NASA/TM—2016-218923

Tool for the Integrated Dynamic Numerical Propulsion System Simulation (NPSS)/Turbine Engine Closed-Loop Transient Analysis (TTECTrA)

User's Guide

Jeffrey C. Chin and Jeffrey T. Csank
Glenn Research Center, Cleveland, Ohio

January 2016

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., "quick-release" reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 757-864-6500

- Telephone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Program
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM—2016-218923

Tool for the Integrated Dynamic Numerical Propulsion System Simulation (NPSS)/Turbine Engine Closed-Loop Transient Analysis (TTECTrA)

User's Guide

Jeffrey C. Chin and Jeffrey T. Csank
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

January 2016

This report contains preliminary findings, subject to revision as analysis proceeds.

This work was sponsored by the Fundamental Aeronautics Program at the NASA Glenn Research Center.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at http://www.sti.nasa.gov/ and http://ntrs.nasa.gov/

Contents

List of Figures

List of Tables

Tool for the Integrated Dynamic Numerical Propulsion System Simulation (NPSS)/Turbine Engine Closed-Loop Transient Analysis (TTECTrA)
User's Guide

Jeffrey C. Chin and Jeffrey T. Csank
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

The Tool for Turbine Engine Closed-Loop Transient Analysis (TTECTrA ver2) is a control design tool that enables preliminary estimation of transient performance for models without requiring a full nonlinear controller to be designed. The program is compatible with subsonic engine models implemented in the MATLAB/Simulink (The Mathworks, Inc.) environment and Numerical Propulsion System Simulation (NPSS) framework. At a specified flight condition, TTECTrA will design a closed-loop controller meeting user-defined requirements in a semi or fully automated fashion. Multiple specifications may be provided, in which case TTECTrA will design one controller for each, producing a collection of controllers in a single run. Each resulting controller contains a setpoint map, a schedule of setpoint controller gains, and limiters; all contributing to transient characteristics. The goal of the program is to provide steady-state engine designers with more immediate feedback on the transient engine performance earlier in the design cycle.

Nomenclature

accel	Acceleration
alt	Altitude*
decel	Deceleration
dTamb	Free-stream static temperature minus standard atmosphere temperature*
DWS	Dynamic systems analysis workspace (WS) variable
Fdbk	Feedback signal
Fnet	(Uncorrected) thrust*
i	Index for input or output vectors of linear models (u and y)
inputs	Variable containing inputs for configuring simulation of engine model with TTECTrA
Kp	Controller proportional gains
Ki	Controller integral gains
MN	Mach number
Nc	(Uncorrected) core speed*
Ncdot	Core speed acceleration*
NcR25	Core speed corrected at station 25*
Nf	(Uncorrected) fan speed*
outputs	Variable containing outputs from simulation of engine model with TTECTrA
P2	Pressure at station 2*
Ps3	High-pressure compressor static pressure*

u	Input vector for linear models
Wf/Ps3	Ratio of fuel flow to static high-pressure compressor static pressure
y	Output vector for linear models

* Units of these variables are model-specific

Acronyms

EPR	Engine Pressure Ratio
GUI	Graphical User Interface
HPC	High-Pressure Compressor
IWP	Integral Wind-up Protection
LM	Linear Model
LPC	Low-Pressure Compressor
NPSS	Numerical Propulsion System Simulation
PI	Proportional Integral controller
TTECTrA	Tool for Turbine Engine Closed-loop Transient Analysis

Introduction

Conceptual cycle design often begins with steady-state thermodynamic analyses, which incrementally graduate into higher and higher fidelity models. Historically, there have been completely separate code-bases for each level of fidelity. “Low Fidelity” (Lo-Fi) models can be recognized by their flexibility; capable of exploring large design spaces relatively quickly. As the engine design starts to become more concrete, higher fidelity models are developed. These higher detail models are intrinsically more sensitive to design tweaks, and in general take longer to setup. They inherently require more stringently defined configurations and boundary conditions. A large variation in the baseline design could potentially render entire higher fidelity analyses obsolete. Due to this weakness in adaptability, higher fidelity models are generally not created until the low-fidelity design has fully matured. This friction can partly be attributed to the differences in toolsets when transitioning to higher fidelity models.

Naturally, a trade-off exists on the amount of time spent in each analysis phase. Spending more time in the low fidelity phase could have large payoffs in the long run; however, without higher fidelity models it is hard to identify possible shortcomings such as those arising from operability and control issues. To optimize an engine across multiple disciplines, it becomes necessary to balance each trade-off in the design process as early as possible. This effort is intended to introduce transient and control considerations into the design cycle, while maintaining the rapid iteration and flexibility of a low-fidelity code.

Creating a single unified program to satisfy every design consideration runs a high risk of becoming overly complex or cumbersome. As models grow in capability, flexibility and program modularity is often lost. A natural reaction is for various disciplines to diverge and create unique tools satisfying their particular needs using frameworks and programming languages that are already of familiarity. This is acceptable as long as the codes preserve flexibility and rapid adaptability. An intermediate fidelity code can only be introduced earlier into the low fidelity analysis phase if it can keep pace with the rapid redesigns. TTECTrA is an attempt to provide intermediate fidelity insights, while maintaining the development speed of steady-state design tools.

The Numerical Propulsion System Simulation (NPSS) has become the industry standard for steady-state air breathing engine design. Written in C++, the object-oriented framework is generalized to solve any thermodynamic cycle, but has been primarily used to design gas turbine engines. The framework has been well integrated with noise, weight, and mission analysis codes for multi-point design, but has seen limited use in transient and control analyses. Despite having basic support for transient

operation, this barrier can be attributed to a historic difference in programming paradigms and environments between each discipline. Furthermore the controls design process has traditionally had a human-in-the-loop iteration cycle, making it challenging to integrate it into an automated multi-design point optimization. TTECTrA also aims to narrow this design gap.

At a specific flight condition TTECTrA produces a basic controller designed to meet user-defined goals, consisting of only the fundamental limiters that affect the transient performance of the engine. The controller provides a preliminary estimate of the transient performance by modifying user-defined engine control variables based on feedback received from simulation outputs. The tool is developed in the MATLAB/Simulink environment, which allows users to access a standard library of functions and to add on toolboxes such as the Control System Toolbox, which can be used to simplify the control design process. This user's guide is written assuming the user is familiar with NPSS, MATLAB and Simulink. TTECTrA consists of MATLAB functions and scripts written to perform control design calculations, based on interactions with a Simulink engine model. Inside the *NPSS_TTECTrA.mdl* file the *TTECTrA Simulink Block* implements a scheduled proportional integral (PI) controller with the designed set points, gains, and limiters and supplies the fuel flow input to the NPSS S-function block. More information regarding the integration of TTECTrA with NPSS is contained in the following section.

Version two of this program marks the integration of NPSS and TTECTrA, providing NPSS engine designers the ability to run native NPSS engine models wrapped directly within Simulink. This process enables designers to obtain an estimate of the closed-loop performance without redefining the engine in a separate language or codebase. This version of TTECTrA has been tested and verified using the 32-bit MATLAB version 7.10.0 (R2010a) with Simulink version 7.5 (R2010a). The 32-bit version of MATLAB must be used since the NPSSv1.6.x S-function is compiled with a 32-bit compiler. Later versions of MATLAB cannot be used since they do not support *.dll* files. Note that throughout this paper, MATLAB commands or MATLAB variable names will be in `Courier`, while generic variables, function names and file names will be *italicized*.

The Benefits of TTECTrA

Transient engine simulation provides another degree of merit for evaluating overall performance. Multiple turbomachinery limits and operational margins are usually baked into rule-of-thumb empirical constraints. Even with a roughly tuned controller, transient allowances and stall margin stack-ups can be more accurately bookkept. This gives the engine designer more latitude and confidence in designing closer to engine limits. Performance during large accelerations and decelerations can be better quantified, and designs can be checked against various constraints such as FAA mandated takeoff response time. This level of analysis is necessary when analyzing the feasibility of engine technologies where engines must be designed very close to their operability limits, or investigating slow moving actuators, such as shape memory alloys.

TTECTrA's ability to design first-cut controllers also provides a more familiar starting point for control engineers to further develop transient engine models. Compatible for both NPSS and MATLAB/Simulink based engine models, TTECTrA serves a common thread across development tools. By introducing NPSS into the MATLAB/Simulink environment, many additional toolboxes and development tools become available to the developer.

Controller Architecture

The general architecture of the controller designed and implemented by the TTECTrA tool is shown in Figure 1, where the details of each block are omitted for simplicity. A switch is used to select the fuel flow calculated using one of five possible control methods (identified by the "loop selection" flag). Closed-loop controllers modify fuel flow (the control variable) based on target outputs as well as measured outputs, while open-loop controllers determine plant inputs solely based on target outputs.

1. Closed-loop control with the controller designed using TTECTrA.
2. Open-loop control with a minimum fuel flow limit defined; this method is used to tune the deceleration limiter and can also get the open loop results while protecting the minimum limits, such as minimum fuel to air ratio and LPC surge margin.
3. Open-loop control where a fuel flow profile is defined; this method may be used for obtaining data for the acceleration limiter or deceleration limiter, or to obtain unlimited open loop responses.
4. Open-loop control with an acceleration schedule/limiter defined; this method is used to test the acceleration schedule and acceleration controller.
5. Closed-loop control using only the linear Proportional Integral controller and set point map function.

The NPSS model must promote at least one control variable feedback signal along with numerous output signals. Example outputs include corrected core speed and *Ps3*, which are used by the "acceleration" and "deceleration" logic blocks respectively. The fuel flow command is provided to the engine to close the loop with the TTECTrA controller, but it is important to note that this is not a closed-loop simulation if the loop selection is not set to either 1 or 5 (top and bottom paths shown in Figure 1).

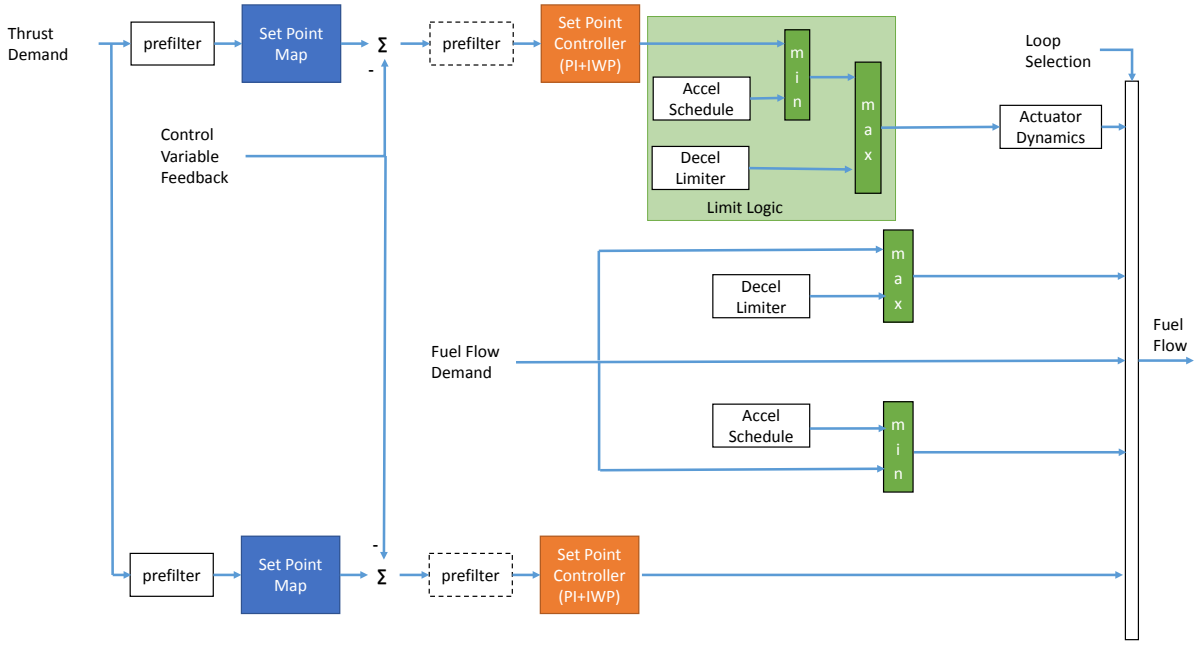

Figure 1.—Schematic of the controller implemented by the TTECTrA; parameters in the highlighted blocks are designed by TTECTrA.

The set point controller drives the control variable to a reference value which is determined from the relationship to the thrust calculated for the specific engine model; consequently, if the control variable tracks the demand, it is expected that the thrust produced by the engine also tracks the demanded thrust. The tracking response is dependent on the tuning of the proportional and integral gains of the controller and how tight the defined limiters for acceleration and deceleration are.

The acceleration limiter is designed to prevent high-pressure compressor surge during engine acceleration. The corrected core acceleration is compared to an acceleration limit that is dependent on the core speed and if the speed is too high, fuel flow to the engine is restricted. The core acceleration limit as a function of the corrected core speed is often referred to as an acceleration schedule. The acceleration schedule can be found by applying fuel flow profiles, transitioning from an idle fuel flow to a takeoff fuel flow, at varying rates until the minimum high-pressure compressor surge margin requirement is met (within some defined accuracy). The acceleration limiter uses a PI controller with integral wind-up protection to produce a fuel flow command designed to drive the engine acceleration to the limit value from the acceleration schedule based on the current (corrected) core speed.

A two-step process is used to determine deceleration limits. An initial *Wf/Ps3* limit value is defined based on the desired minimum fuel to air ratio. This value is found by analyzing the relationship between fuel to air ratio and *Wf/Ps3* in steady-state. The user can also determine this value based on the low-pressure compressor surge margin, however many NPSS models already include an operational variable bleed valve in its map and will not operate at such a low surge margin value. The second step is to fine tune this value by applying fuel flow transitions from a takeoff fuel flow to an idle fuel flow and finding the *Wf/Ps3* value which would preserve the acceptable fuel to air ratio (or low-pressure compressor surge margin). The fuel flow command produced by this limiter is calculated by multiplying the combustor pressure (*Ps3*) by the *Wf/Ps3* limit and does not contain any type of feedback controller.

The fuel flow command provided to the "actuator dynamics" block in Figure 1 and ultimately to the engine, is determined by first selecting the minimum of the fuel flow from the set point controller and that from the acceleration limiter. This fuel flow is then compared to the command from the deceleration limiter and the largest of these commands is selected. In this way, the fuel flow provided to the engine represents the controller that is operating closest to its respective set point. For more information regarding the design of an aircraft engine controller, the reader is referred to References 1 to 4.

Running TTECTrA allows the user to design parameters for implementation of the three highlighted blocks in the figure: the set point map (relationship between thrust and control variable), the set point controller's gain schedules for the PI controller, and the limit logic that depends on the acceleration schedule and the deceleration limiter.

The Integrated NPSS-TTECTrA Model

The TTECTrA tool leverages built-in MATLAB and Simulink functionality to design and implement a scheduled PI controller, with acceleration and deceleration limiters, for an NPSS engine model called from an S-Function block in Simulink. The following sections describe how the NPSS model must be integrated with TTECTrA prior to proceeding with control design.

Installation and Model Setup

Software Requirements for Using the Integrated NPSS/TTECTrA

The integrated NPSS/TTECTrA tool provides an interface for designing and implementing a gain scheduled PI controller with acceleration and deceleration transient limiters for a given NPSS model. The following requirements must be met in order to use this tool:

1. MATLAB/Simulink R2010a has been tested. Later versions of MATLAB will not work due to MATLAB no longer supporting the .dll extension.
2. 32-bit NPSS (version 1.6.4 has been tested)
3. Windows Operating System

As of June 2015, support for the newest releases of 32/64-bit MATLAB, Simulink, and NPSS are currently in progress.

TTECTrA Installation

The TTECTrA code consists of a Simulink block and a folder of custom MATLAB scripts and functions that are called during the execution of TTECTrA. To install TTECTrA for use with an engine model:

1. Right click on the zip package and choose Extract All. This will start the extraction wizard.
2. Select "Next" to bring up the Select a Destination screen.
3. Select "Browse" to open the file browser window and navigate to the folder that contains the engine model that will interact with TTECTrA. Select OK to return to the extraction wizard.
4. Select "Next" to extract all files to the desired location.
5. Select "Finish" to close the extraction wizard.

NPSS Installation

NPSS can be obtained from its commercial vendor Southwest Research Institute. Interested parties can download a trial version or purchase either a university or commercial license. NPSS is export controlled, and limited to countries not listed on the U.S. Department of Commerce Anti-Terrorism watch list (http://www.swri.org/npss/).

Installation is required to set proper environmental variable paths; but can also be run by manually calling the executable with the proper paths defined. Batch files are commonly created to properly setup custom path variables and run NPSS scripts more conveniently.

MATLAB Configuration

TTECTrA creates such a batch file, but requires the user to provide paths to the required files in *set_paths.m*. (Basic error handling messages are provided in TTECTrA to confirm paths are properly set.)

1. npss_location – Path to the parent directory of the NPSS executable
2. npss_engine_name – Name of the engine to be loaded and run
3. ttectra_engine_name – Name for saving simulation output
4. model_flag – NPSS preprocessor flags to be called by TTECTrA for engine information
5. ss_flag – NPSS preprocessor flags to be called by TTECTrA for steady-state information
6. ttectra_model_name – Name of the Simulink model to be run
7. model_location – Path to the NPSS engine model

TTECTrA also requires an NPSS engine that has been appropriately configured for transient operation. Details on this process are beyond the scope of this guide, and can be found in other materials (Refs. 5 and 8). After creating a transient engine within NPSS, the user must create a run file for an engine that can be configured from the command line with a limited number of flags to do the following.

1. Remove all active constraints from the solver that would conflict with TTECTrA, such as fuel, speed or temperature limiters.
2. Switch the solver solutionMode to "TRANSIENT"
3. Switch the burner mode to "Wfuel"
4. Add an independent/dependent pairs for transient conditions to be controlled by TTECTrA.
5. Specify the engine name, run file name, and any necessary flags in *set_paths.m*

Simulink Configuration

The Simulink model must be configured using the steps outlined in the S-function user's guide provided with official distributions of NPSS (Ref. 6). The config file supplied in the S-function dialog box defines the promoted input and output engine variables, along with NPSS run configuration variables.

TTECTrA Operation

This section focuses on setting up and operating TTECTrA and is an extension of Reference 7. The control design process using TTECTrA involves three main steps (1) calculating the set points, (2) finding the controller gains,
(3) calculating the acceleration and deceleration limiters. Once the integrated NPSS/TTECTrA tool has been initiated, TTECTrA will begin defining the steady-state relationships based on steady-state data obtained directly from NPSS. Next, the tool calculates the controller gains based on linear models, developed at defined operating points using NPSS. The final set of calculations addresses the need for implementing transient limiters to protect the engine. These protect against high- or low-pressure compressor surge, exceeding the maximum turbine inlet temperature, and exceeding a minimum fuel to air ratio. The current TTECTrA software only considers these two limiters, but it is possible to expand the limit logic to include additional constraints, such as core speed or *Ps3*.

After the controller has been designed a simulation will be executed to test the functionality of the controller, which includes small and large changes in thrust demand. The results for the control design and verification for the example model will be presented here along with discussion of each step of the design process using TTECTrA.

Before operating TTECTrA, the user has the option to specify default values and preferences for the parameters listed in Table 1 in the file *TTECTrA_Inputs.m*. The values in this file are loaded by TTECTrA and called when the Graphical User Interface (GUI) is started, but may be changed during the design process if necessary. Once the NPSS and TTECTrA installations have been completed and the default TTECTrA parameters have been modified (previous section), the TTECTrA software can be executed. There exists several ways to operate TTECTrA; through a custom GUI, from the MATLAB command prompt, or using an automated function, all of which will be discussed in one of the following sections.

TABLE 1.—USER DEFAULT INPUTS FROM TTECTrA_INPUTS.M FILE, WHICH ARE LOADED INTO THE GUI

SubField	Field name	Description
in	alt	Altitude (scalar)
	MN	Mach number (scalar)
	dTamb	Ambient temperature deviation from standard day (scalar)
	simTime	Length of the simulation (scalar)
	simFileName	File name (and extension) of the user's Simulink engine model with the TTECTrA Simulink Block controller
	filename	Optional name and extension to save the TTECTrA controller
	Ts	Sample time, in seconds
	setpoint_vector	Vector containing minimum and maximum thrust and other thrust points of interest for linear models
	linearModelfilename	Default name of the piecewise linear model developed by NPSS (DO NOT CHANGE)
controller	FdbkFilterBW	Feedback filter bandwidth if > 0, otherwise no filter is used
	PreFilterBW	Prefilter bandwidth
	CVoutput	Element of the linear model output vector (y_i) corresponding to the controlled variable
	bandwidth	Default bandwidth for tuning the controller
	phasemargin	Default phase margin for tuning the controller
	IWP_gain	Integral Windup Protection gain
	accel_k	Acceleration controller gain
	accel_bw	Acceleration controller bandwidth
	Accel_IWP	Acceleration controller Integral Windup Protection gain
actuator	wf_bw	Fuel flow actuator bandwidth
SMlimit	Accel	Minimum allowed surge margin during acceleration (for limiter design)
	T40	Maximum allowed high pressure turbine inlet temperatures (for limiter design)
	Decel	Minimum allowed surge margin during deceleration (for limiter design)
	FARmin	Minimum allowed high pressure turbine fuel to air ratio allowed during an acceleration (for limiter design)

TTECTrA Auto (Non-GUI Operation)

Once the user has entered the default inputs it the *TTECTrA_Inputs.m* file, the user can design a controller by running the *TTECTrA_Auto.m* file. This file is intended to design the control system for the given engine without any additional feedback or interaction from the user. To design the controller, the following functions are executed using the default values provided from *TTECTrA_Inputs.m*: set point controller design, acceleration limiter design, deceleration limiter design, integral windup protection tuning, and finally testing and verification.

Set Point Control

The TTECTrA setpoint controller contains a PI controller with integral wind-up protection, where the controller gains are scheduled as a function of the control variable. The controller gains are found using a custom automated PI tuning function contained in the file *custom_PIDtune2.m*. The algorithms designs a controller based on the linear model at each of the thrust points defined in the *setpoint_vector* variable of the *TTECTrA_Inputs.m* file from NPSS. The user can specify the controller bandwidth, phase margin, feedback filter (optional), desired control variable, and prefilter bandwidth which are listed in Table 1. The set point function will produce the controller gains (*Kp* and *Ki*) and the control variable feedback values at each linear model (*Fdkb*) and are stored in the *ttectra_in.gains* subfield. The actual controller gains are interpolated based on the current feedback. The next step is to design the acceleration limiter.

Acceleration Limiter

The acceleration limiter is designed to protect against a high-pressure compressor engine surge and exceeding the high-pressure turbine inlet temperature (T4) during a rapid acceleration. The acceleration schedule is designed using the default parameters in *TTECTrA_Inputs.m*. An example acceleration schedule produced by TTECTrA is shown in Figure 2, which shows the maximum acceleration allowed (y-axis) at a particular corrected core speed (x-axis). The acceleration schedule is saved in the *NcR25_sched* and *Ncdot_sched* fields of *ttectra_in.Limiter*.

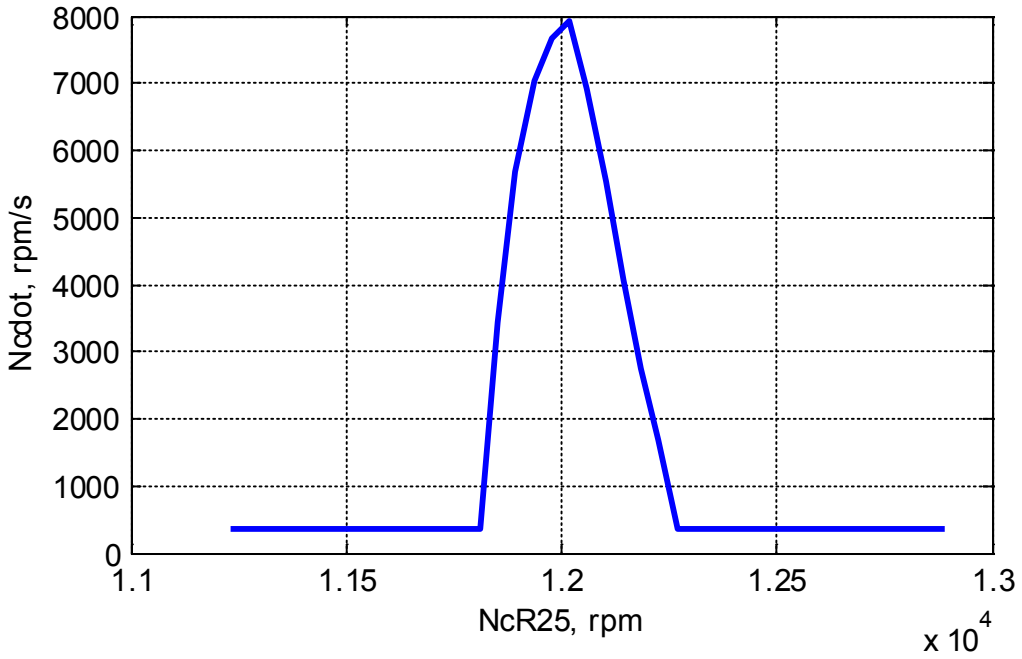

Figure 2.—Example acceleration schedule.

Deceleration Limiter

The deceleration limiter is designed to protect against a low pressure compressor engine surge and exceeding the minimum fuel to air ratio during a rapid deceleration. In this application the ratio unit limiter, defined as the fuel flow divided by the high pressure compressor static pressure (*Wf/Ps3*) is found which preserves both the minimum LPC surge margin and minimum fuel to air ratio. The deceleration limiter is saved in the *ttectra_in.Limiter.WfPs3lim* variable.

Integral Windup Protection

Integral windup is a common problem in control systems that contain integral action, especially those that have more than one controller. For any inactive controller (not producing an input to the plant at the current time), the integrator will continue to increase in magnitude due to a non-zero error (difference between set point and current feedback). To reduce the effect of the integral term in the non-active controllers, integral wind-up protection (IWP) is utilized. The approach implemented here[1] was adapted from CMAPSS40k (Ref. 2). The main idea with this approach is to decrease the error seen by the integrator of the inactive controller rather than zero the error out. This allows the integrator to increase to an appropriate value and decrease the size of the instantaneous change in magnitude when the inactive controller becomes active.

Verification/Simulation

To test the final design, a test profile has been designed which consists of a burst and chop profile, followed by small throttle transients from idle to full power to idle, followed by a very fast chop and burst profile. Figure 3 shows a comparison of the command and feedback for both thrust and the control variables (top and bottom plots, respectively). These plots show that the control variable drives the thrust to the commanded values, even though the controller does not have knowledge of the thrust produced by the engine.

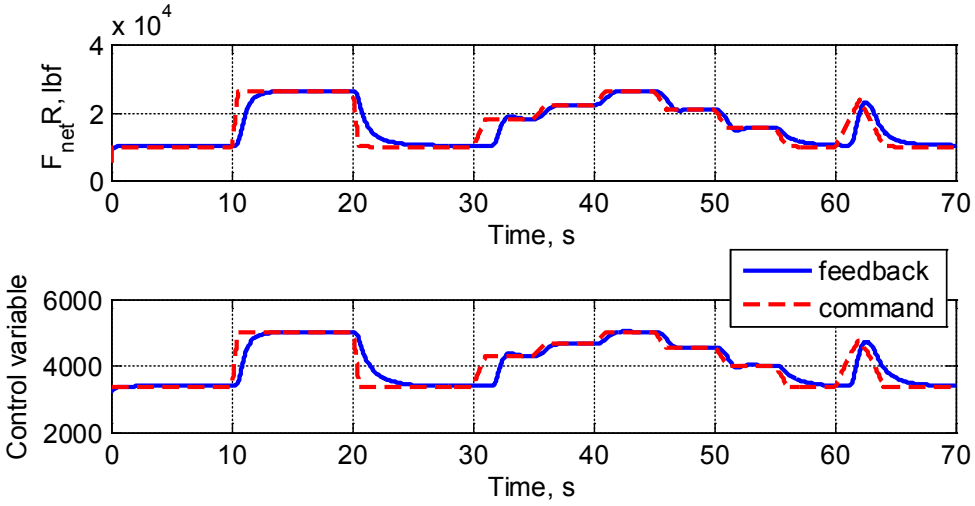

Figure 3.—Thrust and control variable commands and outputs.

[1] Martucci, A., and Volponi, A.J., "Fuzzy Fuel Flow Selection Logic for a Real Time Embedded Full Authority Digital Engine Control," *Journal of Engineering for Gas Turbine and Power*, 2003.

Even though the controller is able to drive the engine to the desired thrust, via the control variable, the presence of limiters slows this response, as can be seen in Figure 3. When the engine begins to accelerate from a low power (such as at 10 and 60 sec), it is operating near the acceleration schedule limit and the controller restricts the fuel flow to the engine to protect against HPC surge. This is reinforced by Figure 4, which shows that the engine does not reach the HPC surge margin limit or the T40 limit. Similarly, the controller ensures the engine does not exceed the fuel to air ratio limit on the decelerations (from high power to low power) as seen in Figure 5.

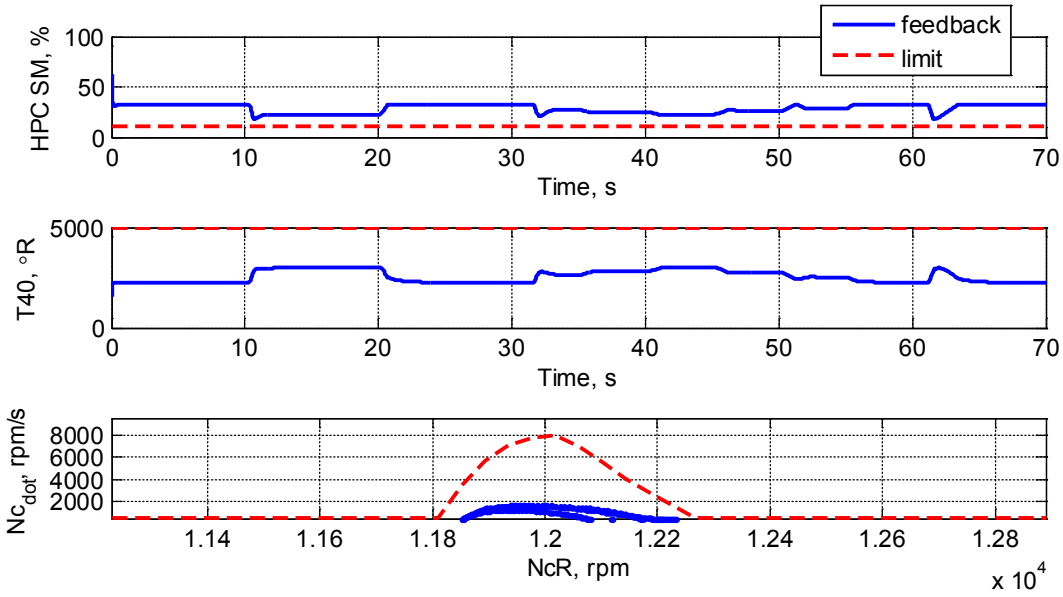

Figure 4.—The HPC surge margin and acceleration schedule for the large thrust transient.

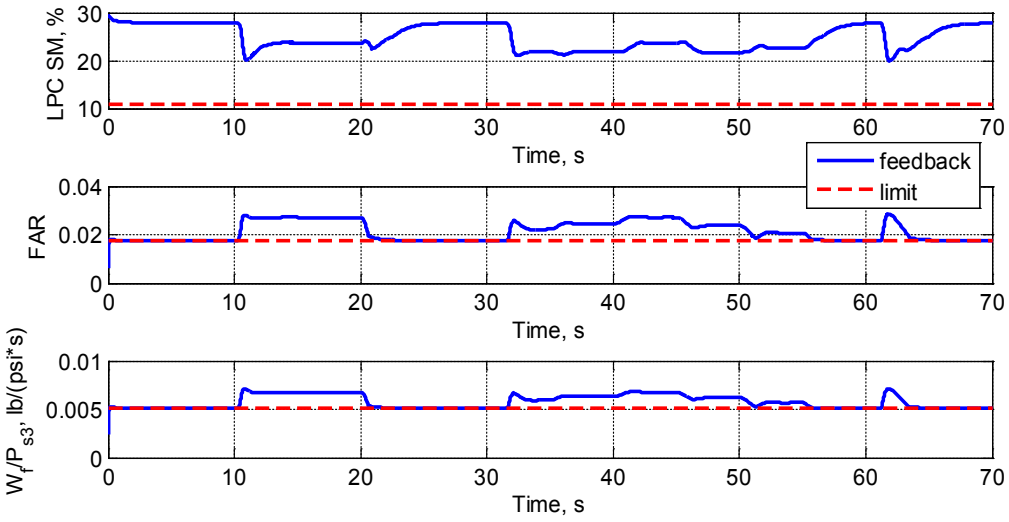

Figure 5.—The LPC surge margin and *Wf/Ps3* limit for the large thrust transient.

Performance Map Plotting

Turbomachinery performance maps are defined across five dimensions, often making it challenging to evaluate operating points based purely on numeric value alone. TTECTrA provides plotting scripts that automatically convert NPSS maps to MATLAB compatible matrices, and provides additional plotting utilities for visualizing operating points over transient runs as shown in Figure 6, Figure 7 and Figure 8.

Located within TTECTrA_Auto/plot is a GUI that automatically loads all NPSS map files with prefix "*mapData<name>.m*". The GUI also allows the user to dynamically slide and re-interpolate the maps for various variable geometry settings. Example standalone scripts are also provided for the provided sample engine. The user is required to save operating point variables to the workspace during transient runs, and provides map scalars (derived from the NPSS model).

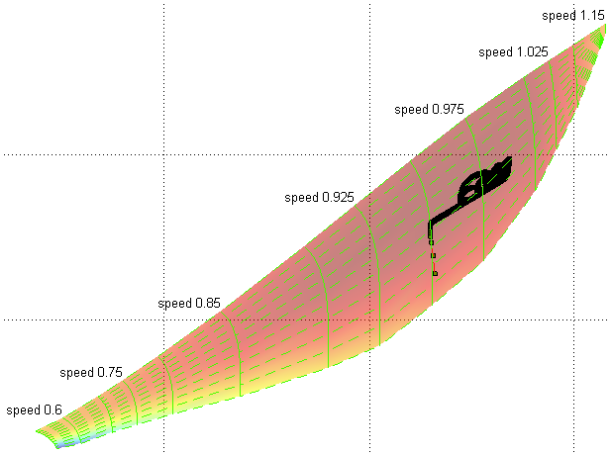

Figure 6.—Low pressure compressor performance map with operating point overlay.

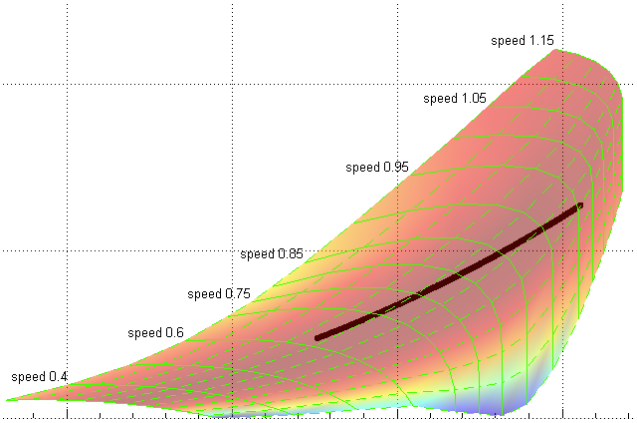

Figure 7.—High pressure compressor performance map with operating point overlay.

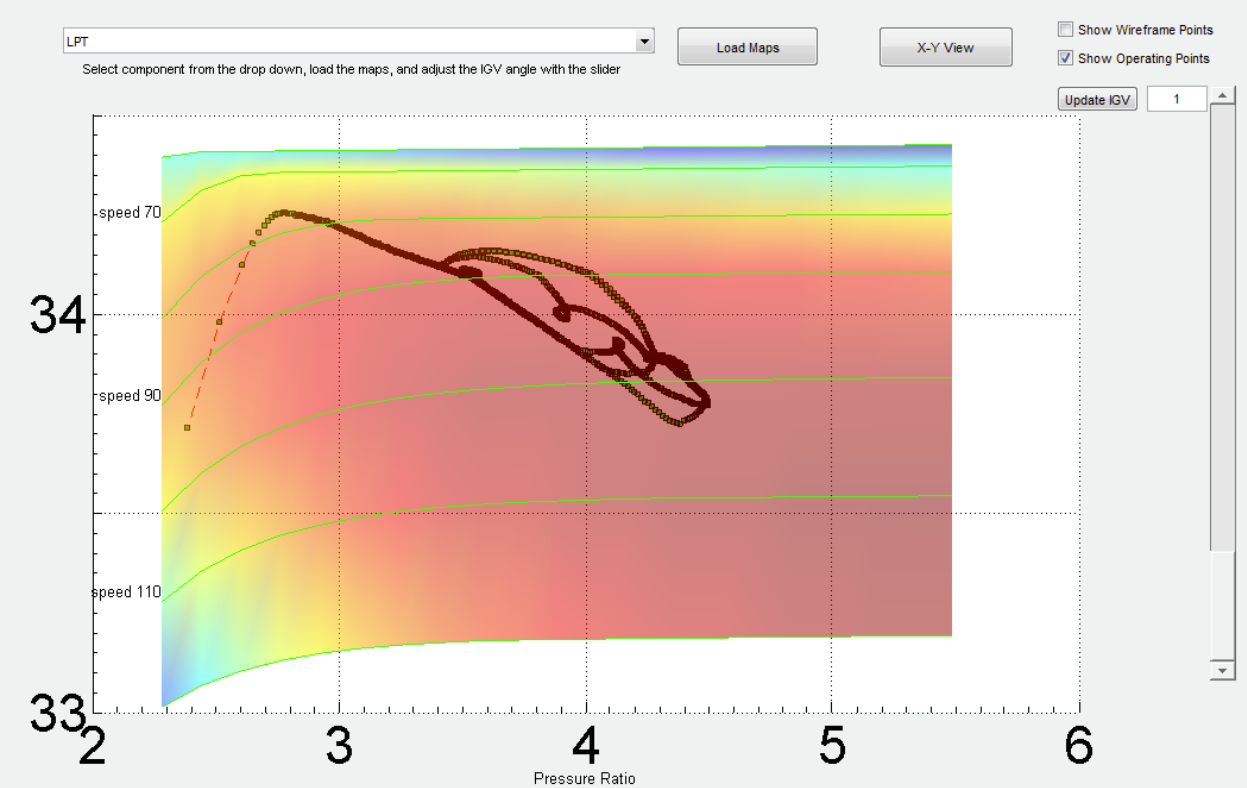

Figure 8.—Turbine performance maps with operating point overlay.

GUI Instructions:

1. Run *plotGUI3d.m,* **OR** one of the individual <*name*>*_plot.m* files and skip steps 2–7.
2. Select a map from the dropdown
3. Click “Load Maps”
4. Select Wireframe to see a wireframe for all variable geometry (IGV alpha) settings
5. Select a slider position to re-interpolate the map for an intermediate alpha setting
6. Optionally toggle operating points, using the checkbox provided
7. Click X-Y view for top down view (alpha in the Z-plane)

Saving Controller Data

TTECTrA will automatically save the controller data to the file specified in `ttectra_engine_name`, which is specified in the *set_paths.m* file. If the user does not want to save the file, then replace the name with empty brackets ‘[]’.

GUI Operation

To operate TTECTrA using the GUI, run the file *TTECTrA_gui.m* located in the MATLAB directory. Once this file is executed, the NPSS environment will be setup and called to obtain the necessary steady-state values and linear models. Once the design is complete, the verification and simulation will be the same as presented in the TTECTrA Operation section as well as saving the controller data.

Set Point Controller

TTECTrA begins the set point controller design by using the default parameters defined in *TTECTrA_Inputs.m* and plots the results for small thrust transients as shown in Figure 9. The top plot compares the thrust command and actual thrust (feedback). The bottom plot compares the control variable command and feedback, which in this example is the fan speed (*Nf*).
In order to modify the control variable setpoints using the GUI:

1. Enter the new bandwidth (Hz), phase margin (degrees), and feedback filter bandwidth (Hz) from the set point controller tuning GUI shown in Figure 10.
2. To recalculate the controller gains, press the execute button which will produce a new plot.
3. To accept the current design, press finish. Otherwise repeat the process with step 1.

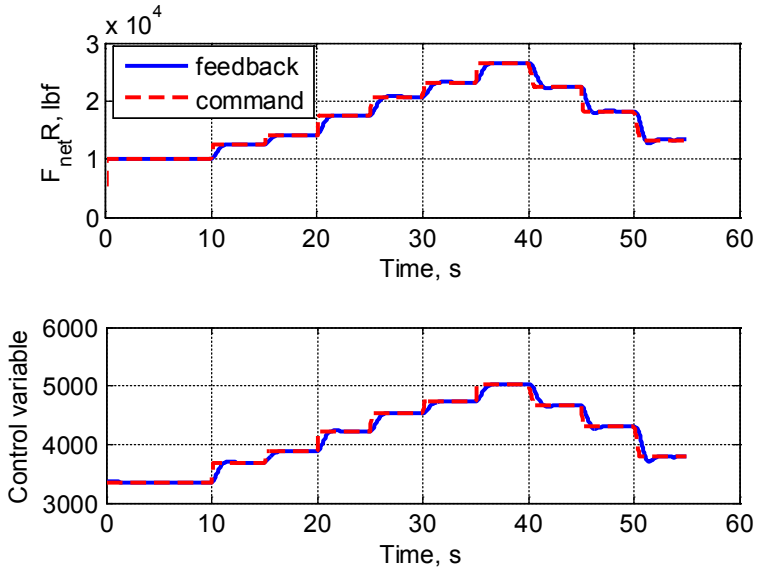

Figure 9.—Set point controller output plot.

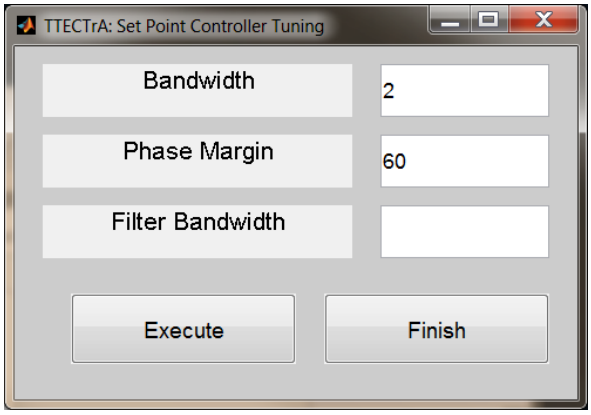

Figure 10.—Set point controller tuning GUI.

Acceleration Schedule and Limiter

The acceleration schedule can be tested at multiple conditions by applying a step change in *Wf* and selecting the minimum of two fuel flow results. The fuel flow that will drive the core acceleration to its limit defined by the acceleration controller versus the fuel input. This response is shown in Figure 11, where the top left plot shows the thrust response to the fuel flow transient (fractional units), top right plot shows the actual acceleration compared to the acceleration schedule, bottom left plot shows the surge margin compared to the minimum surge margin limit, and bottom right plot shows the T40 temperature compared to the max T40 temperature. If the results are unacceptable, the user can modify both the limits (minimum surge margin and T40) the schedule is defined for and the controller gains. If the feedback does not meet one or more of its limit, as shown by the top right, bottom left, and bottom right plots, then the controller most likely needs to be modified. However, if one or more of the plots show that a limit is being met, then the limiter most likely needs to be modified.

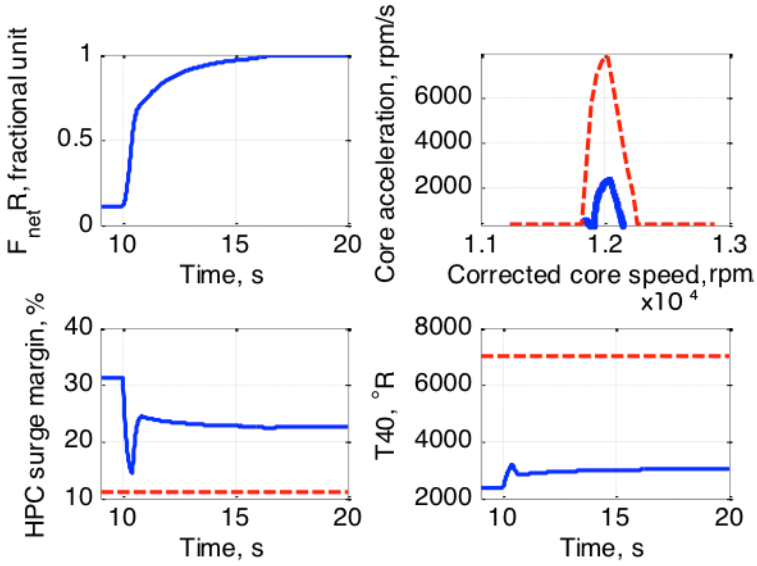

Figure 11.—Response of the acceleration controller to an applied fuel flow transient.

Modify the Acceleration Schedule

1. Enter the desired minimum high-pressure compressor surge margin in the Acceleration Schedule section of the Acceleration Limiter GUI, shown in Figure 12.
2. Enter the desired maximum T40 temperature in the Acceleration Schedule section of the Acceleration Limiter GUI (Figure 12).
3. Press the "ReCompute Accel Schedule" button (Figure 12).

Modify the Acceleration Controller Gains

1. Enter the desired controller gain in the Acceleration Limiter GUI shown in Figure 12.
2. Enter the desired time constant.
3. To produce the results with the new (currently shown) controller gains, press the Execute button shown in Figure 12.
4. To accept the acceleration schedule and controller gain and proceed, press Finish, otherwise repeat step 1 of either the "Modify the Acceleration Schedule" or this section.

Deceleration Limiter

The deceleration schedule can be tested by applying a fuel flow step change (from max fuel flow to minimum fuel flow) and taking the maximum of fuel flow input and the fuel flow derived by multiplying the current *Ps3* pressure by the *Wf/Ps3* limit. This response is shown in Figure 13, where the top left plot shows the fuel flow input, the top right plot shows the fuel to air ratio feedback and limit, the bottom left plot shows the LPC surge margin feedback and limit, and the bottom right plot shows the actual *Wf/Ps3* as compared to the limit.

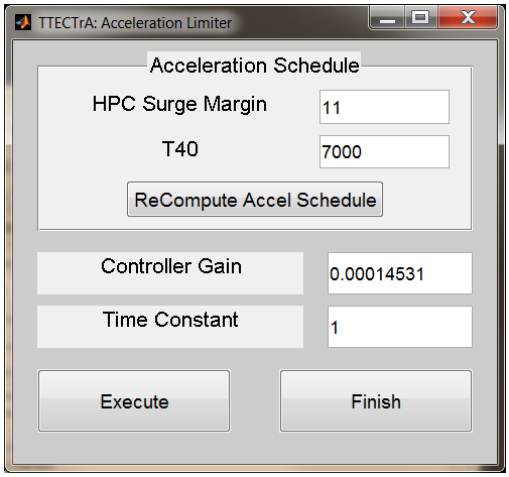

Figure 12.—Acceleration limiter GUI.

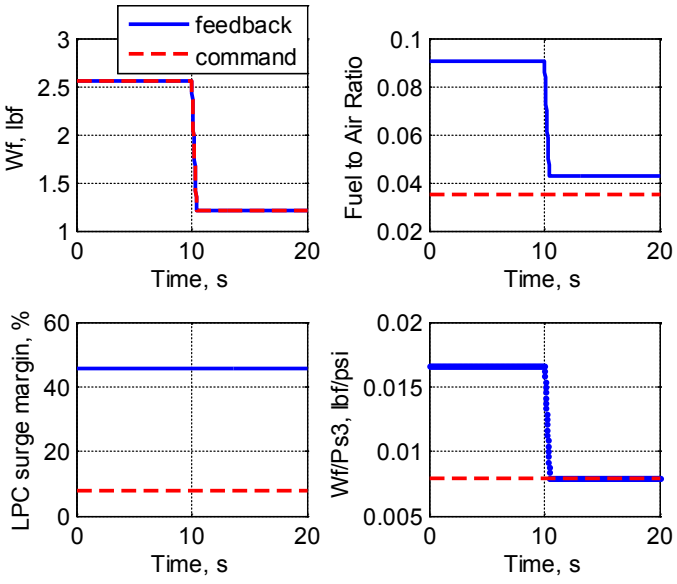

Figure 13.—The deceleration limiter output.

Modify the Deceleration Limiter

1. Enter the desired minimum low pressure compressor surge margin in the GUI shown in Figure 14.
2. Enter the desired minimum fuel to air ratio into the deceleration limiter of Figure 14.
3. To compute the deceleration limiter based on the new (and currently shown) specifications, press Execute.
4. To accept the new deceleration limiter and continue on with the control design process, press Finish.

Integral Windup Protection

The integral windup protection gain is empirically tuned to meet performance requirements using a custom function and algorithm (*TTECTrA_IWP_s.m*). During operation of the GUI, once the algorithm finishes a design by either meeting performance requirements or an iteration limit (watchdog), the final performance is shown in Figure 15.

Modify Integral Windup Protection Gain

1. Enter a new IWP gain and press Execute as shown in Figure 16.
2. To accept the new gain and continue, press the Finish button, otherwise repeat step 1.

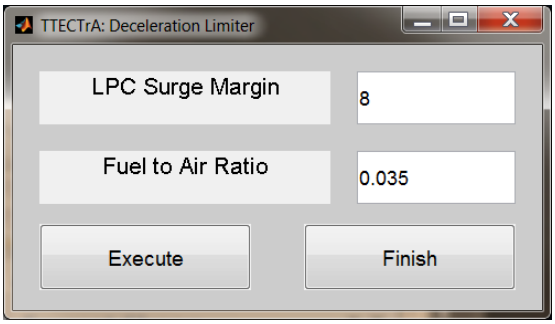

Figure 14.—Deceleration limiter GUI.

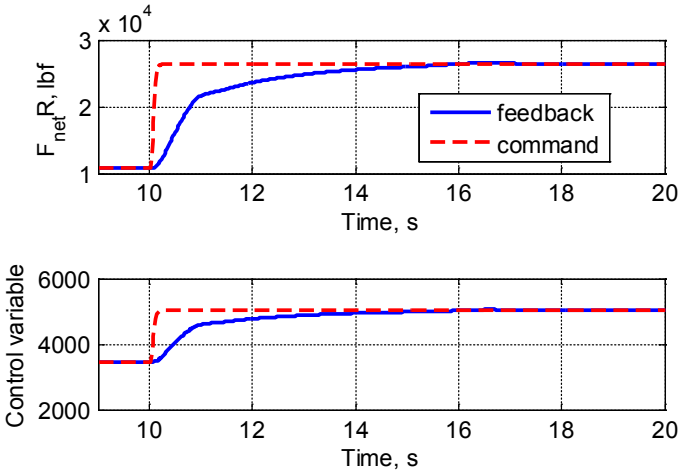

Figure 15.—The integral windup protection output plot.

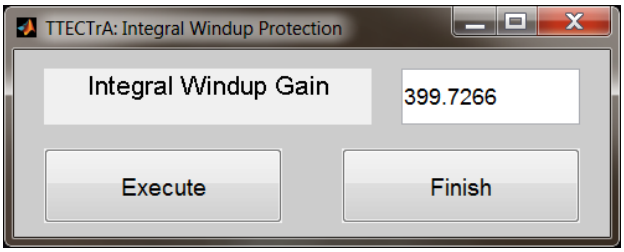

Figure 16.—The integral windup protection gain GUI.

Manual Design and Debugging

The TTECTrA software can be executed directly from the MATLAB command prompt. Even after designing a controller from the GUI or using the automated function, the user can attempt to debug the closed loop controller from the MATLAB command prompt.

Setting up TTECTrA

1. From the MATLAB directory of TTECTrA, add the *TTECTrA_Auto* folder and *TTECTrA_ManualTune* folders by either:
    a. From the current folder view of the MATLAB window, right click on the folder and select *"Add to Path"*
    b. From the command window, type addpath('FOLDER NAME')
2. Once the required folders are added to the MATLAB path, the user can setup the NPSS paths, get the required steady-state data and linear models from NPSS, and design the set point function by running the *ManualTune_BasicSetup* from the command prompt.

Set Point Controller Design

From the command line:

1. Modify any of the controller specifications at the MATLAB command prompt. The variables that can be modified are:
    a. *ttectra_in.controller.bandwidth*
    b. *ttectra_in.controller.phasemargin*
    c. *ttectra_in.controller.FdbkFilterBW* - Note that the filter feedback can be removed from the control system by using empty brackets '[]'.
2. Run the *ManualTune_spc.m* script. Once complete, a plot showing the current performance of the closed-loop controller for small throttle transients will appear as shown in Figure 9.

or by using the set point controller GUI by running *gui_spc.m*

Design of the Acceleration Schedule

From the command line:

1. Modify any of the acceleration schedule parameters at the MATLAB command prompt:
    a. *ttectra_in.SMLimit.Accel* (HPC surge margin)
    b. *ttectra_in.SMLimit.T40*
2. Run the ManualTune_accelschedule.m script. A plot showing the impact of the acceleration controller and schedule will appear as shown in Figure 11.

or the acceleration schedule can be recalculated from the acceleration controller by running the *gui_accel.m*.

Design of the Acceleration Controller

From the command line:

1. Modify any of the acceleration controller parameters at the MATLAB command prompt:
    a. *ttectra_in.controller.accel_k*
    b. *ttectra_in.controller.accel_bw*
2. Run the *ManualTune_accelcontroller.m* script. A plot showing the impact of the acceleration controller and schedule will appear as shown in Figure 11.

or the acceleration controller can be redesigned by running the *gui_accel.m*.

Design of the Deceleration Controller

From the command line:

1. Modify any of the deceleration schedule parameters at the MATLAB command prompt:
    a. *ttectra_in.SMLimit.Decel* (LPC surge margin)
    b. *ttectra_in.SMLimit.FARmin*
2. Run the *ManualTune_decelschedule.m* script. A plot showing the impact of the deceleration controller will appear as shown in Figure 13.

or from the deceleration GUI by running *decel_gui.m*.

Manual Tuning of the Integral Windup Protection Gain

From the command line:

1. Modify the integral windup protection gain at the command prompt, *ttectra_in.controller.IWP_gain*.
2. Run the *ManualTune_iwp.m* script. A plot showing the closed loop response will appear as shown in Figure 12.

or by running *gui_iwp.m* to use the integral windup protection gain GUI.

Verification/Simulation From the Command Line

The verification and simulation of the closed-loop control system can be obtained by running *ManualTune_testfinaldesign.m* script.

Saving the Controller Design

Similar to the GUI-less version of TTECTrA, calculated controller parameters can be saved to a specified folder and loaded in subsequent runs to avoid repeated calculations.

Final Remarks

Defining the boundaries between responsibilities of the TTECTrA controller and NPSS solver are still under development. Although passing engine parameters is fairly straightforward, the developer must take care when invoking multiple data passing techniques, especially when multiple data round-trips are required. Developers are advised to avoid setting up NPSS run files with side effects that may influence subsequent functions depending on the execution order. Furthermore, many existing NPSS models contain solver constraints that can directly conflict with limits and inputs dictated by TTECTrA. The NPSS developer is also responsible for creating engine models where the solver is robust enough to handle the wide range of transients applied by controller algorithm. Future development of the tool would be focused on reducing setup time and improving debugging tools to assist in developing robust NPSS models. To better assist the integration process, TTECTrA provides multiple NPSS engine configurations as examples. The tool is not intended to replace discipline experts, but rather, improve the communication and collaboration between disciplines with historically incompatible toolsets.

Appendix—Controller Elements to be Verified Against Another Model

The *TTECTrA Simulink Block* has been designed and tested on an in-house engine model and includes the following control elements in addition to those designed by TTECTrA:

- A gain correction on the setpoint controller based on *P2*, primarily to decrease the gain of the controller at higher altitudes.
- IWP gain in the setpoint controller, calculated by running simulations of the closed-loop model to find the gain that reduces overshoot during acceleration and deceleration below a specified threshold
- PI gains in the thrust setpoint controller, designed using linear models of the in-house model, and further adjusted to improve the model response
- PI and IWP gains in the acceleration schedule, designed (using the in-house model) to depend on the ambient pressure (altitude) at which the model is being simulated

Although necessary to obtain acceptable results from simulations of the in-house model, these modifications may not be required when the controller is implemented with other engine models. A piecewise-linear version of this in-house model has been developed and tested successfully with the tool, but it is necessary to test the controller block with other engine models (independent from the in-house model) to verify the necessity of these additional elements.

References

1. Jaw, L., and Mattingly, J.D., *Aircraft Engine Controls: Design, Systems Analysis, and Health Monitoring*, American Institute of Aeronautics and Astronautics, Inc., Virginia, 2009.
2. Mattingly, J.D., Heiser, W.H., and Pratt, D.T., *Aircraft Engine Design*, American Institute of Aeronautics, Inc., 2nd Edition, Virginia 2002.
3. Csank, J., May, R.D., Litt, J.S., and Guo, T.-H., "Control Design for a Generic Commercial Aircraft Engine," AIAA–2010–6629, 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Nashville, TN, July 25–28, 2010.
4. May, R.D., Csank, J., Lavelle, T.M., Litt, J.S., and Guo, T.-H., "A High-Fidelity Simulation of a Generic Commercial Aircraft Engine and Controller," 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Nashville, TN, July 25–28, 2010
5. The Ohio Aerospace Institute, on behalf of the NPSS Consortium. NPSS User Guide, 2010.
6. Southwest Research Institute, on behalf of the NPSS Consortium. NPSS Level 2 S-function Interface to Simulink, 2015.
7. Csank, J. and Zinnecker, A.M. "Tool for Turbine Engine Closed-loop Transient Analysis Users Guide," 2014.
8. Chin, J., Csank, J., Haller, W., and Seidel, J. "An Introduction to Transient Engine Applications Using the Numerical Propulsion System Simulation and Matlab," 2015.