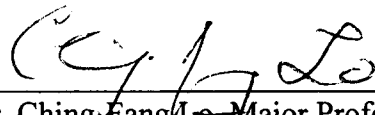
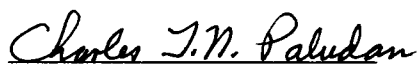
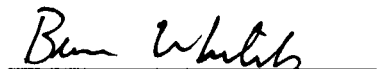


To the Graduate Council:

I am submitting herewith a thesis written by Mary Louise Craig entitled "Graphical User Interface for Off-line Trajectory Generation." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Engineering Science.


Dr. Ching-Fang Lo, Major Professor

We have read this thesis and
recommend its acceptance:

Accepted for the Council:


Associate Vice Chancellor and
Dean of the Graduate School

GRAPHICAL USER INTERFACE FOR
OFF-LINE TRAJECTORY GENERATION

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Mary Louise Craig
December 1997

DEDICATION

This is dedicated to my husband

Mr. Michael David Craig

and

my parents

Mr. Roy Watson Ferguson, Sr.

and

Mrs. Joan Crutcher Ferguson

who have always believed in me.

ACKNOWLEDGEMENTS

There are many people to whom I am grateful for making my time at the University of Tennessee so rewarding. I have benefited greatly from knowing several members of faculty in many different departments over the last few years. I am particularly grateful to Dr. Ching Lo for his advice and guidance during my UTSI years. I am also grateful to my Thesis Committee, Dr. Ching Lo, Dr. Ted Paludan, and Dr. Bruce Whitehead.

There are a number of other people whose assistance should be recognized. I would like to thank Ricky Arterbury, Andy Moore, Scott Keen, and Ric Clippard for invaluable inputs into my project. I would like to especially thank Ric Clippard who was my partner in the design, development, and coding of the software. I would like to thank my management, Dr. Bill Baker and Frank Langham, for allowing me to use my work for my thesis.

I would like to thank my husband, Mike. He has encouraged me through the entire program and kept me going when I wanted to quit. I would have never made it to this point without his help and support.

Abstract

Off-line trajectory generation is a method used to compute standard store (any object carried on an aircraft that is designed to be released in flight) separations using wind tunnel data. A six-degree-of-freedom (6-DOF) trajectory generation program (TGP) was originally developed at Arnold Engineering Development Center (AEDC) to control the Captive Trajectory Support (CTS) testing mechanism in aerodynamic tunnels 4T, 16T/S and A. To supplement the six-DOF prediction capability used on-line in the wind tunnels, a family of off-line analysis applications were developed based on the TGP. Variants of the TGP are used to compute standard store separations using wind tunnel data, Multi-Dimensional Interpolation Trajectory Generation Program (MDITGP) and ballistic predictions (BalTGP). The same six-DOF prediction capability is also used in the Helicopter Armament Store Separation Trajectory Generation Program (HassTGP) that analyzes the trajectory from a rotary-wing aircraft.

This suite of programs uses similar inputs to run simulations. The input process in the starting version was very extensive and time-consuming and required an experienced analysis engineer. Errors could easily occur and error checking was at a minimum in the old system. The previous file structure was binary direct access and was not portable between hardware platforms. The analysis technical staff has a strong need to transfer files between different hardware systems. This need for portability required writing a conversion program that had to run on each platform to put it into the machine specific binary form. The target computer platforms were the IRIS®, SUN® and PC.

The TGP programs run simulations quickly, however, an enormous amount of time was spent in the setup process. This process was chosen as an area that needed vast improvements in efficiency, error checking, quickness and ease of use. The programs and data files are now transportable directly without any software changes or conversion. The improvements to the system were implemented in two phases. Phase one consisted of the determination of an easily transportable file structure. Once the structure had been designed, software was written and tested to use the new file structure in the TGP software. Phase two consisted of the design and implementation of a Graphical User Interface (GUI) that used the new file structure.

Preface

Since 1987 the author has been involved with development of the Multi-Dimensional Interpolation Trajectory Generation Program (MDITGP) input structure. The original software resided on the Amdahl® 5860 at Arnold Engineering Development Center (AEDC). The input files were card image files, which had to be hand edited, and direct access unformatted files, which required use of a program for editing. In 1987 the author redesigned the card image inputs into a form that created several unformatted direct access files. This structure improved system flexibility but it also added several small programs that had to be used to modify the files. At that time, editors on the Amdahl® 5860 were not very user friendly so this file structure was the best that could easily be developed on that system. Over the past twelve years hardware platforms have changed along with standard editors that come with the hardware. This made the old question/answer type programs inefficient and harder to use. The need for improvements was seen as the working environment changed. The research in this paper reviews technical reports, existing and new software, and first hand experience.

The first four chapters deal with the trajectory generation process. Chapter One provides a brief description of the trajectory generation process and a brief description of some of the problems associated with the old method. Chapter Two provides a description of the Multi-Dimensional Interpolation Trajectory Generation Program (MDITGP) (which is now referred to as AutoTGP), Ballistic TGP (BalTGP) and Helicopter Armament Store Separation Trajectory Generation Program (HassTGP). This

describes what each program does and the different kinds of input each requires. Chapter Three provides a description of the setup procedure for the programs. It provides a description of each of the input files and how they are assembled and edited. A correlation between the inputs for the different software packages is given. Chapter Four describes in detail the process deficiencies that are currently in the TGP process.

The next two chapters deal with the software improvements. Chapter Five details the improvements recommended for the file structure and how these changes were implemented. The second phase of improvement lies on top on this structure. Chapter Six details the Graphical User Interface (GUI) design recommendations and how these changes were implemented. All software changes were made on the PC and Silicon Graphics Iris® Workstations.

Chapter Seven summarizes the research and the work accomplished through the two improvement phases. A brief description on the direction of future improvements are addressed.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
Background	1
Areas to be Studied	5
II. TRAJECTORY GENERATION PROGRAMS (TGP)	6
Multi-Dimensional Interpolation Trajectory Generation Program (MDITGP)	6
Ballistic Trajectory Generation Program (BalTGP)	8
Helicopter Armament Store Separation Trajectory Generation Program (HassTGP)	9
III. SETUP PROCEDURES	11
Description of File Types and Structures	11
Differences Between Program Inputs	19
IV. PROCESS DEFICIENCIES	21
Error Checking	21
Ease of Use	22
Portability	24
File Modifications	24
V. IMPROVEMENT PHASE ONE - FILE STRUCTURE IMPROVEMENTS	28
Breakdown of Data into Groups	28
File Types	30
RunTGP	33
VI. IMPROVEMENT PHASE TWO - GRAPHICAL USER INTERFACE DEVELOPMENT	41
Choosing Software Package	41
Designing Screens	43
Design Principles	44
Layout of Screens	46
Benefits	68
VII. SUMMARY	69
Future Development	70
BIBLIOGRAPHY	72

VITA	74
------------	----

LIST OF FIGURES

FIGURE	PAGE
1. Trajectory Generation Program Flow	4
2. Example Setup with Nomenclature	13
3. Example Constants Group (Group D).....	14
4. Example Interpolation Control Set (I)	15
5. Example Free-stream Data Control Set (FR).....	18
6. Example Grid Data Control Set (GR).....	18
7. Input File for HassTGP	20
8. Sample Menu from CONPREP Utility Program	26
9. Sample Menu from SETED, INTERP and SORTEDIT Utility Programs	27
10. Sample RUN File.....	32
11. Sample RIF File	35
12. Sample Setup File	38
13. Sample GRUN File.....	39
14. Sample Main Window	48
15. Sample Rotary Main Window	50
16. Sample Modify Window.....	51
17. Sample Rotary Control Parameter Window	52
18. Sample Main Fixed Wing Window	54
19. Sample File Options for Windows.....	54

20. Sample Modify Store Properties Windows.....	56
21. Sample Interpolation Window	56
22. Sample Nominal Window.....	57
23. Sample Nominal Window with Input Values	59
24. Sample Grid Interpolation/Extrapolation Window.....	60
25. Sample Alpha/Beta Window.....	61
26. Sample Cross Grid Window	62
27. Sample Control Parameters Window.....	64
28. Sample Ejector Forces Window.....	65
29. Sample Rail Geometry Window	66
30. Sample Release Conditions Window.....	67

LIST OF SYMBOLS AND/OR ABBREVIATIONS

A	control constants main deck
AC	aircraft identification number
ADS	Analysis Display System
AEDC	Arnold Engineering Development Center
ALPHA	aircraft angle of attack
ATCOM	Army Aviation and Troop Command
AutoTGP	Automatic Trajectory Generation Program
B-H	control constants groups
BalTGP	Ballistics Trajectory Generation Program
BETA	sideslip angle
CONF	configuration identification number
CONPREP	constant preparation program
CTL	input control file
CTS	captive trajectory support
DOF	degree-of-freedom
EVMCEP	Evasive Maneuver Criteria Evaluation Program
FlipTGP	Flow-field Load Influence Prediction Trajectory Generation Program
FlowTGP	Flow Trajectory Generation Program
FR	free-stream run number
FSRUN	free-stream run number file

GR	grid run number
GRUN	grid run number file
GUI	graphical user interface
HassTGP	Helicopter Armament Store Separation Trajectory Generation Program
I	interpolation control set number
INTERP	interpolation edit program
M	mach number
MDITGP	Multi-dimensional Interpolation Trajectory Generation Program
OPF	operational flight program
RIF	required inputs file
RNG	range file
RUN	ASCII input file
SET	set file
SETED	set edit program
SORT	sort program
SORTEDIT	edit sort file program
STORE	store identification number
TGP	Trajectory Generation Program
TR	trajectory number
Δt	delta time

CHAPTER I

INTRODUCTION

Historically, a period of several years might occur between the time a new weapon system was developed and the time it was certified and placed with military units in the field. This included testing at the full range of anticipated aircraft loadings and flight conditions which was time consuming and expensive. Off-line trajectory generation was used to predict store trajectories analytically and to supplement existing experimental data which reduces the time and costs of store certification efforts (AEDC, 1994). With the downsizing of the Department of Defense budget, analysis tools such as these are being used more extensively than ever. The defense industry's need to be competitive has led to streamlining processes and keeping up with current technology. The off-line trajectory analysis interfaces need to be easy to use and take advantage of the latest technology.

Background

A six-degree-of-freedom (6-DOF) trajectory generation program (TGP) was originally developed at Arnold Engineering Development Center (AEDC) to control the Captive Trajectory Support (CTS) testing mechanisms in Aerodynamic Tunnels 4T, 16T/S, and A. To supplement the six-DOF prediction capability used on-line in the wind

tunnels, a family of off-line analysis applications was developed based on the TGP.

Variants of the TGP code are used to compute standard store separations using wind tunnel data (MDITGP), store separations from bays and for non-standard situations (FlipTGP), ballistic predictions (BalTGP), and store separations using flow angle maps in an aircraft flowfield (FlowTGP). TGP derivatives are the primary tools used to analyze store trajectories at AEDC.

Regardless of whether store trajectory information is determined in a wind tunnel simulation or computationally, the overall trajectory generation process is very similar. The primary difference is the manner in which the store dynamics in the aircraft flow field are derived. Store trajectories are generated basically by a repetitive solution of the Newtonian (i.e., force equals mass times acceleration) equations of motion for the store (Carmen, 1980). Beginning in most cases with predicted or measured forces and moments on the store at its installed position on the aircraft (the store carriage position), the equations of motion are evaluated using the known inertial properties of the store to determine the translational and rotational accelerations of the store which occur as a result of the applied forces and moments. The calculated accelerations represent the time rate of change of the initial store position and orientation 'states'. Integration of these 'state variables' over some small time step allows the prediction of the new store position and orientation relative to the aircraft at the end of the time step (Carmen, 1980). For example, if Euler integration (which is the simplest form of the numerical time integration) is used, the state equations take the form:

$$\begin{aligned} &(\text{position and orientation state variables at time } t + \Delta t = \\ &(\text{state variables at time } t) + (\Delta t) * (\text{time rate of change of state variables}) \end{aligned}$$

Forces and moments on the store are then predicted or measured at the new position and orientation. The equations of motion are re-evaluated using the new loads to predict the store movement over the next time interval and the process is repeated until the entire separation trajectory is defined as shown in Figure 1 (Carmen, 1980).

Many factors may contribute to the total forces and moments acting on a store which are needed for a trajectory simulation. These factors may be categorized as being either of an aerodynamic nature or of a non-aerodynamic nature. Non-aerodynamic loads include ejector forces, thrust of powered vehicles, weight, umbilical and lanyard forces, and any other loads that are not caused by airflow around the vehicle. The non-aerodynamic loads are usually fixed inputs to the store equations of motion and are defined prior to the trajectory simulation (Keen, 1990). Aerodynamic store loads are more difficult to determine. There are three major classes of aerodynamic effects acting on a store during its trajectory. Aerodynamic forces and moments are induced by the flow angles and local flow velocities in the aircraft flow-field; aerodynamic loads are also induced by the relative translational motion of the store through the flow field; and finally, loads are induced by the relative rotational motion of the store within the flow field (Keen, 1990). The determination of all three contributors to the total store aerodynamic loads is the major task of both experimental and computational trajectory generation.

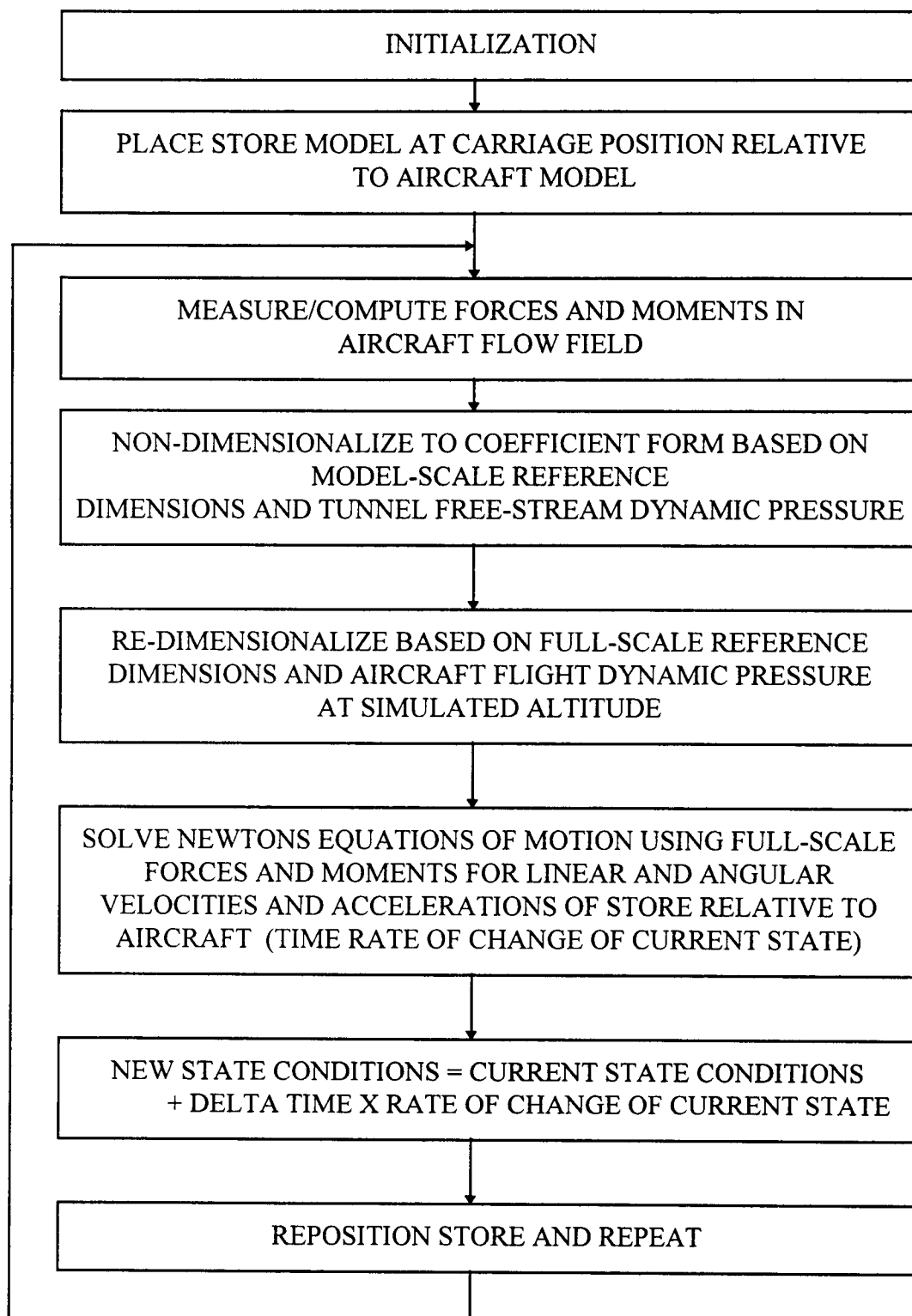


Figure 1. Trajectory Generation Program Flow.

Areas to be Studied

Because the vast nature of the physical problem, a tremendous amount of input is needed to run trajectories off-line. The data are segregated into several different categories such as control constant data, grid data from the wind-tunnel, interpolation information, and run number files that tell which wind tunnel data runs correspond to a particular configuration and flight condition. With such a large amount of manual data input, the likelihood of errors is significant. These files had to be edited with utility programs because they were originally unformatted direct access. This led to portability issues between platforms as well as a crippling of flexibility. With the high importance of these separation programs in the weapons certification process, a re-evaluation of the whole input methodology was necessary. This system needed to be very error-free, flexible, transportable and easy to understand so that the work could be completed in a timely and cost-effective manner.

CHAPTER II

TRAJECTORY GENERATION PROGRAMS (TGP)

The suite of analysis programs that will be discussed is the Multi-Dimensional Interpolation Trajectory Generation Program (MDITGP), Ballistic Trajectory Generation Program (BalTGP) and Helicopter Armament Store Separation Trajectory Generation Program (HassTGP). There are several other variants of the trajectory generation program that will not be covered because they have not yet been included in this development work.

Multi-Dimensional Interpolation Trajectory Generation Program (MDITGP)

The MDITGP was developed from the six-degree-of-freedom (six-DOF) trajectory generation program (TGP) that was originally designed to control the Captive Trajectory Support (CTS) testing mechanism in aerodynamic tunnels 4T, 16T/S and A (Carmen, 1980). This variant of the TGP is used to compute standard store separations using wind tunnel data. The data are taken in grid testing mode wherein aerodynamic loads are measured on the store at a spatial grid of predetermined positions and orientations relative to the aircraft model. Trajectories can then be generated off-line based on the information in the grid data base. Once the grid data are obtained for a particular aircraft configuration, multiple trajectories may be generated from the same

data base by parametrically varying the store launch initial conditions. The captive trajectory rig-positioning and force-measurement processes used in the wind tunnel are replaced in the off-line code, with an interpolation package designed to determine the forces and moments acting on the store by searching grid and free-stream data bases as a function of the store position and orientation state variables obtained from the trajectory equations of motion (Keen, 1990).

An input method had to be designed to incorporate the information that is normally read from the data acquisition system in the tunnels along with the standard input files used in the tunnel. The tunnel uses a binary direct access file that contains the control constants. Input of the free-stream and grid run numbers for particular configuration and flight conditions and the interpolation information which describes the spatial distribution of the aerodynamic grid data had to be added to the standard TGP to allow off-line processing to occur. The actual grid and free-stream files taken from the tunnel also had to be accessed. The first version of the MDITGP used a single card deck for everything other than the control constants and the actual free-stream and grid data.

In 1988 a variant of the MDITGP (called AutoTGP) was created that replaced the single card file input with unformatted direct access files. A setup file structure, improved off-line processing of data, and invaluable audit trail and data retrieval capabilities were added. Flexibility was incorporated with new input files that were created and edited with their own utility programs. The AutoTGP was the software that was used as the primary basis for this study.

Ballistic Trajectory Generation Program (BalTGP)

In 1989 a request was made to explore new ways to reduce the time and cost required to develop or improve the ballistic trajectory prediction algorithm for an aircraft/store system. A Ballistic Trajectory Generation Program (BalTGP) was developed that permits application of wind tunnel test data to the problem of ballistic trajectory prediction. The BalTGP is an extension of the AutoTGP version of the off-line program. The use of wind tunnel data, equations of motion and aerodynamic calculations were already proven so this was an obvious starting point for development (AEDC, 1994). With the equations of motion of a falling object readily solved, the integrated method in the TGP was extended to predict the trajectory of a store from release, through the interference field, all the way to impact. Trajectory information like this can then be analyzed and used to set the bombing algorithm that is part of the aircraft operational flight program (OFP). Before the development of the off-line capability, the bombing algorithms in most aircraft had previously been determined through a sequence of expensive and time-consuming flight tests.

The BalTGP is composed of the AutoTGP and several enhancements to increase program efficiency and to facilitate more accurate modeling of some factors that may have effect upon an extended trajectory. Features addressed by the BalTGP to improve trajectory modeling include variations in local atmospheric temperature and pressure along the trajectory path, Magnus force and moment, Coriolis force, variation of aerodynamic damping coefficient derivatives as a function of store aeroballistic angle of

attack, and atmospheric winds. Program efficiency enhancements include increasing integration time increments when possible, use of linearized free-stream coefficients when possible, and bypassing the flow-field interference coefficient calculation module when the store has exited the aircraft flow field (AEDC, 1994).

Helicopter Armament Store Separation Trajectory Generation Program (HassTGP)

The U.S. Army Aviation and Troop Command (ATCOM) has sponsored a simulation model of helicopter flight known as the code Evasive Maneuver Criteria Evaluation Program (EVMCEP). A six-DOF simulation similar to the MDITGP was added to EVMCEP to allow simultaneous simulation of a rotary-wing aircraft and a released store (AEDC, 1994).

One of the major concerns in store separation from a helicopter is insuring that the store trajectory does not penetrate the rotor path. To address this concern, the tip path plane of the rotor must be predicted during store separation. The trajectory is based on initial conditions, a store aerodynamic math model, and a rotor wake flow field model. A partial list of inputs required to predict a separation trajectory includes: 1) math models for the aerodynamics of the stores, 2) ejector forces and velocities, 3) thrust functions, 4) store inertial data, 5) atmospheric data and 6) aircraft flight conditions (AEDC, 1994).

This analysis tool can be used to assist the airworthiness and certification process of weapons and other stores on tri-service (Army, Navy and Air Force) rotary-winged

aircraft. This capability is easily expanded to include additional aircraft and stores as the evolving mission definitions require.

CHAPTER III

SETUP PROCEDURES

The setup procedures for the AutoTGP and the BalTGP were the same. The most efficient method of planning and executing trajectory predictions was the same for on-line and off-line operations. To maintain commonality with the optimized methods devised for store separation testing, the off-line TGP simulation was controlled by specialized files in a process designed to emulate wind tunnel operations while integrating functions needed for off-line TGP applications with minor modification. Additional automation features designed into the off-line TGP take advantage of the setup file structure and the possibilities it offered for improved off-line processing of data. The HassTGP setup structure was a single input card deck that had namelist capabilities. This structure allowed one simulation to be run at a time with file modification between each trajectory.

Description of File Types and Structures

The programs described in this section originally used unformatted direct access files as input which meant that the user had to use a program to view any element in the file. The files could not be edited or viewed using a standard text editor. The examples shown in this section were produced using the display programs that were used to build

and modify these files. They are shown for information purposes only. Each of these elements were stored in its file according to the set number in a corresponding record.

Overall control of the program was directed from the setup files and all other files were called from the setup. Each setup had to be a unique set of values. The values in the setup indicated the control constants, data files, trajectory number, and other control inputs for a particular run. The information necessary for overall control of the simulation was drawn from the setup file. The values in the setup indicate the simulated flight condition (M, ALPHA), the input control constants (A-H), the interpolation control (I), the set containing the interference flowfield grid data run numbers (GR), the set containing the aerodynamic store free-stream data run numbers (FR), the unique trajectory number (TR), and the aircraft/store configuration identification for a particular run (CONF, AC, STORE). The setup methodology originated from tunnel operations and was expanded to meet the needs of off-line analysis. If the files were managed as intended, each setup would be unique, allowing the setup files to double as the primary audit trail and organizational tool. The setup file allowed the user to run multiple trajectory simulations by specifying a range of setups to run at any given time. This allowed the AutoTGP and BalTGP to run in a batch mode scenario. A sample line of information from a setup sheet file is shown in Figure 2 along with a description of each entry.

The control constant inputs represent store parameters, ejector forces, etc. The constants are organized into a main deck (A) and constant groups (B-H). Typically each store (or family of stores) was assigned a main deck and nominal values were stored for

SET	M	ALPHA	A	B	C	D	E	F	G	H	I	GR	FR	TR	CONF	AC	STOR
1	0.80	1.1	2	5	1	0	0	0	1	0	23	35	17	12078	52	1	5
M			Mach Number														
ALPHA			Aircraft angle of attack														
A			Control Constants Main Deck														
B-H			Control Constants Groups														
I			Interpolation Control Set Number														
GR			Grid Run Number Set Number														
FR			Free-stream Run Number Set Number														
TR			Trajectory Number, assigned identification														
CONF			Configuration Identification Number														
AC			Aircraft Identification Number														
STORE			Store Identification Number														

Figure 2. Example Setup with Nomenclature.

use as defaults. Constant groups contain control constants usually perturbed as sets, such as ejector controls, store mass properties, or aircraft carriage orientations. The total number of control constants in the main deck was fixed at three-hundred-sixty for AutoTGP and seven-hundred-twenty for BalTGP. The constant groups could have any subset of the total number up to ninety-nine. The constant groups were used with each main deck and store specific data were established in set number ranges within groups. As the constants were read in alphabetical order, values were reassigned. If a value for the same constant was in "A" and in "D", the value in "D" would be retained for that trajectory. Figure 3 shows a sample of a constant group.

The Interpolation Control "I" files contained nominal spatial information about the input aerodynamic data, the method of interpolation and/or extrapolation to be used

Group D			
SET 1:			
136.	137.	138.	139.
1.77700E+00	-5.38000E+02	6.33000E-02	4.50000E+02

Figure 3. Example Constants Group (Group D).

for the simulation, and control parameters used to calculate the six aerodynamic delta-coefficient values as a function of the six grid data independent variables, XG, YG, ZG, PSIG, THAG, and PHIG. The AutoTGP and BalTGP use a six-dimensional data array which requires the primary variable to be identified. The primary variable is the independent position coordinate in which the primary interpolation/extrapolation will be performed. Nominal values are given for all the non-primary independent variables that would be used when reading the data from the data file. A tolerance was specified for all six variables. A sample of an interpolation file set is shown in Figure 4. The value of IT denoted the primary independent variable. Every variable with a 1-6 at the end corresponded to the six independent variables (XG, YG, ZG, PSIG, PHIG, THAG, or PHIG respectively). The NP value associated with the primary variable was the maximum number of data points along any traverse in the primary direction at a single combination of the nominal values of the other variables. The remaining five NP values define the number of locations of the non-primary independent variables at which data along traverses in the primary direction are available. The INT values defined the method

```

                                INTERPOLATION PARAMETERS

SET NUMBER 3
IT NP1 NP2 NP3 NP4 NP5 NP6 INT1 INT2 INT3 INT4 INT5 INT6 IX1 IX2 IX3 IX4 IX5 IX6
3   1   2  20   4   1   2   0   1   2   1   1   0   0   0   1   0   0   0

VTOL1 VTOL2 VTOL3 VTOL4 VTOL5 VTOL6
0.50   0.50   0.20   2.50   0.50   10.0

NOMINAL VALUES
0.
0.0    4.0
-10.   -5.    0.    10.
5.
0.     5.

NA  INTA  IXA  MIRA  NB  INB  IXB  MIRB
23   2    0    0    4   1    0    0

ATOL  BTOL
1.000  1.000

BETA1  BETA2  BETA3  BETA4
-20.00  -10.00   0.0   10.0

FREESTREAM CHANNEL LOCATIONS
95 96 97 98 99 100 45 77

GRID CHANNEL LOCATIONS
101 102 103 99 100 98 77 78 79 67 68 69 71 72

DELTA COEFFICIENT CHANNEL LOCATIONS
0 0 0 0 0 0 0 0 0 0 0 0 0

STORE      AC      WING
BOMB      A-PLANE  RIGHT

PRINT FLAGS --- FREESTREAM  GRID  TRAJECTORY SUMMARY PAGES
                1          1  3  3  1  2  3  2  3

```

Figure 4. Example Interpolation Control Set (I).

of interpolation to be used in each independent variable. A value of one denotes linear interpolation while a value of two denotes Akima spline interpolation. A zero should be used if data only exists at one nominal value. The value of IX for each independent variable tells what type of extrapolation the user wanted for delta coefficients, one for linear or zero for none. The VTOL values specified the tolerances for delta-coefficient interpolation for each independent variable. The nominal values should be specified for each of the non-primary variables. The number of variables given should correspond to the NP specified above. There were only five sets of these specified since they do not apply to the primary variable. The value of NA was the maximum number of values of store angle of attack, ALPHAS, along any ALPHAS sweep at fixed store angle of sideslip, BETAS. The value of NB was the number of nominal BETAS values. The values of INTRPA and INTRPB determined the type of interpolation to be used in ALPHAS and BETAS, respectively, with 1 denoting linear interpolation and two denoting Akima spline interpolation. The values of IXA and IXB determine whether extrapolation is to be allowed. The value of IMIRA denotes whether symmetry was to be assumed in the event that data were available for only positive or negative ALPHAS values. IMIRB was defined analogously, but applies to BETAS values. The ATOL and BTOL defined the tolerance for ALPHAS and BETAS, respectively. The BETA1-BETA(NB) contained the nominal BETAS values at which ALPHAS-sweep data were available. These values had to be input in ascending order. Input data file locations for the aerodynamic data was specified for the desired axis system. There are eight free-stream channels corresponding to ALPHA, BETA, CN, CY, CA, CLL, CLM, and CLN.

The fourteen grid data channels corresponded to X, Y, Z, PSI, THA, PHI, ALPHA, BETA, CN, CY, CA, CLL, CLM, and CLN. If pre-calculated delta coefficients were used, the twelve delta coefficient channels were input and corresponded to X, Y, Z, PSI, THA, PHI, DCN, DCY, DCA, DCLL, DCLM, and DCLN. The labels for the aircraft, wing and store were input and used on the summary page printouts. The print flags told whether output was to be printed or not. A flag was defined for the free-stream and grid coefficient printout and seven hard-set summary pages.

The aerodynamic data input used for the AutoTGP and BalTGP simulation were stored in C binary files that were obtained from the tunnel test. To direct the correct run numbers into the program, free-stream and grid data input run numbers were stored and input as "sort" sets, identified in the setup files as FR and GR, respectively. A sufficient amount of information to permit sorting an entire test data set into individual configuration/condition sets was input as parameters (name, value, and tolerance). The name had to accurately reflect the channel name in the input file, the value for the respective channel was the nominal value, and the tolerance had to be adequate to include all the desired data and exclude data belonging to other configuration/condition sets. The parameter sets could be sorted automatically by a utility program or input manually. The number of run numbers would change with the respective sets of data and was not input directly. Once the GR and FR files were assembled with run numbers, the files could be called by the setup files. A sample FR file is shown in Figure 5 and a sample GR file is shown in Figure 6.

SORT PARAMETERS				
SET NUMBER 6				
NUMBER OF PARAMETERS 3				
NAME:	CONFIG		MACH	PHIS
VALUE:	9002		0.9	45.0
TOLERANCE:	0.4		0.2	4.0
NUMBER OF RUN NUMBERS:			4	
RUNS:	45	645	745	845

Figure 5. Example Free-stream Data Control Set (FR).

SORT PARAMETERS									
SET NUMBER 4									
NUMBER OF PARAMETERS 3									
NAME:	CONFIG		MACH		ALPHA				
VALUE:	11200		0.9		5.0				
TOLERANCE:	0.4		0.2		0.5				
NUMBER OF RUN NUMBERS: 10									
RUNS:	1220	1221	1222	1224	1226	1227	1228	1229	1231
1233									

Figure 6. Example Grid Data Control Set (GR).

The HassTGP input structure is different from the above structures for the AutoTGP and BalTGP. The HassTGP inputs are the standard namelist structure and is a single input ASCII file that will only allow one simulation to be run at a time. A sample HassTGP input file is shown in Figure 7.

Difference Between Program Inputs

The three programs AutoTGP, BalTGP, and HassTGP need much of the same information to run simulations. The AutoTGP and BalTGP use Analysis and Display System (ADS) files for their input data while the HassTGP uses a rotor wake flowfield model file. The ADS files come directly from the tunnel testing area. A math model file is used to build the rotor wake model file for the HassTGP. The HassTPG allows for maneuvering information to be input along with the standard separation variables. The three software packages all need aircraft properties, reference properties, mass properties, carriage and initial conditions, pivot/rail values, program control, thrust and ejector information. The BalTGP has more program options, which in turn means it needed more input information for control constants (720 versus 360 for AutoTGP). In addition the BalTGP has options that require weather and drag data.

```

INPUT DECK FOR 19 TUBE LAUNCHER
PROGRAM CONTROL (TIME IN SECONDS)
TEJECT   TBO   RUNOUT
  1.0    0.0   5001.

ITHRST   IEJECT
  0       1

STORE CARRIAGE DISPLACEMENTS (FSPA,BLPA & WLPA IN INCHES, ANGLES IN DEG)
  FSPA   BLPA   WLPA   PSIPA   THTPA   PHIPA   XE1
  322.3  -80.0  225.75   0.0    4.0    0.0   0.649

STORE DESCRIPTION (UNITS ARE FT**2, FT AND FT RESPECTIVELY)
  S      CBAR    B
  1.39   1.33    1.33

STORE MASS PROPERTIES (MASS IN SLUGS, CG IN FT, INERTIAS IN SLUG-FT**2)
LAUNCH MASS   XCG   YCG   ZCG   IXX   IYY   IZZ   IXZ
  2.519  2.983   0.0   0.0   0.629   7.12   7.12   0.0

BRNOUT MASS   XCG   YCG   ZCG   IXX   IYY   IZZ   IXZ
  2.519  2.983   0.0   0.0   0.629   7.12   7.12   0.0

XBTCG   YBTCG   ZBTCG   BALPHT   BPHIT   THRUST   THRO
  0.0    0.0    0.0    0.0    0.0    0.0    0.0

TIME   THRUST
  0.0   0.0
  0.1   0.0

```

Figure 7. Input File for HassTGP.

CHAPTER IV

PROCESS DEFICIENCIES

Off-line store separation trajectory generation has grown significantly over the years, especially for the testing of new weapons systems. With the greater demand, the whole store separation process had to be evaluated for process deficiencies. A group consisting of programmers and experienced store separation engineers was assigned to study this problem. The group started by looking at how the process currently works and at the hardware system used by the current implementation. The nature of software development had changed since the initial input system was designed. The group decided that a graphical user interface (GUI) should be used for the input of information; however, the current file structure was seen as not being conducive to the development of a GUI. The group decided that the underlying file structure would have to be redesigned before the process could be improved. There were several main areas that were identified as process deficiencies. These included error checking, ease of use, portability, and file modifications.

Error Checking

Error checking was one area where the system was not very effective. Some error checking was built into the editing programs but there were places where the user could

input bad information and there was no way to catch it until the TGP program calculated incorrect data results. This was due in part to the use of so many different files with many using different edit utilities. This made error tracking very time-consuming and overwhelming for the inexperienced user. The control constant utility program had the user inputting information by a number rather than by a name. The numbers were defined in a user's manual that the user had to have at their disposal while building files. Experienced users depended on their memory when building these files and could easily forget or mistype the constant number they were trying to set or change. The channel numbers that were input into the interpolation file were representative of certain variables but only the index numbers were input into the file. It was a common mistake to mistype the channel number and the program would not catch it because any number was valid. The user had to get a copy of the file variable list and find each variable and then find the channel number associated with it. A good interface system should allow the user to specify input using the name instead of the number or getting the information from a pull-down menu that lists all acceptable variable name choices, thereby reducing the potential for error.

Ease of Use

The process of setting up the necessary input files using the appropriate edit program could be very intimidating to the uninitiated user and could be difficult even for an experienced user. Even though there was a common thread that ran throughout the

edit programs, each was different and each required familiarity developed from experience to be used well. In addition, this process was not flexible enough to allow for the addition of new variables easily. If a change needed to be made then the file structure had to be rebuilt and space allocated for a larger record along with modification to the edit programs to accommodate the new parameter. Also, because of the linear nature of the inputs, if the user wanted to change one variable in any one of the files, they had to go through several steps to even get to the point to change the variable. It took five different programs to get the data ready for input into the appropriate TGP. This meant a new user would usually need several user's manuals just to do the setup work. There was also a high possibility that the manuals were not up to date and they would have to rely on one of the experienced analysts for help. The system was not intuitive. There were several places where a value was wanted in an edit option and the user did not get a description of what that value was. For example, in the interpolation file the user was allowed to choose what type of interpolation they would like to do. The program knew that zero, one, or two were valid inputs but it did not have any information as to what those numbers meant. Descriptions should inform the user a zero gives no interpolation, a one gives linear interpolation and a two gives Akima interpolation. A brief description may be all the user needs to make an intelligent decision.

Portability

Portability had been a big problem with this system for years. To get around the problem, conversion programs had been written on all the different platforms. The conversion programs converted the unformatted direct access data into an ASCII form that could be transferred to another system which also had a conversion program that took the ASCII form and converted it back to an unformatted direct access file. This made the process of transporting data very time-consuming and inefficient. The users routinely exchanged information back and forth between the PC, Silicon Graphics®, and Sun® workstations. Also, direct access files had to be skeletonized on some platforms (built to be filled with zeros with as many records as the user thinks they will ever need) and this limited the expandability of the files. If there was a requirement to add another set and the file was at its maximum then the user would have to build new files or delete one of the already existing sets. This destroys the audit trail feature of the system. Portability was a huge design consideration.

File Modifications

To do modifications to the files, the user had to run one of the appropriate edit utilities for the type of file being modified. The programs used were CONPREP, SETED, INTERP, SORT, and SORTEDIT. These were written specifically to allow a user to interactively manipulate the files described earlier. The CONPREP program

allowed the constant main deck (A) and constant groups (B-H) to be edited separately.

The options available allowed the user to update or display a main deck, update or display a constants group set, delete constants from constants group sets, establish a current main deck from scratch, or copy a constant group set by adding or overwriting existing constant groups. The user was prompted for needed information at each executable step which can delay the process of getting to the variable that was desired to be changed. The menu is shown in Figure 8 for the CONPREP utility. The actual menu text shown in all caps, all other text is only there for descriptive purposes.

The SETED, SORTEDIT, and INTERP programs all had similar menu options which included the following: list existing sets that contain information, display a set number or range of sets to the screen, clear a set or range of sets, add a set by edit where the user is prompted for inputs sequentially, edit set number selectively, edit range of sets so that values are changed for a range of sets simultaneously, and copy one set number of data to another set number either to new locations or over existing sets. The interactive utility program prompted the user for needed inputs with explanations provided. Default values were displayed inside pointed brackets, "<>", while examples were shown inside parenthesis, "()". With the SORTEDIT program the user must first specify whether the program would be editing free-stream or grid run number sets. Sometimes this process took several prompting iterations before the user got to the point of changing the value they were looking for. A sample menu is shown in Figure 9. As before, the actual menu is only the wording in all caps, all other text is only there for descriptive purposes.

- 1 - UPDATE A MAIN DECK
Individual constant locations can be edited.
- 2 - DISPLAY A MAIN DECK
The specified set is written to the screen.
- 3 - UPDATE A CONSTANTS GROUP
Individual constant locations in group B-H can be edited.
- 4 - DISPLAY A CONSTANTS GROUP SET
The specified group will be written to the screen.
- 5 - UPDATE CONSTANTS GROUPS BY CONSTANT NUMBER STRING
A constant location can be edited for a specified range of sets.
- 6 - DELETE CONSTANT FROM CONSTANTS GROUP SETS
Deleting empty locations prevents unwanted overwriting with zero.
- 7 - ESTABLISH A CURRENT MAIN DECK
Adding a set by edit, from scratch.
- 13 - POSTLAUNCH VARIABLES
Not used.
- 14 - COPY A CONSTANTS GROUP SET
Add or overwrite existing constant groups.
- 15 - EXIT PROGRAM

Figure 8. Sample Menu from CONPREP Utility Program.

1 - LIST EXISTING SETS

The file numbers of the available sets are displayed.

2 - DISPLAY A SET # (RANGE OR SINGLE)

Sets are printed to the screen.

3 - CLEAR A SET # (RANGE OR SINGLE)

Unused sets can be eliminated entirely.

4 - ADD A SET BY EDIT

User is prompted for the needed inputs sequentially.

5 - EDIT SET #

Sets are edited selectively

6 - EDIT RANGE OF SETS

Values are changed for a range of sets simultaneously.

7 - COPY SET # TO ANOTHER SET #

Sets are copied to new locations or over existing sets.

8 - QUIT

Figure 9. Sample Menu from SETED, INTERP and SORTEDIT Utility Programs.

CHAPTER V

IMPROVEMENT PHASE ONE - FILE STRUCTURE IMPROVEMENTS

Phase one consisted of redesigning the file structures which needed to be highly flexible and easily transportable between platforms. The entire design had to keep in mind the final product which was a GUI. The file structure had to be flexible enough that new variables could be added easily without having to change the file structure. This flexibility is needed for all the test peculiar applications that occur with tunnel data and autopilot integration. The new design should allow the same interface to be used for multiple versions of the TGP such as AutoTGP, BalTGP, and HassTGP along with all future versions of the TGP without having to change the file system. The file structure needed to be intuitive which meant it had to be easy to build and the user did not have to have a manual to build it. It was decided that the file structure change would be implemented first and tested and then the GUI would be designed to operate under this structure.

Breakdown of Data into Groups

After thoroughly examining the data it became apparent that it fell into groups: group1, group2, etc. Several files in the old process were deemed to be important. The interpolation data and the grid and free-stream run number files would still need to be

grouped into different files as before. The concept of the setup file was considered very important and beneficial to the user. It provided a means of setting up several different parametric studies using the same data files. The setup would need to allow the user the capability to run multiple trajectories in a batch mode. The constants file which consisted of three-hundred-sixty variables in one large file could be broken down into smaller groups that were more functional. The data that were in the constant file are sometimes used for parametric studies and are usually grouped according to what variables are being studied. It is the intent of this new file structure to allow the data to be grouped according to functionality and at the user's discretion.

Some of the constant data fell easily into distinct groups. The store data consist of several small groups within each store consisting of the store mass properties, store reference dimensions, and store damping derivatives. Other groups that developed were the store aerodynamic offset coefficients, aircraft flight conditions, store carriage orientation, store/ejector references, ejector parameters, and postlaunch parameters. In documentation of the different TGP's this is the functional breakdown that is used to describe the variables. It was determined that the data structure should be flexible to have as many groups as the user deems necessary for their particular study. The user may group the data in any number of groups that they would like and it can and will vary from user to user. The file structure would also have to ensure that all data parameters needed for a particular version of the TGP be defined and specified.

File Types

The old file structures were found to be inflexible and very limiting to the user. The files had to be changed if a new variable was added and the editing programs had to be changed. There was no flexibility in the use of the files. Everything had to be defined in a certain order and pattern. The file structures were very dependent on the hardware platform on which it was created. Since, the most portable format for any file is ASCII, it was decided that an ASCII file input would be used for the new system. When using the old file structure a considerable amount of code was incorporated in the TGP to allow for processing the different file inputs. The TGP should be removed from trying to consolidate all the different groups for input, so it was decided that an intermediate process would do the consolidation. The TGP itself would read a single ASCII file that had all the information given in it. The TGP would only run one set at a time and the batch processing that consisted of multiple sets being run would reside outside the TGP as an intermediate program.

The versions of the various TGP's that use this file structure have been called the Toolkit. This toolkit allows the TGP to be executed from a single ASCII file called the RUN file. The RUN file contains all of the variable names and respective values required for program execution for a single run. The RUN file is 'human readable' and can be edited with any standard text editor and is fully portable between PC and UNIX platforms.

The data in the RUN file are keyed off a constant name not a number like the previous version. The descriptive name gives the user a short description of the variable being input. The name of each constant appears in the RUN file followed by its value for the current run. The names and values can be entered in several formats:

- The variable name followed by its value
- A line of variable names followed by a line of their respective values
- An array followed by its size in parentheses (direct size specification) and its values on the next line
- An array followed by a constant name containing the array size in parentheses (indirect size specification) and the array values on the next line

Actually these are merely suggested arrangements. The combination of names and values is not important as long as there are the same number of variable names (including arrays) as there are values and they are in the same order; i.e., the tenth name corresponds to the tenth value. The only exception is that a variable value must be specified before it can be used as an indirect array size specification. An example of the various entries are shown in Figure 10.

There are several rules that hold true for the RUN file.

- Every variable required by the TGP program must appear in the RUN file.

Neither the case nor order are important; however, it must be spelled exactly as the TGP expects it. Leading and trailing spaces are ignored. All variables must begin with a letter and cannot exceed twenty-four characters.

```

! * Store properties *
L      10.
WT     3000.
C(2)   1.    2.
IT      3
NP(IT) 1.    2.    3.
VAL1 VAL2 VAL3
0.      1.    2.

```

Figure 10. Sample RUN file.

- Time order is only important when a variable name is used to dimension an array. In this case the dimensioning variable must be defined and its value assigned to it before it is used to specify an array size. All elements of the dimensioned array must be set explicitly; that is, there must be a value for every index.
- All variable names are treated as floating point numbers (no integers nor characters). If the variable value is used to dimension an array, only the integer part is relevant.
- Toolkit files are 'human-readable' ASCII files. They may be created using a standard text editor or word processor on the PC, UNIX workstation or by a FORTRAN or C program or even an Excel® or Lotus® spreadsheet. If they are created by a word processor or spreadsheet program, however, they must be saved as text-only files.
- All files must end in a carriage return regardless of what is used to create them – that is, they must have a blank line at the end. The reason for this is that the

files are read with a standard FORTRAN read for formatted input that does not read a line unless it ends with a carriage return.

- Any characters after an exclamation mark (!) to the end of the line are ignored. The exclamation mark may appear anywhere on a line. If it is at the beginning of the line, the entire line is ignored.
- Blank lines are ignored.
- The separator between variable names or between variable names and their values can be any number of spaces or tabs.

RunTGP

It was determined that the file structure and batch capabilities needed to be implemented and tested before the GUI software was written. A software package was written to emulate the functionality of the GUI and is called RunTGP. It consolidates all the data from the setup file and creates the RUN file that is passed to the appropriate TGP program. The TGP program will be forked from the RunTGP program and after execution control will return to the RunTGP program so that the next set can be executed. When the TGP is executed it will read the RUN file and extract the information that is required. If a required variable is not found during the consolidation process of the RunTGP then the RUN file will not be created and an error message will be written. In order for the user to know which variables are required for the TGP they are trying to execute, a Required Inputs File (RIF) is created by the TGP programmer and provided

with each version of the software to the user. The RIF contains a list of all the variable names required by the TGP. Each name is usually followed by a comment that provides a brief description. Typically the variables in the RIF are also grouped according to a function with a descriptive comment preceding each group. Array variables are specified with the maximum number of elements enclosed in parenthesis. The TGP program does not use the RIF in any way; it is used primarily for input validation by the user and to aid in building the RUN file that is passed to the TGP. It is also used as a help file for the user to look for a description of variables. A sample RIF file is shown in Figure 11.

The RunTGP program uses a set of auxiliary files to repeatedly assemble and validate a RUN file and execute the TGP. The toolkit files required for automated execution of RunTGP are:

- The setup file (SET) is the primary program control file.
- The group files contain related variables and their respective values.
- The FSRUN and GRUN files contain related free-stream and grid run numbers.
- The range file (RNG) specifies the range of setups to be executed by the RunTGP.
- The control (CTL) identifies the RIF, SET and RNG files.

The SET file is where the user describes what information needs to be assembled and how it is assembled for each trajectory. The SET file does not contain any TGP constants, interpolation information, free-stream or grid run numbers, but rather, has a list of file names that contain the necessary information. The RunTGP program creates

```

!
! ** Store Properties **
!
L          ! Length
WT         ! Weight
XCG        ! cg location from the nose, x
IXX        ! Inertia, Ixx
IYY        ! Inertia, Iyy
IZZ        ! Inertia, Izz
IXY        ! Inertia, Ixy
IXZ        ! Inertia, Ixz
IYZ        ! Inertia, Iyz
SREF       ! Area, ft**2, full-scale
XL1        ! Store pitching-moment reference length, ft, full-scale
XL2        ! Store yawing-moment reference length, ft, full-scale
XL3        ! Store rolling-moment reference length, ft, full-scale
CLP        ! Clp
CMQ        ! Cmq
CNR        ! Cnr

```

Figure 11. Sample RIF file.

a RUN file by combining contents of each file listed on a single line of the SET file. The setup number is used to select a line to assemble. Given a set number RunTGP reads each group file on the same line and combines them with the interpolation file and run numbers from the FSRUN and GRUN files. Finally, the values of any variables matching the overlay variables are changed to their new values and the RUN file is written. The TGP is then invoked to process the RUN file. Once the TGP finishes, the whole process repeats with the next requested set number.

The SET file is broken into two parts: the setup section and the path section. The setup section contains all the information for each trajectory. The path section contains

the required fully qualified file names for INTERP, FSRUN and GRUN as well as those files that were referenced in the setup section. The name used in the setup section that references an actual file name in the path section is called the alias. This feature allows the user to use a short name rather than a fully qualified name for each of the groups.

Each line in the setup section contains the information necessary to assemble a single RUN file. The first non-commented line of the file is a header line that must contain the words: SETUP, FSRUN, GRUN, INTERP followed by zero or more overlay constant names. The overlay variables are used to override a constant value in the assembled groups. After all the files have been read and before the RUN file is written, the RunTGP replaces all the variable values listed in the overlay with the corresponding specified value. All the overlay variables must be specified in the header line. If one is not used for a particular setup, the user puts an X instead of a value for those overlays to ignore on the setup line. Also overlay variable names must also appear in a group file; i.e., the name is not added to the RUN file but merely the value of an existing name is replaced.

The header line is followed by the trajectory setup lines that contain the SETUP, FSRUN, GRUN, INTERP, OVLY and GROUP values. Under the SETUP heading the set number is specified which is used as the trajectory number in the TGP. The set number is a single integer value that will be used to select the setup lines when RunTGP is executed. The FSRUN value is the set number(s) in the free-stream run number files. The values can be single, multiple, range or a combination (e.g. 1,2-4). The GRUN value is the set number(s) in the grid run number file and can be specified like the FSRUN.

The INTERP value is specified in one of two ways. Either the file name containing one set of the interpolation information or an index indicating the set number to be read from the file associated with the alias INTERP. The OVLY variable names are specified on the header line and the values to be used in place of those read from the other files. The GROUP specifies the group aliases/files for this setup. The GROUP section is a list of aliases and/or filenames, separated by commas, which contain the run data for this trajectory. RunTGP will check each group name to see if it is an alias to another file. If an alias is found, the file associated with the alias will be loaded. Otherwise, RunTGP will try to load the file as specified. The number of groups used may vary for each trajectory as long as there is at least one group on each line and every variable has a value specified. Each major field (SETUP, FSRUN, GRUN, INTERP, overlays, and groups) is separated by one or more spaces and/or tabs. Within each major field, multiple values are separated by commas or dashes where appropriate. In Figure 12 a sample setup file is shown.

The FSRUN and GRUN files have identical formats. Zero or more variable names are listed at the beginning of each file. If variable names are specified, the value for each will be supplied in the run number set specification. Each set starts with the set number followed by the values for the overlay variables. Values would not be supplied if there are no variables specified for this file or if they should be ignored for this set. The set number and the overlay variables must appear on the same line and can only occupy one line. The next line contains the run number count and the run numbers for the set. The run numbers may span several lines and the number per line is not fixed. It has been

```

!
! Setup Section
!
SETUP   FSRUN  GRUN  INTERP  Mach  Alpha  !Groups
  2       1    1,2    3,4    0.8   2.0   MAIN EJECT1 MASS1
  1       1-3   3-5    1,5-6  0.9   5.0   MAIN EJECT1 MASS2

!
! Path Section
!
PATH = .
FSRUN = b1b.fsrn
GRUN = b1b.grun
INTERP = b1b.interp
MASS1 = b1b.mass1
MASS2 = b1b.mass2

```

Figure 12. Sample Setup File.

found that while this format is very simple, it is sometimes hard to remember error-prone since if the run number count is wrong the file is misread. It is the intent of the GUI to have checks that allow this to be an error-free file when all the software checks are in place. A sample GRUN file is shown in Figure 13.

The INTERP field is reserved for the interpolation information. Either a file name, alias, or a number may be specified. If a file alias or name is given, the entire contents of that file will be added to the RUN file; otherwise, the interpolation set corresponding to the value specified is added from the file associated with the alias INTERP. The structure of the interp file is the same as a standard group if it is a stand-

GRIDMACH						!Overlay variable names on first 3 lines
GRIDALPHA						
GRIDSTORE						
#1	0.9	1.1	11115.			! Run set number - Overlay values
5	101	102	103	104	105	! Number runs - run numbers
#2	.85	1.1	11115			! Run set number - Overlay values
3	202	203	204			! Number runs - run numbers

Figure 13. Sample GRUN File.

alone file. If, on the other hand, a file contains multiple sets of information the sets are separated by #setNumber. This is another file that is in a simple format but can be very prone to errors and should benefit extensively by the addition of the GUI.

Following the interpolation information on the setup line are the values for the overlay variables separated by commas. While overlay variables are not required, they provide a convenient way of parametrically changing a subset of variables for each trajectory. The overlay variables can also provide a convenient way to have the parameters that the user is studying on one line in the setup as an audit trail instead of having them hidden in several group files.

The final entries on the trajectory setup line are the group file aliases and file names. The structure of the group files is identical to the RUN file. While the limit to the number of variables in a group file is the same as that for the RUN file, the group file

provides another mechanism to organize related information and typically a file only contains a small subset of the RUN file. Each file is opened, moving left to right and the contents copied to the current RUN file. If the name specified is found in the alias section, the file to which it is aliased is opened; otherwise, the name is assumed to be that of a file and it is opened as specified. Every trajectory line must have at least one group file, but the number of group files for each trajectory is arbitrary. The only rule is that, for each trajectory defined, all the group files combined with the INTERP information must provide a value for every variable in the RIF and that a variable assigned a value only once for all group and INTERP files. This means that any overlay variables, in either the OVLY section or the FSRUN/GRUN files, must also appear in a group or INTERP file.

With the new file structure design and implementation several ideas surfaced as to how the GUI could be added. The benefit from breaking the design into two phases was greatly seen as the file design was completed. The task of designing the GUI became much simpler.

CHAPTER VI

IMPROVEMENT PHASE TWO - GRAPHICAL USER INTERFACE DEVELOPMENT

Phase two consisted of the taking the file structure that was developed in phase one and adding a Graphical User Interface (GUI) to it. By developing, implementing and testing the RunTGP program in the phase one process, a logical concept design of the GUI functionality had been defined. Initial design issues had to be resolved such as GUI development software, platform specifications, which high level language to develop code with and how to design the screens. It was decided that the GUI was to be more than just a means of inputting data. It leads the user through the input by using very intuitive and descriptive screen layouts so that the inexperienced user would have ample on-screen documentation. Error checking would be very important at every step in the input process in order to correct errors as early in the process as possible.

Choosing Software Package

The target platforms were determined to be the PC and UNIX workstation. The next step was to find GUI development software that would run on all the platforms and also be inexpensive. Cost was a big determining factor in the choice of software packages because funds were very limited and therefore did not allow for paying a large

sum of money for software. There was a requirement for no licensing fees or distribution restrictions on the software development package. The high level language the development package used was also important since it was undesirable to develop all the underlying code necessary to support graphical displays on all the platforms chosen for support. The developers wanted to use an object-oriented approach to the design and use C++.

The look and feel, or style, which guides the appearance and behavior of the interface was very important in the software choice. The look of the interface is what the user sees on the computer screen. This includes colors, buttons, menus, and the general appearance of the windows. The feel of an interface involves the actions a user takes to interact with what is seen on the screen to accomplish the desired function (DOD, 1984). The look and feel of the Microsoft® Office products was chosen since most of the users were familiar with those type packages and the familiarity would help the user feel at ease using the package.

There were several inexpensive packages that fit the software development criteria. The software package from Rogue Wave® called zApp Developer's Suite®, which is a visual builder, interface tester, and code-generation system for building GUIs that are portable to 10 platforms, was chosen as the development package. The Developer's Suite is based on and includes zApp, a C++ applications frame work. The zApp® interactive GUI builder called Factory® was purchased for the PC's and the executable libraries purchased for the PC's and the UNIX platforms. All interactive GUI

design and development work was developed on the PC and then transported to the UNIX workstations for testing.

Designing Screens

Designing the screens involved both the programmers and engineers working closely in design layout. The type of window controls had to be determined initially. Check buttons would be needed to allow a physical toggle switch. Activating the control button toggles the value of the state, but need not invoke any further action. Radio buttons would allow exclusive settings. These would be used when selecting from multiple options where only one can be selected at any given time. A push button or command button, which would be used to initiate an action when the button is released would be needed in the GUI. Text fields would be needed for the inputting of information. A title or label would be appended to the field to identify or describe the data that is to be entered (DOD, 1992). Drop-down menus would be used for displaying lists of information from which an item would be chosen by highlighting and double-clicking the mouse. Pop-up menus would be used for command options when needed.

The color scheme for all the screens had to be determined. The proper use of color, background, patterns and sound has the potential to aid the user significantly (DOD, 1992). Colors chosen for display on a screen must consider these factors: the human visual system, the possible problems that its use may cause, the contextual effects that may occur, the viewing environment in which the display is used, and the task of the

user. The primary objective in using color is communication, to aid the transfer of information from the screen to the user (Galitz, 1993). It was decided that color would not be a means of identifying functionality in the system. Color would only accent the screen and needed to be harmonious. Harmonious colors are those that work well together or meet without sharp contrast. Harmony is most easily achieved with a monochromatic palette (Galitz, 1993). The color scheme that was chosen for the GUI was a gray scale, sometimes called black-gray-white scale (Galitz, 1993). The five tonal values for a black-white scale are black, dark gray, medium gray, light gray, and white. Consistency in color usage had to exist within a screen, a set of screens, and the system (Galitz, 1993).

It had been determined early on that the screens had to be grouped according to functionality and help lead the user through the input process. There were several places that would need more description than just words and control options. It was decided that bitmaps would be used in places where only a picture could describe the function. Help screens would be built into the system along with descriptive layouts and text to help the user.

Design Principles

Several design principles were enforced when considering the layout of the screens in order to maintain a consistency throughout the entire program. Title bars appear with a short descriptive phrase indicating the purpose of the screen at the top of

each window. All screen menus, such as “File Save”, “File Print”, etc., will be displayed beneath the title bar at the top of the window. On screens where no file options are given, the “Cancel” and “OK” buttons will be located in the upper right corner of the window. All windows will appear in the center of the monitor and the push and pop features will behave as standard Windows® screens. The different types of input and buttons will carry the same features throughout the entire design. For example command buttons will all look the same as will input boxes, radio buttons, etc. no matter which screen they appear on. Any feature, which includes input boxes, command buttons, radio buttons, etc, that is disabled will be colored light gray and will be inactive.

In all windows the default values will be loaded for all variables that have defaults. Input of all required values that are not set to a default value will have to be input before the user can accept the screen. Bitmaps will be used on difficult input windows with labels appearing on the diagrams that correspond to input boxes that appear at consistent locations on the screen.

Single data input boxes will have a textual descriptive caption to the left of or above the data field. The type such as integer, real, or character, of input, will be determined at the time the input boxes are designed. Data type error checks will occur as soon as input is complete thereby ensuring that the wrong type of input is not stored. All error checks would occur as soon as the input can be checked. The error message will appear in a dialog box is displayed in the window and a beep will occur to alert the user of an error.

Program variables that use integer values as flags to represent an option will use drop-down menus to provide a list of descriptive terms for each flag setting. These menu options will be tied to an integer value that corresponds to a flag in the TGP software. This will allow the actual integer value to be hidden from the user. Since the user is only interested in what function is performed not the value. Drop-down menus will also be used to list variables such as data variable name, aircraft, and store names. These menus will be sorted alphabetically to aid the user in finding the desired selection and to allow the user to immediately jump to a word by typing the first letter. This will keep the user from having to scroll through a long list of variables when the variable being sought is already known. When choosing from a drop-down menu or from a list, the user would have to double click on the desired item in order to select it.

The edit list box will be used for input of an array of values. In all cases where the number of values input is important, the program will automatically calculate this for the user and display the value in a text box.

Layout of Screens

All screens were designed using Powerpoint® to do the screen layout initially. The engineers had extensive input into how the screens were to be designed by functionality. The functionality of the inputs discovered during phase one development was used to define the screens in phase two. Each screen was prototyped and checked by the engineering group. The prototype gave the engineer the opportunity to sit down with

the screen and observe the look and feel for correctness and double check the functional breakdown of the variables. The screens were designed so that the system could be used to build input for any of the three TGP's discussed in this study.

All the screens were built using the zApp Factory® software, which is an interactive window building software package. This package allows the programmer to design the screen using icons to interactively create the buttons and all controls. The programmer chooses what type of control is needed, such as control buttons, menus, radio buttons, etc. that will be used and then the user can interactively position it on the screen. The function that the user wants associated with each control object can be determined at this point such as user function, double click, etc. The menus can also be built this same way along with textual information and input boxes. The zApp Factory® will produce the main C++ code controlling the GUI and create the necessary include files and stub routines which can be used to execute the design without software modification. Stub routines are empty subroutines that do not have any function but are used as placeholders. This code will lack the functionality part of the software so the programmer has to go back and insert the code that will provide the interaction in the stub routines. Rogue Wave® has provided a mechanism in Factory® that provides the developer specified areas to add code modifications. If the user will add all modifications within these predefined locations then the software is smart enough to preserve them when later changes have been made to the interface using Factory®.

The main screen or window shown in Figure 14 allows the user to choose which aircraft, weapon station and store they are going to be studying. A catalog of aircraft and

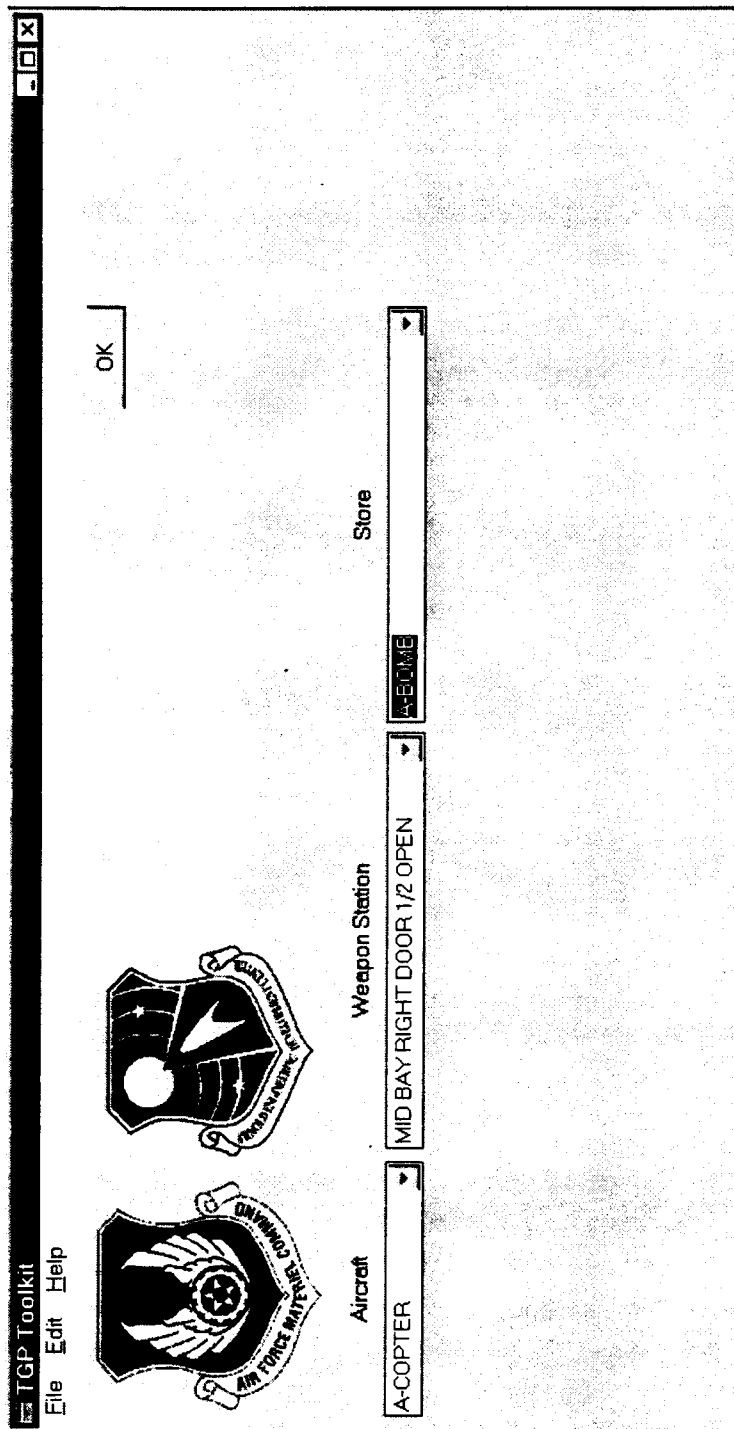


Figure 14. Sample Main Window.

store properties are accessed to give the user choices. The catalog is read and the menus built as each input box is entered. The aircraft chosen will determine what weapon station is available on that particular aircraft. The store menu list is built from the catalog directory. The aircraft chosen also has information that tells the GUI whether rotary (helicopter) or fixed wing (AutoTGP or Ballistics) windows are to be used. Each window contains a title bar that specifies the screen name, like TGP Toolkit. All of the windows are icon capable and movable by dragging.

If a helicopter is chosen, then the main window will call the appropriate windows for a rotary wing interface. The main rotary wing window is shown in Figure 15. This shows the different inputs that are needed to run the HassTGP. The Control Parameters button will bring up a control window that allows for control information to be input. The information in the list boxes were filled with information that was gathered from the input given at the main screen and retrieved from the catalog. The Modify buttons will execute a window that allows the user to change the value of any parameter in that group. A sample screen for modifying variables is shown in Figure 16. In this figure there is an input box that has a pull-down menu attached that has Damping in it. The data are broken into smaller groups within the large store properties group such as lanyard, damping coefficients, mass properties, etc. This menu will allow the user to jump to a certain group or the user can use the large scroll bar on the side of the table. Any variable that has Unknown as its default must be specified by the user before the user can click OK. A pop-up error box will appear if the user tries to proceed without having values for every variable. A sample of the control parameter window is shown in Figure 17. The

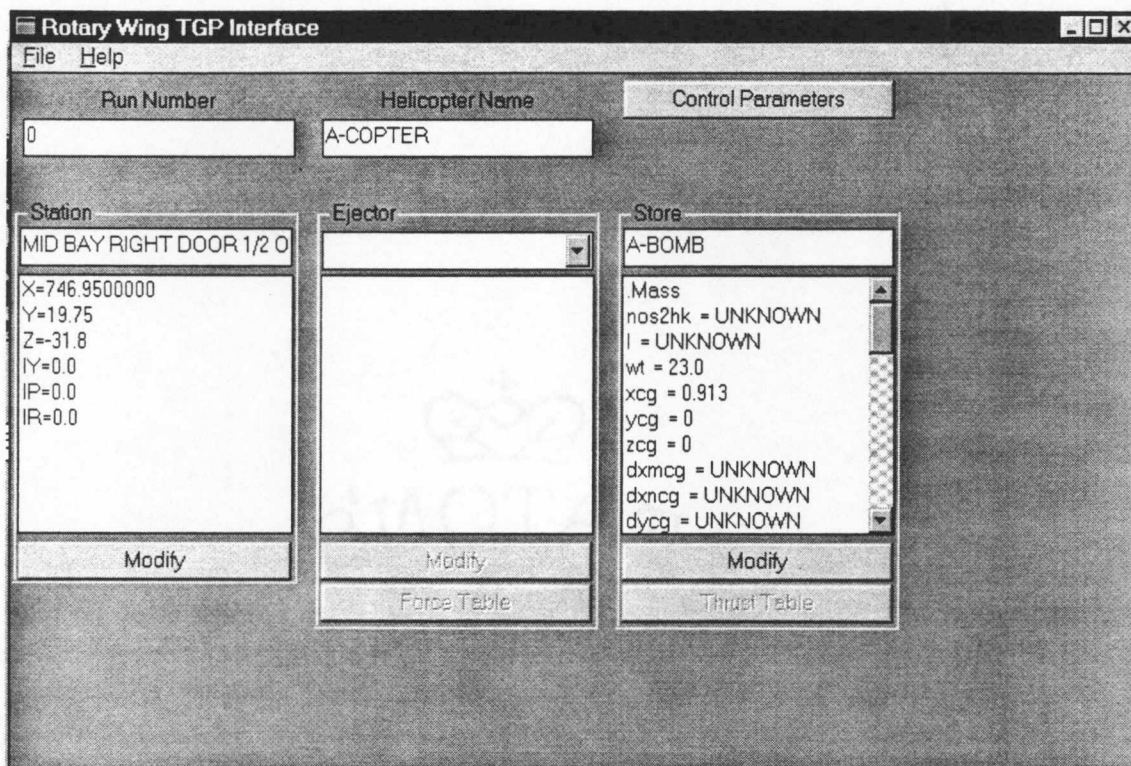


Figure 15. Sample Rotary Main Window.

Store Edit Window

Damping

Description	Default	User
.Damping		
clp	0.0	
cmq	0.0	
cnr	0.0	
clpfd	Unknown	1.23
cmqfd	Unknown	2.23
cnrfd	Unknown	1.12
.Lanyard		
rlen	Unknown	4.9
x1	Unknown	1.0

OK Cancel

Figure 16. Sample Modify Window.

Rotary Wing Control Parameters

Maneuver

Create Maneuver

Motion Constraint

Pivot Geometry

Postlaunch ☐ Thrust

Time (sec)

In maneuver to

Begin	End
1.0	3.0

In simulation to begin ejection

Integration Time Step

OK Cancel

Figure 17. Sample Rotary Control Parameter Window.

rotary and fixed wing options both use pivot geometry, rail geometry and postlaunch information so the windows were designed once and used for both applications. After the user has filled in all the information a RUN file is created which will be used by HassTGP.

The fixed wing interface starts with the same main window but the user has to choose a fixed wing aircraft to initiate the fixed wing portion of the software. The ballistics option has not been defined on these screens because of funding constraints. The screens have been designed but not yet implemented pending future funding. All screens shown for the fixed wing apply to BalTGP but more screens would have to be implemented for the additional ballistics data requirements. The first screen after the main window is the Fixed Wing TGP Interface Toolkit screen shown in Figure 18. This screen has the four main options as choices for execution. Each button represents a unique program, that is executed from this main window. An option to be considered for future enhancement will allow the user to add any miscellaneous variables that may be needed for test peculiar data. The file for each group will be built in the individual program or window depending on the connectivity of the data. As shown in Figure 19, if the window has a file button on it then the data need to be saved at that level. This is the first time the file option has become available because files are saved as group files and this is the first time a group level has been reached in this example. The program will not let the user leave without verifying the data save option. At this level the user can create a new file that is filled with default values, open and load an existing group file, save, or

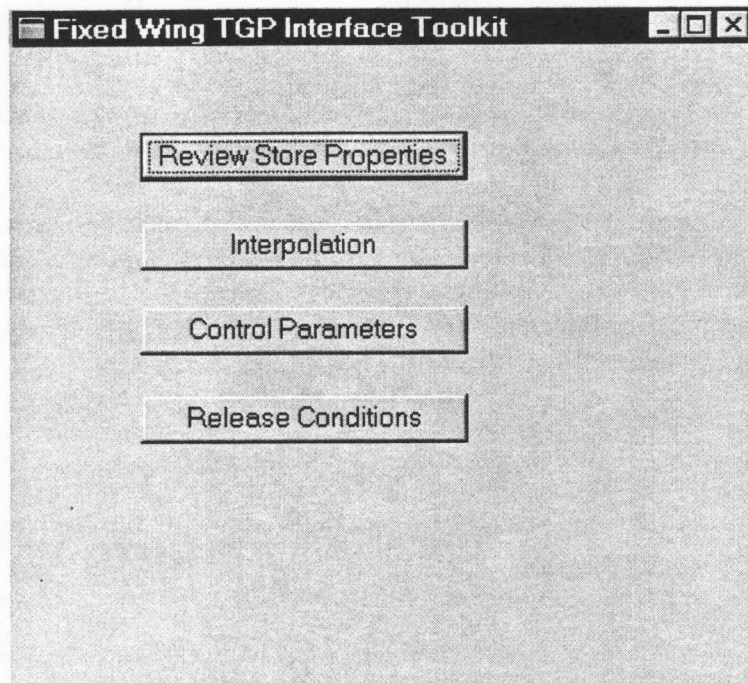


Figure 18. Sample Main Fixed Wing Window.

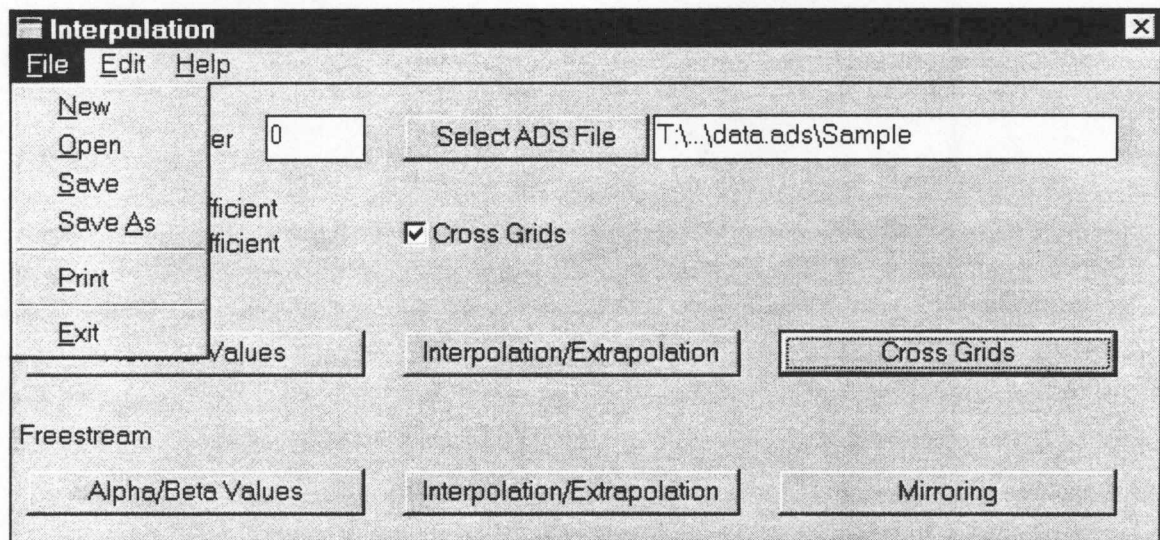


Figure 19. Sample File Options for Windows.

save as the data to a file with a name the user specifies, print the information, or exit the option.

The first option in Figure 18 is to review store properties, such as the rotary wing interface, which will be loaded from the catalog based on input supplied on the main window regarding the aircraft and store. A sample data set is shown in Figure 20. As with the rotary option, all unknowns require input values and the different small groups can be chosen in the edit/menu box. The only difference is that the user has to save this into a group file before exiting the window.

The second option facilitates the input of interpolation values used by MDITGP and BalTGP but are not required by rotary wing. This is one of the largest and most complicated groups that the system contains and has been further divided into subgroups of related interpolation values. While all the interpolation information is written to the same file, each screen specifies a small subset of data. In Figure 21 a sample of the interpolation main window is shown. The user has to specify an ADS data input file before going to any of the other options so that the data channel names can be used for input. The system will help walk the user through the appropriate order of input. If the user tries to go to an option before the needed information has been given, the user will get a warning telling them what has to be done first. For example, the cross grid option is initially grayed because the cross grid option has not been checked. In Figure 22 the nominal values window shows the default settings for the file. The user will always have to input the nominal values for the five variables since no standard set of inputs exist so no defaults have been supplied, however the inputs at the bottom have been filled with

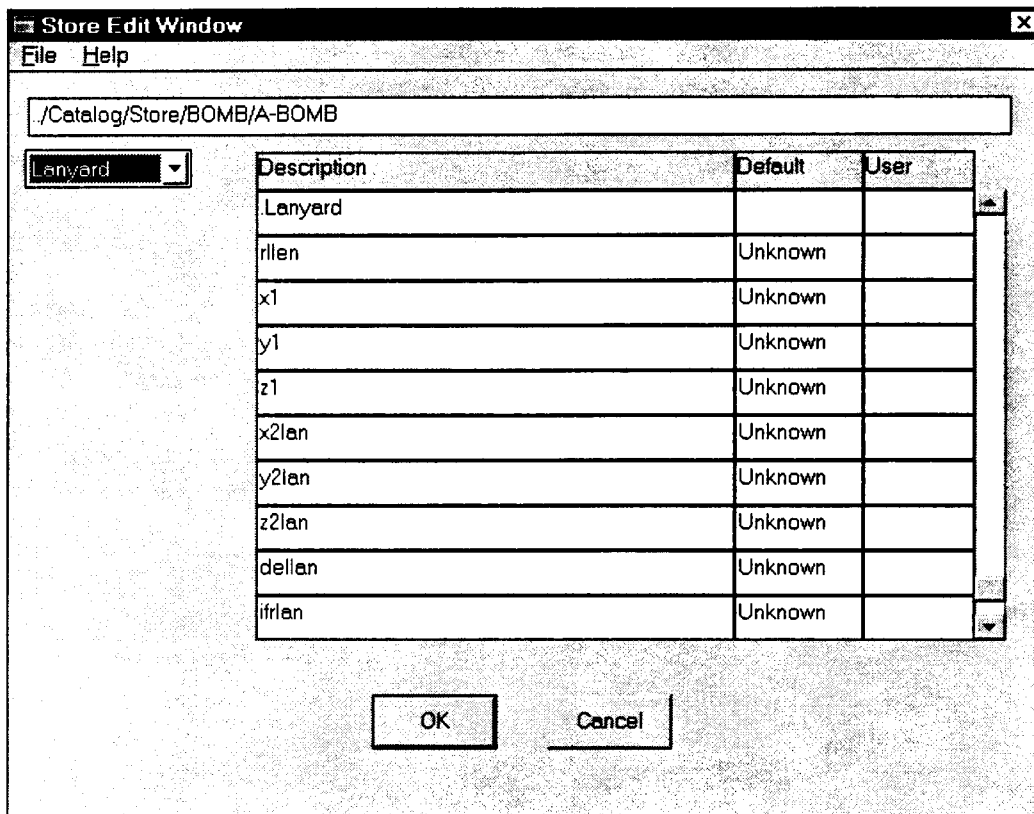


Figure 20. Sample Modify Store Properties Window.

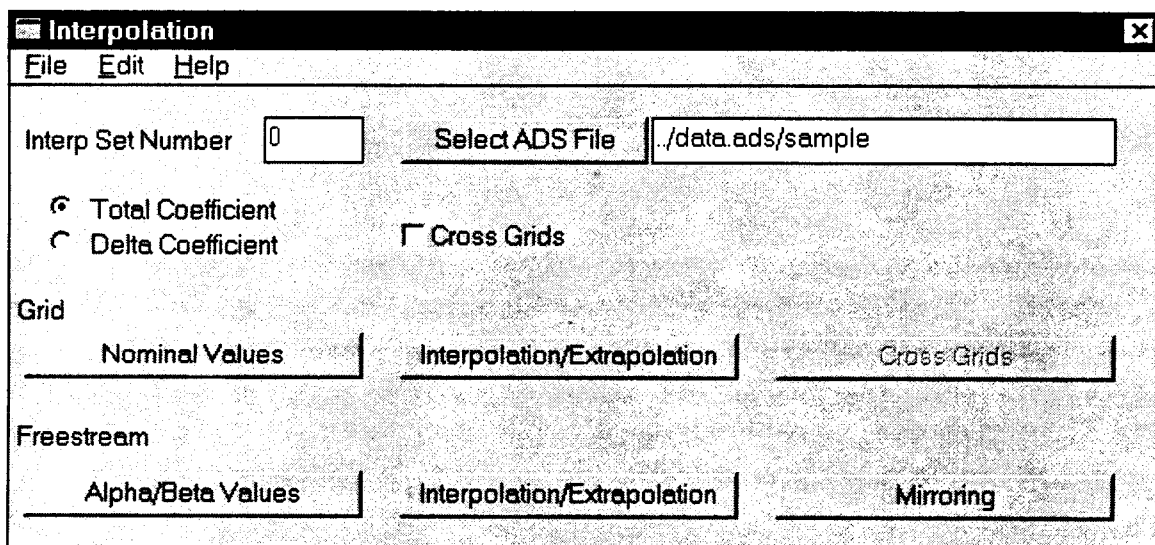


Figure 21. Sample Interpolation Window.

Nominal Values

OK

Cancel

Enter Nominal values excluding the primary variable

X'

☐ Primary

☐ Automatic

0

Y'

☐ Primary

☐ Automatic

0

Z'

☒ Primary

☐ Automatic

0

Yaw'

☐ Primary

☐ Automatic

0

Pitch'

☐ Primary

☐ Automatic

0

Roll'

☐ Primary

☐ Automatic

0

Tolerances

0.50

0.50

0.250

2.0

2.0

5.0

Axis System

XP

Pylon

Pylon

Reference

Aircraft

Absolute

Body

ZP

DPSI

DTHA

DPHI

Coefficients

Axial

Side

Normal

Yaw

Pitch

Roll

CA

CY

CN

CLN

CLM

CLL

Alpha

Beta

ALPHAS

BETAS

Figure 22. Sample Nominal Window.

the most commonly used values. The window is set up so that if the tab key is used the user will jump to the next input location and it will walk the user through the filling out of the form. The values for tolerances and the six independent parameters, six coefficients, alpha and beta would not have to be given if the defaults are sufficient. After the user inputs the nominal values, the program will automatically count the number of values input and set the count at the top of the column. In the old system the user was responsible for counting the variables and errors could occur easily. The axis system input box is tied to a pull-down menu that allows the user to choose from several pre-defined axis systems. These axis systems define the six independent variable names that are used to look up data in the ADS data file. When an axis system is chosen, the names of the six variables are loaded into the appropriate input boxes in the lower portion on the window. There is an option in the menu that will allow the user to specify the channels one input box at a time to handle those rare situations using a non-standard axis system. The user can type in the variable name or use the pull-down menu that contains all valid variable names from the input ADS file. This is another area in which the GUI has improved performance and reduce errors. The old system had the user input the number location of the data in the file. It was very easy to mistype the number; however, with the Toolkit and GUI the variable name is always used so the user can easily tell that the appropriate variable is chosen. The selection of the coefficients is handled the same as the axis system variables. As with the coefficients, alpha and beta can either be typed into the input box or chosen from the menu of names from the input data file. Figure 23 shows how the window looks after inputting several nominal values. It is seen that the

Nominal Values [X]

Enter Nominal values excluding the primary variable

OK Cancel

X'	Y'	Z'	'Yaw'	'Pitch'	'Roll'
☐ Primary	☐ Primary	☒ Primary	☐ Primary	☐ Primary	☐ Primary
☐ Automatic	☐ Automatic	☐ Automatic	☐ Automatic	☐ Automatic	☐ Automatic
5	3	0	0	0	0
-20 -10 0 10 20	-10 0 10		-20		
Tolerances					
0.50	0.50	0.250	2.0	2.0	5.0
Axis System: Pylon					
XP	YP	ZP	DPSI	DTHA	DPHI
Coefficients: Body					
Axial	Side	Normal	Yaw	Pitch	Roll
CA	CY	CN	CLN	CLM	CLL
Alpha		Beta			
ALPHAS		BETAS			

Figure 23. Sample Nominal Window with Input Values.

count has been updated in the box above the input column. These are light grayed to indicate that there is no required input.

The interpolation and extrapolation for the free-stream and grid data are very similar in design, therefore their parameters are presented in the same window. In Figure 24, a sample is shown for the grid data. For each parameter there are three choices for the type of interpolation and two choices for extrapolation. Each radio button is labeled and represents a value. In the old system the user had to give a value for each variable but the choices for that value were specified in a manual. This is a much needed improvement which simplifies the task for inexperienced users and practically eliminates the possibility of error.

The Alpha/Beta value window shown in Figure 25 depicts a design similar to the nominal values screen. The variables are selected by name using pull-down menus

	X	Y	Z	Yaw	Pitch	Roll
Interpolation						
None	☐	☐	☐	☐	☐	☐
Linear	☐	☐	☐	☐	☐	☐
Akima	☒	☒	☒	☒	☒	☒
Extrapolation						
None	☒	☒	☒	☒	☒	☒
Linear	☐	☐	☐	☐	☐	☐

Figure 24 . Sample Grid Interpolation/Extrapolation Window.

Alpha/Beta Values [X]

Alpha

Number of Alphas:

Tolerance:

AlphaS:

OK Cancel

Beta

Number of Betas:

Tolerance:

BetaS:

Beta Values

0
2
3
4

Axial:

Side:

Normal:

Yaw:

Pitch:

Roll:

Figure 25. Sample Alpha/Beta Window.

containing the data names from the ADS data file. The beta values are input into the column input box and the count is updated beside it.

If the user has checked the cross grid option then the cross grid control button will turn dark gray. The user can then choose the cross grid option which is shown in Figure 26. The nominal value information that was input in the nominal values window will be displayed from which the user selects the index to use in the cross process. The program sets the nominal index which is shown at the bottom of the screen for the parameters the user wants as cross variables. In the old system the user had to keep up with the index manually and this led to more errors and confusion. After all information has been input the user can save the interpolation information in a file on the main interpolation window. All variables in this group have a predefined value with the exception of the nominal and

The screenshot shows a window titled "Cross Grids" with a standard Windows interface including "OK" and "Cancel" buttons. The window is divided into two main sections: "Interpolation Method" and "Nominal Values".

Interpolation Method: This section contains six columns, each with a radio button labeled "Primary" and a selection box. The columns are labeled "X", "Y", "Z", "Yaw", "Pitch", and "Roll". The "X" column has "Normal" selected, while the others have "Cross" selected. Below the selection boxes are input fields for the nominal index, with values 0, 2, 0, 0, 0, and 0 respectively.

Nominal Values: This section contains six columns, each with a list of values. The columns are labeled "X", "Y", "Z", "Yaw", "Pitch", and "Roll". The "X" column has values 0.000000, 10.000000, -10.000000, 20.000000, and -20.000000. The "Y" column has values 0.000000, 10.000000, and -10.000000. The "Z" column is empty. The "Yaw" column has values 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, and 0.000000. The "Pitch" column has values 0.000000, 5.000000, and -5.000000. The "Roll" column has values 10.000000 and 5.000000.

Interpolation Method	X	Y	Z	Yaw	Pitch	Roll
Primary	☒	☐	☒	☐	☐	☐
Method	Normal	Cross	Normal	Normal	Normal	Normal
Nominal Values	0.000000 10.000000 -10.000000 20.000000 -20.000000	0.000000 10.000000 -10.000000		0.000000 0.000000 0.000000 0.000000 0.000000 0.000000	0.000000 5.000000 -5.000000	10.000000 5.000000
Nominal Index	0	2	0	0	0	0

Figure 26. Sample Cross Grid Window.

the alpha/beta values. The user could accept all the defaults in the system and only have to enter information on two screens to build a valid interpolation file.

The third option, after store properties and interpolation information, is the input of control parameter information. The control parameter window has several drop-down menu input boxes as shown in Figure 27 with the coefficient offset option selected. The drop-down menus provide a list of legal options for each of the parameters. This is another example of parameters in the old system that used to be defined only with a number now being selected by name in the new system. When an option has been selected in any of the input boxes a variable is set and the appropriate command option buttons are enabled. This leads the user through the input process by showing what must to be done at each step. All the variables in this window are flags that control different TGP program options so they are all displayed on this one screen. However, the options are functionally grouped into Ejector, Post Launch, etc., and all the variables associated with each group are stored in a group file named after the sets.

For an example of the control parameter command option buttons, the ejector force using equations will be used. In Figure 28 the ejector force input variables are shown. Also bitmaps are used to explain the inputs more explicitly. The pictorial representation of the input data adds much needed explanation for the user because the user is able to tell visually what is being set. Most of the options under the control parameter window have bitmaps in the windows for pictorial explanations of the inputs. Another example of this is the rail geometry as shown in Figure 29.

TGP Control Parameters [X]

File Edit Help

Ejector

Force and cutoff by stroke

Forces

☒ Equation ☐ Geometry

☐ Table

Post Launch

Inertial axis position/orientation

Initial Conditions

Coefficient Offset

Constant values, All coefficients

None

Constant values, All coefficients

Ramp axial force coefficients

Test peculiar

Time Delay

Forces

☒ Equation ☐ Table

Motion Constraint

Pivot, Pitch and Yaw

Pivot Geometry ☐ Rail Geometry ☐

Autopilot

☐ Autopilot

Autopilot Inputs

☐ Output to plot file

Integration

Time step .0001

Integrations / print 4

Maximum time 0.50

Printout

Grid Set

☐ First ☐ Second

☒ Group Inputs

Freestream Set

☐ First ☐ Second

☒ Summary Pages

Figure 27. Sample Control Parameters Window.

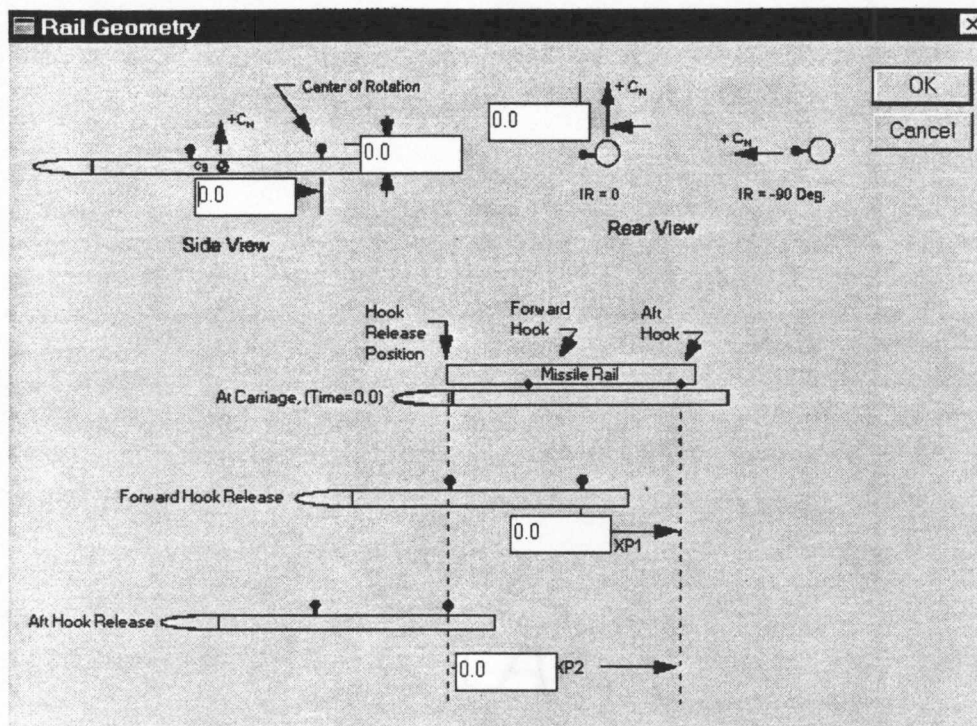


Figure 29. Sample Rail Geometry Window.

The last option in the current fixed wing program is the release conditions. In Figure 30 an example is shown. This window contains several input boxes and check boxes. The input data are very straight forward and will be stored as one group file.

The screenshot shows a window titled "Aircraft Flight Conditions" with a menu bar containing "File", "Edit", and "Help". The window contains several input fields and checkboxes arranged in a grid-like fashion.

Run Number	Configuration	Store Number
234	45	246

Mach	☐ Interpolation	Altitude	Load Factor
0.90		10000.0	10.0

Angle of Attack	☐ Interpolation	Dive Angle	Bank Angle
45.0		45.0	20.0

Sideslip	☐ Interpolation
10.0	

Figure 30. Sample Release Conditions Window.

Benefits

There are several benefits of the GUI being used for the input of TGP information. There is a decline in the number of typographical errors, numerical code option errors, the number of input parameters being counted improperly and other simple errors. The more complex errors such as logical or functional errors are greatly reduced since the GUI helps define a set path for the user. The system also guarantees that erroneous information is not entered. The GUI ensures the integrity of the input data by doing data validation at various points throughout the interface and provides for immediate notification of an input error so that it can be corrected as soon as possible. In the areas where more descriptive terms and pictures are used, the user is less likely to misunderstand what the parameters represent. This will greatly benefit new and inexperienced users as well as the more experienced users. By using the GUI it should make the building of input files more intuitive and less of a struggle. The amount of time the user spends doing setup for trajectory generation should be shortened. This time can now be spent studying the engineering problem rather than building input files.

CHAPTER VII

SUMMARY

Off-line trajectory generation provides a valuable tool to predict store trajectories analytically and to supplement existing experimental data. This reduces the time and costs of store certification efforts in new weapon system design. Setting up the inputs for the off-line trajectory generation programs is very time-consuming. The turnaround time, ease of use, portability and reduction of errors have greatly been improved with the development of the GUI. This cuts the cost and time of off-line trajectory generation drastically and in turn reduces the time and cost of the store certification of new weapons system designs.

The system has been used extensively in the helicopter trajectory generation process and is used at AEDC and at Army and Navy Installations. The improvement in setup time has greatly improved their prediction capability to the point that with the new GUI the customers are able to do more analysis work which helps contain the cost of certification. This increase in production is due to the GUI design that minimized human learning time and the error checking which led to higher productivity.

The fixed wing section is using the GUI in a beta testing scenario at this time. There are several options that need to be completed before it will be a fully integrated package for the AutoTGP users. Even without these options, however, the GUI provides a valuable tool. With the defined defaults that the GUI provides for each group, the user

can set up basic cases with minimal effort. The interpolation information windows are an area where considerable improvements are noted in variable inputs and error detection. The use of bitmap graphics gives the user a better understanding of what inputs are expected and a more intuitive understanding of how they are used. When the system is in full production the increase in productivity should be very high.

Once all three TGP's, MDITGP, HassTGP, and BalTGP, are using the same user interface the need for cross-training between the analysis engineers will be decreased. It will allow engineers that are typically working in one area to move to another area as needed. If the engineer has a basic understanding of the physical problem, the GUI will help the engineer go through the paths that are different with little or no additional human guidance.

Future Development

The setup input building process is slated to be completed.. There are several other enhancements that are currently being studied. The ballistics inputs need to be added to the system as soon as funding becomes available. There are several tools that could easily be added to the system. By using a consistent graphical interface on several of our tools, the user could learn the tools at a much faster rate. These tools would supplement other aspects of the trajectory generation process. There is a sort program that would build the user GRUN and FSRUN files automatically from the data using

several independent parameters that are specified by the user. These options would add value to the process.

A deficiency of the system has already been identified and plans are being made to rectify the problem which is the use of characters in the RUN file. The use of only real numbers is not a problem if an integer is needed but character values are a whole new subject. An addition of hot keys would allow the more experienced to bypass the walk-through method of input. As the system is used more, changes will be made that will help make the system more intuitive and user-friendly.

BIBLIOGRAPHY

BIBLIOGRAPHY

AEDC. Test Highlights, Spring/Summer 1994.

Apple, Inc. Human Interface Guidelines: The Apple Desktop Interface. Addison-Wesley Publishing Company, Inc., 1983.

Carman, J.B., Jr. "An Overview", Store Separation Testing Techniques at the Arnold Engineering Development Center, Volume I. ARO, Inc., June 1980, AEDC-TR-79-1.

Carman, J.B., Jr., Hill, D. W., Jr. and Christopher, J.P. "Description of Captive Trajectory Store Separation Testing in the Aerodynamic Wind Tunnel (4T)", Store Separation Testing Techniques at the Arnold Engineering Development Center, Volume II. ARO, Inc., June 1980, AEDC-TR-79-1.

DOD - Defense Information Systems Agency. Human Computer Interface Style Guide, Version 1.0. Center for Information Management, February 12, 1992.

Galitz, Wilbert O. User-Interface Screen Design. QED Publishing Group, 1993.

Keen, K. Scott. New Approaches to Computational Aircraft/Store Weapons Integration, Arnold Engineering Development Center. Calspan Corporation, January 1989, AIAA-90-0274.

Rogue Wave Software, Inc. 850 S.W. 35th Street, Corvallis, OR. 97333 U.S.A.

VITA

Mary Louise Craig was born in Shelbyville, Tennessee on February 14, 1963. She attended schools in the public system of Bedford County, where she graduated from Cascade School in May, 1981. She entered Middle Tennessee State University, Murfreesboro, Tennessee during August of 1981 where in May 1985 she received the Bachelor of Science in Mathematics. That same summer she entered work with ITT Avionics as a computer programmer doing navy and radio simulations. In February, 1986 she became employed by OAO at Arnold Engineering Development Center in Tullahoma, Tennessee. In August of 1994, she continued her education by entering the Master's program in Engineering Science at The University of Tennessee Space Institute in Tullahoma. The Masters Degree was granted December 1997.

She is presently employed as a Senior Programmer Analyst with Sverdrup Technology at Arnold Engineering Development Center in Tullahoma.