ARL-TR-9433 ● APR 2022

DEVCOM ARMY RESEARCH LABORATORY

Adaptive Control Validation Using a MATLAB-Based CFD/RBD Coupled Simulation

by Jubaraj Sahu, Benjamin Gruenwald, and Bradley T Burchett

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9433 ● APR 2022

Adaptive Control Validation Using a MATLAB-Based CFD/RBD Coupled Simulation

Jubaraj Sahu
Oak Ridge Associated Universities

Benjamin Gruenwald and Bradley T Burchett
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
April 2022	Technical Report	1 October 2021–28 February 2022

4. TITLE AND SUBTITLE
Adaptive Control Validation Using a MATLAB-Based CFD/RBD Coupled Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jubaraj Sahu, Benjamin Gruenwald, and Bradley T Burchett

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLW-WD
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-9433

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Bradley T Burchett, 0000-0002-1934-0537

14. ABSTRACT
A new high-fidelity coupled computational fluid dynamics (CFD)/rigid body dynamics (RBD) simulation was built with several novel features aimed at validating adaptive control laws on a high-speed missile. First, the MATLAB programming language was used to implement the RBD and control laws, making the environment user-friendly for a wide community of control engineers. Second, the simulation provides for hundreds of dynamic states allowing for actuator dynamics, reference models, adaptive gains, reference signal integrators, and so forth. Third, numerical integration is performed using a multistep Adams–Moulton method such that force-moment information from prior time steps is included to smooth the predictions. With this new simulation, we demonstrate regulation and tracking control for a supersonic projectile using model reference adaptive control. The adaptive feedback law is shown to provide superior stability and tracking performance in the presence of significant modelling errors, thus achieving specified performance while reducing the precision required in gain scheduling.

15. SUBJECT TERMS
adaptive control, projectile aerodynamics, wind tunnel, CFD, high-speed missile, Weapons Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Bradley T Burchett
Unclassified	Unclassified	Unclassified	UU	36	19b. TELEPHONE NUMBER (Include area code) (812) 201-0390

Contents

List of Figures

List of Tables

1. Introduction

Improved computer technology and state-of-the-art numerical procedures now enable solutions to complex 3-D problems associated with projectile and missile aerodynamics. Detailed understanding of controlled flight behaviors is critical for enhanced vehicle maneuverability of these munitions. Advanced computational techniques are being developed to understand flight behaviors of both unguided and guided projectiles. One such technique couples computational fluid dynamics (CFD) and rigid body dynamics (RBD) for the simultaneous prediction of unsteady aerodynamics and flight dynamics in an integrated manner.[1] The coupled approach is to capture static and dynamic aerodynamic behavior over short time durations with different motions. Performing coupled simulations in this manner allows for screening of situations where conventional aerodynamic models based on static wind tunnel or CFD techniques break down. These instances are encountered more often as wider classes of munitions (small-medium-large caliber) feature control inputs and the associated flow complexity such as interactions, unsteadiness, and high angle of attack. Thus, a major benefit of these coupled simulations is to mitigate risk of unanticipated flight behaviors during unguided and especially guided free-flight experiments.

Coupled CFD/RBD simulations have been used to simulate complete projectile bodies in flight for about 15 years. US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory researchers have published many works describing the development and application of coupled simulations during this period.[1–10] Over this short lifespan, such simulations have been applied to many problems including projectile ballistics, store separation, booster separation, and aerodynamic decelerators. Computed results obtained from the coupled simulations were validated with available free-flight test data at subsonic, transonic, and—in some cases—supersonic speeds. The coupled approach has also been used for high-fidelity RBD and prescribed motion maneuvers of aircraft and other flying vehicles.[11–13] Due to the increased fidelity available with such tools, CFD packages such as Kestrel are now offering RBD simulation coupling as part of the off-the-shelf software build.[12] As computational resources continue to grow, more researchers are seeking to apply the method to other high-speed and high-Reynolds number problems.[14]

Coupled CFD/RBD was also integrated to the CFD/RBD technique by adding a flight control system (FCS) module for computation of controlled maneuvers using canard control. In this coupled procedure, the aerodynamic forces and moments were computed at every time step in the CFD solver and transferred to a Fortran-based RBD/FCS code.[15] The RBD/FCS code not only provides RBD state

variables, but also the flight control variables based on a given FCS design as output. The FCS design allows for modelling of both controlled and prescribed motions. The output of the RBD state and the control variables are transferred to the CFD flow solver, which then computes the aerodynamic forces and moments subject to these RBD state and control variables. For example, for a canard-controlled projectile the output of the FCS variables would be the canard deflection angles. As canards are deflected with the FCS-generated deflection angles, the flow solver must account for the canard motion. A recent study[7] investigated roll-only control of a canard-controlled finned projectile using the coupled procedure. Coupled CFD/RBD/FCS simulations were conducted with canard deflections derived from a roll controller and results from coupled simulations were validated with available wind tunnel data. This research was extended further to a highly maneuverable canard-controlled, finned projectile for roll-pitch-yaw motions, but with open-loop canard control[9] and closed-loop control.[10] Although some progress has been made, a fully coupled and validated CFD/RBD/FCS capability does not yet exist for computation of unsteady aerodynamics and flight dynamics associated with complex guided maneuvers using roll-pitch-yaw. Additionally, the FCS design is limited because advanced control algorithms are difficult to design and test using Fortran-based RBD/FCS code.

Therefore, current research efforts are focused on integrating an in-house FCS into the coupled CFD/RBD method for simulations of guided-control maneuvers. The resulting coupled CFD/RBD/FCS technique can be used for open- and closed-loop control maneuvers where canards or fins could be deflected as desired based on the control algorithm to provide the control authority needed. The canard/fin deflections output of the flight control element is used to move the grid (locations and velocities) for the next CFD time step computation. CFD computes the aerodynamic forces and moments that dictate the projectile flight motion and subsequent controlled deflections subject to the control algorithm.

In this work, we present the development of a new coupled CFD/RBD simulation using the CFD++ commercial software, and MATLAB RBD and FCS simulations. A ".c" language interface ties the two main simulation environments together. This simulation was developed specifically to make flight control validation accessible to a wide audience of control engineers who prototype their control designs in the MATLAB/Simulink environment. In addition to using the MATLAB environment, we added three significant enhancements to the existing in-house simulation framework.

First, continuous states from all subsystems (plant, actuator, control laws) are concatenated in one long system-state vector and integrated simultaneously for more realistic continuous time simulation. The current implementation provides for

up to 300 continuous states; however, expansion is easily facilitated by modifications to the .c-based interface. To avoid difficulty compiling the MATLAB codes, we chose to make the MATLAB memory space completely volatile, which required the state history to be stored entirely on the CFD++ side.

Second, actuator and control system-state derivatives are sequestered to a subfunction to achieve modularity. Simulation users can provide their own actuator models and dynamic control systems without modifying the baseline CFD/RBD simulation engine.

Third, the numerical integration scheme was modified from fourth-order Runge–Kutta to a third-order Adams–Moulton multistep method. The previous method suffered from a subtle inaccuracy caused by force/moment predictions not being updated at RK4 fractional time-step predictions. The small time step required by coupled simulations made this a minor error; however, multistep methods use derivative information from previous time steps and will therefore use past force/moment predictions. This provides an extra measure of smoothness in the state history prediction. Because this method requires storage of the previous two state derivative estimates, the storage required for continuous states is trebled. However, when control laws require state derivatives, these are easily gleaned from storage. Also, by demonstrating one multistep method, we make it easy for the user to choose from others.

In this report, we validate and demonstrate the simulation capabilities through several scenarios: open-loop CFD/RBD motion, single- and multiaxis regulation, linear quadratic regulator (LQR) tracking control without adaptation, LQR tracking control with model reference adaptive control (MRAC) adaptation and hedging, and LQR tracking control with MRAC-Extended Reference Model (ERM) adaptation. The authors are not aware of any previous coupled CFD simulations of adaptive control.

2. Coupled CFD/RBD: A Brief Overview

Research efforts are ongoing at the DEVCOM Army Research Laboratory (ARL) to perform multidisciplinary-coupled CFD/RBD computations for the flight trajectory of a complex guided projectile system. A time-accurate approach is used in the present work that requires significant computer resources. The time-accurate coupled approach requires that the six-degrees-of-freedom (6-DOF) body dynamics be computed at each repetition of a flow solver. In the coupled CFD/RBD procedure, the forces and moments are computed at every CFD time step and transferred to a 6-DOF module that computes the body's response to the forces and moments. The response is converted into translational and rotational accelerations

that are integrated to obtain translational and rotational velocities and then integrated once more to obtain linear position and angular orientation.

The CFD capability used here solves the Navier–Stokes equations and incorporates advanced boundary conditions and grid motion capabilities. The complete set of 3-D time-dependent Navier–Stokes equations is solved in a time-accurate manner for simulations of virtual fly-outs. A commercially available code, CFD++,[16,17] is used for the time accurate and unsteady CFD simulations. The basic numerical framework in the code contains unified-grid, unified-physics, and unified-computing features. Details of the basic numerical framework can be found in works by Peroomian et al.[16,17] Only a brief synopsis of this framework and methodology is provided.

The 3-D time-dependent Reynolds-averaged Navier–Stokes equations are solved using the following finite volume method:

$$\frac{\partial}{\partial t}\int_V \mathbf{W} dV + \oint [\mathbf{F} - \mathbf{G}] \cdot dA = \int_V \mathbf{H} dV \tag{1}$$

where **W** is the vector of conservative variables, **F** and **G** are the inviscid and viscous flux vectors, respectively, **H** is the vector of source terms, V is the cell volume, and A is the surface area of the cell face.

Several techniques such as implicit scheme and relaxation are used to achieve faster convergence. Use of an implicit scheme circumvents the stringent stability limits suffered by their explicit counterparts, and successive relaxation allows update of cells as information becomes available and thus aids convergence. Second-order discretization was used for the flow variables and the turbulent viscosity equation. The turbulence closure is based on topology-parameter-free formulations. A realizable two-equation k-ϵ turbulence model[18] was used for the computation of turbulent flows. These models are ideally suited to unstructured meshing and massively parallel processing due to their independence from constraints related to the placement of boundaries and/or zonal interfaces.

For time-accurate simulations of virtual fly-outs that are of interest here, a dual-time-stepping procedure was used to achieve the desired time accuracy in the time-accurate solutions. The term “dual-time-step” implies the use of two time steps. The first is an “outer” or global (and physical) time step that corresponds to the time discretization of the physical time variation term. This fixed time step is applied to every cell (not separately varying). An artificial “inner” or “local” time variation term is added to the basic physical equations. This time step and corresponding “inner-iteration” strategy is chosen to help satisfy the physical transient equations to the desired degree. An order of magnitude reduction in the

residues is usually sufficient to produce a good transient iteration. This typically requires 10–20 internal iterations to achieve and depends on the magnitude of the outer time step, the nature of the problem, the nature of the boundary conditions, and the consistency of the mesh with respect to the physics.

Grid velocity is assigned to each mesh point. The entire grid was moved to account for the motion of the projectile. To account for rigid body dynamics, the grid point velocities were set as if the grid is attached to the rigid body with 6-DOF. The basic CFD solution technique described here is coupled with an RBD simulation (described in Section 3) for simultaneous prediction of unsteady aerodynamics and flight dynamics for uncontrolled and controlled flights.

Typically, the coupled solution procedure requires three steps. First, we begin with a computation performed in the "steady-state mode" with the grid velocities prescribed to account only for the translational motion component of the complete set of initial conditions. At the second step, we also impose the angular orientations from the initial conditions. At this stage spin is normally added. Computations are performed in a time-accurate mode for a desired number of time steps (usually 500 time steps for nonspinning and 720 time steps for spinning cases). Converged solution from this second step provides the initial condition for the third step. In the third step, the remaining rotational velocity components (pitch and yaw) are added and a coupled CFD/RBD computation is performed in time-accurate mode. The solution from the third step should correspond to the complete set of initial conditions that includes all translational and rotational velocity components and accounts for initial position and angular orientations.

For controlled flights, an FCS is integrated into the coupled CFD/RBD method, and the resulting CFD/RBD/FCS can then be used for simulations of control maneuvers (open- or closed-loop). Canards or fins are deflected as desired based on the control algorithm to provide the control authority needed. The canard/fin deflections output of the flight control element is used to move the grid (locations and velocities) for the next CFD time-step computation. The CFD computes the aerodynamic forces/moments that dictate the projectile flight motion and subsequent controlled deflections subject to the control algorithm.

3. Six-Degree of Freedom Rigid Body Dynamics

The CFD++ simulation uses a 12-state input vector consisting of the vehicle position and mass center velocity in ground-fixed Cartesian coordinates, $\mathbf{X} = \{x \quad y \quad z\}$ and $\dot{\mathbf{X}} = \{\dot{x} \quad \dot{y} \quad \dot{z}\}$, respectively, the vehicle orientation expressed as the standard set of aircraft Euler angles ($\mathbf{\Theta} = \{\phi \quad \theta \quad \psi\}$), and the three body-

fixed angular rates ($\boldsymbol{\omega} = \{p \quad q \quad r\}$). From these inputs it renders force/moment predictions in the ground-fixed frame.

The relationships between ground- and body-fixed coordinates are illustrated in Figs. 1 and 2.

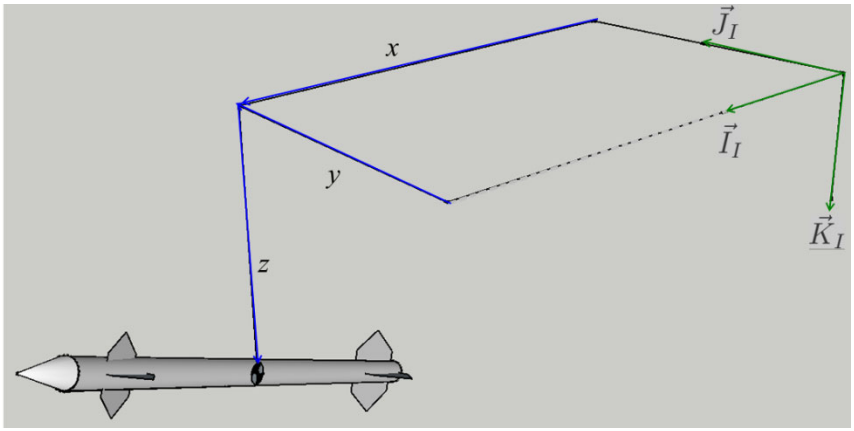

Fig. 1 Ground-fixed Cartesian coordinates for projectile cg position defined

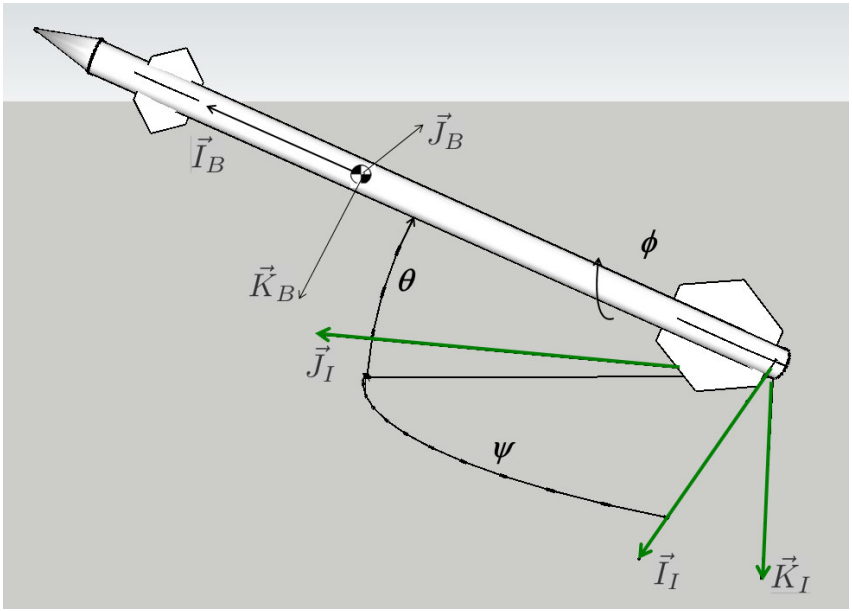

Fig. 2 Standard Euler angles defined

Our 6-DOF RBD model uses a standard-state vector consisting of the concatenation of vehicle cg position in ground-fixed Cartesian coordinates, $\mathbf{X} = \{x \quad y \quad z\}$, orientation as the customary set of Euler angles, $\boldsymbol{\Theta} = \{\phi \quad \theta \quad \psi\}$, velocity in the body-fixed frame, $\mathbf{U} = \{u \quad v \quad w\}$, and body-angular rates in the body frame, $\boldsymbol{\omega} = \{p \quad q \quad r\}$. Thus, the overall state is $\boldsymbol{\zeta} = \{\mathbf{X} \quad \boldsymbol{\Theta} \quad \mathbf{U} \quad \boldsymbol{\omega}\}$.

Since the CFD uses ground-fixed velocity, forces, and moments, the first step in the RBD simulation is to transform these quantities into the body frame. Given

$$\mathbf{T}_{i2b} = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{bmatrix}$$

the body-fixed velocity vector is found as:

$$\mathbf{U} = \mathbf{T}_{i2b}\dot{\mathbf{X}}\,, \tag{2}$$

the body-fixed force vector $\mathbf{F^{RBD}} = \{X \quad Y \quad Z\}$ is given as:

$$\mathbf{F^{RBD}} = \mathbf{T}_{i2b}\mathbf{F^{CFD}}\,, \tag{3}$$

and the body-fixed moment vector $\mathbf{M^{RBD}} = \{l \quad m \quad n\}$ is given as:

$$\mathbf{M^{RBD}}\mathbf{=T}_{i2b}\mathbf{M^{CFD}}\,. \tag{4}$$

The CFD force vector does not include gravity, which is found as:

$$\vec{\mathbf{F}}_{\boldsymbol{grav}} = \begin{Bmatrix} -s_\theta \\ s_\phi c_\theta \\ c_\phi c_\theta \end{Bmatrix} mg = \mathbf{T}_{i2b} \begin{Bmatrix} 0 \\ 0 \\ mg \end{Bmatrix}.$$

And simply added to render the total force on the projectile:

$$\mathbf{F}_{\boldsymbol{tot}} = \mathbf{F^{RBD}} + \vec{\mathbf{F}}_{\boldsymbol{grav}}\,.$$

Once the required quantities are found in the body frame, the derivatives of the state vector follow a standard formulation:

$$\begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{Bmatrix} = \mathbf{T}_{i2b}^{T} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{5}$$

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \mathbf{T}_{b2e} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{6}$$

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = \begin{Bmatrix} X/m \\ Y/m \\ Z/m \end{Bmatrix} - \mathbf{S} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{7}$$

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = [\mathbf{I}]^{-1} \left[\begin{Bmatrix} l \\ m \\ n \end{Bmatrix} - \mathbf{S}[\mathbf{I}] \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \right] \tag{8}$$

where

$$\mathbf{T}_{b2e} = \begin{bmatrix} 1 & s_\phi t_\theta & -c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}.$$

And [**I**] is the 3-D body-fixed inertia tensor.

3.1 Actuator Dynamics

The actuators in any coupled CFD/RBD study must have finite bandwidth. If the actuator makes a large deflection between time steps, CFD solution may encounter a singularity since the mesh is required to make a corresponding large adjustment. For this study, we implemented a simple first-order actuator model with a bandwidth of 330 rad/s. Thus, the actuator position δ_i is related to the commanded position δ_{CMDi} through the first order ordinary differential equation (ODE):

$$\dot{\delta}_i = -330\delta_i + 330\delta_{CMDi} \, . \tag{9}$$

The projectile simulated here has four independently controlled trailing edge flaps; therefore, four instances of this ODE are implemented, requiring four state variables. More numerous actuators and higher order actuator dynamics may be implemented in our simulation environment by assigning the requisite dynamic states and programming the ODEs.

3.2 Numerical Integration

To limit calls to the CFD solver and capitalize on the history of force/moment predictions, a third-order Adams–Moulton numerical integration scheme was used. This method can be compactly written as

$$y_{n+1} = y_n + \frac{h}{12}(23\dot{y}_n - 16\dot{y}_{n-1} + 5\dot{y}_{n-2}) \tag{10}$$

$$y_{n+1} = y_n + \frac{h}{12}(5\dot{y}_{n+1} + 8\dot{y}_n - \dot{y}_{n-1}) \, , \tag{11}$$

where y is the full system-state vector, and h is the integration time step. Dot notation indicates the state time derivative prediction from Eqs. 5–8 and auxiliary equations included in the control laws and actuator dynamics. The subscript n indicates the current time step, $n + 1$ is one step in the future, $n - 1$ is one step in the past, and so on. The method is *implicit* because the state prediction at the next time step depends on a derivative estimate at the next time step.

Equation 10 serves as a predictor step where the state at the next time step is predicted to estimate the state derivative $\dot{y}_{n+1}$ one step into the future. This future state derivative prediction will rely on the current force/moment predictions from CFD; therefore, the algorithm suffers slightly from this approximation. Equation 11 serves as a corrector step, using the one-step-ahead derivative prediction in conjunction with the current and one-step-ago predictions. This

implicit predictor–corrector scheme was chosen as opposed to the explicit Adams–Bashforth to enhance the numerical stability of the integration.

Using such a multistep method has two notable advantages. First, the costly CFD force/moment predictions from previous time steps are used to inform the state prediction at the current time step. In contrast, Runge–Kutta methods base all derivative predictions on the current force/moment estimates. Time steps are kept small to provide continuity as the CFD mesh is moved at each time step; therefore, the error induced is small. Second, storing recent estimates of the state derivative makes it easy to access these terms as they are required by certain control laws (see Section 4).

4. Model Reference Adaptive Control with an Extended Reference Model

A major challenge for the implementation of MRAC architectures is the exclusion of actuator dynamics in the theoretical development. This is done by assuming that the actuator dynamics are fast enough such that the actuation system is properly applying the desired control signal. In this section, we present an overview of the ERM approach developed by Gruenwald et al.[19,20] We begin by providing the problem formulation as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}(\mathbf{\Lambda}\mathbf{v} + \mathbf{W}^{\mathbf{T}}\mathbf{x}) \tag{12}$$

$$\dot{\mathbf{v}} = -\mathbf{M}(\mathbf{v} - \mathbf{u})\,. \tag{13}$$

In Eq. 12, $\mathbf{x}$ is a measurable state vector, $\mathbf{A}$ and $\mathbf{B}$ are known system matrices, and $\mathbf{W}$ is an unknown weight matrix. In addition, $\mathbf{\Lambda}$ is an unknown control effectiveness matrix parameterizable as $\mathbf{\Lambda} = \mathbf{I} + \boldsymbol{\delta\Lambda}$, with $\boldsymbol{\delta\Lambda}$ being unknown. Using $\mathbf{W}$ and $\boldsymbol{\delta\Lambda}$ we capture potential uncertainty in the aerodynamic parameters of the flight vehicle system. Equation 13 represents the actuator dynamics where $\mathbf{M}$ is a diagonal matrix with each element being the actuator bandwidth for the control channels, $\mathbf{u}$ is the control input, and $\mathbf{v}$ is the actuator output that is applied to the system dynamics in Eq. 12. The individual actuator dynamics given by Eq. 9 can be combined and written in the compact form of Eq. 13.

The classical objective of the model reference adaptive control problem is to design an adaptive feedback control law such that—in the presence of system uncertainties captured by the unknown matrices $\mathbf{W}$ and $\boldsymbol{\delta\Lambda}$—the state vector $\mathbf{x}$ follows a reference state vector $\mathbf{x_r}$ satisfying the reference model dynamics given by

$$\dot{\mathbf{x}}_\mathbf{r} = \mathbf{A}_\mathbf{r}\mathbf{x}_\mathbf{r} + \mathbf{B}_\mathbf{r}\mathbf{c}\,, \tag{14}$$

where $\mathbf{A_r}$ is a Hurwitz reference model matrix, $\mathbf{B_r}$ is the command input matrix, and $\mathbf{c}$ is a vector of the desired uniformly continuous, smooth and bounded, reference command. This reference model captures the ideal closed-loop behavior achieved when there are no system uncertainties. A standard model reference adaptive control law will then suppress the effect of the system uncertainties, allowing the trajectories of the actual uncertain system to converge to the trajectories of the reference system, and thereby capturing the ideal closed-loop behavior. However, owing to the presence of the actuator dynamics given by Eq. 13, a standard model reference adaptive control law $\mathbf{u}$ does not have direct access to the system uncertainties to suppress them and allow for tracking the reference system.

For this reason, the expanded reference model approach[19] is considered. The expanded reference model is given by

$$\begin{bmatrix}\dot{\mathbf{x}}_\mathbf{r}\\ \dot{\mathbf{v}}_\mathbf{r}\end{bmatrix} = \begin{bmatrix}\mathbf{A}+\mathbf{B}\widehat{\mathbf{W}}^\mathbf{T} & \mathbf{B}\left(\mathbf{I}+\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\right)\\ -\mathbf{M}\left(\mathbf{K_1}+\widehat{\mathbf{W}}^\mathbf{T}\right) & -\mathbf{M}\left(\mathbf{I}+\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\right)\end{bmatrix}\begin{bmatrix}\mathbf{x}_\mathbf{r}\\ \mathbf{v}_\mathbf{r}\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{MK_2}\end{bmatrix}\mathbf{c}\,. \qquad (15)$$

This reference model is constructed using a model of the actuator dynamics $\dot{\mathbf{v}}_\mathbf{r}$, a nominal feedback gain $\mathbf{K_1}$ designed such that $\mathbf{A_r} = \mathbf{A} - \mathbf{BK_1}$ is Hurwitz, and a nominal feedforward gain $\mathbf{K_2}$ designed to be nonsingular and to satisfy $-\mathbf{EA_r^{-1}BK_2} = \mathbf{I}$, with $\mathbf{E}$ being a matrix that allows the selection of a subset of states in $\mathbf{x}$ to follow a desired command in the vector $\mathbf{c}$. In addition, $\widehat{\mathbf{W}}$ and $\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}$ are the estimates of the uncertainties in $\mathbf{W}$ and $\boldsymbol{\delta\Lambda}$, respectively. The $\widehat{\mathbf{W}}$ weight update law is given by

$$\dot{\widehat{\mathbf{W}}} = \boldsymbol{\Gamma}\,\mathrm{Proj}\left(\widehat{\mathbf{W}}, \mathbf{x}\tilde{\mathbf{z}}^\mathrm{T}\overline{\mathbf{P}}\mathbf{G}\right), \qquad (16)$$

with $\boldsymbol{\Gamma}$ being a diagonal learning rate matrix, $\tilde{\mathbf{z}} = [(\mathbf{x}-\mathbf{x_r})^{\boldsymbol{T}}, (\mathbf{v}-\mathbf{v_r})^{\boldsymbol{T}}]^{\boldsymbol{T}}$ being the augmented error for the system tracking error and the actuator output error, $\overline{\mathbf{P}}$ being the solution of a matrix inequality, and $\mathbf{G} = [\mathbf{B}^\mathbf{T}, \mathbf{0}]^{\boldsymbol{T}}$. The $\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}$ is diagonal such that

$$\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}} = \begin{bmatrix}\delta\hat{\lambda}_1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \delta\hat{\lambda}_{nref}\end{bmatrix}$$

and the $\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}$ weight update law is constructed from the elemental weight update laws

$$\dot{\delta\hat{\lambda}}_\iota = \alpha_i\,\mathrm{Proj}\left(\delta\hat{\lambda}_i, \mathbf{v}_i\tilde{\mathbf{z}}^\mathrm{T}\overline{\mathbf{P}}\mathbf{G}\boldsymbol{e_i}\right), \qquad (17)$$

where α_i are the learning rates for each respective element, $\mathbf{v}_i$ is the i-th element of the actuator output vector, and $\boldsymbol{e}_i$ is the standard basis. The projection operator notated by $\mathrm{Proj}(\cdot,\cdot)$ is used in both update laws, the definition of which can be found in Gruenwald et al.[19]

This is a standard modification used in model reference adaptive control to bound the evolution of the parameter estimates.

To further improve performance, a command governor architecture is augmented to the expanded reference model in Eq. 15. For this purpose, the command signal $\mathbf{c}$ is replaced with the signal given as

$$\mathbf{c_g} = \mathbf{c} + \mathbf{D\xi}\,, \tag{18}$$

where $\mathbf{c}$ is the desired uniformly continuous, smooth and bounded reference command (this allows one to assume that $\dot{\mathbf{c}}$ is bounded and available), $\mathbf{D\xi}$ is the command governor signal with $\mathbf{D} = \mathbf{K_2^{-1}M^{-1}}$, and $\boldsymbol{\xi}$ being the command governor output given by

$$\boldsymbol{\xi} = \left(\mathbf{I} + \boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\right)^{-1}(\mu\boldsymbol{\rho} - \boldsymbol{\phi_1}) - \boldsymbol{\phi_2}\,. \tag{19}$$

In Eq. 19, μ is a gain value on the command governor, $\boldsymbol{\rho}$ is the command governor state vector designed as

$$\boldsymbol{\rho} = \left(\mathbf{K_1} + \widehat{\mathbf{W}}\right)\mathbf{x_r} + \left(\mathbf{I} + \boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\right)\mathbf{v_r} - \mathbf{K_2c}\,,$$

and the signals $\boldsymbol{\phi_1}$ and $\boldsymbol{\phi_2}$ are designed using a backstepping approach as

$$\begin{aligned}\boldsymbol{\phi_1} = \left(\mathbf{K_1} + \widehat{\mathbf{W}}\right)\Big(&\left(\mathbf{A} + \mathbf{B}\widehat{\mathbf{W}}^{\boldsymbol{T}}\right)\mathbf{x_r} + \mathbf{B}(\mathbf{I} + \boldsymbol{\delta\Lambda})\mathbf{v_r}\Big) \\ &-\mathbf{K_2}\dot{\mathbf{c}} + \boldsymbol{\delta}\dot{\widehat{\boldsymbol{\Lambda}}}\mathbf{v_r} + \dot{\widehat{\mathbf{W}}}\mathbf{x_r}\,,\end{aligned} \tag{20}$$

$$\boldsymbol{\phi_2} = -\mathbf{M}\left(\mathbf{K_1} + \widehat{\mathbf{W}}\right)\mathbf{x_r} - \mathbf{M}\left(\mathbf{I} + \boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\right)\mathbf{v_r} + \mathbf{MK_2c}\,. \tag{21}$$

The feedback control law is then given by

$$\mathbf{u} = -\mathbf{K_1x} + \mathbf{K_2c_g} - \widehat{\mathbf{W}}^{\mathbf{T}}\mathbf{x} - \boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\mathbf{v}\,, \tag{22}$$

to achieve tracking of the expanded reference model with a command governor.

As noted for the weight update laws, a matrix inequality is used to obtain a solution $\overline{\mathbf{P}}$. The main feature of this is that one can determine ahead of time for given projection bounds on the elements of $\widehat{\mathbf{W}}$ and $\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}$ if the actuator bandwidths in $\mathbf{M}$ are sufficiently large enough for the actuator to properly apply the adaptive control signal. For this purpose, let $\overline{\mathbf{W}}_\mathbf{i}$ and $\boldsymbol{\delta}\overline{\boldsymbol{\Lambda}}_{\boldsymbol{i}}$ represent all possible variations in the parameter estimates. We can use the projection bounds to do this. One can then write

$$\overline{\mathbf{A}}_\mathbf{i} = \begin{bmatrix} \mathbf{A} + \mathbf{B}\overline{\mathbf{W}}_\mathbf{i}^{\mathbf{T}} + \dfrac{\epsilon}{2}\mathbf{I} & \mathbf{B}(\mathbf{I} + \boldsymbol{\delta}\overline{\boldsymbol{\Lambda}}_\mathbf{i}) \\ -\mathbf{M}\left(\mathbf{K_1} + \overline{\mathbf{W}}_\mathbf{i}^{\mathbf{T}}\right) & -\mathbf{M}(\mathbf{I} + \boldsymbol{\delta}\overline{\boldsymbol{\Lambda}}_\mathbf{i}) + \dfrac{\epsilon}{2}\mathbf{I} \end{bmatrix}, \tag{23}$$

to represent all the combined permutations of $\bar{\mathbf{W}}_{\mathrm{i}}$ and $\boldsymbol{\delta}\bar{\boldsymbol{\Lambda}}_{\boldsymbol{i}}$, with ϵ being an additional design parameter. For given actuator bandwidth values in $\mathbf{M}$, one can then solve the matrix inequality given by

$$\bar{\boldsymbol{A}}_{\boldsymbol{i}}^{\boldsymbol{T}}\bar{\boldsymbol{P}} + \bar{\boldsymbol{P}}\bar{\boldsymbol{A}}_{\boldsymbol{i}} < \mathbf{0}, \;\; \bar{\boldsymbol{P}} > \mathbf{0} \tag{24}$$

to calculate $\bar{\mathbf{P}}$ for use in the weight update laws.

This concludes the overview of the expanded reference model approach with a command governor that is used as the flight control for the coupled simulation. For more details on the expanded reference model approach, theoretical proofs of stability, added discussion of the benefits of the expanded reference model and the command governor augmentation, and comparisons to a well-known hedging approach, the reader is directed to work by Gruenwald et al.[19] In simulation we make comparisons with the hedging approach of Johnson and Calise.[21] While more information can be found in the literature,[19,21] it is sufficient to know that hedging is another tool used in the model reference adaptive control literature to account for the presence of actuator dynamics. Similar to the expanded reference model, a hedging control architecture makes a modification to the standard reference model given by Eq. 14 and adds an error term between the actuator output and the control input. Table 1 summarizes the hedging approach.

Table 1 MRAC with hedging

Reference model	$\dot{\mathbf{x}}_{\mathbf{r}} = \mathbf{A}_{\mathbf{r}}\mathbf{x}_{\mathbf{r}} + \mathbf{B}_{\mathbf{r}}\mathbf{c} + \mathbf{B}(\mathbf{v} - \mathbf{u})$
Control law	$\mathbf{u} = -\mathbf{K}_{\mathbf{1}}\mathbf{x} + \mathbf{K}_{\mathbf{2}}\mathbf{c} - \widehat{\mathbf{W}}^{\mathbf{T}}\mathbf{x} - \boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}\mathbf{v}$
Update laws	$\dot{\widehat{\mathbf{W}}} = \boldsymbol{\Gamma}\,\mathrm{Proj}\left(\widehat{\mathbf{W}}, \mathbf{x}(\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{r}})^{\mathrm{T}}\mathbf{P}\mathbf{B}\right)$ $\dot{\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}} = \boldsymbol{\alpha}\,\mathrm{Proj}\left(\boldsymbol{\delta}\widehat{\boldsymbol{\Lambda}}, \mathbf{B}^{\mathbf{T}}\mathbf{P}(\mathbf{x} - \mathbf{x}_{\mathbf{r}})\mathbf{v}^{\mathbf{T}}\right)$
Lyapunov equation	$\mathbf{A}_{\mathbf{r}}^{\mathbf{T}}\mathbf{P} + \mathbf{P}\mathbf{A}_{\mathbf{r}} + \mathbf{R} = \mathbf{0}, \qquad \mathbf{R} = \mathbf{R}^{\mathbf{T}} > \mathbf{0}$

5. Results

All results in this report use the DEVCOM ARL Laboratory Technology Vehicle (LTV) projectile (Fig. 3). An LTV is a 105-mm-diameter projectile with long strakes to increase lift-to-drag ratio. The vehicle is equipped with four independently actuated trailing edge flaps for control. Several cases of increasing complexity were tested. These include open-loop tests to validate the new code against the legacy Fortran-based code, a simple LQR regulation of the projectile pitch rate to demonstrate closed-loop operation, and a full adaptive control tracking problem to display the full capabilities of the simulation. All tracking controllers were prototyped and tested in a pure MATLAB environment where the CFD

computations were replaced by a locally devised aerodynamic model. Apart from minor changes to the frame transformations in Eqs. 2–4, the MATLAB prototype controller was imported into the coupled simulation without modification.

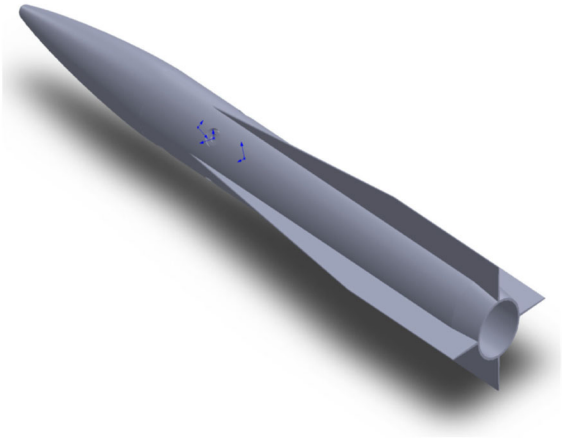

Fig. 3 LTV

5.1 Open Loop

The MATLAB-based coupled simulation was validated against the legacy Fortran-based code in several open-loop shots. Figure 4 shows the results of a shot with all fins deflected statically to induce a pitch response. The missile shows a typical underdamped response in pitch, angle of attack, and pitch rate. The angle of attack reaches about –7° with a peak time just less than 0.1 s. It appears that once the oscillations damp out, the response will settle around –4.5° angle of attack. The new simulation matches well with the legacy simulation.

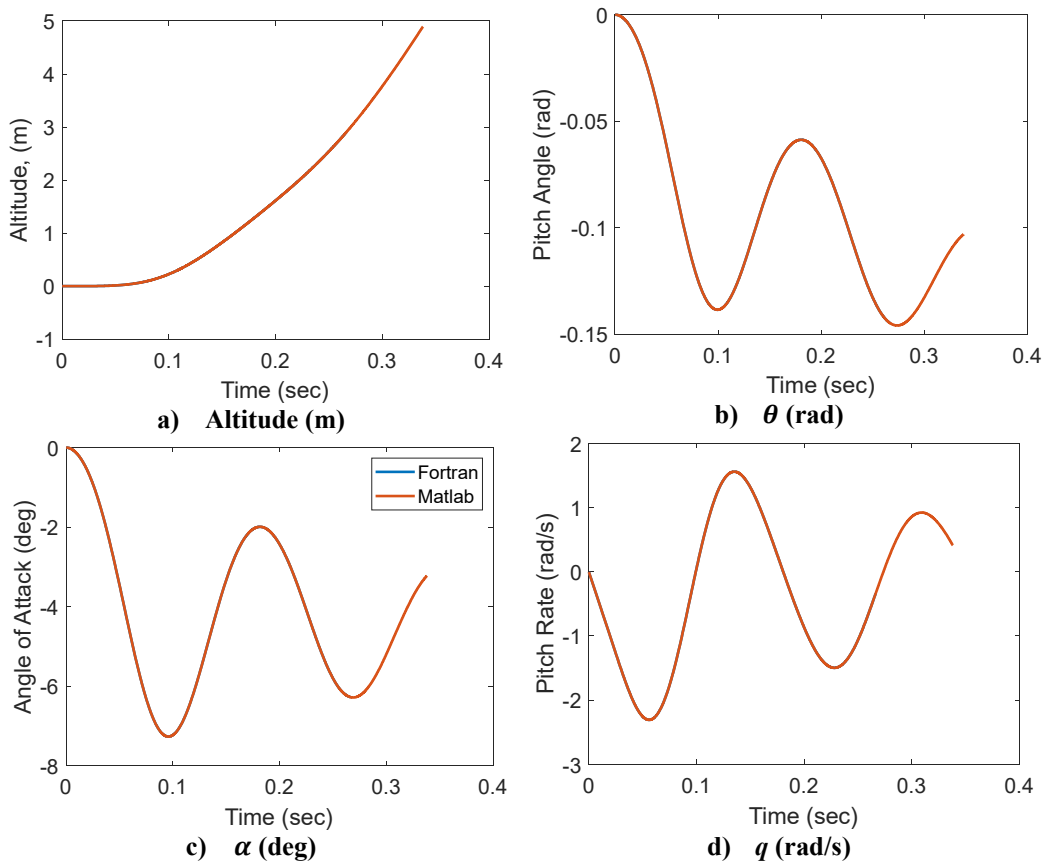

Fig. 4 Open-loop validation, all four flaps deflected to command a nonzero angle of attack. Legacy code prediction shown in blue; MATLAB code shown in red.

5.2 Simple Regulation

The simulation was used to demonstrate regulation on two cases with nonzero initial conditions. To achieve this, an LQR state feedback gain matrix was computed based upon the six-state linear model

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{25}$$

where the linearized state is $\mathbf{x} = \{\phi \quad p \quad q \quad r \quad \dot{v} \quad \dot{w}\}$. The dynamics are linearized at a trim condition at Mach 2 and sea level. Commanded feedback is computed as

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{26}$$

where

$$\mathbf{K} = \begin{bmatrix} \mathbf{1} & \mathbf{0.074} & \mathbf{0.} & \mathbf{0.00} & \mathbf{0.00} & \mathbf{0.00} \\ \mathbf{0} & \mathbf{0.0} & \mathbf{3.21} & \mathbf{0.00} & \mathbf{0.00} & \mathbf{0.058} \\ \mathbf{0} & \mathbf{0.0} & \mathbf{0.00} & \mathbf{2.97} & \mathbf{0.067} & \mathbf{0.00} \end{bmatrix}. \tag{27}$$

The commanded deflections are found by multiplying the virtual control $\mathbf{u}$ by the control allocation matrix

$$\boldsymbol{\delta}_{\mathbf{CMD}} = \mathbf{C}_{\mathbf{A}}\mathbf{u} \tag{28}$$

where

$$\mathbf{C}_{\mathbf{A}}^{T} = \begin{bmatrix} -\mathbf{1} & -\mathbf{1} & -\mathbf{1} & -\mathbf{1} \\ -\mathbf{1} & \mathbf{1} & \mathbf{1} & -\mathbf{1} \\ -\mathbf{1} & -\mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix}.$$

Four auxiliary states are appended to the simulation to integrate four instances of Eq. 8 to determine the actual deflections, assuming finite actuator bandwidth. This also prevents the flaps from making large instantaneous deflections, causing a singularity in the CFD solution.

Figure 5 shows the results of two simulations with LQR state feedback regulation as described previously. The blue lines show the response with initial conditions $\{\phi = 22.5° \quad p = 0 \quad q = 3\mathrm{r/s} \quad r = -0.35\mathrm{r/s} \quad u = 680\mathrm{m/s} \quad \theta = 0\}$. The red lines show the response for all zero initial conditions except $q = 3\mathrm{r/s}$ and $u = 680\mathrm{m/s}$.

In Fig. 5e the pitch rate settles within 2% of steady state at 0.175 s for both initial conditions. Both conditions result in similar responses in pitch (Fig. 5b) and angle of attack as well (Fig. 5d). Figures 5a, 5c, and 5f show the regulator response to nonzero initial conditions in roll and yaw rate. Yaw rate settles around 0.234 s. Due to the large slew required in roll, the roll settling time is 0.419 s. These cases illustrate that we have successfully implemented feedback control in multiple axes, including realistic actuator dynamics. Subsequent cases will illustrate command tracking control and real time adaptation.

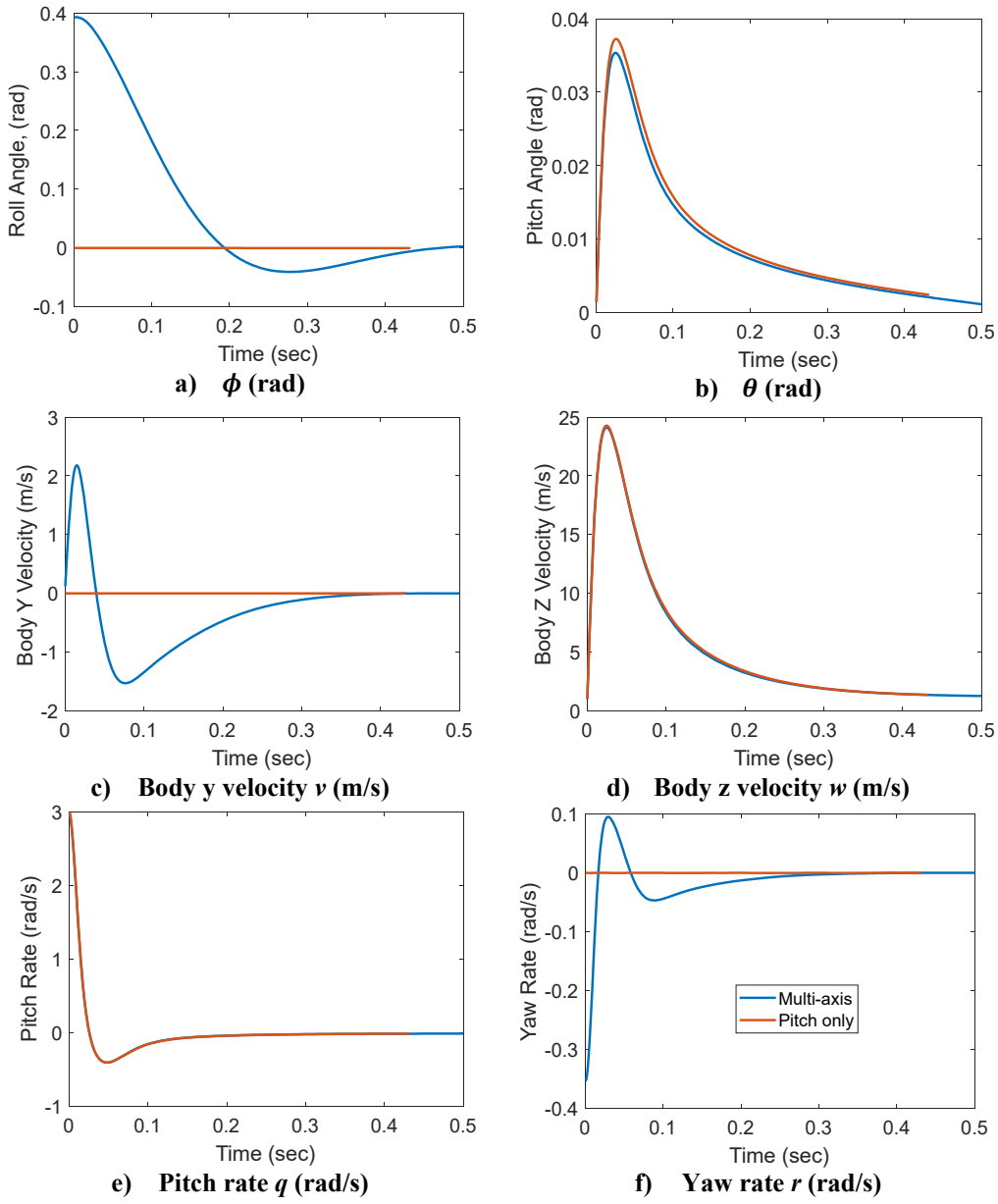

Fig. 5 Demonstrating state regulation, M = 2.0, sea level, red lines are pitch only and blue lines are multiaxis

5.3 Adaptive Control

We implemented the MRAC design using an expanded reference model described in Section 4. The feed-forward control was designed to track the body Z velocity (w), which maps essentially to angle of attack. Although this choice requires strong assumptions about the availability of w for feedback, it allows for a fast closed-loop settling time such that the system could be driven through several command cycles in a reasonable time. Choosing body acceleration, for instance, would produce a much slower response, and prevent demonstration of adaptation in the coupled

simulation within a reasonable time horizon. In essence, the goal of this simulation is to demonstrate performance of the adaptive controller, independent of any challenges that may be induced by state estimation in an actual implementation.

To contrast performance, a nonadaptive LQR controller was tested under identical conditions. An adaptive controller with hedging[21] (as mentioned at the end of Section 4) was also tested. As highlighted by Table 1, the hedging approach uses a different reference model and replaces the matrix inequality solution $\bar{\mathbf{P}}$ from Eqs. 23 and 24 with a solution $\mathbf{P}$ to the Lyapunov equation where $\mathbf{R}$ is a small positive definite matrix. There are also changes to the weight update laws.

Figure 6 shows the results of these tests. In Figure 6a, it is obvious that the default LQR controller underestimates the control authority, resulting in poor tracking at the high dwells. The LQR controller was designed based on a model that also overestimated the system damping. This results in weak feedback and excessive overshoot, which is also evident in each of the high dwells. All three controllers use the same baseline feedback $\mathbf{K_1}$ from this design to demonstrate how adaptation can achieve specified performance in the presence of modelling errors.

The red lines in Fig. 6 show the response for the hedging controller. In general, it results in greater oscillation than the MRAC with the ERM; however, it does an adequate job of canceling out the model error in DC gain. The MRAC-ERM outperforms the other control schemes in tracking (Fig. 6a) while reducing the fast oscillations that appear with hedging control.

Even though tracking performance is similar between MRAC-ERM and hedging control (Fig. 6a), the off-axis response shows that MRAC-ERM greatly improves tracking over the other two methods. In Fig. 6b, the blue line representing MRAC-ERM has much less oscillation from zero, especially early in the response. In Fig. 6d, MRAC-ERM holds the roll angle very close to zero (note the axis scaling), while the other two controllers allow it to drift slowly away from neutral. In Fig. 6f, MRAC-ERM shows much less oscillation in the yaw rate, while hedging produces significantly more than the LQR.

Figure 7 displays the actuator response for the simulations shown in Fig. 6. The LQR controller behavior is identical for each cycle since it does not adapt. The LQR drives all four deflections to steady-state positions larger than the other schemes since the underlying linear model underestimates the control authority. The adaptive schemes show similar responses; however, the MRAC-ERM causes less actuator oscillation at high dwells after the first. Thus, although hedging adapts to the poor model of DC gain, it does not completely adapt for the poor model of damping. Hedging continues to have large actuator oscillations for the full simulation time.

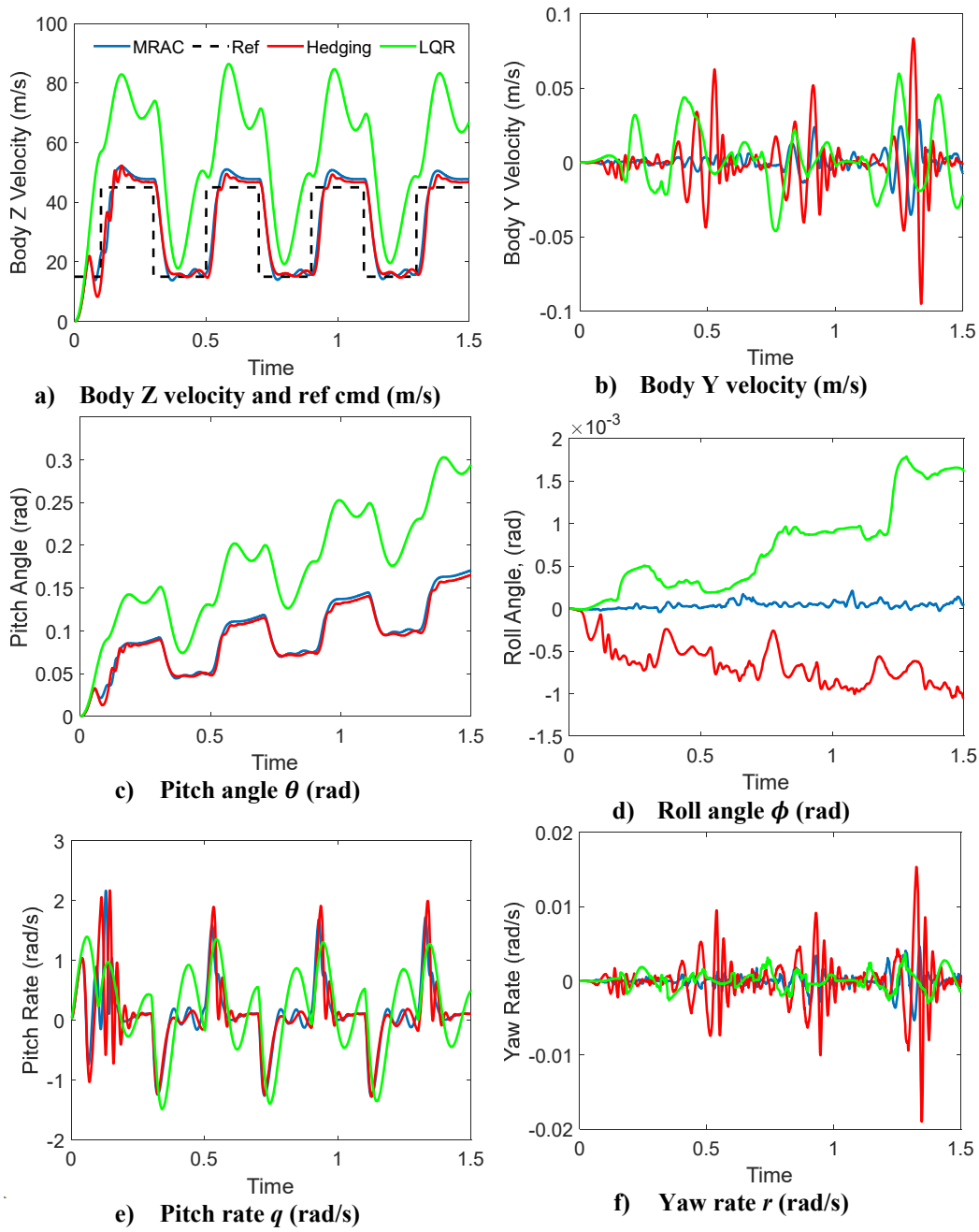

Fig. 6 Subset of state history, baseline LQR controller designed at M = 2.0 with poor estimates of DC gain and damping

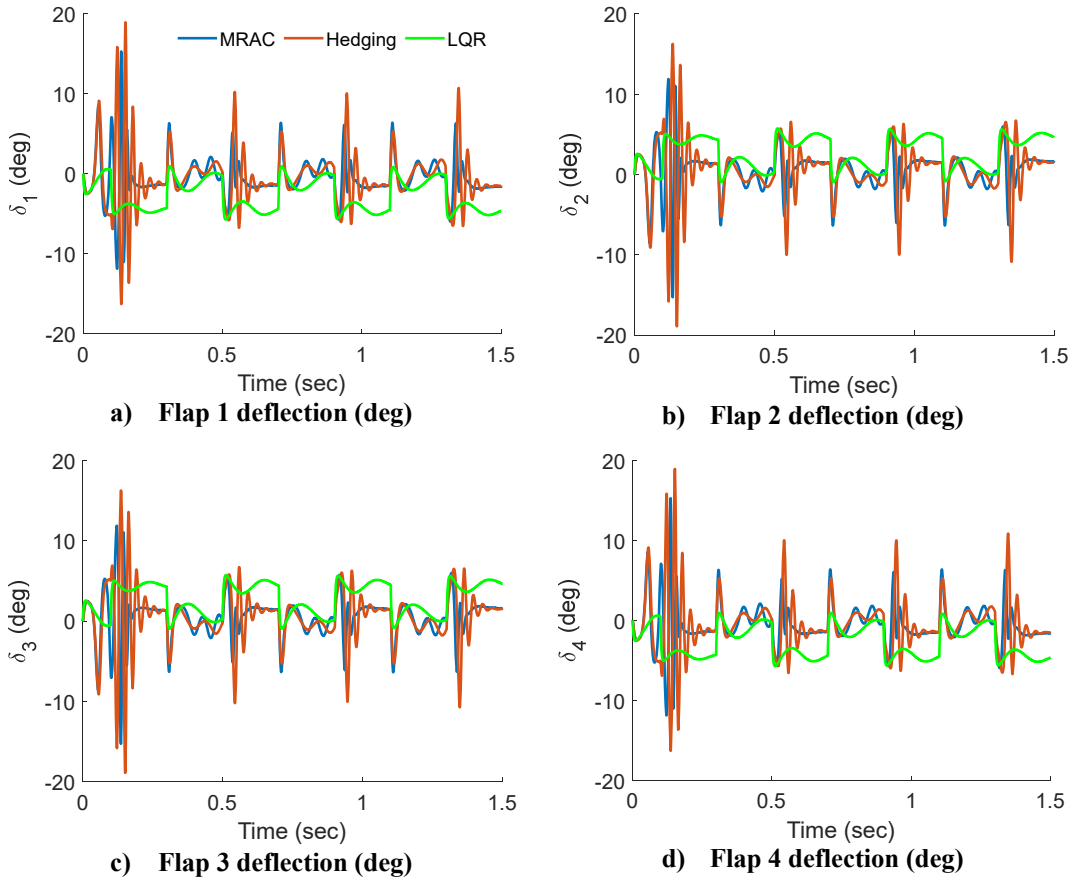

Fig. 7 Flap deflections, demonstrating MRAC, M = 2.0, sea level, baseline controller designed with poor estimates of DC gain and damping

6. Conclusion

A novel coupled CFD/RBD/FCS simulation was devised using CFD++ and MATLAB to make closed-loop coupled simulation more accessible to the aerospace controls community. The simulation incorporates several specific improvements over an existing Fortran-based simulation:

- Concatenating all continuous states into a system-state vector so that all are integrated simultaneously with a common time scale.
- Providing up to 300 continuous states for actuator, control system, and other dynamic systems in addition to projectile motion.
- Using an Adams–Moulton multistep method so that derivative information from previous time steps is leveraged without additional calls to CFD.
- Sequestering actuator and control system-state derivatives to separate subfunctions for modularity.

- Making the MATLAB memory space completely volatile to aid in compilation.

To demonstrate the capabilities and flexibility of the simulation, several examples were presented including the following:

- Open-loop CFD/RBD motion
- Simple single- and multiaxis regulation
- LQR tracking control without adaptation
- LQR tracking control with MRAC adaptation and hedging
- LQR tracking control with MRAC-ERM adaptation

The first scenario was used to validate the new simulation against open-loop predictions from the existing CFD-Fortran simulation. The closed-loop scenarios are listed in order of increasing complexity. Each one showed both the expanded capabilities of the simulation and the ease with which the user can import highly complex control systems directly from pure MATLAB simulations.

Due to the high-fidelity solution rendered by CFD, the user can be confident that simulation results will imitate reality with a high degree of accuracy. The adaptive control results demonstrated that adaptation can overcome modelling errors to achieve specified performance in a realistic scenario.

The work outlined in this report introduces a new coupled CFD/RBD/FCS simulation using the CFD++ commercial code, and an RBD-FCS simulation coded in the MATLAB programming environment. Although we have attempted to make the MATLAB code intuitive, some specifics are offered in the Appendix to help otherwise experienced MATLAB users with accessing our simulation tool.

7. References

1. Sahu J. Time-accurate numerical prediction of free-flight aerodynamics of a finned projectile. AIAA J Spacecraft Rockets. 2008;45(5):946–954.

2. Costello M, Sahu J. Using computational fluid dynamic/rigid body dynamic results to generate aerodynamic models for projectile flight simulation. J Aerospace Eng, 2008;22(G7):1067–079.

3. Sahu J. Virtual fly-out simulations of a spinning projectile from subsonic to supersonic speeds. AIAA Applied Aerodynamics Meeting; 2011 June; Honolulu, HI.

4. Sahu J. Time-accurate computations of free-flight aerodynamics of a spinning projectile with and without flow control. Army Research Laboratory (US); 2006 Sep. Report No.: ARL-TR-3919.

5. Sahu J, Fresconi F, Heavey K. Unsteady aerodynamics of a finned projectile at a supersonic speed with jet interaction. AIAA Aviation 2014 Forum; 2014 June; Atlanta, GA. AIAA Paper No.: 2014-3024.

6. Sahu J. Unsteady aerodynamic simulations of a canard-controlled projectile at low transonic speeds. AIAA Atmospheric Flight Mechanics Meeting; 2011 Aug; Portland, OR.

7. Sahu J, Fresconi F, Heavey K. Control performance, aerodynamic modeling and validation of coupled simulation techniques for guided projectile roll dynamics. AIAA Aviation 2014 Forum, 2014 June; Atlanta, GA. AIAA Paper No.: 2014-2191.

8. Sahu J, Fresconi F. Flight behaviors of a complex projectile using a coupled CFD-based simulation technique: free motion. Aviation Forum; 2015 June, Dallas, TX. AIAA Paper No.: 2015-2585.

9. Sahu J, Fresconi F. Flight behaviors of a complex projectile using a coupled CFD-based simulation technique: open-loop control. SciTech Forum and Exposition; 2016 Jan; San Diego, CA. AIAA Paper No.: 2016-2025.

10. Sahu J, Fresconi F. Flight behaviors of a complex projectile using a coupled CFD-based simulation technique: closed-loop control. Aviation Forum, 2016 June; Washington, DC. AIAA Paper No.: 2016-4332.

11. Morton SA. Rigid, maneuvering, and aeroelastic results for Kestrel – a CREATE simulation tool. US Air Force SEEK EAGLE Office; 2010. AIAA Paper No.: 2010-1233.

12. Dean JP, Clifton JD, Bodkin DJ, Ratcliff CJ. High resolution CFD simulations of maneuvering aircraft using the CREATE-AV/Kestrel Solver. 49th AIAA Aerospace Sciences Meeting, 2011 Jan 4–7; Orlando, FL. AIAA Paper No.: 2011-1109.

13. Guo D, Xu M, Chen S-L. Time-accurate simulation of longitudinal flight mechanics with control by CFD/RBD coupling. Appl Mechan Mater. 2012 Nov;226–228:788–792.

14. Ernst Z, Drosendahl M, Robertson B, Mavris D. Development of a trajectory-centric CFD-RBD framework for advanced multidisciplinary/multiphysics simulation. AIAA SciTech Forum; 2022 Jan 3–7; San Diego, CA. doi: 10.2514/6.2022-1793.

15. Costello M, Rogers J. BOOM: A Computer-aided engineering tool for exterior ballistics of smart projectiles. Army Research Laboratory (US); 2011 June. Report No.: ARL-CR-670.

16. Peroomian O, Chakravarthy S, Goldberg U. A 'grid-transparent' methodology for CFD. AIAA; 1997. AIAA Paper No.: 97-07245.

17. Peroomian O, Chakravarthy S, Palaniswamy S, Goldberg U. Convergence acceleration for unified-grid formulation using preconditioned implicit relaxation. AIAA; 1998. AIAA Paper No.: 98-0116.

18. Goldberg UC, Peroomian O, Chakravarthy S. A Wall-distance-free K-E model with enhanced near-wall treatment. ASME J Fluids Eng. 1998;120;457–462.

19. Gruenwald BC, Yucelen T, Dogan KM, Muse JA. Expanded reference models for adaptive control of uncertain systems with actuator dynamics. J Guidance Control Dynamics. 2020;43(3):475–489.

20. Gruenwald BC, Bryson JT. Adaptive control for a guided projectile using an expanded reference model. AIAA Scitech 2020 Forum; 2020 Jan. Paper No.: AIAA 2020-1822.

21. Johnson EN, Calise AJ. Limited authority adaptive flight control for reusable launch vehicles. J Guidance Control Dynamics. 2003;26(6):906–913. doi: 10.2514/2.6934.

Appendix. MATLAB User's Guide

A.1 Calling Syntaxes and Inputs

The rigid body dynamics (RBD)/flight control system (FCS) tool consists of two MATLAB functions such that RBD and FCS have intentionally separate workspaces. The calling syntaxes are fixed by the ".c" interface with the computational fluid dynamics (CFD) tool. Each is called once per simulation time step.

The RBD simulation is called by the syntax

```
[doutrbdsim] = mc_rbdsim(rbd_tomatlab).
```

The FCS simulation is called by the syntax

```
[fcs_frmatlab] = mc_fcssim(dinfcssim).
```

The inputs `rbd_tomatlab` and `dinfcssim` are identical and go by the moniker "`dinrbdsim`" in the .c interface. The first 11 values of `dinrbdsim` contain the projectile mass properties as shown in Fig. A-1. A block of the .c interface comments spell out the contents explicitly. This is shown in Fig. A-2. The comments that follow use zero-based indexing as customary in the .c language. In the MATLAB functions all indices are increased by one from those shown below. Elements 11–19 (zero-based indices) are currently reserved for future expansion. Elements 20–39 contain time, time step, 12 vehicle states, 3 force components, and 3 moment components as shown in Fig. A-2.

```
dinrbdsim[ 0]: gravity (m/s^2, equals 9.81)
dinrbdsim[ 1]: weight (n) (mass*grav)
dinrbdsim[ 2]: stationline of cg (m)
dinrbdsim[ 3]: buttline of cg (m)
dinrbdsim[ 4]: waterline of cg (m)
dinrbdsim[ 5]: ixx (kg m^2)
dinrbdsim[ 6]: iyy (kg m^2)
dinrbdsim[ 7]: izz (kg m^2)
dinrbdsim[ 8]: ixy (kg m^2)
dinrbdsim[ 9]: ixz (kg m^2)
dinrbdsim[10]: iyz (kg m^2)
```

Fig. A-1 Vehicle properties in the MATLAB input vector

```
  --- 11-19 left as future expansion
  dinrbdsim[20]: time (sec)
  dinrbdsim[21]: delta time (sec)
  dinrbdsim[22]: x cg position (m, inertial frame)
  dinrbdsim[23]: y cg position (m, inertial frame)
  dinrbdsim[24]: z cg position (m, inertial frame)
  dinrbdsim[25]: x cg velocity (m/sec, inertial frame)
  dinrbdsim[26]: y cg velocity (m/sec, inertial frame)
  dinrbdsim[27]: z cg velocity (m/sec, inertial frame)
  dinrbdsim[28]: euler roll angle (rad)
  dinrbdsim[29]: euler pitch angle (rad)
  dinrbdsim[30]: euler yaw angle (rad)
  dinrbdsim[31]: roll rate (rad/s, body frame)
  dinrbdsim[32]: pitch rate (rad/s, body frame)
  dinrbdsim[33]: yaw rate (rad/s, body frame)
  dinrbdsim[34]: x cfd force (n, inertial frame)
  dinrbdsim[35]: y cfd force (n, inertial frame)
  dinrbdsim[36]: z cfd force (n, inertial frame)
  dinrbdsim[37]: l cfd moment (n m, inertial frame)
  dinrbdsim[38]: m cfd moment (n m, inertial frame)
  dinrbdsim[39]: n cfd moment (n m, inertial frame)

  Additional values are sent to RBDSIM at the end of the
dinrbdsim array

  dinrbdsim[40]: how many BC families need auxiliary output
(nbaux)
  next nbaux*11 values are the auxiliary output values

  next 12 values used to save returned 12 values from doutrbdsim
(after 1st call)

  next nfcs values used to save returned nfcs values from
doutfcssim
    (after first call to fcssim, returned in doutfcssim)

  next 300 values used to save 300 values returned after the
needed 12 values in doutrbdsim
```

Fig. A-2 Dynamic states in the MATLAB input vector

The input `dinrbdsim[40]` was zero for all examples shown in this report. Its purpose is to provide for passing additional data to the MATLAB interface, such as forces and moments on specific parts of the projectile (e.g., canards or flaps).

Twelve additional inputs are reserved to echo the projectile body states calculated by the RBD sim. The next "nfcs" (number of flight controls) inputs store the actuator positions from the last time step.

The actuator positions in CFD are driven directly by the first "nfcs" elements in the FCS sim output vector, so the values stored here are largely ignored. The user can specify the nfcs although all the examples presented here use nfcs = 4.

Finally, there are 300 elements of the input used to pass system states not included in the 12 body states. The user has complete flexibility in how these are used. We illustrate by unpacking how these were used for the model reference adaptive control (MRAC) example presented in this report.

A.2 The Use of Auxiliary States in the Examples

In the MRAC example shown, there are a plethora of continuous states among the RBD/-FCS and actuator subsystems. A primary objective in designing this coupled tool was to ensure that all system continuous states were integrated simultaneously by first concatenating them into a common system-state vector. To effectively use our simulation with a complex controller, the user must first account for all continuous states in their respective domains. Table A-1 illustrates this enumeration for our most complicated example.

Table A-1 Continuous states used in MRAC example

Domain	Description	Number
RBD	Body states	$n_{body} = 12$
Actuator	Flap positions	$n_{fcs} = 4$
FCS	Virtual controls	$n_{ref} = 3$
	Reference model states	$n_{state} = 6$
	Adaptive gain $\widehat{W}$	$n_{state} \cdot n_{ref} = 18$
	Adaptive gain $\widehat{\Lambda}$	$n_{ref} = 3$

We chose to reserve the first n_{fcs} of the auxiliary states to contain the commanded actuator positions since they need to be passed through the RBD sim to integrate the actual actuator positions. Inside the RBD sim, these are shuffled to the bottom of the system state vector to keep the integrable quantities in one contiguous part of the state vector. This streamlines two aspects moving forward—a single function call is used to determine the state vector derivative, such as

```
g1 = RBDDOT(T,XRBD,CFDFORCEVEC,CFDMOMENVEC,properties);
```

where `XRBD` contains the *entire system state* as outlined in Table A-1. Second, past values of the system state vector derivative are stored in the next contiguous block of `dinrbdsim` as described henceforth.

Next, the function `RBDDOT` is written to compute the derivatives of the 12 body states given the states themselves and current force and moment vectors. Its code should remain fixed for any simulation that models the projectile as a single rigid body. To sequester the FCS calculations and limit the inputs thereof, derivatives of

the remaining states are computed in a subfunction that is called at the bottom of `RBDDOT` using:

```
XRBDDOT(13:end) = FCSDOT(time,XRBD(13:end),bodystates);
```

where `bodystates` passes a limited number of RBD states to be used in the computations where a measurement model may or may not be applied.

Returning to the parsing of `dinrbdsim`, the remaining elements contain past values of the system state vector derivative, which are stored in MATLAB in columns 2 and 3 of an `XRBDDOT` array so that Eq. 10 is computed by adding `g2` to the current state vector where

```
g2=DT/12.0*(23.0*g1-...
                    16.0*XRBDDOT(:,2)+5.0*XRBDDOT(:,3));
```

and `g1` was given above.

A.3 Remining Details of "mc_rbdsim"

Much of "mc_rbdsim" should be left unmodified by the typical user. Steps that will be common in all applications are:

- Parse the projectile body states, force, and moment vectors.
- Transform body linear velocities, forces, and moments into body frame.
- Integrate forward in time.
- Parse the projectile body states.
- Transform body linear velocities back to ground fixed frame.

The user will need to concatenate auxiliary states from the actuator and control system to the output vector in an order consistent with their positions in the input vector. The output vector is specified in the comments block of the .c code shown in Fig. A-3.

```
doutrbdsim[0-11] are 12 RBD state values
 like dinrbdsim[22-33]
+ up to 300 values can be returned after these 12 values
```

Fig. A-3 Output vector definition for "mc_rbdsim.m"

A.4 Details of "mc_fcssim"

Since all numerical integration is performed in "mc_rbdsim", "mc_fcssim" will need only to parse the common input vector for quantities required to evaluate the control law and compute the control commands. These quantities and their dimensions were introduced in Table A-1. To quickly review, the reference model states are formed into a vector consisting of six linear plant states and three reference virtual controls. $\widehat{W}$ is formed as an $n_{state} \times n_{ref}$ rectangular matrix. $\widehat{\Lambda}$ is a diagonal matrix with n_{ref} nonzero terms.

The control law described earlier also requires the time derivatives of $\widehat{W}$ and $\widehat{\Lambda}$. These are captured from the state derivative history contained near the bottom of the system state vector. $\dot{\widehat{W}}$ is then formed as an $n_{state} \times n_{ref}$ rectangular matrix and $\dot{\widehat{\Lambda}}$ is a diagonal matrix.

Remaining quantities required are the virtual control and a reference signal. The user defines the reference signal as a vector function of time with dimension equal to the number of virtual controls. The virtual controls in our case are found by pulling the current actual flap deflections from the system state vector, and multiplying by the pseudo-inverse of the control allocation matrix

$$\mathbf{u_V} = \mathbf{C_A^\dagger \delta_{ACT}} \tag{A-1}$$

where

$$\mathbf{C_A^\dagger} = \frac{\mathbf{1}}{\mathbf{4}} \cdot \begin{bmatrix} \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{1} & \mathbf{1} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{-1} & \mathbf{1} & \mathbf{1} \end{bmatrix}.$$

After determining the commanded virtual control from Eqs. 18–22, Eq. 28 is invoked to determine the commanded physical control. These commands are then limited to $-30^\circ \leq \delta_{CMDi} \leq 30^\circ, \forall\, i$. Finally, "mc_fcssim" concatenates $\mathbf{\delta_{ACT}}$ with $\mathbf{\delta_{CMD}}$ to form its output vector. Users can certainly define any number of controls using nfcs, but the simulation convention is that the "mc_fcssim" output contains the actual control positions followed by the commanded control positions.

List of Symbols, Abbreviations, and Acronyms

3-D	three-dimensional
6-DOF	six-degrees-of-freedom
ARL	Army Research Laboratory
CFD	computational fluid dynamics
DC gain	ratio of steady state response to commanded response
DEVCOM	US Army Combat Capabilities Development Command
ERM	Extended Reference Model
FCS	flight control system
LQR	linear quadratic regulator
LTV	Laboratory Technology Vehicle
MRAC	model reference adaptive control
ODE	ordinary differential equation
RBD	rigid body dynamics

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
11 (PDF)	ARL DEVCOM FCDD RLW WD L STROHM V A BHAGWANDIN J BRYSON B BURCHETT I CELMINS J DESPIRITO L D FAIRFAX B GRUENWALD J PAUL J SAHU J D VASILE