsymbol{\theta}, \qquad \forall \boldsymbol{\nu} \in \mathbb{R}^3 \tag{32}$$

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

$$\Lambda(\boldsymbol{\nu}) \triangleq \begin{bmatrix} \nu_1 & 0 & 0 & \nu_2 & \nu_3 & 0 \\ 0 & \nu_2 & 0 & \nu_1 & 0 & \nu_3 \\ 0 & 0 & \nu_3 & 0 & \nu_1 & \nu_2 \end{bmatrix} \tag{33}$$

The terms on the left hand side of Eqn.27 can be written as

$$\begin{aligned} I_a^*\ddot{\boldsymbol{\sigma}} &= P_a^T I P_a \ddot{\boldsymbol{\sigma}} \\ &= P_a^T \Lambda(P_a\ddot{\boldsymbol{\sigma}})\boldsymbol{\theta} && (34)\\ C_a^*\dot{\boldsymbol{\sigma}} &= -P_a^T I P_a \dot{J}_a P_a \dot{\boldsymbol{\sigma}} + P_a^T[\widetilde{P_a\dot{\boldsymbol{\sigma}}}]IP_a\dot{\boldsymbol{\sigma}} \\ &= P_a^T\{-\Lambda(P_a\dot{J}_aP_a\dot{\boldsymbol{\sigma}}) + [\widetilde{P_a\dot{\boldsymbol{\sigma}}}]\Lambda(P_a\dot{\boldsymbol{\sigma}})\}\boldsymbol{\theta} && (35)\end{aligned}$$

Combining Eqns.34 and 35 we have the linear minimal parametrization for the Inertia Matrix.[25]

$$\begin{aligned} & I_a^*(\boldsymbol{\sigma})\ddot{\boldsymbol{\sigma}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}} \\ =\ & P_a^T\{\Lambda(P_a\ddot{\boldsymbol{\sigma}}) - \Lambda(P_a\dot{J}_aP_a\dot{\boldsymbol{\sigma}}) + [\widetilde{P_a\dot{\boldsymbol{\sigma}}}]\Lambda(P_a\dot{\boldsymbol{\sigma}})\}\boldsymbol{\theta} \\ =\ & Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})\boldsymbol{\theta} && (36)\end{aligned}$$

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

$$I_a^*\ddot{\boldsymbol{\epsilon}} = I_a^*\ddot{\boldsymbol{\sigma}} - I_a^*\ddot{\boldsymbol{\sigma}}_{\boldsymbol{r}} \tag{37}$$

Adding $(C_{da} + C^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\ \boldsymbol{\epsilon}$ on both sides, where C_{da} and K_{da} are the design matrices,

$$\begin{aligned} & I_a^*\ddot{\boldsymbol{\epsilon}} + (C_{da} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} \\ =\ & I_a^*\ddot{\boldsymbol{\sigma}} - I_a^*\ddot{\boldsymbol{\sigma}}_{\boldsymbol{r}} + (C_{da} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} && (38)\end{aligned}$$

The RHS of Eqn.38 can be written as

$$\begin{aligned} (I_a^*\ddot{\boldsymbol{\sigma}} &+ C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}}) - (I_a^*\ddot{\boldsymbol{\sigma}}_{\boldsymbol{r}} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}})\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}}) \\ &+ C_{da}\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} && (39)\end{aligned}$$

From Eqn.27 and the construction of Y similar to Eqn.36, the RHS of Eqn.38 can be further written as

$$P_a^T\mathbf{M} - Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)\boldsymbol{\theta} + C_{da}\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} \tag{40}$$

So the control law can be now chosen as

$$\mathbf{M} = P_a^{-T}\{Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)\boldsymbol{\theta} - C_{da}\dot{\boldsymbol{\epsilon}} - K_{da}\boldsymbol{\epsilon}\} \tag{41}$$

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,[21] adaptive estimates for the inertia parameters $\hat{\boldsymbol{\theta}}$ will be used for calculating the control.

$$\mathbf{M} = P_a^{-T}\{Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)\hat{\boldsymbol{\theta}} - C_{da}\dot{\boldsymbol{\epsilon}} - K_{da}\boldsymbol{\epsilon}\} \tag{42}$$

With the control law given in Eqn.42 the closed loop dynamics takes the following form

$$I_a^*\ddot{\boldsymbol{\epsilon}} + (C_{da} + C_a^*(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}}))\dot{\boldsymbol{\epsilon}} + K_{da}\boldsymbol{\epsilon} = Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\ddot{\boldsymbol{\sigma}})\tilde{\boldsymbol{\theta}} \tag{43}$$

where $\tilde{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}$. 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

$$V = \frac{1}{2}\dot{\boldsymbol{\epsilon}}^T I_a^*\dot{\boldsymbol{\epsilon}} + \frac{1}{2}\boldsymbol{\epsilon}^T K_{da}\boldsymbol{\epsilon} + \frac{1}{2}\tilde{\boldsymbol{\theta}}^T\tau^{-1}\tilde{\boldsymbol{\theta}} \tag{44}$$

where τ^{-1} is a symmetric positive definite gain matrix. Taking the derivative of the lyapunov function along the closed loop trajectories given by Eqn.43 and simplifying further

$$\begin{aligned} \dot{V} &= \dot{\boldsymbol{\epsilon}}^T[\frac{1}{2}\dot{I}_a^* - C_a^*]\dot{\boldsymbol{\epsilon}} - \dot{\boldsymbol{\epsilon}}^T C_{da}\dot{\boldsymbol{\epsilon}} \\ &+ (\dot{\boldsymbol{\epsilon}}^T Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r) + \dot{\tilde{\boldsymbol{\theta}}}^T\tau^{-1})\tilde{\boldsymbol{\theta}} && (45)\end{aligned}$$

The first term vanishes because $[\frac{1}{2}\dot{I}_a^* - C_a^*]$ is skew symmetric. We set the coefficient of $\tilde{\boldsymbol{\theta}}$ to 0, to obtain the adaptive laws. From the definition of $\tilde{\boldsymbol{\theta}}$ and assuming that the true parameter $\boldsymbol{\theta}$ remains constant,

$$\dot{\tilde{\boldsymbol{\theta}}} = -\tau Y_a(\boldsymbol{\sigma},\dot{\boldsymbol{\sigma}},\dot{\boldsymbol{\sigma}}_{\boldsymbol{r}},\ddot{\boldsymbol{\sigma}}_r)^T\dot{\boldsymbol{\epsilon}} \tag{46}$$

This update law renders the derivative of the lyapunov function

$$\dot{V} = -\dot{\boldsymbol{\epsilon}}^T C_{da}\dot{\boldsymbol{\epsilon}} \tag{47}$$

Thus the derivative is negative semi definite. From Eqn.44 and Eqn.47 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, asymptotic stability of the tracking error dynamics can be concluded for bounded reference trajectories.

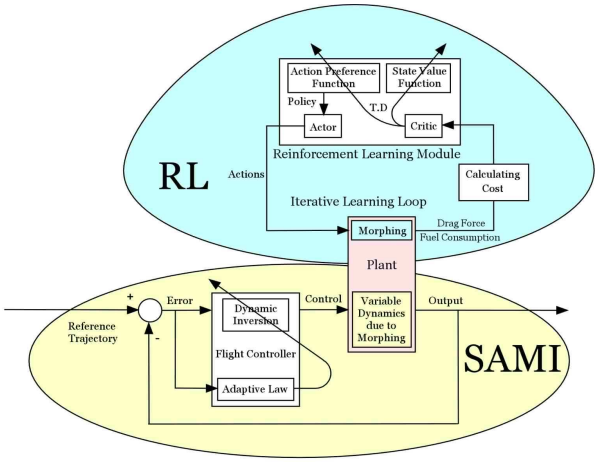


Fig. 9 Adaptive-Reinforcement Learning Control Architecture.

Mathematical Formulation of the SAMI Controller for Control of Linear States

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

$$I_l^* \ddot{\mathbf{p}}_c + C_l^* \dot{\mathbf{p}}_c = P_l^T \mathbf{F} \tag{48}$$

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

$$P_l \triangleq J_l^{-1} \tag{49}$$

$$I_l^* \triangleq P_l^T m P_l \tag{50}$$

$$C_l^* \triangleq -I_l^* \dot{J}_l P_l + P_l^T \tilde{\boldsymbol{\omega}} m P_l \tag{51}$$

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

$$\mathbf{F} = P_l^{-T} \{ Y_l m - C_{dl} \dot{\boldsymbol{\varepsilon}} - K_{dl} \boldsymbol{\varepsilon} \} \tag{52}$$

where

$$\begin{aligned} Y_l &= P_l^T P_l \ddot{\mathbf{p}}_{c(ref)} - P_l^T P_l \dot{J}_l P_l \dot{\mathbf{p}}_{c(ref)} \\ &+ P_l^T \tilde{\boldsymbol{\omega}} P_l \dot{\mathbf{p}}_{c(ref)} \end{aligned} \tag{53}$$

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

$$\dot{\hat{m}} = -\Gamma Y_l^T \dot{\boldsymbol{\varepsilon}} \tag{54}$$

where Γ^{-1} is a symmetric positive definite matrix.

A-RLC Architecture Functionality

The Adaptive-Reinforcement Learning Control Architecture is composed of two sub-systems: Reinforcement Learning (RL) and Structured Adaptive Model Inversion (SAMI) (Fig.9). 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 RL sub-system at the top of Fig.9 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 drag force and fuel consumption, is evaluated with the cost function and then sent 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.9, 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.

Numerical Example

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. The numerical example consists of an unsupervised learning simulation which learns the optimal shape change policy for the morphing smart block, and tracks a specified trajectory, using only a limited number of unsupervised learning episodes. Learning performance is demonstrated first, followed by a single episode with new arbitrary conditions to demonstrate trajectory tracking performance. The optimal shape change policy is learned using a sequence of action selection policies to avoid the exploration-exploitation dilemma. The sequence consists of a uniform probability policy in the initial stage, a Gibbs softmax policy in the intermediate stage, and a greedy policy in the final stage. A total of 200 unsupervised learning episodes are used, each consisting of a single 200 second transit through a 100 meter long path. The sequence of the flight conditions is random, and changes twice during each episode. For learning purposes, the shape change is commanded at ten second intervals. The reference trajectory to be tracked in each episode is generated arbitrarily. Finally, trajectory tracking performance is demonstrated 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.

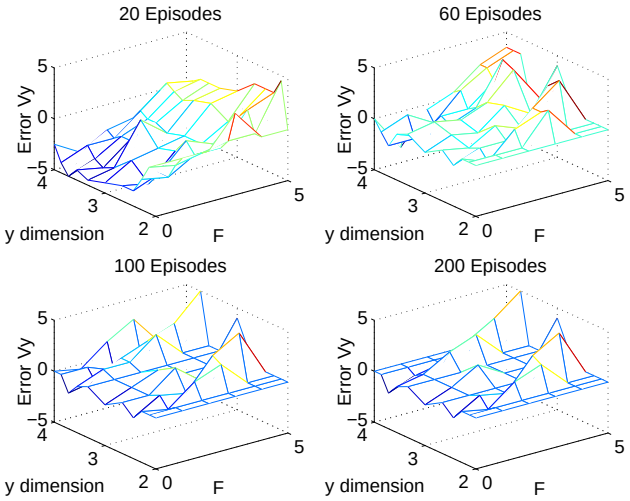


Fig. 10 Error in the Action Preference Function after a) 20 episodes, b) 60 episodes, c) 100 episodes , d) 200 episodes.

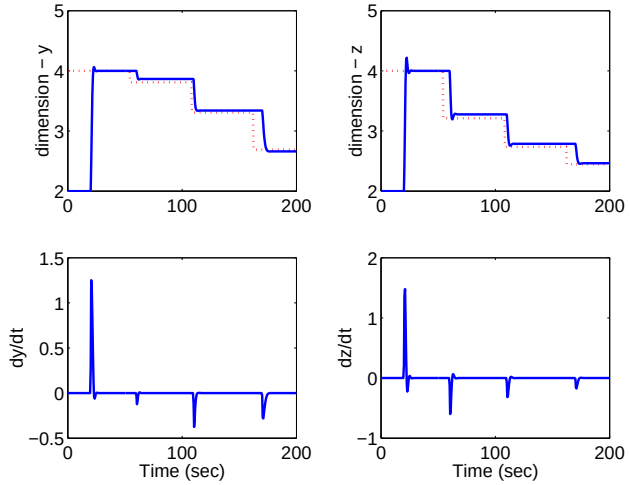


Fig. 11 Comparison between the True Optimal Shape and the Shape Learned by the RL Agent, for a Sequence of Flight Conditions.

Learning the Optimal Policy for Shape Change

For the definitions of the morphing control functions and the cost functions defined by Eqn.1 to Eqn.6, this RL problem can be decoupled into two independent problems that correspond to the y and z dimensions respectively. For the y dimension RL 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 $\{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$. For simplicity, the training is conducted only at six discrete flight conditions: 0,1,2,3,4,5. Fig.10 is a graphical representation of the errors in the preferred voltages of the y-dimension selected by the Actor, when compared

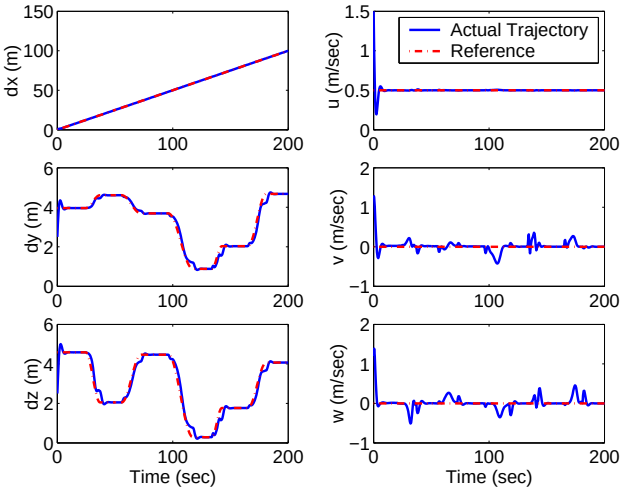


Fig. 12 Time Histories of the Linear States.

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 errors decrease, indicating the action preference functions are being learned accurately. However, we see that the decrease in error is less significant from 100 to 200 episodes than from 20 to 60 episodes because, firstly, the focus shifts from exploration to exploitation, and secondly, the action preference functions asymptotically converge to the ones of the optimal policy. Fig.11 compares the time histories of the learned 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 RL agent. Values of the learned shape are seen to be close, but not exact to, the optimal values. The RL agent cannot learn the optimal policy exactly because the applied control has discrete values, with a minimum resolution of 0.5 volts. Therefore, it cannot take any on arbitrary continuous values. This result is typical for all episodes in the later learning stages.

Trajectory Tracking Performance

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 and Fig.13 show the time histories of the linear and angular states respectively. The deviations from the reference trajectory seen in Fig.12, subplot 2 are due to rapid changes in the smart block's dimensions. The adaptive control has to undergo rapid adaptation during a shape change, at ten second intervals, as seen in Fig.14. The adaptive controller formulation assumes that the true parameters are constant, hence the shape change causes disturbances in the tracking behavior. Fig.15 shows that the control forces and moments are well behaved and within saturation limits.

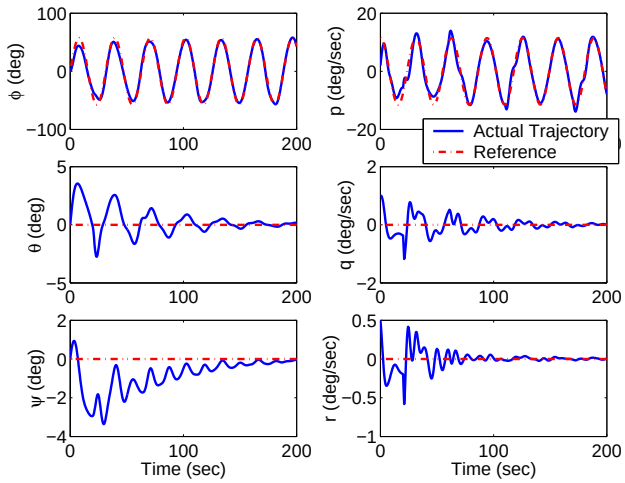


Fig. 13 Time Histories of the Angular States.

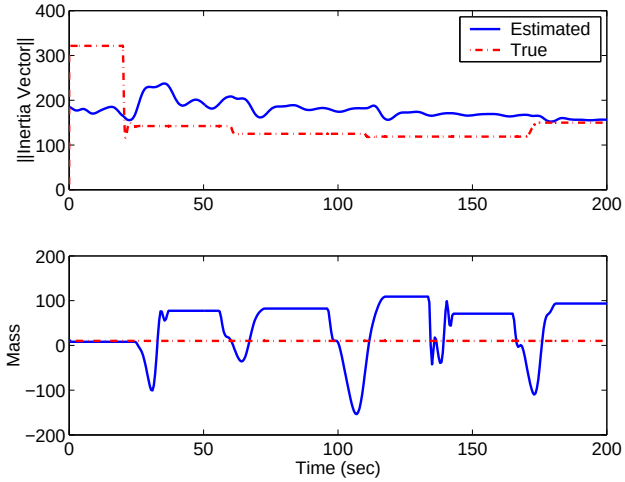


Fig. 14 Time Histories of the Adaptive Parameters.

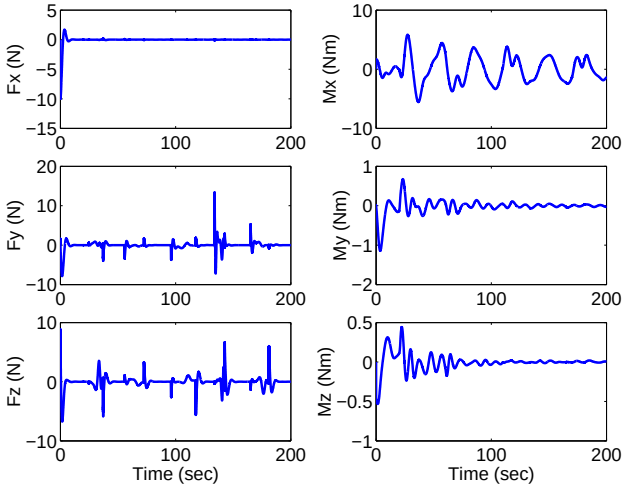


Fig. 15 Time Histories of the Control Forces and Moments.

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 term $\dot{I}\boldsymbol{\omega}$ in the Eqn.11, so $\dot{I}\boldsymbol{\omega}$ acts as unmodeled dynamics and perturbs the tracking. However, the SAMI controller is able to maintain adequate tracking performance.

The stability proof guarantees asymptotic stability of the tracking error only for constant true parameters. Here it has been applied it for a morphing smart block. Considering the timescales for morphing for mission adaptation, the mission specifications change very slowly compared to the rate of adaptation for the SAMI controller. Also, after the shape changes for a mission specification it will be held constant for the mission duration. Thus, due to the timescale separation and the time interval between two successive changes in mission specification, the shape change behaves like a piecewise constant parametric change for the adaptive controller, which can be handled effectively.

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.

Conclusions

This paper proposed and developed a control methodology for morphing aircraft, combining machine learning and adaptive dynamic inversion control. 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 learning the optimal shape change policy, the sequence of action selection policies consisting of uniform probability policy to Gibbs softmax policy to greedy policy works well, and avoids the exploration-exploitation dilemma. Learning times and learning performance are strongly affected by both the morphing dynamics of the structure (stiffness, frequency, damping ratio), and the cost function.

2.Coordination and timing between elements in the system must be tuned, as shape change commands, morphing dynamics, learning performance, and adaptive control must be balanced to achieve synergy and therefore good performance. Since Structured Adaptive Model Inversion Control assumes that unknown

varying parameters are piecewise constant and change slowly, the time interval between shape change commands cannot be arbitrarily small. However, Morphing for Mission Adaptation should not pose a problem, since compared to time constants of system elements, the time interval between shape changes is relatively large and the shape remains constant for long durations.

3.Adaptive-Reinforcement Learning Control is a promising candidate methodology for the control of Morphing for Mission Adaptation. For the numerical example presented, the controller maintains asymptotic tracking in the presence of initial condition errors, external disturbances, and unmodeled dynamics. In addition, the reinforcement learning module can function online, which may lead to better performance since learning continues as the system operates.

Acknowledgment

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.

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]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.

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

[10]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.

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

[12]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.

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

[14]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.

[15]Subbarao, K., *Structured 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.

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

[17]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.

[18]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.

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

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

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

[22]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.

[23]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.

[24]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.

[25]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.