C-17/PARATROOPER RISK ASSESSMENT ANALYSIS

THESIS

Jose C. Belano III, Captain, USAF

AFIT/GOR/ENS/97M-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/97M-01

C-17/PARATROOPER RISK ASSESSMENT ANALYSIS

THESIS

Jose C. Belano III, Captain, USAF

AFIT/GOR/ENS/97M-01

AFIT/GOR/ENS/97M-01

C-17/PARATROOPER RISK ASSESSMENT ANALYSIS

THESIS

Presented to the Faculty of the Graduate School of
Engineering
Air Education and Training Command
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Jose C. Belano III, B.S. Operations Research
Captain, USAF

March 1997

AFIT/GOR/ENS/97M-01

THESIS APPROVAL

Student: Jose C. Belano III, Captain, USAF Class: GOR-97M

Title: C-17/Paratrooper Risk Assessment Analysis

Defense Date: 4 March 1997

Committee:	Name/Title/Department	Signature
Advisor	T. Glenn Bailey, Lieutenant Colonel, USAF Assistant Professor Department of Operational Sciences	
Reader	Jack M. Kloeber Jr., Lieutenant Colonel, USA Assistant Professor Department of Operational Sciences	

ACKNOWLEDGMENTS

First and foremost, I give all praise and honor to God who, in His great providence and mercy, has given me grace and has shown me my reason for being here. Salvation and eternal life with Him can only be realized through His Son, Jesus Christ without whom I would be nothing and apart from whom I could do nothing. It took me 25 years to come to this, and I am thankful that God whispered out my name to raise me up again. All that I have to offer that is worth anything to anyone is the truth of God's love for us made manifest through Jesus.

Working on this thesis effort with Lieutenant Colonel Glenn Bailey as my advisor has made the experience more than just an academic exercise. It has been a growth process for me in terms of being an officer and an analyst. His guidance has been extremely invaluable, and his example as a true Air Force officer will always be with me. It is few and far between to come across a senior Air Force officer willing to impart professional military growth experiences from which junior officers may learn, especially in an academic environment where the military aspect is not stressed. I am very fortunate to have come across Lt. Col. Bailey.

Another officer who holds the same standards is Lieutenant Colonel Jack Kloeber, US Army. LTC Kloeber has been an integral member of this effort contributing valuable insight into the early stages of the modeling and providing the Army perspective of the world. His capacity on this committee has greatly enhanced the communication of important information that needed to be conveyed.

Many people have contributed to this thesis effort and have endured my consistent inquiries into subjects I know little of. First there is Richard Benney of Army Natick Research Development Engineering Center whose guidance and advice got me up to speed in the world of aerodynamic decelerator technology modeling. Mark Kuntavanish of the C-17 System Program office has been a great help in providing the needed support to accomplish this endeavor. Lieutenant Colonel William Klimack has been a welcome gold-mine of information about Army Airborne issues, always making the time to help me in the post-processing portion of this thesis. Lastly, I want to thank Major Hans Petry for his partnership throughout this effort. I am fortunate to have his talents and intelligence working on a very closely related thesis effort that practically joined us at the hips.

There are many people who work behind the scenes in thankless positions, but whose efforts are essential to the functionality of this institution. In ENS, there is Jackie Logan, Kim Grandison, and Joel Coons. Jackie and Kim keeps the endless paper work going in support the theses efforts. Joel keeps the computers running in the Modeling and Simulation Center, a job which places him in constant contact with the confusion of those of us who are less computer literate.

Finally, I wish to thank my friends who have kept me motivated when I needed it the most and basically becoming my family here. Brutus, Fess, Heathro, Dan'o, Neal, Christine, Sonia, Angie, Matt—thank you for your support and your friendship.

Joey C. Belano

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The C-17 test and evaluation community has been testing different aircraft formation geometries in search of a configuration which minimizes paratrooper encounter with the wake vortices of upstream aircraft. This thesis develops a simulation tool that the C-17 test and evaluation community can utilize as an advanced risk assessment model to use on proposed formation geometries prior to live testing. The model is developed under the architecture of object-oriented simulation using MODSIM III and parallels similar efforts by the Aerodynamic Decelerator Technology community in creating object-oriented counterparts to already developed trajectory models of various degrees of freedom.

This thesis develops the paratrooper object portion of the simulation model while the Petry thesis (1997) develops the C-17 aircraft and vortex objects. Once integrated with the Petry C-17 aircraft and vortex objects, and after verification and validation, the simulation model is applied to a simplified airborne operation scenario using the mean distance of paratrooper impact location to assembly areas and DZ dispersal distribution as MOEs for different aircraft formation geometries. Lateral separation is shown to have the most influence on both MOEs, while trail distance has minimal effects. For the airborne commander, this translates into operational parameters applicable to the choice of assembly areas and formation geometries. Further operational parameters of any significance are gained when coupled with the results from Petry on encounter rates between paratroopers and wake vortices where trail distance has a significant impact.

C-17/PARATROOPER RISK ASSESSMENT ANALYSIS

1. INTRODUCTION

This thesis' purpose is to provide the C-17 test and evaluation community with the capability to assess paratrooper performance during C-17 drop formations in both training and combat environments. Paratrooper performance takes into account the risk of wake vortex encounter and ground dispersal patterns on the drop zone (DZ). Object-oriented modeling is used to convert current static/deterministic parachute/payload system trajectory models of any degree of freedom into dynamic/stochastic models through the development of a class of parachute/payload system objects which are expandable to model not only personnel but equipment and different types of parachutes. The immediate impact of this thesis is assessing the risk of C-17 formations for brigade-size personnel airborne operations. However, the parachute/payload system objects can be expanded for use in a combat modeling environment.

1.1 Background

Paratrooper interaction with wake vortices has been studied by the US Air Force since 1987 conducting the first tests to determine the effects of aircraft wake vortices on parachute/payload systems using static-line deployed parachutes which included personnel and equipment (Johnson 1988a:vii). These first tests investigate the effects of C-130 and C-141 aircraft on parachute/payload systems, while follow-on tests investigate the effects from C-5 wake vortices. Conclusions from both studies show that the aircraft generate significant wake vortices in size, velocity (strength) , and duration. Although the wake vortices dissipate with time, there still exists the danger of encounter with parachute/payload systems in formation air drops. When interacting with a parachute/payload system, these wake vortices can induce potentially hazardous conditions to include parachute collapse, partial deflation, severe oscillation, increased rate of descent, collision, entanglement, and hard landings (Johnson 1988b:13-14). Recommendations of both studies indicate the need to develop and evaluate formation tactics for large scale airborne drop activities, along with safety guidelines to avoid conditions conducive to paratrooper/wake vortex encounter and interaction. The C-5 study also found a need to begin study on C-17 wake vortex characteristics and parameters.

With the advent of the C-17 phasing into operational roles (as a C-141 replacement) it has encountered several operational problems, one being the problem of paratrooper/wake vortex encounter when used as a jump platform for US Army Airborne units. During operational testing of airborne activities with the C-17 in June 1995, a paratrooper/wake vortex encounter prompted the re-examination of formation tactic geometry (Blake 1996:1). Due to the size, weight, and the aerodynamics of the C-17, the wake vortex strength is greater than those previously encountered with the C-130 or the C-141 aircraft (Natick 1996c:1). As with the previous studies' recommendations, these wake vortices are key considerations in developing and evaluating formation tactics for mass airborne airdrop activities involving paratroopers and their equipment.

In developing formation tactics, Air Force and Army planners decided that the following conditions must be met: (1) total wake vortex avoidance must be assured where the wake vortex strength presents an unacceptable risk to the paratrooper; and, (2) if wake vortex avoidance cannot be assured, the encounter must not take place before the wake has decayed to the point where risks associated with wake vortex strength and probability of encounter are acceptable (Natick 1996c:1). In the summer of 1996, a series of tests were conducted at the drop zones located at Edwards AFB, CA (EAFB) in an effort to find the probabilities associated with wake vortex encounter by flying in formations most conducive to paratrooper/wake vortex interaction. A total of 57 test passes were made with a total of 672 dummies dropped from both C-141 and C-17 aircraft. The results of these tests were carried over in the development and evaluation of formation tactics at Fort Bragg, NC in late summer of 1996. Seeking to avoid paratrooper/wake vortex encounter, these tests used the only working model that predicts paratrooper/wake vortex encounters specific to the C-17 (Blake 1996). The formation tactics still create the conditions which induced paratrooper/wake vortex interaction, extending the testing of new formation tactics until the conditions set forth by the Air Force and Army planner can be met.

1.2 Problem Statement

Costs of test and evaluation precludes test of new formation tactics, therefore the need for a modeling tool which can assess the probability of encounter between paratroopers and wake vortices becomes essential. The use of object-oriented simulation in modeling airborne airdrop activities can predict the probability of paratrooper/wake vortex encounter under different formation tactic geometries. Two aspects are involved in the simulation of the paratrooper/wake vortex interaction system: (1) modeling the wake vortices specific to the C-17; and, (2) modeling a parachute/payload system's trajectory from exit to impact.

1.3 Organization of Research

Research of work already accomplished can be generalized into three categories: (1) research and development of parachute/payload systems for use in space related applications (i.e., interplanetary vehicle atmospheric-entry deceleration, space launch rocket booster recovery); (2) pure complete fluid dynamics of parachute/payload systems' behavior studies; and, (3) research and development of parachute/payload systems for airdrop applications. Due to the wide range of behavioral characteristics modeled in the three categories, the research for this thesis effort concentrates mainly on the last two categories.

The modeling of the parachute/payload system and its surrounding equations of motion when influenced by gravitational and aerodynamic forces has been tackled by engineers for years. The wide array of approaches differ in levels of approximation, numbers of degrees of freedom, and the numbers of independent bodies in the system (Maydew 1991: 121). The primary research focuses on the areas of parachute/payload system dynamics and parachute trajectory models, and searches for models with various degrees of freedom to achieve a specific degree of accuracy. Secondary objectives include finding studies on the interaction of parachute/payload systems and aircraft wake vortices, and mathematical models defining the dynamics of the parachute/payload system in all stages of descent.

The scope of this thesis effort revolves around modeling a class of parachute/payload systems' trajectory from aircraft exit to impact using MODSIM III, an object-oriented simulation language. This thesis effort parallels research efforts in the aeronautical community as evidenced by object-oriented simulation of parachute trajectories emerging at the forefront of a new wave of modeling techniques at the 14th Annual AIAA Aerodynamic Decelerator Systems Technology Conference for 1997. A defining characteristic of this thesis effort is converting well-defined static/deterministic models into dynamic/stochastic objects to provide a better method of assessing risk of parachute/payload system/wake vortex encounter, and assessing ground dispersal patterns on the drop zone using different C-17 airdrop formation geometries.

The main premise driving this thesis effort is that there are stochastic aspects which play an integral part of the parachute/payload system trajectory from the time the system exits the aircraft through open-canopy descent propagating all the way to impact dispersed around a target drop point. To simplify initial development of the parachute/payload system object, the system is assumed to be a point-mass system with two degrees-of-freedom (2-DOF). After successful completion of the 2-DOF point-mass object, a rigid, single-body system using higher DOF replaces the 2-DOF model creating a 6-DOF model. A rigid, single-body system assumes that the parachute axis of symmetry remains aligned with the longitudinal axis of the payload (Maydew 1991:122).

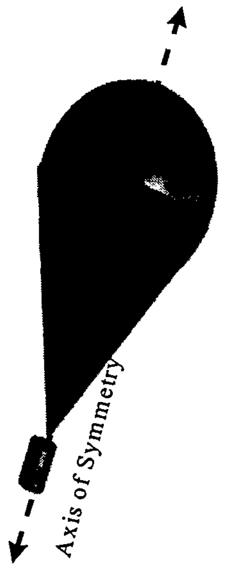

Figure 1. Rigid, Single-Body System

A further assumption is that the payload is unable to steer the parachute system. The end product will be a paratrooper object with 6-DOF modeled by an unconscious rigid, single-body system.

1.4 Thesis Objectives

MODSIM III, an object-oriented simulation language, provides the architecture with which to model the parachute/payload system. Chapter II provides the information necessary to model a parachute/payload system, and introduces trajectory models of various DOF. Chapter III provides in detail the modeling of the parachute/payload system trajectory using MODSIM III.

The simulation itself is developed in three phases. The first phase develops a single parachute/payload system object in MODSIM III from an already validated and academically accepted deterministic model. The second phase modifies the parachute/payload system object into a paratrooper object, and validates the results with the US Army Natick Research, Development, and Engineering Center. The final phase integrates the paratrooper objects with the Petry (1997) C-17 aircraft and wake-vortex objects, using both discrete and continuous simulation, to simulate mass airborne drop formations; and, measures dispersal patterns, with no interaction from wake vortices, using conventional X/Y coordinates.

Chapter IV presents the results of the simulation when used with a simplified airborne drop scenario: (1) the mean distance from company assembly areas; and, (2) the dispersion distribution of the paratrooper object across the width and down the length of the DZ. Chapter V summarizes the thesis effort and suggests follow-on efforts in expanding the capability of the object-oriented simulation.

2. LITERATURE REVIEW

The literature review covers two principal areas of research interest--the fluid dynamics of parachute-payload systems' behavior, and the development and modeling of simulations of parachute-payload systems' trajectory during descent. However, an understanding of the standard operating procedures for airborne operations is fundamental to the understanding of the need for reliable paratrooper trajectory modeling.

2.1 Airborne Operations

The US Army airborne commander and staff employs a form of "top-down analysis," referred to as the backward planning sequence, in planning successful airborne operations (82nd ABN DIV ASOP 1985:2-1). The commander follows basic joint airborne operations planning principles to allow for maximal effectiveness in the execution of the airborne operation. The airborne commander has a wide view of the operations, taking into account the limitations of the airborne units, the appropriateness of the mission for airborne forces, and the responsibilities of the joint task force (JTF) or theater commander.

There are four basic plans in the backward planning sequence, all focused toward the most effective way of acquiring the objectives (82nd ABN DIV ASOP 1985:2-6). Each plan supports the one preceding it, as the planning process backs out from the objectives. The four basic plans are:

- Ground Tactical Plan.

- Landing Plan

- Air Movement Plan

- Marshaling Plan

In the Ground Tactical Plan, the conduct of operations in the objective area is carefully orchestrated with the specifics of how airborne forces are to maneuver on the ground. The Landing Plan directly supports the Ground Tactical Plan. The main focus of the Landing Plan concentrates on the dispersion of the airborne units in the DZ in such a way as to be effectively positioned to move efficiently into their area of responsibility (AOR) as designated in the Ground Tactical Plan. The Air Movement Plan encompasses the period of time from aircraft loading to arrival of airborne units in the objective area. Specific flight routes are assigned with the order of flight and formation geometry in the area around the DZ chosen that will best support the Landing Plan. Incorporated into the Air Movement Plan are specific arrival times over the DZ. The Marshaling Plan completes the inverse planning sequence. The Marshaling Plan sees airborne units assemble and complete their final preparations for combat, and coordinates the airborne units' movement to departure airfields. Also, in the Marshaling Plan airborne units are cross-loaded onto the aircraft such that the Landing Plan is best supported given the Air Movement Plan.

2.2 Airdrop Studies

In developing airdrop models, particular interest in the backward planning sequence focuses around the Landing Plan and the Air Movement Plan. The Landing Plan concerns itself, in part, with the dispersion of airborne units on the DZ, and the Air Movement Plan concerns itself with formation geometry. A fundamentally sound method of modeling the movement of paratroopers along their trajectories will give an accurate estimate of paratrooper dispersion along the DZ. Dispersion is a function, in part, of aircraft formation geometry during airdrop.

Martin (1978) addresses the issue of missed distance errors experienced during Canadian Forces Paradrop Exercises. Martin develops a simple model which meets the following self-imposed requirements - the model reasonably describes the activity in question; the parameters affecting airdrop performance are correctly varied over the effective ranges; the range of possible outcomes is not so wide as to impair the usefulness of simulation; and, the model is verified, if possible, using empirical data (Martin 1978:223). Martin, instead of using a parachute-payload trajectory model, uses a commonly used approximation method called the Computed Air Release Point (CARP) system of release to model the impact point of the parachute-payload system. The simulation involved varying those input parameters that are considered the most important: winds, position at release point, and ballistics or hesitation error used in CARP calculations. Martin, however, does not consider formation geometry since the simulation focused only on individual parachute-payload systems and not on mass formations.

Further studies done on airdrop operations were done by Johnson (1988) which focused more on vortex encounter, rather than on DZ dispersion, with the C-130, C-141 and C-5 aircraft. Blake (1996) went further into paratrooper/wake vortex encounters in an effort to predict the relative locations of paratroopers and wake vortices of the C-17 aircraft during formation airdrops. The Blake Model can be modified to take into account a specific formation geometry; however, it does not attempt to predict dispersion on the DZ.

2.3 Dynamics of a Parachute-Payload System

Basic deceleration system equations can be derived starting with Newton's Second Law and building upon its fundamental premise. In order to understand the equations behind parachute-payload system trajectory models, a basic understanding of the equations that govern trajectory dynamics is needed. Seaman's (1975) memorandum develops the fundamental equations needed to understand the dynamics of a decelerating system such as a parachute-payload system. Seaman breaks down the components that act upon a basic decelerating system to mathematically model its trajectory. Knacke (1992) confirms Seaman's derivation process by developing the basic system of equations describing trajectory dynamics.

In describing a more specific application of a parachute-payload system, more equations are added to the basic system of equations to model characteristics associated with the application. When parachute-payload systems are released from aircraft, new factors (or degrees of freedom) help describe the system's trajectory, such as translation and rotation (Jones 1987:538). Jones builds on both the rigid single-body and rigid two-body models to describe the oscillation effects which translation and rotation causes. For an even more specific application, Heinrich, Noreen, and Saari (1973) describe the characteristics of a parachute-payload system using a static-line deployed T-10 personnel parachute.

The modeling of a parachute-payload system trajectory can be expanded with the hierarchical break down of the components that make up the parachute-payload system. The trajectory that a parachute-payload system follows can be broken down into discrete stages. The stages are payload free flight (before the parachute is deployed); parachute deployment; parachute inflation; system deceleration and turnover; and, steady-state descent. Within each stage, different analytical techniques can be used; however, Maydew (1991) shows that trajectory dynamics is applicable throughout all phases (Table 1). Maydew defines trajectory dynamics as the analytical solution of a set of time-dependent differential equations that describe the trajectory of a system. Maydew lists several references that discuss the equations of motion, the selection of an axis system, and simplifying assumptions. He furthers lists several other references that have made

Table 1
Modeling the Stages of Parachute/Payload System Flight (Maydew 1991:121)

	STAGES				
Modeling Technique	Payload Free Flight	Decelerator Deployment	Inflation	System Deceleration and Turnover	System Descent
Trajectory Dynamics	•	•	•	•	•
Elastic Structure Dynamics		•	•		
Decelerator Steady-State Aerodynamics		•	•	•	•
Decelerator Unsteady Aerodynamics			•	•	

significant contributions in the development of parachute-payload system trajectory models.

Maydew also presents several models that describe the system state of a parachute-payload system's trajectory. The most basic trajectory model is that of a point mass system in which all forces act upon a central point in the system in the vertical plane. This is considered a 2-DOF system (Figure 2) because it contains only horizontal and vertical components to describe its motion. The next level in trajectory modeling is a 3-DOF system (Figure 3) which adds an oscillation component to the 2-DOF system of equations, although it is still limited to the vertical plane. Within the family of 3-DOF models, Maydew shows that the system itself can be modeled several ways: massless, rigid single-body, rigid two-body, and elastic. A massless decelerator model maintains the alignment of the decelerator along the payload velocity vector, and the only force that the decelerator produces is drag. A rigid single-body system is one where the parachute axis of symmetry remains aligned with the payload's longitudinal axis; and, the forces produced by the decelerator are drag, lift, and a moment in which all are assumed to act in the payload's center of gravity. The rigid two-body system is a point mass pinned to a

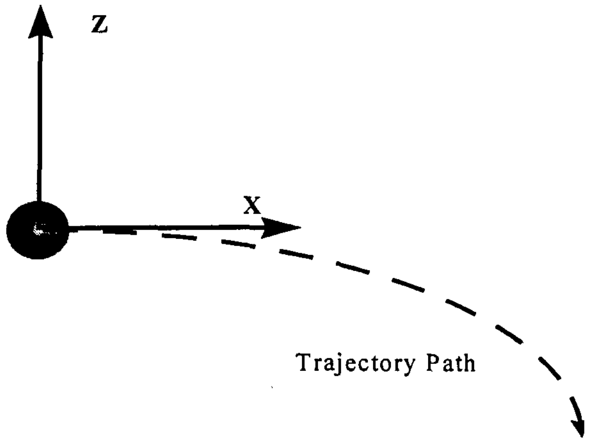

Figure 2. 2-DOF System

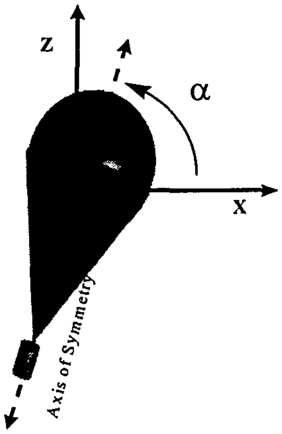

Figure 3. 3-DOF System

rigid body by a massless rigid connection. Finally, the elastic model has mass node equations of motion, defining the decelerator and elastic suspension lines, which are solved simultaneously with the rigid body equations. Doherr (1992) summarizes the different parachute models and their capabilities (Table 2). For the 6-DOF model, ψ, θ, and ϕ describe rotational motion of the system (Figure 4). The 9-DOF model adds to this the variables ψ_p, θ_p, and ϕ_p, which describe the relational and rotational motion of the payload with respect to the canopy.

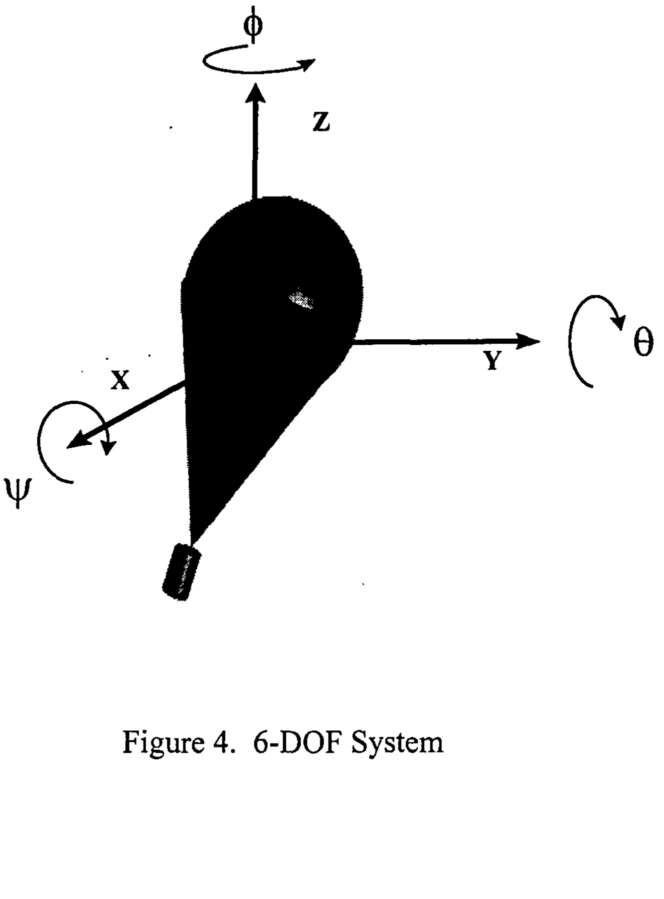

Figure 4. 6-DOF System

Table 2
Comparisons of Parachute Models (Doherr, 1992:6-4)

Trajectory Analysis	Point Mass or Ballistic	Planar Rigid Body	6-DOF Rigid Body	9-DOF Rigid Body
Degrees of Freedom (DOF)	2	3	6	9
Major Variables	x, z	x, z, θ	x, y, z, ψ, θ, ϕ	x, y, z, ψ, θ, ϕ, ψ_p, θ_p, ϕ_p
PAYLOAD INPUTS				
Mass	•	•	•	•
Inertias		•	•	•
Cx	•	•	•	•
Cy			•	•
Cz		•	•	•
Cl			•	•
Cm		•	•	•
Cn			•	•
DECELERATOR INPUTS				
Mass				•
Inertias				•
C_DS (drag area)	•	•	•	•
C_N (normal)				•
C_l (roll)			•	•
x_{CP} (center of pressure)				•
α_{ij} (apparent mass)				•
Coupling Conditions				•
OUTPUT				
Deceleration	•	•	•	•
Velocity	•	•	•	•
Down-Range	•	•	•	•
Off-Range			•	•
Altitude	•	•	•	•
Heading Angle			•	•
Pitch Angle		•	•	•
Roll Angle			•	•
Relative Angles				•

In communicating trajectory-modeling methods in the airborne test and evaluation community, the stages of system flight are commonly referred to as zones (Figure 1). *Zone 1* is defined from the time of exit to canopy inflation just after first vertical. This is normally 130 below exit for the T-10C. *Zone 2* is the region from 130 feet below exit to 450 feet above ground level (AGL). In combat jumps, *Zone 2* is non-existent. *Zone 3* is the region from 450 AGL to ground. In combat jumps, *Zone 3* is defined as the region from 130 feet below exit to ground level (Natick 1996).

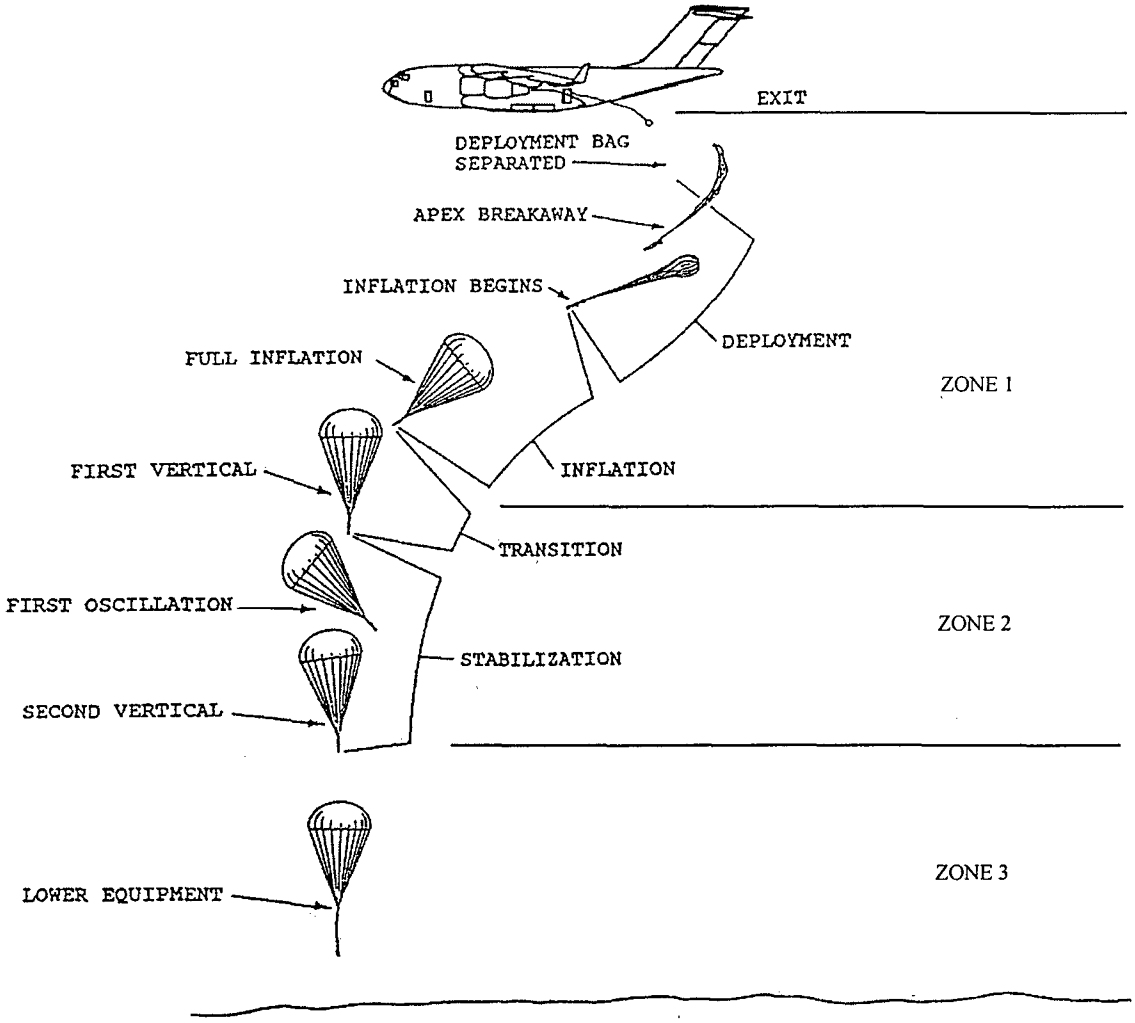

Figure 5. Parachute Behavior Zone Regions (Natick 1996)

2.4 Current Models

The Blake Model (1996) uses a simple point-mass system with 3-DOF, one each for the vertical, horizontal, and lateral components of motion. Although the model uses

3-DOF, both the horizontal and lateral components are simplified by using the same components of crosswind as the defining relation.

Another 3-DOF model (Benney 1996) uses the rigid two-body model that oscillates in a two-dimensional wind profile. This model uses the basic system of equations developed from Newton's Laws of Motion to estimate the response of a parachute-payload system when encountering a changing wind profile such as a wake vortex approximation.

Natick RDEC has developed another 3-DOF model for a different purpose (Wallace 1996). The motivation in this model is to show an inherent Dispersion Error Probability (DEP) for a non-gliding (unconscious) parachute-payload system. The model uses a point-mass system with random oscillation affecting both lateral and horizontal components of the trajectory. Factors that affect the DEP are identified and are taken into account if considered to possess a significant affect on DEP.

A 6-DOF model increases the resolution of the trajectory and can account for the position components of the trajectory which are the vertical, horizontal, and lateral movements (with corresponding axes) of the parachute-payload system, as well as three added DOF for rotation about each axes. One such model, developed by Tory and Ayres (1977), uses a rigid two-body system and models the trajectory of the 6-DOF parachute-payload system after full canopy inflation.

Models that use high DOF are more useful when analysis of the relative motion between the parachute and the payload needs to be considered (Doherr 1992:774). Doherr (1992) develops a more complex model having 9-DOF, which uses a non-rigid two-body system where two masses, the parachute and the payload, are connected by a joint. There are 6-DOF associated with the payload (position and attitude) and 3-DOF associated with the parachute (attitude only).

This is by no means an exhaustive list of models available, but is one that are useful for the level of resolution this thesis effort aspires toward. In a complete fluid dynamic (CFD) study of parachute structural dynamics in a close surface investigation, thousands of DOF are required due to the ever-increasing complexities involved, and therefore is beyond the scope of this effort.

3. METHODOLOGY

There exists a need to develop a usable airborne risk assessment model helpful in both the Landing Plan and the Air Movement Plan in the backwards planning sequence of airborne activities, and addressing both paratrooper/wake vortex encounter and DZ dispersion. Given the current trial and error method of formation testing in the C-17 airdrop test and evaluation community, aided by the Blake (Wright Lab) Model, the problem is to develop a stochastic paratrooper object which predicts paratrooper trajectories to be used in conjunction with the Petry C-17 aircraft and vortex objects. The technique uses object-oriented simulation subject to similar self-imposed, but necessary, requirements of Martin's (1978) model.

3.1 Solution Technique

Exhaustive testing of possible formation geometries only provides data for those specific cases. An alternative to the trial and error approach that the airborne test and evaluation community uses is to approximate the complex system environment of simultaneous activities and events involved in airdrop operations using simulation methods. Smart and effective testing methods exist with the use of experimental design analysis (EDA) and response surface methodology (RSM); but, at the same time questions arise as to whether or not the formation geometry being tested is best suited for the conditions which exist. Given the cost of testing, repetition of sample runs may not be feasible. Using a simulation model, repetitive testing can be performed under a varied range of controllable parameters to determine feasible configurations prior to live testing.

Different forms of simulation methods exist, ranging from spreadsheets to highly specialized languages specifically catered for simulation. MODSIM III is one such specialized simulation language in that it is object-oriented. Objects are self-contained data structures that have their own methods (Banks *et al.* 1996: 128). They can be looked upon as the building blocks of a system, behaving independently of other building blocks yet interacting with them. When taken as a whole, object-oriented programming provides a more natural way to approach the building of a system simulation by dynamically adding more instances of an object as they are needed. Object-oriented programming facilitates multiple, concurrent instances of similar objects.

In airborne operations, several entities exist concurrently with each entity exhibiting specific behaviors. There are multiple aircraft in formation, dropping multiple paratroopers and generating wake vortices, all in the same airspace. Although the aircraft, vortices, and paratroopers behave independently of each other, they continually interact with one another--the aircraft fly in a formation, the paratroopers exit the aircraft, and the wake vortices may or may not disturb the paratroopers' trajectories. In short, the type of system environment involved with airborne operations lends itself well to object-oriented simulation.

3.2 Implementation

US Army Natick Research Development and Engineering Center remains at the forefront of aerodynamic decelerator systems modeling specific to US Army needs. From the outset, Natick has been involved in the development of the paratrooper object. With Natick, the decision to pursue a 6-DOF model consistent with Table 2 is substantiated because of the need for the information that a 6-DOF model provides over a 3-DOF model and because a 9-DOF model would provide no added benefit (Benney 1996). The Purvis 6-DOF Model (1987), written in FORTRAN, is a simple yet effective model providing a straight forward approach to trajectory propagation by using a second-order Euler integration scheme (Doherr 1992:6-1). The development of a paratrooper object takes place in three stages. First, the original Purvis FORTRAN Model code is translated into an object using the object-oriented simulation language MODSIM III. Once a successful translation is achieved, the object is modified to reflect the aerodynamic properties of a combat equipped paratrooper using a T-10C parachute. The last stage integrates the paratrooper objects with the Petry C-17 aircraft and wake vortex object and incorporates the addition of stochastic elements in the trajectory propagation scheme.

3.2.1 Translating Into MODSIM III

The Purvis Model is organized using a single main program, four subroutines, and a function routine. However, it can be looked upon as performing three distinct roles: input of parameters specific to the parachute-payload system, initialization of starting conditions, and trajectory propagation. Similar roles are needed in order to perform the same calculations in MODSIM III. An initial structure of four MODSIM III modules has been used to properly convert the Purvis Model. The modules are:

- *Main Module*
- *Global Module*
- *Calculation Module*
- *Jumper Module*

The *Main Module* is a simple module that most accurately describes the activities taking place in the simulation. In the *Main Module*, a new instance of a jumper object, emulating the same type of parachute-payload system used in the Purvis Model, is created and told to jump at the start of the simulation. Once the jumper object impacts ground level, it is disposed.

The *Global Module* defines and initializes those constants and looping variables that are used throughout the simulation. It includes the method *initializeData* that simply initializes the constants used in trajectory propagation. The *Main Module* imports this method and implements it prior to creating a new instance of a jumper object.

The *Calculation Module* defines two functional procedures that are used repetitively during trajectory propagation. The first procedure calculates the gravitational effect at a given altitude. The other procedure calculates the density and speed of sound for a given altitude above sea level (ASL).

The *Jumper Module* contains the bulk of the MODSIM III code. There are two methods written into the *Jumper Module*: *ObjInit* and *jump*. When the *Main Module* creates a new instance of a jumper object, the *ObjInit Method* is automatically called upon to initialize all the parameters specific to the jumper in its trajectory propagation. The *jump Method* houses the trajectory propagation loop. Trajectory information continues to be calculated until ground impact. During this propagation, the jumper maintains all output information consistent with Table 2 and displays the information at specified time intervals.

The difficulty in translating from the Purvis FORTRAN Model to a MODSIM III object is two-fold. Since a basic understanding of the logic flow of the Purvis Model must be discerned from the FORTRAN coding, the first level of difficulty is understanding of the structural flow of logic, i.e., what is the Purvis Model doing? The second level of difficulty is translating the FORTRAN structure into a MODSIM III architecture. Both difficulties have been overcome with an end result of a MODSIM III object that correctly emulates the FORTRAN parachute-payload system of the Purvis Model.

3.2.2 Modification Into Paratrooper Objects

Modifying the parachute-payload system of the Purvis Model is a matter of modifying the input parameters to correctly represent the aerodynamic properties of a combat equipped paratrooper using a T-10C. Under the guidance of Natick, the following input parameters have been changed in order to model a paratrooper's trajectory: weight, payload (forebody) center of gravity (cg), payload length, inertia in the x-coordinate (roll), inertia in the y-coordinate (pitch), inertia in the z-coordinate (yaw), and effective T-10C inflation area. The Purvis Model has an example that changes all these aerodynamic properties in its input routine to show that the model can be used with different parachute-payload systems. The modifications needed in the development of a paratrooper object follow closely to Purvis' own changes.

Several assumptions are made regarding the physical properties of the paratrooper-parachute system in order to find its aerodynamic properties. The paratrooper object assumes the physical geometry of a cylinder, with the T-10C assuming the form of a half-sphere. This is a typical assumed configuration used in parachute-payload systems modeling. There is also the conical section of the system formed by the risers connecting the paratrooper to the parachute. And lastly, as the parachute inflates, there is added mass to the system due to the air trapped under the parachute canopy. The cylinder is assumed to be six feet high and one foot wide. Although joined at a point where the harness meets the paratrooper, the system is considered a rigid body. In other words, the joint does not pivot, and the payload has no relative motion in relation to the parachute. Initially, the weight of a 360-pound paratrooper is assumed since there are known parameters for such a paratrooper, such as mean descent velocity and mean descent time. Moments of inertia around each of the axes are found easily using a set of equations found in basic physics texts (Benney 1996).

In finding the moments of inertia, the following parameters are first defined:

a = radius of payload cross-section

d_c = distance from canopy cg to system cg

d_p = distance from payload cg to system cg

d_s = distance from suspension line cg to system cg

l_p = length of payload

M_A = added mass

M_P = mass of payload

r = radius of canopy

ρ = air density

W_c = weight of canopy

W_p = weight of payload

W_s = weight of suspension line

First, the added mass of an inflated canopy is calculated using

$$M_A = \rho \frac{4}{3} \pi \cdot r^2 .$$

The payload moment of inertia about the x and y-axes are calculated using

$$I_{P_{XX}} = I_{P_{YY}} = \frac{1}{12} M_P (3a^2 + l_P^2)$$

and about the z-axis using

$$I_{P_{ZZ}} = \frac{1}{2} M_P a^2 .$$

Finally, using M_A and I_P, the system moment of inertia about the x and y-axes are

$$I_{SYS_{XX}} = I_{SYS_{YY}} = M_A (\frac{2}{5} r^2 + d_c^2) + \frac{W_c}{32.2} d_c^2 + \frac{W_s}{32.2} d_s^2 + I_{P_{XX}} + \frac{W_p}{32.2} d_p^2$$

and the z-axis is

$$I_{SYS_{ZZ}} = \frac{2}{5} (\frac{W_c}{32.2}) r^2 + \frac{2}{3} M_A r^2 + I_{P_{ZZ}} .$$

The last change to the parameters involves the effective inflation area of the T-10C. Lying down flat the coverage area of the T-10C is 953 ft^2 (Natick 1996:1). However, once inflated, the effective inflation area, assuming the inflated T-10C becomes a half-sphere, is approximately 690 ft^2.

3.2.3 Integration and Enhancements

The building block ease of MODSIM III allows for easy integration between different objects. The paratrooper object combines with the Petry C-17 aircraft and vortex objects to form a cohesive system to model airborne operations. During the integration process, the need for further modifications becomes apparent. The following new modifications are made:

- management of positional information using three coordinate systems
- separate right and left paratrooper objects
- a system to pass relevant information between the C-17 aircraft objects and the individual paratrooper objects
- a *greenLight* Method which initiates the paratrooper object jump formation
- a method for each paratrooper object to calculate its own distance from vortices originating from upstream aircraft
- the introduction of stochastic variables into the paratrooper objects

Coordinate Systems. The use of three related coordinate systems becomes a convenient way of managing relative positional information between the C-17 aircraft, vortices, and paratrooper objects. The first coordinate system has its origin centered on the lead aircraft of the lead element, is referred to as the Aircraft Coordinate System (ACS), and is continuously moving. Positive direction is measured aft of the aircraft for the x-position, starboard for the y-position, and above for the z-position. The Ground Coordinate System (GCS), the stationary system, has its origin at leading edge of the drop zone. Positive direction is measured in the direction of flight path for the x-position, to the right for the y-position, and above for the z-position. A third coordinate system, the Inertial Coordinate System (ICS), is unique to the paratrooper objects. Its origin is at the point of exit for the paratrooper object and remains stationary from exit to impact. Positive direction is measured along the direction of flight for the x-position, starboard for the y-position, and downward for the z-direction.

Left and Right Paratrooper Objects. The addition of distinct right and left paratrooper objects adds to the resolution of the simulation by modeling the exit of paratroopers from either the right or left rear doors of the C-17. The only difference between a right and a left paratrooper object is its point of origin.

Communication of Information. Communication between the aircraft and the paratrooper objects is essential for several activities that occur in the simulation. The first instance of shared information is in the *greenLight* Method of the C-17 aircraft objects. The *greenLight* Method starts the stick of paratrooper objects exiting the aircraft. In this method, the C-17 aircraft object passes on its positional information to the paratrooper objects. Once free of the aircraft, the paratrooper objects act independently of any other object; however, several pieces of information need to be continuously passed into the paratrooper objects, since they change concurrently with the paratrooper objects' trajectories. MODSIM III allows for easy exchange of information between different objects as long as the requesting object specifies which object to import the information from. Table 3 lists the categories of information which are continuously changing and are continually used by each paratrooper object.

Table 3
Location of Imported Information

INFORMATION	IMPORTED FROM
Exit Positions	Vortex Control Module
Position of Lead Aircraft in Lead Element	Vortex Control Module
Variable Winds	Global Module
Vortex Positions	Vortex Module

Green Light. In airborne operations, aircraft are flying in a specified formation. Green Light is called when the CARP is encountered. The CARP in the simulation is similar to the CARP in the model done by Martin in that it is that point in airspace where paratrooper objects begin their exit from the aircraft. However, it differs in that it is not calculated from a planned impact point on the DZ, but rather, it is used as a starting point for the simulation. Once Green Light is reached, paratrooper objects begin to exit the aircraft until the last paratrooper object in the stick has departed. Paratrooper objects exit in a static inter-departure time of 0.5 seconds.

Vortex Polling. Management of positional information allows for each paratrooper object to calculate its position in all three coordinate systems in order to keep track of relative positions. Relative positioning is used in the determination of paratrooper object distances from known vortex positions called *missed distance*. This is accomplished in the *pollVortices Method.* The linear algebra method of vector projection is used to find the orthogonal distance between the paratrooper object and the vortex object's core. The core positions of vortex objects are defined as points in space at 100-foot intervals, originating 100 feet behind its generating aircraft. Since the vortex objects are defined under the ACS, each paratrooper object must translate its position from the ICS to the ACS to begin a search along the vortex body for its closest orthogonal distance. Before calculating orthogonal distances, the paratrooper object searches for that position along the vortex body where it is within a tolerance box; i.e., a region that is reasonably close to the vortex. Once within this tolerance box, the paratrooper object calculates the orthogonal distance by first defining two vectors having the same origin. The first vector is defined from a vortex object core position to the paratrooper object position v_j. The second vector is defined from the same vortex core position to the subsequent vortex core position v_v. In Figure 6, projection of v_j onto v_v is calculated using the following equation (Strang 1986: 147)

$$v_p = \frac{v_v^T v_j}{v_v^T v_v} v_v$$

where v_p is the projection, and the orthogonal distance is $\|v_j - v_p\|$. If this orthogonal distance is within the effective vortex radius (defined by a critical vortex strength, or swirl velocity) then an encounter is recorded. Major and minor encounters are not distinguished. The paratrooper object makes these calculations for every upstream vortex generated.

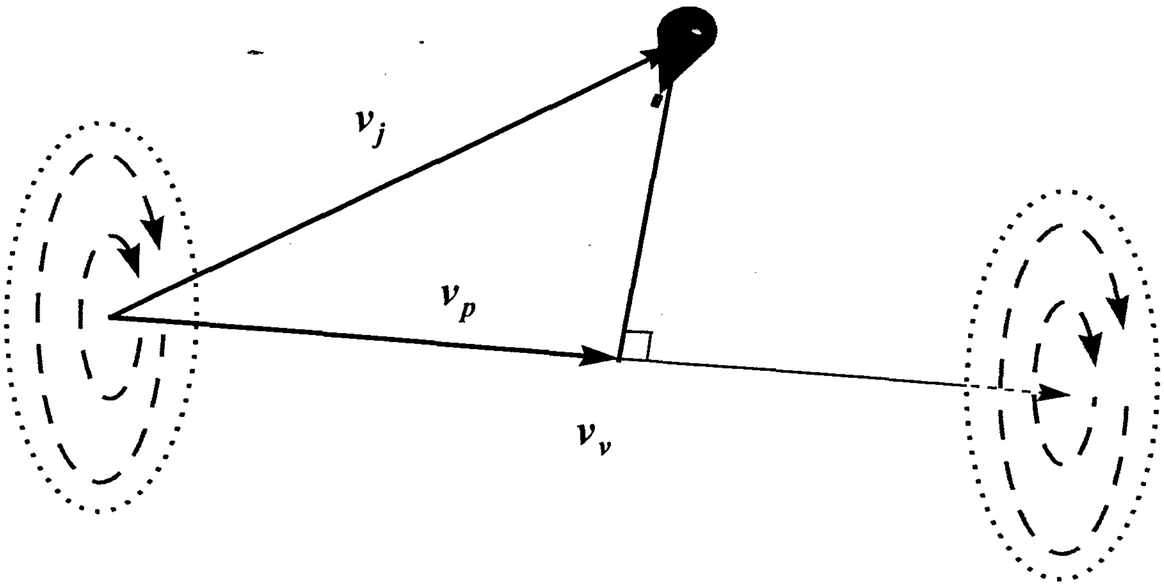

Figure 6. Orthogonal Distance of Paratrooper from Vortex Core

Inclusion of Stochastic Elements. Without any form of stochastic elements present in the model, the vortex encounter rate and the DZ dispersion can be calculated deterministically. The trajectories of each paratrooper object mirrors exactly the trajectory of any other paratrooper object resulting in either all or no paratrooper/wake vortex encounters. The addition of stochastic behavior enhances the simulation's usefulness by modeling their random behavior. The variables to randomize are chosen such that the model reflects actual occurrences in an airborne operation. Although any numbers of elements are candidates for stochastic modeling, this thesis limits the choices to paratrooper weights and T-10C glide. In keeping with resolution and capabilities of the Purvis Model, these two elements are easily modeled without changing the fundamental aerodynamic methodology used. (With the integration of aircraft and vortex objects, winds are a stochastic element that affects trajectory propagation.) From the weights of paratroopers used in D-bag clearance testing in March 1996, a normal distribution is fitted to model paratroopers with a mean weight of 247 pounds and a standard deviation of 24.35 pounds.

A harder problem is the random glide inherent in the T-10C when no wind is present and no oscillation is being experienced. Natick recognizes the presence of glide and defines it as occurring in a random direction, changing in a random manner with an initial horizontal velocity of 2 to 4 feet per second (fps) (Natick 1996:4). Under Natick advisory correspondence, the modeling of the glide behavior is made to mimic (as best as possible) the true behavior of glide under similar conditions. When the paratrooper object has reached steady state descent, the onset of glide is modeled from a uniform draw between 0 and 360 degrees for direction and between 0 to 4 fps for velocity. Subsequent changes in the glide follows the initial direction with little deviation, while glide continues to vary between 0 to 4 fps (Watkins 1996).

With the inclusion of random winds, the Purvis method of trajectory propagation reveals limitations which does not vitiate the model by any means. In the presence of winds, the paratrooper object takes on the direction and velocity of the winds immediately. Additionally, in the presence of winds, the model becomes highly sensitive to the propagation time step. A trade off exists with time step and simulation run length, where the smaller the time step the longer a simulation run takes until completion. The section on verification and validation discusses further the effects of different time steps.

3.3 Verification and Validation

Both *verification* and *validation* are continual processes along every stage of simulation model development. *Verification* involves the determination that a simulation model is performing as the developer intends while *validation* concerns whether or not the conceptual model on which the simulation model is based accurately represents the system being studied (Law *et al.* 1991:299).

3.3.1 Verification

A critical milestone in verification is the correct translation of the Purvis FORTRAN Model into MODSIM III. Due to the vast amounts of information provided by both models, a graphical form of verification is employed using MATLAB graphing capabilities. Table 4 displays the largest absolute error in the state variables of the paratrooper object when compared to the state variables of the FORTRAN model. Based on these small errors, we conclude that the Purvis FORTRAN Model has been correctly translated into MODSIM III, since the errors are small enough to be attributable to computational differences in the hardware or software environment.

Table 4
Error Between FORTRAN and MODSIM III Models

Paratrooper State Variables	Largest Absolute Error
Altitude (ft)	0.1827
Down Range (ft)	0.2840
Off Range (ft)	0.0000
Velocity (fps)	0.3990
Airspeed (fps)	0.3990
Mach Number	0.0003
Dynamic Pressure (psf)	0.2540
Axial Acceleration (gees)	0.0294
Trajectory Angle (deg)	0.0143
Pitch Angle (deg)	0.4538
Alpha (deg	0.4513
Drag Area (ft^2)	0.0000

In integrating the paratrooper objects with the Petry C-17 aircraft and vortex objects, the interaction of the objects is another area where verification comes into play. Of particular interest are the *greenLight* and *pollVortices Methods*; i.e., the interactions of the aircraft/paratrooper and paratrooper/wake vortex, respectively. The individual paratrooper objects keep track of all relational information; e.g., data on which aircraft a particular paratrooper object jumped from, the time of exit, the (x, y, z)-coordinate of the exit point, and airspeed at time of exit are all maintained within the paratrooper object. Thus, verification is greatly facilitated. The paratrooper objects also keep track of all relational information regarding the wake vortices from upstream aircraft. Again, each paratrooper object tracks all known wake vortex core locations and computes relational distances. Verification, although cumbersome when several paratrooper objects and wake vortex objects are in the same airspace at the same time, is facilitated with the capability of the paratrooper objects to display its search of, and distance from, the wake vortices.

During large preliminary test runs with multiple aircraft and several hundred paratrooper objects, the need to either reduce run time or acquire a faster computer becomes apparent. The addition of more aircraft and more jumpers further reduces the speed of the simulation model. The obvious choice of focus is the trajectory propagation time step, *dt*, of the paratrooper objects. As in the Purvis Model, the paratrooper object's original *dt* is 0.0005 of a second; every paratrooper object calculates its new trajectory position every 0.0005 seconds! Sensitivity analysis done on different *dt*s show that certain system state variables of the paratrooper object are highly sensitive to the *dt*s under different wind conditions. Two configurations are considered--no winds, and head/cross winds. In both cases, increasing the *dt* can make the trajectory propagation calculations unstable, usually starting with the rotational angles and spreading to all other state variables.

Larger Propagation Time Step/No Winds. With no winds present, three propagation time steps are used in finding the sensitivity of model stability to changes in the time step using the same parachute-payload system configuration as the original Purvis Model. The three *dt*s are 0.0005, 0.001, and 0.05. As the *dt* increases, run length decreases, although larger absolute errors become apparent as the propagation scheme becomes unstable (particularly in the rotational state variables). Table 5 shows the maximum absolute error experienced at each of the *dt*s. The large errors experienced in the descent velocity and airspeed are not due to larger values but rather can be attributed to an earlier inflation of the canopy, thus causing the parachute-payload system to decelerate sooner at the larger *dt*s. The descent velocity and the airspeed are the same once steady state is reached in all three propagation time steps. The instability in the rotational state variables is readily seen graphically as *dt* is decreased (Appendix E). The positional state variables, however, still have relatively small absolute errors associated with decreased *dt*s, with the final (x, y, z)-coordinate fairly close to the original location using the original *dt* of 0.0005. This is an important observation, as shorter run length is desired.

The same experimentation is done on the parachute-payload system configured as a paratrooper object. Again, similar results are experienced using the same three *dt*s. A favorable run time length is gained using the larger propagation time step of 0.05 seconds, and hence becomes a candidate for use in the paratrooper objects with the knowledge of the instability experienced by the rotational state variables and an earlier canopy inflation. The selection of this *dt* is conditional on system performance under the case where head/cross winds are present.

Larger Propagation Time Step/Winds. Further experiments under different *dt*s are made under the conditions with winds present, again using the same parachute-payload system as in the original Purvis Model. A constant crosswind velocity of 5 fps is used to determine the effects of the increased *dt*s on the system state variables. Similar results of the case with no winds are observed; however, now with error in the y-axis, increased instability is experienced in the rotational state variables. Table 5 shows the maximum absolute error experienced under wind conditions with the increased *dt*s.

Table 5
Max Error From Changes In Propagation Time Step

System	No Winds			Winds		
State		*dt*			*dt*	
Variables	***0.0005***	***0.001***	***0.05***	***0.0005***	***0.001***	***0.05***
Altitude (ft)	0.183	0.182	10.166	0.202	0.149	11.203
Down Range (ft)	0.284	0.806	23.251	0.282	0.811	23.250
Off Range (ft)	0.000	0.000	0.000	0.143	0.131	1.000
Velocity (fps)	0.399	0.332	25.558	0.389	0.366	25.542
Airspeed (fps)	0.399	0.332	25.558	0.389	0.328	25.542
Mach Number	0.000	0.000	0.023	0.000	0.000	0.023
Dynamic Pressure (psf)	0.254	0.153	22.731	0.258	0.148	22.817
Axial Acceleration (gees)	0.029	0.170	38.560	0.029	0.171	38.578
Trajectory Angle (deg)	0.014	0.094	1.310	0.019	0.095	1.304
Pitch Angle (deg)	0.454	19.995	176.378	0.523	19.696	152.811
Alpha (deg	0.451	19.981	202.024	0.477	19.655	202.737
Drag Area (ft^2)	0.000	0.000	4.998	0.003	0.003	4.998

When applied to the paratrooper objects under wind conditions, the entire propagation scheme becomes uncontrollably unstable early in the trajectory life of the paratrooper object prior to the inflation of the canopy. After a mechanistic breakdown of the trajectory calculations, it turns out that this instability has always been present, but is muted due to the null value of wind effects and is therefore zeroed out. With the presence of winds, this instability manifests earlier in the trajectory as the *dt* increases.

In the pursuit to achieve a more efficient run length, it becomes necessary to use two separate propagation time steps at different stages of the paratrooper object's trajectory. Through trial and error, the large time step of 0.001 seconds prior to full canopy inflation and the smaller time step of 0.01 seconds after full canopy inflation gives a reasonable solution to a shorter run length while maintaining stability.

3.3.2 Validation

Under the guise of object-oriented programming, the paratrooper object is the Purvis Model. Henceforth, validation centers on the configuration from the parachute-payload system used in the Purvis Model into a combat-gear outfitted paratrooper using a T-10C. Our validation first looks into the distribution of a representative sample of paratrooper weights. Secondly, a graphical comparison between actual paratrooper trajectories and trajectories generated by the integrated airdrop model is accomplished. Finally, Petry (1997) compares actual testing results from Edwards AFB and Fort Bragg to results generated by the integrated airdrop model.

Paratrooper Weights. The jumper manifest from C-141 Jumper-Head to D-Bag Clearance testing is used as a representative sample in finding a distribution for paratrooper weights. The paratroopers in this testing are outfitted with combat gear and use the T-10C parachute and reserve. Using the software package BestFit, any one of the distributions in Table 6 makes an appropriate fit using the Kolmogorov-Smirnov (K-S) Test Statistic with a level of significance of $\alpha = 0.05$, and $n = 82$.

Table 6
Fitted Distributions and Associated K-S Test Statistics

Distribution	**Rank**	**K-S Test Statistic**
Logistic(247, 13.34)	1	0.04008
Normal(247, 24.35)	2	0.06968
Beta(5.35, 4.87) x 151 + 167	3	0.07252
$\chi^2(246)$	4	0.07974

The normal distribution may be the easiest and most readily recognizable distribution from among the top choices with good fits. By examining both the probability-probability (P-P) and the quantile-quantile (Q-Q) plots, the use of the normal distribution is a good choice as the representative distribution for paratrooper weights.

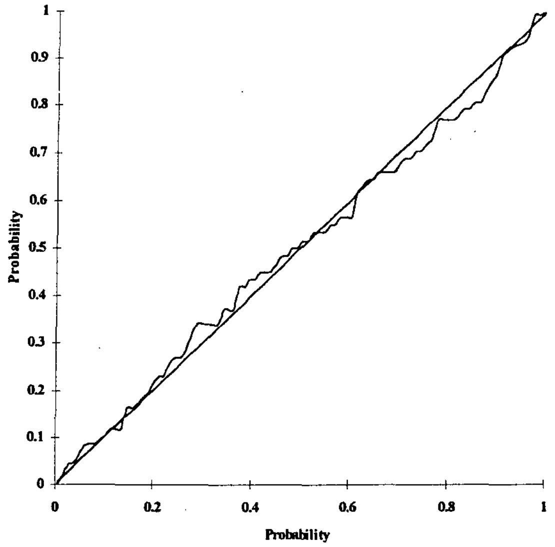

Figure 7. Normal P-P Plot of Paratrooper Weights

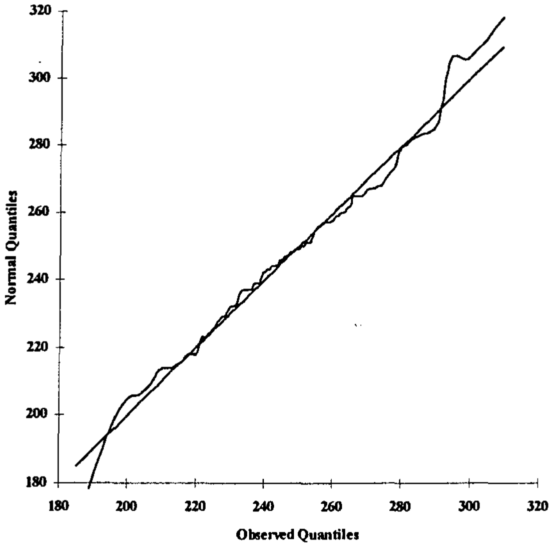

Figure 8. Normal Q-Q Plot of Paratrooper Weights

Cinetheodolite Data Comparison. A method of recording actual paratrooper trajectories is the cinetheodolite (Cine-T) method, which uses six synchronized cameras recording the same paratroopers from different angles. This itself is a crude recording method subject to human error during the post-processing data collection, which is essentially watching the recorded video and extrapolating the paratrooper positions from frame-by-frame playback of the recorded trajectory (Dassow 1997). According to Dassow the positions are extrapolated using the known angles of the individual cameras and triangulating using the information from the other cameras. A further limitation of the Cine-T is that trajectories are not recorded to impact on the DZ. Using the Cine-T data from actual jumps, only a cursory visual comparison can be done between actual trajectories recorded from paratrooper jumps using the C-17 as the jump platform, and trajectories generated by the simulation model. This comparison is essentially the implementation of a Turing Test. Although the exact conditions surrounding the Cine-T jumps in Figure 9 are not known (i.e., wind conditions and airspeed of the aircraft) and hence cannot be duplicated, valuable information is still gained from this comparison. Judging from the trajectories of the Cine-T trajectories, a reasonable assumption is made that both head wind and cross wind are present, and subsequently are incorporated in the generation of the trajectories seen in Figure 10.

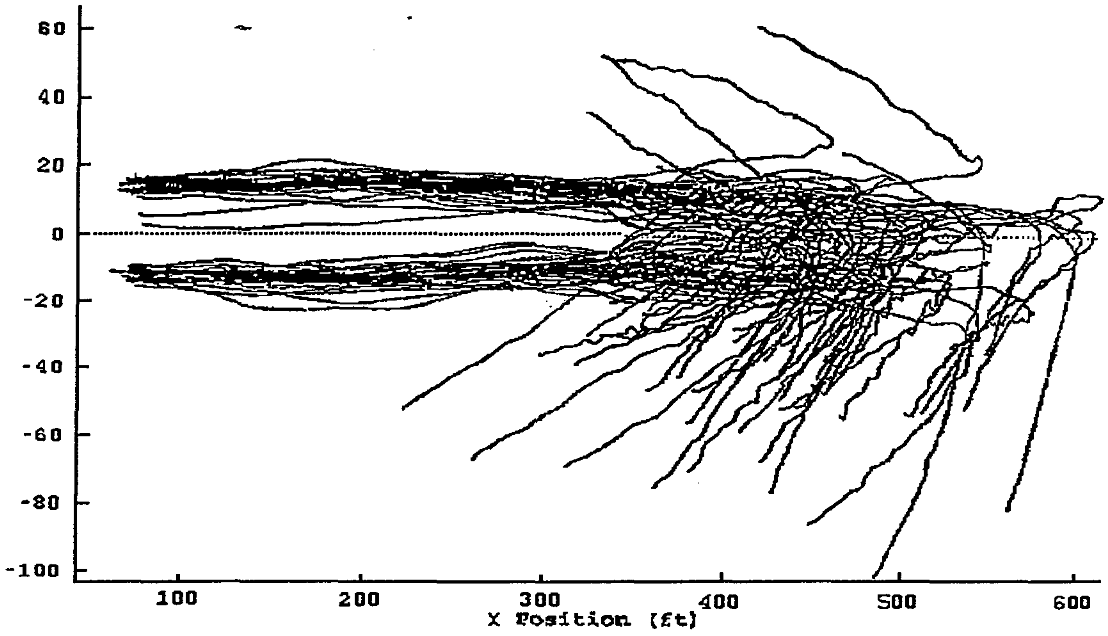

Figure 9. Cine-T Trajectory Plots

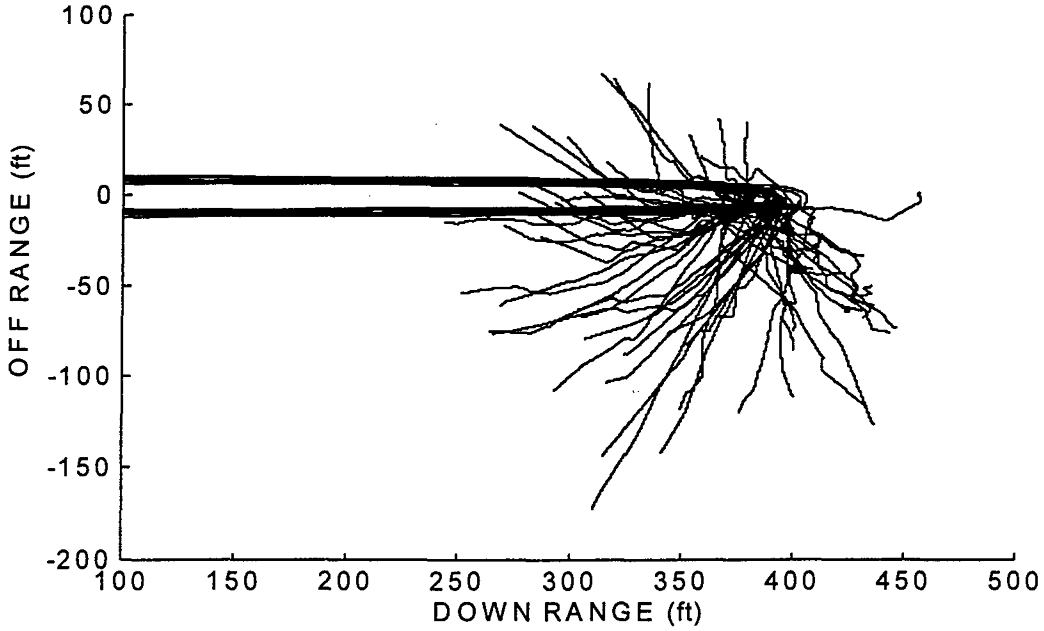

Figure 10. Trajectory Plots With Head/Cross Winds.

Furthermore, given the current assumptions of the simulation, further refinement in trajectory generation suggests using Turing test for further validation.

Encounter Rate Comparisons. Petry (1997) compares actual vortex encounter rate test results from Edwards AFB (EAFB) testing with results generated by the simulation model.

Table 7
Results Comparisons of Simulation and Actual EAFB Test

	Simulation	*Flight*
Mean	**0.13500**	**0.16250**
Variance	0.03171	0.03470
Observations	50.00000	20.00000
Pooled Variance	0.03255	
Hypothesized Mean Difference	0.00000	
df	68.00000	
t Stat	**-0.57611**	
P(T<=t) one-tail	0.28322	
t Critical one-tail	1.66757	
P(T<=t) two-tail	0.56644	
t Critical two-tail	**1.99547**	

3.4 Limitations

With the integration of the Petry C-17 aircraft and vortex objects, the limitations of the paratrooper objects define the limitations of the simulation. One of the major assumptions of all parachute/payload system models encountered during the research stages of this thesis effort is that the payload is non-intelligent or "unconscious." This is also the case with the paratrooper objects. In reality, paratroopers train to control the direction of flight under a T-10C canopy. This is an important limitation in that with non-intelligent paratrooper objects, all trajectory and scatter information only provides at best an upper bound, or worst case performance measure. Furthermore, starting from exit, other limitations are as follows. First, the right and left paratrooper objects exit the

C-17 aircraft object at discrete time intervals every 0.5 seconds. Second, paratrooper objects immediately assume the velocity conditions of the head/cross winds. This relationship tends to overestimate actual wind effects on the paratrooper objects, and thus, the paratrooper objects' impact points are again overly conservative. Third, the paratrooper trajectory propagation scheme in not affected by the presence of wake vortices. The paratrooper objects do not interact with the wake vortices, thus causing the impact points to be determined without taking into account the effect of an encounter. Fourth, the modeling of the paratrooper objects assumes that the Earth is flat; thus, all paratrooper objects land at the same ground altitude.

4. ANALYSIS

In the development of the C-17 Paratrooper/Wake Vortex Simulation Model, the primary measure of effectiveness (MOE) is the encounter rate between paratrooper and vortex objects (Petry 1997) which the model provides. The model, however, also provides other information that can be subjected to post-processing procedures. The model provides DZ dispersion information of the paratroopers--information that is useful in the Landing Plan of the backward planning sequence used by airborne commanders. First and foremost, the Landing Plan must support the Ground Tactical Plan and be supported by the Air Movement Plan. Elements of the Landing Plan are found in both the Ground Tactical Plan and the Air Movement Plan. Embedded in the Landing Plan, an Assembly Plan is generated. Data provided by the simulation model on DZ dispersion is useful for the Assembly Plan. With guidance from Klimack (1997) and the 82nd ABN DIV ASOP, a simplified, though not trivialized, airborne drop is constructed using Nijmegen DZ at Fort Bragg. Two MOEs are used: (1) mean paratrooper distance from assembly area (AA); and, (2) distribution of the dispersion across the width as well as down the length of the DZ.

4.1 The Scenario

Nijmegen DZ is one of the smaller DZs used at Fort Bragg. It measures 4950 feet long and 3000 feet wide. Nijmegen DZ has a personnel point of impact (PI) at 1050 feet into the DZ, meaning green light is not lit until there is high confidence that paratroopers will land at least 1050 feet into the DZ. When a C-17 flies at 135 knots, Nijmegen is referred to as a 17 second DZ. When paratroopers exit the aircraft at 0.5-second intervals, this allows for a stick size of 18 jumpers with one stick exiting each door per pass.

Three airborne rifle companies are used with three separate AAs for each company. An AA for the airborne unit is the place where, upon landing, the unit is tactically organized and ready to fight (82nd ABN DIV ASOP 1985:3-35 - 3-45). The AA should be as close as possible to where the paratroopers will land in the DZ, with the ASOP suggesting areas that lie along the flanks (instead of the ends) of the DZ. The three AAs chosen for this scenario all lie on the very edge of the DZ. The first AA is to port of the flight path at the PI. The second AA is located 2000 feet down range from the PI and to starboard of the flight path. The last AA is located 4000 feet down range from the PI, and is also to starboard of the flight path. Figure 11 depicts the scenario used.

With three companies assembling at three different areas, the concept of cross loading is essential to the successful execution of the Assembly Plan. When the C-17 is loaded, the different units are partitioned and loaded among the aircraft such that the paratroopers land according to their designated AAs. Cross-loading both enhances unit survivability, and maintains the tactical integrity of the operation. Cross loading can also expedite assembly if done correctly. It is in this Assembly Plan where the mean distances from the AA and the dispersal distributions on the DZ are used as MOEs.

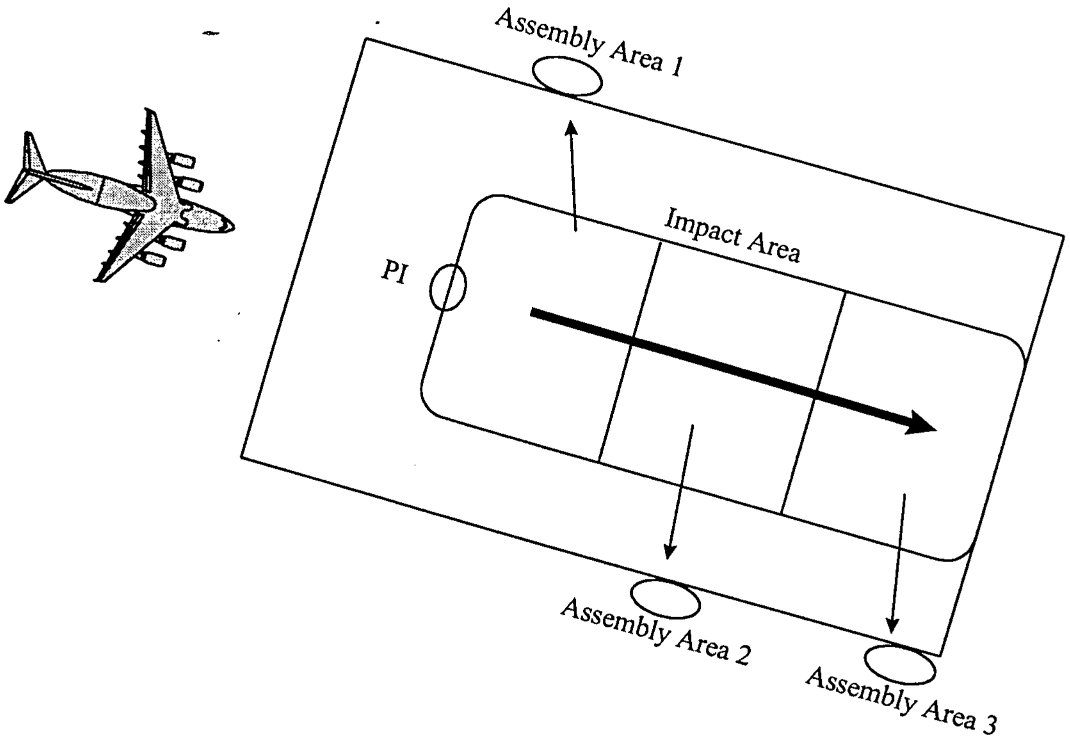

Figure 11. Drop Zone Assembly Scenario

4.2 The Design

Similar to the Petry experimental design done with encounter rates, two features of aircraft formation geometry are used in finding the MOEs: trail distance and lateral distance. In other words, how do these two features affect the MOEs?

Table 8
Design Point Description

Design Point	**Seed Used**	**Trail Distance (ft)**		**Lateral Separation (ft)**	
		Real	*Coded*	*Real*	*Coded*
1	1	15,000	-	0	-
2	2	15,000	-	500	+
3	3	32,000	+	0	-
4	4	32,000	+	500	+
5	5	23,500	0	250	0
6	6	15,000	-	250	0
7	7	23,500	0	0	-
8	8	32,000	+	250	0
9	9	23,500	0	500	+

In approaching this experimental design, multiple replications of the model are performed at different set values for the trail and lateral distances. Table 8 defines the design points used while Figure 12 shows how both the trail and lateral distances are varied. A simple 2^2 factorial design is used with center point runs. However, additional points of interest are also considered. The additional face points are included in the design to gain better insight while staying within our region of operation. The idea of adding axial points with $\alpha_d > 1.0$ extends the region in areas which would not provide any added benefit (α_d is the distance, in coded terms, from the center point and is different from the α in statistical significance levels). An axial point for lateral spacing, for example, merely places the trail aircraft on the opposite side of the lead aircraft, an area that is already accounted for

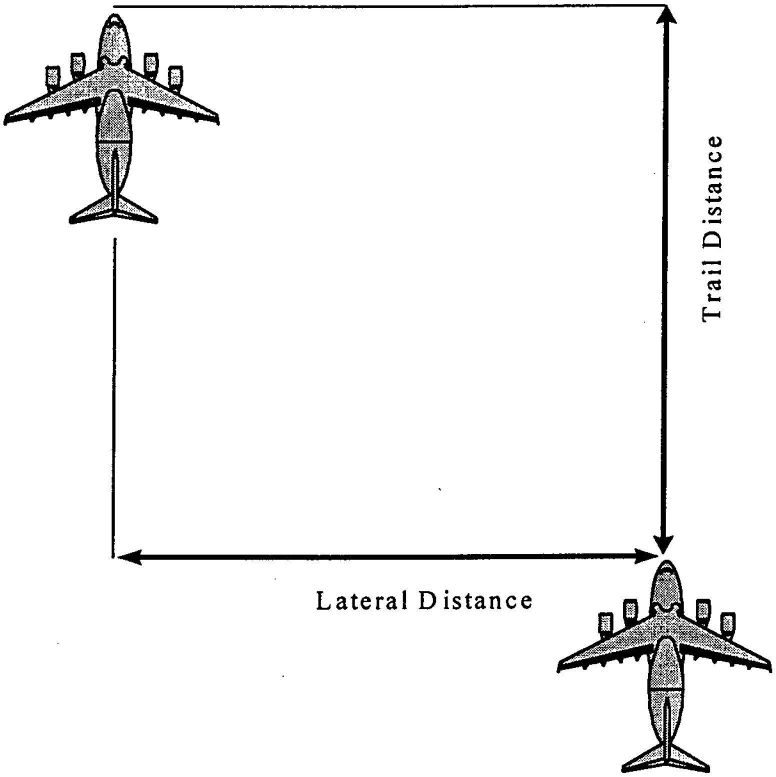

Figure 12. Variations On Formation Geometry

by center-point observations. In short, the design is a central composite structure with α_d = 1.0, or a face-centered design (Neter *et al.* 1996:1282-1286).

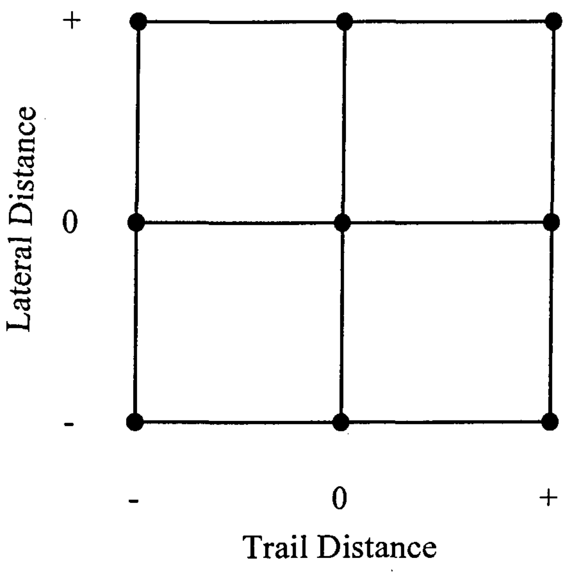

Figure 13. Face-Centered Design

In the design, the distance of each individual paratrooper impact point to its respective AA is not looked at as an individual observation. The mean distance from the AAs of all the paratroopers in the entire two-ship formation defines the MOE for a single run. It is similar to the principle applied when looking at the distribution on the DZ. An individual impact point cannot give meaningful information on the distribution. It is only when the dispersion is looked at in aggregate that a meaningful distribution can be found. (This is similar to the method of batch means used for non-terminating simulations, where the batch size is the number of paratroopers used in each of the replications.) Each run has an associated mean distance per company $\overline{Y}_i$ and sample variance S_i^2 where

Y_{im} = distance from designated AA for company *i*

c = number of runs per design point

i = designator for company number and respective AA

j = run number

k = the design point

n = number of paratroopers

and

$$\overline{Y}_i = \frac{1}{n}\sum_{m=1}^{n} Y_{im}$$

and

$$S_i^2 = \frac{1}{n-1}\sum_{m=1}^{n}(Y_{im} - \overline{Y}_i)\,.$$

Furthermore, each design point has an associated overall mean distance $\overline{\overline{Y}}_{ki}$ such that

$$\overline{\overline{Y}}_{ki} = \frac{1}{c}\sum_{j=1}^{c}\overline{Y}_{ij}$$

with variance

$$S_{\overline{\overline{Y}}_{ki}} = \frac{1}{c-1}\sum_{j=1}^{c}(\overline{Y}_{ij} - \overline{\overline{Y}}_{ki})$$

and mean variance $\overline{S}^2_{ki}$ where

$$\overline{S}^2_{ki} = \frac{1}{c}\sum_{j=1}^{c} S^2_{ij}$$

which itself has variance

$$S_{\overline{S}^2_{ki}} = \frac{1}{c-1}\sum_{j=1}^{c}(S^2_{ij} - \overline{S}^2_{ki})\,.$$

The dispersion of the paratroopers across the width and down the length of the DZ when fitted to a distribution as a paired set defines the second MOE of interest to the airborne commanders (Klimack. 1997). Although the 82nd ABN DIV ASOP does not specifically define a favored distribution for these dispersions, it is beneficial to see whether the dispersion on the DZ supports the Ground Tactical Plan. It is desirable that the paratroopers disperse around the flight path of the aircraft with some form of distribution. Intuitively, the dispersion across the width of the DZ may depend greatly on the lateral spacing between the aircraft. These paired MOEs are extracted during post-processing of the design results.

The initial number of replications used at each design point is twelve. In considering the number of runs needed, the mean distances from each of the three AAs are used with a very conservative tolerance of 5 feet. If the need to create more replications becomes evident, performing further runs is a simple task to do. (Post-processing reveals that the 12 replications are more than enough to be within the 5 feet tolerance at an alpha level of $\alpha = 0.05$.)

4.3 The Results

4.3.1 Mean Distance From Assembly Areas

Table 9 summarizes the results of the 12 runs at each design points. It contains only the $\overline{\overline{Y}}_{ki}$ and $\overline{S}^2_{ki}$ of each design point for each of the three companies. Table 24 in Appendix F shows the complete results with standard deviations for each of the $\overline{\overline{Y}}_{ki}$ and $\overline{S}^2_{ki}$, including fairly small variations in all the variables. However, looking at the results within each design point, the variation in mean distance is consistent with our intuition--that lateral spacing has a large influence while trail distance has a small effect on mean

Table 9
Summary Of Results For
Mean Distance[‡] To Assembly Areas

	Company		
Design Point[†]	**1**	**2**	**3**
1	436.228	362.8717	387.8438
(-,-)	*74.0577*	*34.5662*	*57.795*
2	504.9642	287.2827	319.209
(-,+)	*98.2338*	*83.8582*	*96.8437*
3	428.0379	361.4047	392.7657
(+,-)	*72.9951*	*30.6491*	*61.6572*
4	497.1976	285.635	327.4557
(+,+)	*96.2339*	*84.306*	*101.5088*
5	468.5109	319.6805	351.9928
(0,0)	*77.5729*	*51.1272*	*72.7923*
6	468.2581	323.4957	352.2123
(-,0)	*79.2695*	*52.4096*	*73.2678*
7	433.0555	359.7963	391.7037
(0,-)	*74.6482*	*31.3088*	*58.2386*
8	460.9896	322.2542	360.2295
(+,0)	*77.7871*	*50.4006*	*76.3632*
9	499.979	286.9205	322.376
(0,+)	*97.8993*	*85.2177*	*100.2291*

[†] Denotes coded factor levels at each design point.
[‡] Mean distance is above its standard deviation. Standard deviation is italicized.

distance. This observation is supported by looking at the correlation matrix, Table 10, of the trail (X_1) and lateral (X_2) distances and all the $\bar{Y}_i$'s.

Table 10
Correlation Matrix

Variables	X_1	X_2	$\bar{Y}_1$	$\bar{Y}_2$	$\bar{Y}_3$
X_1	1.0000				
X_2	0.0000	1.0000			
$\bar{Y}_1$	-0.1112	0.9811	1.0000		
$\bar{Y}_2$	-0.0192	-0.9887	-0.9823	1.0000	
$\bar{Y}_3$	0.1023	-0.9811	-0.9895	0.9839	1.0000

A more sophisticated method of looking at the effects of changes in trail and lateral distance can be accomplished through a RSM approach. With the central

Table 11
Response Surface Parameter Estimates

	$\bar{Y}_1$	$\bar{Y}_2$	$\bar{Y}_3$
Intercept	466.7435	320.6825	353.9703
X_1	-3.8709	-0.7260	3.5310
X_2	34.1366	-37.3725	-33.8788
X_1^2	-1.2359	1.6915	1.2618
X_1X_2	0.1059	-0.0452	0.8312
X_2^2	0.6575	2.1750	2.0808

composite design used, the corner points provide estimations for the linear main effects and interaction effects, the axial points provide an estimation for the quadratic main effects, and the center points provides both a pure error estimate for σ^2 and allows for a lack of fit test (Neter *et al.* 1996:1282). The full functional form of the response surface that is generated with coded variables is provided in Table 11 with the analysis of variance (ANOVA) provided in Table 12.

Statistical significance in the parameter estimates can be used in finding a parsimonious model; however, with an α_d = 1.0 tests of significance for the quadratic terms cannot be performed (Neter *et al.* 1996: 1286). Therefore, all the parameters are kept. For $\bar{Y}_1$, the intercept, X_1, and X_2 are statistically significant. For $\bar{Y}_2$, only the intercept and X_2 are significant. Finally, $\bar{Y}_3$ has the same significant effects as $\bar{Y}_1$. In all

Table 12
Analysis of Variance for Significance of Regression In Multiple Regression

Response	Source	DF	Sum of Squares	Mean Square	F-Ratio	Prob > F
$\bar{Y}_1$	Model	5	85028.350	17005.700	808.795	<0.0001
	Error	102	2144.644	21.000		
	Total	107	87172.995			
$\bar{Y}_2$	Model	5	100782.640	20156.500	983.682	<0.0001
	Error	102	2090.07	20.500		
	Total	107	102872.710			
$\bar{Y}_3$	Model	5	83713.418	16742.500	800.166	<0.0001
	Error	102	2134.225	20.900		
	Total	107	85846.642			

three cases, R^2 values > 0.97 are experienced. The graphical representations of these functions follow.

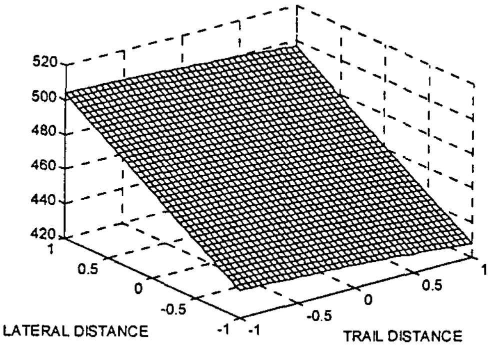

Figure 14. Response Surface Plot for Mean Distance From AA 1

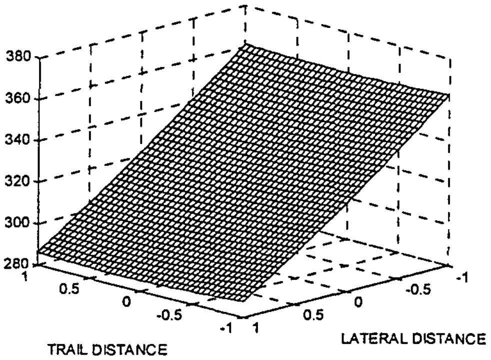

Figure 15. Response Surface for Mean Distance From AA 2

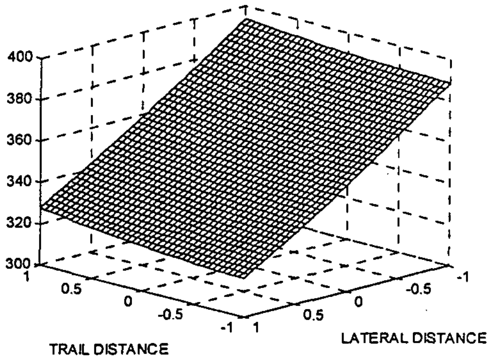

Figure 16. Response Surface for Mean Distance From AA 3

The surface plots of the mean distances for each of the AAs clearly show the small effect which changes in trail distance have compared to the effects of changes in lateral distance.

4.3.2 Distribution On The Drop Zone

In finding a representative distribution of paratrooper dispersal across the width and down the length of a DZ, the primary focus of data description can easily be lost within the post-processing investigative analysis. This occurs during the repetitive nature of post-processing, as there are nine design points to be examined, with each design point having 12 replications and each replication having two variables to work with. Representing the MOEs with 216 separate distributions can distort the underlying reason for the investigation in the first place—to find a descriptive distribution for dispersion.

Much information can be gathered by taking all the impact locations of the paratroopers used in each design point and analyzing them as a whole. Even a cursory look at the basic statistics of the impact locations in Table 13 yields valuable insight into paratrooper performance. In Table 13, location down the length of the DZ is represented by the *x*-component while location across the width is represented by the *y*-component.

Several observations stand out. First, all the summary statistics for the *x*-components at each design point are similar. Secondly, the means of the *y*-components are grouped around the midpoint between the lead and trail aircraft for all three lateral separations used. Finally, as lateral separation increase, the variance of the *y*- component also increases.

Table 13
Summary Statistics Of Dispersion Data

Design Point	*Component*	*Mean*	*Standard Deviation*	*Min*	*Max*
1	*x*	3270.4	1191.1	1170.9	5371.3
(-,-)	*y*	0.9	59.0	-116.9	127.1
2	*x*	3270.3	1191.3	1174.9	5377.7
(-,+)	*y*	251.0	254.3	-123.8	619.3
3	*x*	3272.7	1193.6	1090.4	5327.3
(+,-)	*y*	-0.2	58.2	-127.7	130.1
4	*x*	3223.5	1193.4	1127.7	5340.3
(+,+)	*y*	250.4	255.4	-114.6	612.9
5	*x*	3247.8	1191.2	1128.9	5352.8
(0,0)	*y*	125.1	135.7	-113.3	367.1
6	*x*	3272.0	1193.0	1161.7	5369.6
(-,0)	*y*	123.4	137.0	-128.8	364.2
7	*x*	3245.8	1190.3	1116.4	5349.7
(0,-)	*y*	1.3	57.0	-125.3	118.3
8	*x*	3221.4	1190.8	1125.1	5333.0
(+,0)	*y*	123.1	138.5	-119.2	365.7
9	*x*	3247.1	1194.2	1129.6	5348.8
(0,+)	*y*	249.2	258.8	-110.4	614.4

Interpreting these statistics in relation to the first MOE shows that the two results support one another. For the first and second MOE, trail distance has little effect whereas lateral distance influences the MOEs considerably. Another observation is that variation in the *y*-component increases as lateral distance increases, which occurs because the lateral separation scatters the paratroopers in a wider path down the length of the DZ. In all cases, each design point generates dispersion down the length of the DZ that appears to be uniformly distributed regardless of trail or lateral separation (Appendix G). However, dispersion across the DZ is highly dependent on the lateral separation. With no lateral separation, the dispersal distribution is heavily skewed towards either side of the flight path. As lateral separation increases a bi-modal effect occurs, creating two distinguishable distributions each exhibiting characteristics similar to the distribution of the no lateral separation scenario. Figures 17, 18, and 19 demonstrate this bi-modal distribution and the separation effect experienced with lateral separation at all three trail distances.

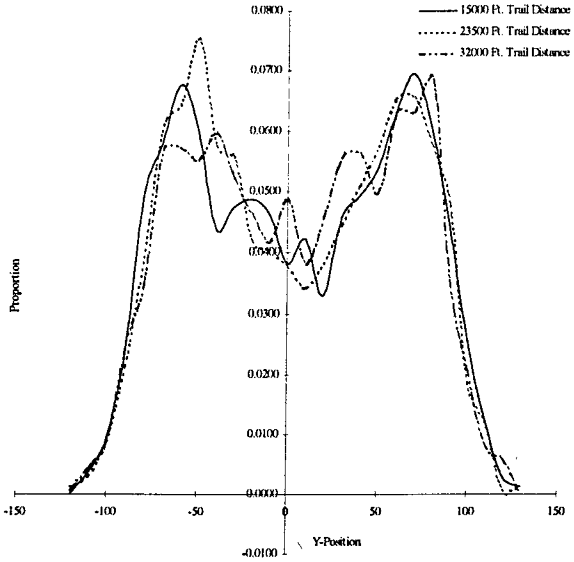

Figure 17. Dispersion Distribution Across DZ With No Lateral Separation

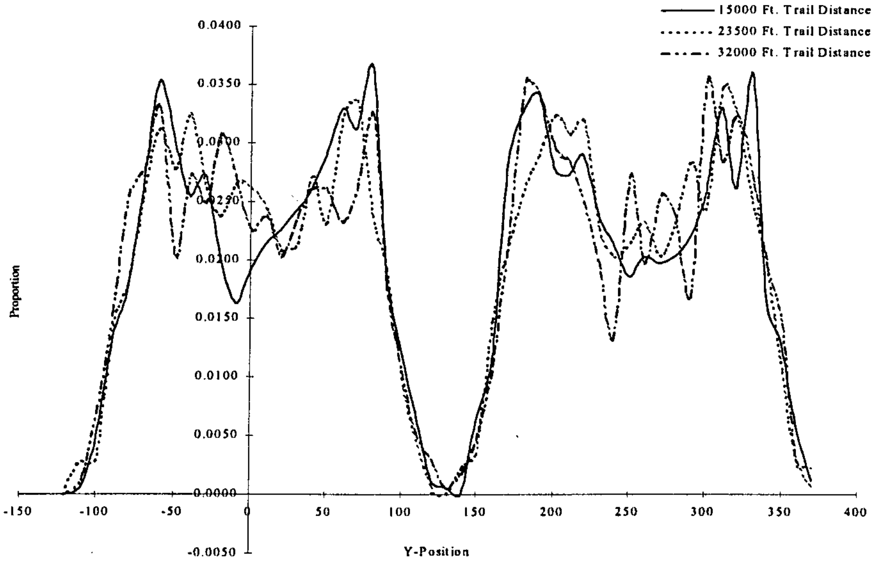

Figure 18. Dispersion Distribution Across DZ With Lateral Separation of 250 Feet

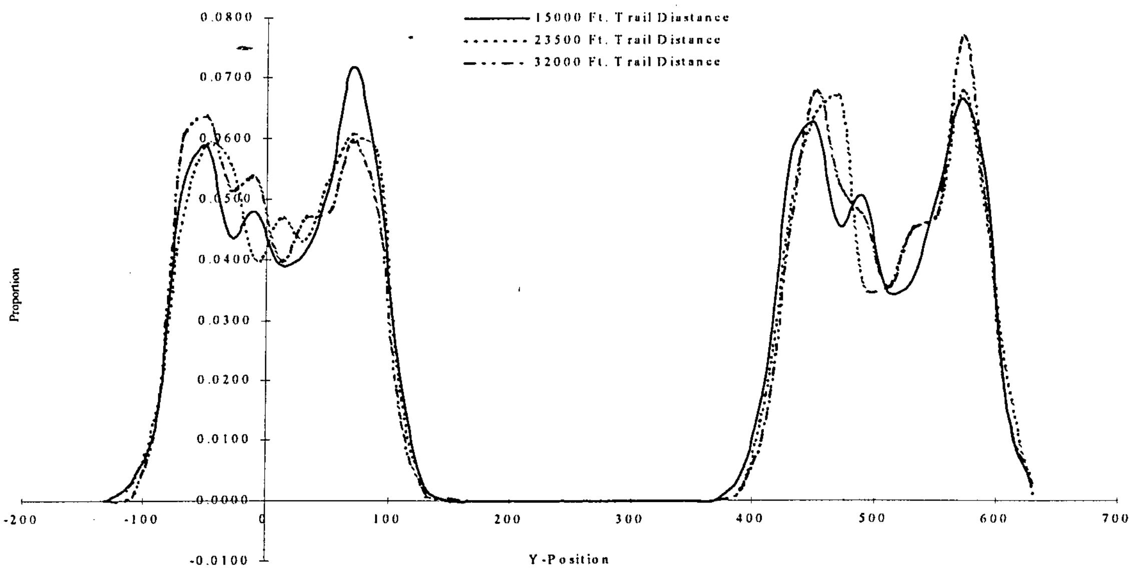

Figure 19. Dispersion Distribution Across DZ With Lateral Separation of 500 Feet

Although the airborne commander does not use mean distance to AAs or DZ dispersal distribution directly, the commander wants to deliver the airborne units onto the DZ in the best configuration possible, which is, in part, a function of the mean distance and the distribution of the paratrooper dispersal (Klimack 1997). How this may affect operational planning relates to the selection of the AAs as to minimize the mean distance from the chosen AAs. In turn, mean distance to AAs and DZ dispersion are both functions, in part, of aircraft formation geometry. This does not suggest changes to the ASOP. It does, however, provide a tool with which commanders can consider both paratrooper safety regarding encounter rates with wake vortices and DZ assembly in using specific operational formations from which to jump.

5. CONCLUSIONS

5.1 Overview

The utility of object-oriented programming is demonstrated with the modeling of the complex airborne operations environment. Taking a *mechanistic* view of the activities involved in airborne operations, a more penetrating look into a *systems* aspect of relational interactions between entities involved in the airborne activities is possible. This Thesis effort with the parallel efforts of Petry (1997) has done just that. In translating an established 6-DOF trajectory model into an object and transforming that object to reflect the aerodynamic characteristics of a combat equipped paratrooper, the *mechanistic* breakdown is completed at the level of resolution desired. In approaching the *systems* view, these paratrooper objects are integrated with the Petry C-17 aircraft and vortex objects into a simulation model that allows for a natural hierarchy in a system build-up.

5.2 Conclusions

In applying the simulation model to a simplified airborne operation, three airborne rifle companies are used to investigate the effects that trail and lateral distances have on (1) mean distance to AAs and (2) dispersion distribution on the DZ. In both cases, trail distances have little influence on the MOEs whereas lateral distances directly affect them. Depending on where the AAs are chosen, mean distance to the AA decreases as lateral separation increases if the AA is chosen on the same side of where separation is towards.

At the same time, it increases the variance of the mean distance because lateral separation scatters the paratroops in a wider path across the width of the DZ. These results are in light of the one major assumption that the paratrooper object is a non-intelligent payload. For the airborne commander, this translates into the operational parameters of aircraft formation geometries when also considering encounter rates between the paratroopers and wake vortices of upstream aircraft. This balancing act of finding an operational aircraft formation which minimizes both the encounter rate and the mean distance to AAs has a direct influence on the type of aircraft formation to implement in addition to all other factors which the airborne commander considers.

5.3 Recommendations and Future Development

The limitations of this model can be used as the starting point for future development and improvements on the objects used. A distribution can be found for inter-exit times for the paratrooper objects. Along with this, however, is the change in the update method of the Petry aircraft object for its positioning. A time delayed wind effect can be incorporated to reduce the overly conservative influence on trajectory which head/cross winds have. Lastly, an object that defines ground terrain and elevation can be incorporated and made to interact with the paratrooper objects to enhance resolution by taking into account natural land formations such as DZs with tree lines. The limitation on the dynamics of paratrooper/wake vortex interactions in terms of how they occur and the resulting effects is best left for aerospace engineers. However, when the resources become available to feasibly study the dynamics of paratrooper/wake vortex interaction, an area of interest which needs to be investigated is the effects the interactions may have on the variation in the paratrooper dispersal distributions on the DZ and then interpret these findings into operational parameters which can be easily applied by the airborne commander.

One aspect of object-oriented simulation that can be applicable to future developments is the idea of inheritance. These paratrooper objects are modeled specifically with the T-10C. An upgrade to the T-10C is being developed, called the Advanced Tactical Parachute System (ATPS), which may cause the re-testing of formation geometries to investigate whether encounter rates change with the change of parachute performance. With inheritance, a new paratrooper object can be developed, inheriting everything from the old paratrooper object but with the aerodynamic characteristics and performance of ATPS.

Creating a graphical user interface (GUI) for the simulation model can improve the model's utility. A GUI can also facilitate the understanding of the dynamics involved in the modeling of the object interactions. However, if rotational information is needed, the original propagation time step problem must be revisited. This addition to the simulation model is already being considered with independent efforts being pursued outside of this Thesis.

5.4 Summary

Without trivializing the complexities with not just the aerodynamics involved with modeling a parachute/payload system but also with the interactions of the parachute/payload system with its environment as in airdrop operations, a simplistic rendition of this airborne world is achieved using object-oriented simulation. With the original purpose of this Thesis effort having been met (to provide the C-17 test and evaluation community with the capability to assess paratrooper performance during C-17 airdrop formations), it is applied to a simplified scenario that parallels real world airborne operations to demonstrate the simulation model's capabilities and applicability. When combined with the findings of the Petry investigation into the paratrooper/wake vortex encounter rates, the airborne commander can have better insight into the environment surrounding the operations of the airborne units.

APPENDIX A

Purvis FORTRAN 6-DOF Code

```
C
C.....SIX DEGREE OF FREEDOM FLICHT SIMULATION
C.....    DIRECTION COSINE-EULER AXES METHOD
C
C     SINGLE BODY - CHUTE DECELERATION PROGRAM
C
C     FOREBODY DRAG VS MACH NO. VERSION
C
C.....EXECUTIVE ROUTINE
C
C.....SOURCE : DR. J. PURVIS
C              CCG-COURSE F12.01
C              MUENCHEN, 1987
C
C     MODIFIED BY DR. K.-F. DOHERR
C     FOR IBM-COMPATIBLE PC
C     IBM PROFESSIONAL FORTRAN WAS USED FOR COMPILATION
C     8087 MATH-COPROCESSOR IS NEEDED
C
C     INPUT FROM DIKFILE SIXD.DAT EXPECTED
C     OUTPUT ON DISKFILE SIXD.GRF AND ON UNIT 6 = SCREEN
C.....
      REAL MASS,IN,JN,MACH
      CHARACTER*1 DUM
      COMMON/INERT/MASS,XCG,XBOD,IN(3,3),JN(3,3)
      COMMON/AEROS/CBAR,SAREA,MACH
      COMMON/SUBS/ XE(3),UE(3),WB(3),B(3,3),VWIND(3)
      COMMON/BURN/ T
      COMMON/CHUTE/ IPTS,PCDS(50),PT(50)
      COMMON/FBDRAG/ MPTS,PCDF(50),PM(50)
      COMMON/AEROCF/ CNA,CNA2,CNQ,CYB,CYB2,CYR,CA0,CAA2
      COMMON/AEROCM/ CL0,CLP,CM0,CMA,CMA2,CMQ,CNB,CNB2,CNR
      COMMON/PROG/ DT,DTPR,TEND,HMIN
      COMMON/REF/ H,SOUND,RHO,RHOZ
  201 FORMAT(F10.3)
  202 FORMAT(A1)
  203 FORMAT(29X,I2)
  204 FORMAT(6X,F10.2,2X,F10.2)
  205 FORMAT(27X,F10.2)
C
C.....OPEN FILES FOR INPUT AND OUTPUT
C
C.....UNIT 11 = INPUT FILE  = SIXD.DAT
C.....UNIT  6 = OUTPUT FILE = CON         TABULATED OUTPUT
C.....UNIT 12 = OUTPUT FILE = SIXD.GRF    ASCII FILE
C
      OPEN (UNIT = 11, FILE = 'SIXD.DAT', STATUS = 'OLD')
      OPEN (UNIT = 12, FILE = 'SIXD.GRF')
C
C.....INPUT SYSTEM INERTIAL PROPERTIES
C
      CALL HEAD
      READ(11,201) WEIGHT
      MASS=WEIGHT/32.17
      READ(11,201) XCG
      READ(11,201) XBOD
      DO 19 J=1,3
      DO 19 L=1,3
      JN(J,L)=0.
      IN(J,L)=0.
   19 CONTINUE
      READ(11,201) IN(1,1)
      READ(11,201) IN(2,2)
      READ(11,201) IN(3,3)
      DO 2 I=1,3
      JN(I,I)=1./IN(I,I)
    2 CONTINUE
C
```

```
C.....INPUT INITIAL CONDITIONS
C
      DO 22 I=1,3
      XE(I)=0.
      UE(I)=0.
      WB(I)=0.
      VWIND(I)=0.
   22 CONTINUE
      CALL HEAD
      READ(11,201) ALT
      READ(11,201) HMIN
      READ(11,201) UE(1)
      READ(11,201) UE(3)
      READ(11,201) THETA
      READ(11,201) VWIND(1)
      READ(11,201) VWIND(2)
      READ(11,201) DENS
      RHOZ=0.002378
      IF(DENS.EQ.0.) GOTO 5
      RHOZ=DENS*EXP(ALT/23111.-.295*SIN(ALT/28860.)-.213*
     $      SIN(ALT/86580.))
    5 CONTINUE
C
C.....INPUT PROGRAM CONSTANTS
C
      CALL HEAD
      READ(11,201) DT
      READ(11,201) DTPR
      READ(11,201) TEND
C
C.....INPUT FOREBODY AERODYNAMIC COEFFICIENTS
C
      CALL HEAD
      READ(11,201) CBAR
      READ(11,201) SAREA
      READ(11,201) CNA
      READ(11,201) CYB
      READ(11,201) CAA2
      READ(11,201) CL0
      READ(11,201) CLP
      READ(11,201) CMA
      READ(11,201) CMQ
      READ(11,201) CNB
      READ(11,201) CNR
C
C.....INPUT FOREBODY DRAG VS MACH NO. TABLE
C
      CALL HEAD
      READ (11,203) MPTS
      READ (11,202) DUM
      DO 1 I=1,MPTS
      READ (11,204) PM(I),PCDF(I)
    1 CONTINUE
C
C.....INPUT PARACHUTE DRAG-AREA VS TIME TABLE
C
      CALL HEAD
      READ (11,205) DEPTIME
      READ (11,203) IPTS
      I1=1
      IF(DEPTIME.LT.0.) DEPTIME=0.
      IF(DEPTIME.LE.0.) GOTO 4
      I1=3
      IPTS=IPTS+2
      PT(1)=0.
      PT(2)=DEPTIME
      PCDS(1)=0.
      PCDS(2)=0.
    4 READ (11,202) DUM
      DO 3 I=I1,IPTS
      READ (11,204) PT(I),PCDS(I)
      PT(I)=PT(I)+DEPTIME
    3 CONTINUE
```

```
                                DO 999 I = 1,4
                              WRITE (*,*) PCDS(I)
                               999 CONTINUE
C                                                                       00012100
C.....CONVERT EULER ANGLES TO DERECTION COSINES                          00012200
C                                                                       00012300
      PSI=0.                                                            00012400
      PHI=0.                                                            00012500
      RAD=1./57.295                                                     00012600
      ST=SIN(THETA*RAD)                                                 00012700
      CT=COS(THETA*RAD)                                                 00012800
      SP=SIN(PSI*RAD)                                                   00012900
      CP=COS(PSI*RAD)                                                   00013000
      SPHI=SIN(PHI*RAD)                                                 00013100
      CPHI=COS(PHI*RAD)                                                 00013200
      XE(3)=-ALT                                                        00013300
      B(1,1)=CP*CT                                                      00013400
      B(1,2)=SP*CT                                                      00013500
      B(1,3)=-ST                                                        00013600
      B(2,1)=-SP*CPHI+CP*ST*SPHI                                        00013700
      B(2,2)=CP*CPHI+SP*ST*SPHI                                         00013800
      B(2,3)=CT*SPHI                                                    00013900
      B(3,1)=SP*SPHI+CP*ST*CPHI                                         00014000
      B(3,2)=-CP*SPHI+SP*ST*CPHI                                        00014100
      B(3,3)=CT*CPHI                                                    00014200
C                                                                       00014300
C.....TRAJECTORY SIMULATION                                             00014400
C                                                                       00014500
      CALL TRAJEC                                                       00014600
C                                                                       00014610
      CLOSE (11)                                                        00014621
      CLOSE (12)                                                        00014622
C                                                                       00014623
      STOP                                                              00014700
      END                                                               00014800
      SUBROUTINE HEAD                                                   00014900
      CHARACTER*1 DUM                                                   00015000
      DO 1 I=1,5                                                        00015100
      READ (11,202) DUM                                                 00015200
  202 FORMAT(A1)                                                        00015300
    1 CONTINUE                                                          00015400
      RETURN                                                            00015500
      END                                                               00015600
      SUBROUTINE TRAJEC                                                 00015700
C                                                                       00015800
C.....EQUATIONS OF MOTION - TRAJECTORY SIMULATION                       00015900
C                                                                       00016000
      REAL MASS,IN,JN,MACH,MB                                           00016100
      INTEGER E                                                         00016200
      COMMON/INERT/MASS,XCG,XBOD,IN(3,3),JN(3,3)                        00016300
      COMMON/AEROS/ CBAR,SAREA,MACH                                     00016400
      COMMON/SUBS/ XE(3),UE(3),WB(3),B(3,3),VWIND(3)                    00016500
      COMMON/BURN/ T                                                    00016600
      COMMON/AEROF/ CN,CY,CA,CSL,CM,CSN,CDS                             00016700
      COMMON/AEROV/ VPO,SALP,CALP,SBET,CBET,PB,QB,RB                    00016800
      COMMON/PROG/ DT,DTPR,TEND,HMIN                                    00016900
      COMMON/REF/ H,SOUND,RHO,RHOZ                                      00017000
      DIMENSION BN(3,3),TEMP(3),E(5),DEL(3,3)                           00017100
      DIMENSION FB(3),MB(3),FE(3),UEDOT(3),WBDOT(3),BDOT(3,3),HB(3)     00017200
      DATA E/1,2,3,1,2/,DEL/1.,0.,0.,0.,1.,0.,0.,0.,1./                 00017300
  200 FORMAT(1H ,21X,4HDOWN,6X,3HOFF,25X,4HMACH,3X,7HDYNAMIC,           00017400

     $        3X,5HAXIAL,3X,10HTRAJECTORY,3X,5HPITCH,12X,               00017500
     $        4HDRAG)                                                   00017600
  201 FORMAT(1H ,1X,4HTIME,3X,8HALTITUDE,4X,5HRANGE,5X,5HRANGE,         00017700
     $        3X,8HVELOCITY,2X,8HAIRSPEED,                              00017800
     $        2X,6HNUMBER,2X,8HPRESSURE,2X,                             00017900
     $        6HACCEL.,4X,5HANGLE,6X,5HANGLE,3X,5HALPHA,4X,             00018000
     $        4HAREA)                                                   00018100
  202 FORMAT(1H ,1X,5H(SEC),4X,4H(FT),7X,4H(FT),6X,4H(FT),5X,           00018200
     $        5H(FPS),5X,5H(FPS),13X,                                   00018300
     $        5H(PSF),3X,6H(GEES),4X,5H(DEG),6X,                        00018400
     $       5H(DEG),3X,5H(DEG),2X,8H(SQ.FT.))
  203 FORMAT(1H ,F6.2,1X,5(F8.1,2X),F6.2,2X,F8.1,2X,F6.1,4X,
     $       F6.1,4X,F6.1,2X,F6.1,2X,F8.2)
C
C.....FIXED INITIAL CONDITIONS AND CONSTANTS
C
      GEES=0.
      CDS=0.
      T=0.
      GMAX=0.
      IEND=0
      TPR=T-DT
      TEND=TEND-0.1*DT
      ALT=-XE(3)
      H=ALT
      WRITE(12,505) TEND,HMIN,ALT
      WRITE(12,505) UE(1),UE(3),VWIND(2)
      WRITE(6,200)
      WRITE(6,201)
      WRITE(6,202)
C
C.....BEGIN TRAJECTORY LOOP
C
    1 CONTINUE
      G=GRAV(H)
      CALL DENS
C
C.....COMPUTE AERO. VARIABLES
C
      PB=WB(1)
      QB=WB(2)
      RB=WB(3)
      UE1=UE(1)-VWIND(1)
      UE2=UE(2)-VWIND(2)
      UE3=UE(3)-VWIND(3)
      VP=SQRT(UE1**2+UE2**2+UE3**2)
      VP0=VP
      MACH=VP/SOUND
      UB1=B(1,1)*UE1+B(1,2)*UE2+B(1,3)*UE3
      UB2=B(2,1)*UE1+B(2,2)*UE2+B(2,3)*UE3
      UB3=B(3,1)*UE1+B(3,2)*UE2+B(3,3)*UE3
      VP13=SQRT(UB1**2+UB3**2)
C
C.....USE SIN(ALPHA) FOR ALPHA AND SIN(BETA) FOR BETA
C
      IF(VP0.LT.1.E-6) VP0=1.E-6
      SBET=UB2/VP0
      CBET=VP13/VP0
      BETA=SBET
      IF(VP13.LT.1.E-6) VP13=1.E-6
      SALP=UB3/VP13
      CALP=UB1/VP13
      ALPHA=SALP
C
C.....AERODYNAMIC AND BODY FORCES AND MOMENTS
C
C.....ISOLATED BODY AERODYNAMICS
C
      CALL AERO
      Q=0.5*RHO*VP
      FPC=-Q*CDS
      QS=.5*RHO*VP*VP*SAREA
      QSD=QS*CBAR
      FB(1)=-QS*CA+MASS*G*B(1,3)+FPC*UB1
      FB(2)=QS*CY+MASS*G*B(2,3)+FPC*UB2
      FB(3)=-QS*CN+MASS*G*B(3,3)+FPC*UB3
      MB(1)=QSD*CSL
      MB(2)=QSD*CM+FPC*UB3*(XBOD-XCG)
      MB(3)=QSD*CSN-FPC*UB2*(XBOD-XCG)
      GEES=-FB(1)/(MASS*G)
      IF(ABS(GEES).GT.ABS(GMAX)) GMAX=GEES
C
C.....PRINT TRAJECTORY DATA
```

```
C
      IF(T.LT.TPR) GO TO 14
      TPR=TPR+DTPR
   30 CONTINUE
      H=-XE(3)
      VPE=SQRT(UE(1)**2+UE(2)**2+UE(3)**2)
      QDYN=0.5*RHO*VP*VP
      BXY=SQRT(B(1,1)**2+B(1,2)**2)
      THETA=57.295*ATAN2(-B(1,3),BXY)
      ALPHAD=57.295*ATAN2(SALP,CALP)
      UXY=SQRT(UE(1)**2+UE(2)**2)
      GAMMAD=57.295*ATAN2(-UE(3),UXY)
C      WRITE(12,505) T,H,XE(1),XE(2),VPE,VP,MACH,QDYN,GEES,
C     $          GAMMAD,THETA,ALPHAD,CDS
C      WRITE(6,203) T,H,XE(1),XE(2),VPE,VP,MACH,QDYN,GEES,
C     $             GAMMAD,THETA,ALPHAD,CDS
      IF(IEND.NE.0) GO TO 31
   14 CONTINUE
C
C.....EULER ROTATION FUNCTION FOR DIRECTION COSINE PROPAGATION
C
      W2=WB(1)**2+WB(2)**2+WB(3)**2
      W=SQRT(W2)
      COSWT=COS(W*DT)
      SINWT=SIN(W*DT)
      COSWTM=1.-COSWT
      IF(W2.GT.1.E-12) GO TO 22
      W2=1.E-12
      W=1.E-6
   22 CONTINUE
C
C.....ANGULAR MOMENTUM CROSS PRODUCT TERMS
C
      DO 20 K=1,3
      HB(K)=IN(K,1)*WB(1)+IN(K,2)*WB(2)+IN(K,3)*WB(3)
   20 CONTINUE
      DO 21 I=1,3
      I1=E(I+1)
      I2=E(I+2)
      TEMP(I)=WB(I1)*HB(I2)-WB(I2)*HB(I1)
   21 CONTINUE
C
C.....FORCE RESOLUTIONS TO EULER SYSTEM
C     TRANSLATIONAL ACCELERATIONS AND DERECTION COSINE ROTATION
C
      DO16 I=1,3
      FE(I)=FB(1)*B(1,I)+FB(2)*B(2,I)+FB(3)*B(3,I)
      UEDOT(I)=FE(I)/MASS
      DO 17 J=1,3
      BN(I,J)=B(I,J)
      J1=E(J+1)
      J2=E(J+2)
      BDOT(I,J)=DEL(I,J)*COSWT+WB(I)*WB(J)*COSWTM/W2+
     $  (WB(J1)*DEL(I,J2)-WB(J2)*DEL(I,J1))*SINWT/W
   17 CONTINUE
C
C.....ANGULAR ACCELERATIONS IN BODY AXES
C
      WBDOT(I)=JN(I,1)*(MB(1)-TEMP(1))
     $+JN(I,2)*(MB(2)-TEMP(2))
     $+JN(I,3)*(MB(3)-TEMP(3))
   16 CONTINUE
C
C.....INTEGRALS
C
      T=T+DT
      DO 11 I=1,3
      XE(I)=XE(I)+DT*(UE(I)+0.5*DT*UEDOT(I))
      UE(I)=UE(I)+DT*UEDOT(I)
      WB(I)=WB(I)+DT*WBDOT(I)
      DO 11 J=1,3
      B(I,J)=BDOT(I,1)*BN(1,J)+BDOT(I,2)*BN(2,J)+BDOT(I,3)*BN(3,J)
   11 CONTINUE
```

```
      IF(T.GE.TEND) GO TO 15
C
C.....END ACCELERATION LOOP - TEST FOR GROUND IMPACT
C
      H=-XE(3)
      IF(H.GT.HMIN) GO TO 1
   15 CONTINUE
      IEND=1
      GO TO 30
   31 CONTINUE
      T=-999.
      WRITE(12,505) T,GMAX
  505 FORMAT(13E10.4)
   98 RETURN
      END
      SUBROUTINE AERO
      COMMON/INERT/MASS,XCG,XBOD,IN(3,3),JN(3,3)
      COMMON/BURN/ T
      COMMON/AEROS/ CBAR,SAREA,MACH
      COMMON/AEROV/ VP0,SALP,CALP,SBET,CBET,PB,QB,RB
      COMMON/AEROF/ CN,CY,CA,CSL,CM,CSN,CDS
      COMMON/AEROCF/ CNA,CNA2,CNQ,CYB,CYB2,CYR,CA0,CAA2
      COMMON/AEROCM/ CL0,CLP,CM0,CMA,CMA2,CMQ,CNB,CNB2,CNR
      COMMON/CHUTE/ IPTS,PCDS(50),PT(50)
      COMMON/FBDRAG/ MPTS,PCDF(50),PM(50)
      REAL MASS,IN,JN,MACH
C
C.....AERO VARIABLES
C
      SAC=SALP*CALP
      SAS=SALP*ABS(SALP)
      SBC=SBET*CBET
      SBS=SBET*ABS(SBET)
      RAD=CBAR/(2.*VP0)
      CNA2=0.
      CNQ=0.
      CYB2=0.
      CYR=0.
      CM0=0.
      CMA2=0.
      CNB2=0.
C
C.....FOREBODY ZERO-LIFT DRAG COEFFICIENT
C
      I=0
    6 I=I+1
      IP=I+1
      IF(I.EQ.MPTS) GO TO 5
      IF(MACH.GT.PM(IP)) GO TO 6
      CA0=PCDF(I)+(PCDF(IP)-PCDF(I))*(MACH-PM(I))/(PM(IP)-PM(I))
      GO TO 4
    5 CA0=PCDF(MPTS)
    4 CONTINUE
C
C.....FORCE AND MOMENT COEFFICIENTS
C
      CN=CNA*SAC+CNA2*SAS+CNQ*QB*RAD
      CY=CYB*SBC+CYB2*SBS+CYR*RB*RAD
      CA=CA0+CAA2*(1.-CALP**2*CBET**2)
      CSL=CL0+CLP*PB*RAD
      CM=CM0+CMA*SAC+CMA2*SAS+CMQ*QB*RAD
      CSN=CNB*SBC+CNB2*SBS+CNR*RB*RAD
C
C.....PARACHUTE DRAG-AREA
C
      CDS=0.
      IF(T.LT.PT(1)) GO TO 1
      I=0
    3 I=I+1
      IP=I+1
      IF(I.EQ.IPTS) GO TO 2
      IF(T.GT.PT(IP)) GO TO 3
      CDS=PCDS(I)+(PCDS(IP)-PCDS(I))*(T-PT(I))/(PT(IP)-PT(I))
```

```
                 WRITE (*,*) T, " ", I, " ", IP, " ", CDS, " T<PT(IP)"
      GO TO 1
    2 CDS=PCDS(IPTS)
    1 CONTINUE
      RETURN
      END
      SUBROUTINE DENS
C
C.....DENSITY AND SPEED OF SOUND VS. ALTITUDE - DOMMASCH EQUATION
C
      COMMON/REF/ H,SOUND,RHO,RHOZ
      RHO=RHOZ*EXP(-H/23111.+.294*SIN(H/28860.)+.213*SIN(H/86580.))
      IF(H.GT.0) GO TO 7
      SOUND=1116.44
      RETURN
    7 IF(H.GT.36152) GO TO 1
      T=518.688-(3.56616E-03)*H
      SOUND=49.02118*SQRT(T)
      RETURN
    1 IF(H.GT.82345) GO TO 2
      SOUND=968.08
      RETURN
    2 IF(H.GT.155348) GO TO 3
      T=254.988+1.64592E-03*H
      SOUND=49.02118*SQRT(T)
      RETURN
    3 IF(H.GT.175346) GO TO 4
      SOUND=1105.7
      RETURN
    4 IF(H.GT.262448) GO TO 5
      T=988.088-2.46888E-03*H
      SOUND=49.02118*SQRT(T)
      RETURN
    5 IF(H.GT.299516) GO TO 6
      SOUND=846.9
      RETURN
    6 T=-349.812+2.19456E-03*H
      SOUND=49.02118*SQRT(T)
      RETURN
      END
      FUNCTION GRAV(H)
      RE=20855531.5
      GRAV=32.1741*(RE/(H+RE))**2
      RETURN
      END
```

Input Parameters A

```
**********************************************************************
**********                                        ********************
********** SYSTEM PARAMETERS                      ********************
**********                                        ********************
**********************************************************************
88.18     SYSTEM WEIGHT (LBS)                     WEIGHT
1.3125    FOREBODY C.G. (FT)                      XCG
2.625     FOREBODY LENGTH (FT)                    XBOD
.1475     ROLL INERTIA IXX (SLUG-FT**2)           IN(1,1)
1.584     PITCH INERTIA IYY (SLUG-FT**2)          IN(2,2)
1.584     YAW INERTIA IZZ (SLUG-FT**2)            IN(3,3)
**********************************************************************
**********                                        ********************
********** INITIAL CONDITIONS                     ********************
**********                                        ********************
**********************************************************************
328.1     ALTITUDE (FT)                           ALT
0.000     MINIMUM ALTITUDE OR GROUND LEVEL (FT)   HMIN
484.65    HORIZONTAL VELOCITY (FPS)               UE(1)
-85.46    EJECTION VELOCITY (FPS) POSITIVE DOWN   UE(3)
15.00     PITCH ANGLE (DEG) NOSE UP POSITIVE      THETA
0.000     HEAD(+) OR TAIL(-) WIND (FPS)           VWIND(1)
0.000     CROSSWIND (FPS)                         VWIND(2)
0.000     DENSITY (0. FOR STD. ATMS.) (SLUG/FT**3)DENSITY
**********************************************************************
**********                                        ********************
********** PROGRAMM CONSTANTS                     ********************
**********                                        ********************
**********************************************************************
0.0005    INTEGRATION TIME STEP (SEC)             DT
0.10      PRINT INTERVAL (SEC)                    DTPR
9.0       MAXIMUM TIME OF FLIGHT (SEC)            TEND
**********************************************************************
**********                                        ********************
********** FOREBODY AERODYNAMIC COEFFICIENTS       ********************
**********                                        ********************
**********************************************************************
.6562     REFERENCE LENGTH (FT) Durchmesser       CBAR
.3382     REFERENCE AREA (SQ.FT.) Querschnitt     SAREA
2.780     NORMAL FORCE CN-ALPHA (/RAD)            CNA
0.000     SIDE FORCE CY-BETA (/RAD)               CYB
0.000     AXIAL FORCE CA-ALPHA**2 (/RAD**2)       CAA2
0.000     ROLL TORQUE COEFFICIENT (NONDIM.)       CL0
0.000     ROLL DAMPING COEFFICIENT (/RAD)         CLP
1.11      PITCH MOMENT CM-ALPHA (/RAD)            CMA
-10.0     PITCH DAMPING (/RAD)                    CMQ
0.000     YAW MOMENT CN-BETA (/RAD)               CNB
0.000     YAW DAMPING (/RAD)                      CNR
**********************************************************************
**********                                        ********************
********** FOREBODY DRAG VS MACH NUMBER           ********************
**********                                        ********************
**********************************************************************
     NUMBER OF TABLE INPUTS:  2
NO.  MACH NO.    DRAG COEFFICIENT
 1   0.00        1.00
 2   1.00        1.00
**********************************************************************
**********                                        ********************
********** PARACHUTE DRAG-AREA VS TIME            ********************
**********                                        ********************
**********************************************************************
     DEPLOYMENT TIME(SEC): 0.25
     NUMBER OF TABLE INPUTS:  2      NOTE: DRAG-AREA VS. TIME IS
NO.  TIME(SEC)   DRAG-AREA(SQ.FT.)   RELATIVE TO START OF DEPLOYMENT.
 1   0.00        0.3382
 2   0.10        10.333
```

Input Parameters B

```
**********************************************************************
**********                                        ********************
********** SYSTEM PARAMETERS                      ********************
**********                                        ********************
**********************************************************************
500.000   SYSTEM WEIGHT (LBS)                     WEIGHT
4.000     FOREBODY C.G. (FT)                      XCG
10.000    FOREBODY LENGTH (FT)                    XBOD
5.000     ROLL INERTIA IXX (SLUG-FT**2)           IN(1,1)
100.000   PITCH INERTIA IYY (SLUG-FT**2)          IN(2,2)
100.000   YAW INERTIA IZZ (SLUG-FT**2)            IN(3,3)
**********************************************************************
**********                                        ********************
********** INITIAL CONDITIONS                     ********************
**********                                        ********************
**********************************************************************
100.000   ALTITUDE (FT)                           ALT
0.000     MINIMUM ALTITUDE OR GROUND LEVEL (FT)   HMIN
500.000   HORIZONTAL VELOCITY (FPS)               UE(1)
0.000     EJECTION VELOCITY (FPS) POSITIVE DOWN   UE(3)
0.000     PITCH ANGLE (DEG) NOSE UP POSITIVE      THETA
0.000     HEAD(+) OR TAIL(-) WIND (FPS)           VWIND(1)
0.000     CROSSWIND (FPS)                         VWIND(2)
0.000     DENSITY (0. FOR STD. ATMS.) (SLUG/FT**3)DENSITY
**********************************************************************
**********                                        ********************
********** PROGRAMM CONSTANTS                     ********************
**********                                        ********************
**********************************************************************
0.005     INTEGRATION TIME STEP (SEC)             DT
0.500     PRINT INTERVAL (SEC)                    DTPR
10.000    MAXIMUM TIME OF FLIGHT (SEC)            TEND
**********************************************************************
**********                                        ********************
********** FOREBODY AERODYNAMIC COEFFICIENTS       ********************
**********                                        ********************
**********************************************************************
10.000    REFERENCE LENGTH (FT)                   CBAR
1.000     REFERENCE AREA (SQ.FT.)                 SAREA
0.000     NORMAL FORCE CN-ALPHA (/RAD)            CNA
0.000     SIDE FORCE CY-BETA (/RAD)               CYB
0.000     AXIAL FORCE CA-ALPHA**2 (/RAD**2)       CAA2
0.000     ROLL TORQUE COEFFICIENT (NONDIM.)       CL0
0.000     ROLL DAMPING COEFFICIENT (/RAD)         CLP
-2.000    PITCH MOMENT CM-ALPHA (/RAD)            CMA
-200.000  PITCH DAMPING (/RAD)                    CMQ
0.000     YAW MOMENT CN-BETA (/RAD)               CNB
0.000     YAW DAMPING (/RAD)                      CNR
**********************************************************************
**********                                        ********************
********** FOREBODY DRAG VS MACH NUMBER           ********************
**********                                        ********************
**********************************************************************
     NUMBER OF TABLE INPUTS:  2
NO.   MACH NO.    DRAG COEFFICIENT
 1    0.00        0.09
 2    2.00        0.09
**********************************************************************
**********                                        ********************
********** PARACHUTE DRAG-AREA VS TIME            ********************
**********                                        ********************
**********************************************************************
     DEPLOYMENT TIME(SEC): 0.20
     NUMBER OF TABLE INPUTS:  2       NOTE: DRAG-AREA VS. TIME IS
NO.   TIME(SEC)   DRAG-AREA(SQ.FT.)   RELATIVE TO START OF DEPLOYMENT.
 1    0.00        0.20
 2    1.00        500.00
```

APPENDIX B

MODSIM III Translation of Purvis 6-DOF Model

```
MAIN MODULE sixDOF;

FROM globalMod IMPORT jumper;
FROM globalMod IMPORT initializeData;

BEGIN

     initializeData;

     NEW (jumper);
     ASK jumper TO jump;

END {MAIN} MODULE {6DOF}.
```

```
DEFINITION MODULE globalMod;

FROM jumperMod IMPORT jumperObj;

CONST

     re = 20855531.5;

TYPE

          eType      = ARRAY INTEGER OF INTEGER;
          delType    = ARRAY INTEGER, INTEGER OF REAL;
          matrixType = ARRAY INTEGER, INTEGER OF REAL;
          vectorType = ARRAY INTEGER OF REAL;

VAR

     i, j   : INTEGER;

     jumper : jumperObj;

     e      : eType;

     del    : delType;

PROCEDURE initializeData;

END {DEFINITION} MODULE {globalMod}.
```

```
IMPLEMENTATION MODULE globalMod;

PROCEDURE initializeData;
     BEGIN

     NEW (e, 1..5);
     NEW (del, 1..3, 1..3);

     e[1] := 1;
     e[2] := 2;
     e[3] := 3;
     e[4] := 1;
     e[5] := 2;

     FOR i := 1 TO 3
          FOR j := 1 TO 3
               IF i = j
                    del[i,j] := 1.0;
               ELSE
                    del[i,j] := 0.0;
               END IF;
          END FOR;
     END FOR;

     END PROCEDURE {initializeData};

END {IMPLEMENTATION} MODULE {globalMod}.
```

```
DEFINITION MODULE calcMod;

PROCEDURE gravCalc (IN a : REAL) : REAL;
PROCEDURE densityCalc (IN h, rhoz : REAL; OUT rho, sound : REAL);

END {DEFINITION} MODULE {calcMod}.
```

```
IMPLEMENTATION MODULE calcMod;

FROM MathMod IMPORT POWER, SIN, COS, SQRT, EXP;
FROM globalMod IMPORT re;

PROCEDURE gravCalc (IN a: REAL) : REAL;
     BEGIN
          RETURN 32.1741*POWER(re/(a+re), 2.0);
     END PROCEDURE {gravCalc};

PROCEDURE densityCalc (IN h, rhoz : REAL; OUT rho, sound : REAL);
     VAR

          t : REAL;

     BEGIN
          rho := rhoz * EXP(-1.0*h/23111.0 + 0.294 * SIN(h/28860.0) + 0.213 *
SIN(h/86580.0));

          IF h > 0.0
               t := 518.688 - (3.56616E-03)*h;
               sound :=  49.02118 * SQRT(t);
               IF h > 36152.0
                    sound := 968.08;
                    IF h > 82345.0
                         t := 254.988 + (1.64592E-03)*h;
                         sound := 49.02118 * SQRT(t);
                         IF h > 155348.0
                              sound := 1105.0;
                              IF h > 262448.0
                                   sound := 846.9;
                                   IF h > 299516.0
                                        t := -349.812 + 2.19456E-03*h
                                   END IF;
                              END IF;
                         END IF;
                    END IF;
               END IF;
          ELSE
               sound := 1116.44;
          END IF;

     END PROCEDURE {densityCalc};

END {IMPLEMENTATION} MODULE {calcMod}.
```

```
DEFINITION MODULE jumperMod;

FROM globalMod IMPORT matrixType, vectorType;

TYPE

     jumperObj = OBJECT

          i1,
          i2,
          iend,
          ipts,
          ip,
          j1,
          j2,
          k,
          loop,
          mpts,
          test   : INTEGER;

          alt,
          alpha,
          alphad,
          beta,
          bxy,
          ca,
          cao,
          caa2,
          calpha,
          cbar,
          cbeta,
          cby2,
          cds,
          clo,
          clp,
          cm,
          cma,
          cma2,
          cmo,
          cmq,
          cn,
          cna,
          cna2,
          cnb,
          cnb2,
          cnq,
          cnr,
          coswt,
          coswtm,
          cp,
          cphi,
          csl,
          csn,
          ct,
          cy,
          cyb,
          cyb2,
          cyr,
          dens,
          deptime,
          dt,
          dtpr,
          fpc,
          g,
          gammad,
          gees,
          gmax,
          h,
          hmin,
          mach,
          mass,
          pb,
          phi,
          psi,
          q,
          qb,
          qdyn,
          qs,
          qsd,
          rad,
          rb,
          rho,
          rhoz,
          sac,
          sas,
          sarea,
          salpha,
          sbc,
          sbeta,
          sbs,
          sinwt,
          sound,
          sp,
          phi,
          St.,
          t,
          tend,
          theta,
          tar,
          ub1,
          ub2,
          ub3,
          ue1,
          ue2,
          ue3,
          uxy,
          vp,
          vp13,
          vpe,
          vpo,
          w,
          w2,
          weight,
          xbod,
          xcg      : REAL;

          pcds,
          pt,
          pcdf,
          pm     : vectorType; {1X2}

          xe,
          ue,
          wb,
          vwind,
          temp,
          fb,
          m,
          mb,
          fe,
          uedot,
          wbdot,
          hb      : vectorType; {1X3}

          in,
          jn,
          b,
          bn,
          bdot : matrixType; {3X3}
```

```
ASK METHOD ObjInit;
ASK METHOD jump;
```

```
  END OBJECT {jumperObj};

END {DEFINITION} MODULE {jumperMod}.
```

```
 IMPLEMENTATION MODULE jumperMod;

 FROM MathMod   IMPORT EXP, SIN, COS, POWER, SQRT, ATAN2;
 FROM globalMod IMPORT re, e, del, i, j;
 FROM calcMod   IMPORT gravCalc, densityCalc;

 OBJECT jumperObj;

      ASK METHOD ObjInit;
      BEGIN

           NEW(pcdf, 1..2);
           NEW(pm  , 1..2);

           NEW(fb   , 1..3);
           NEW(fe   , 1..3);
           NEW(hb   , 1..3);
           NEW(mb   , 1..3);
           NEW(temp , 1..3);
           NEW(ue   , 1..3);
           NEW(uedot, 1..3);
           NEW(vwind, 1..3);
           NEW(wb   , 1..3);
           NEW(wbdot, 1..3);
           NEW(xe   , 1..3);

           NEW(pcds, 1..4);
           NEW(pt  , 1..4);

           NEW(in  , 1..3, 1..3);
           NEW(jn  , 1..3, 1..3);
           NEW(b   , 1..3, 1..3);
           NEW(bn  , 1..3, 1..3);
           NEW(bdot, 1..3, 1..3);

      { system inertial properties }

           weight := 88.18;         { parachute-payload system weight }
           mass   := weight/32.17;
           xcg    := 1.3125;        { forebody c.g. (ft) }
           xbod   := 2.625;         { forebody length (ft) }

           FOR i := 1 TO 3
                FOR j := 1 TO 3
                     jn[i,j] := 0.0;
                     in[i,j] := 0.0;
                END {j} FOR;
           END {i} FOR;

           in[1,1] := 0.1475; { roll  inertia Ixx (slug-ft^2) }
           in[2,2] := 1.584;  { pitch inertia Iyy (slug-ft^2) }
           in[3,3] := 1.584;  { yaw   inertia Izz (slug-ft^2) }

           FOR i := 1 TO 3
                jn[i,i] := 1.0 / in [i,i];
           END FOR;

      { initial conditions }

           FOR i := 1 TO 3
                xe[i]    := 0.0; { (ft)  1: down range, 2: off range, 3: altitude loss}
                ue[i]    := 0.0; { (fps) 1: horizontal velocity, 2: lateral velocity, 3:
ejection velocity positive down}
                wb[i]    := 0.0; { ??? }
                vwind[i] := 0.0; { (fps) 1: head (+) or tail (-) wind, 2: crosswind, 3:
???? }
           END FOR;

           alt       := 328.1;    {altitude (ft) }
           hmin      := 0.0;      {ground level (ft) }
           ue[1]     := 484.65;
           ue[3]     := -85.46;
           theta     := 15.00;    {pitch angle (deg) nose up positive}
```

```
          vwind[1]  := 0.0;
          vwind[2]  := 0.0;
          dens      := 0.0;      {density (0 for standard atms) in slug/ft^3}
          rhoz      := 0.002378; {????}

          IF dens <> 0.0
               rhoz := dens * EXP (alt/23111.0 - 0.295 * SIN(alt/28860.0) - 0.213 *
SIN(alt/86580.0))
          END IF;

     { program constants }

          dt   := 0.0005; { integration time step (sec) }
          dtpr := 0.1;    { print interval (sec) }
          tend := 9.0;    { max time for flight (sec) }

     { forebody aerodynamic coefficients }

          cbar  := 0.6562; { reference length (ft) }
          sarea := 0.3382; { reference area (ft^2) }
          cna   := 2.78;   { normal force cn-alpha (/rad) }
          cyb   := 0.0;    { side force cy-beta (/rad) }
          caa2  := 0.0;    { axial force ca-alpha^2 (/rad^2) }
          clo   := 0.0;    { roll torque coefficient (dimensionless) }
          clp   := 0.0;    { roll damping coefficient (/rad) }
          cma   := 1.11;   { pitch moment cm-alpha (/rad) }
          cmq   := -10.0;  { pitch damping (/rad) }
          cnb   := 0.0;    { yaw moment cn-beta (/rad) }
          cnr   := 0.0;    { yaw damping (/rad) }

     { forebody drag versus mach number table }

          mpts    := 2;
          pm[1]   := 0.00; { mach number }
          pm[2]   := 1.00;
          pcdf[1] := 1.00; { drag coefficient }
          pcdf[2] := 1.00;

     { parachute drag-area versus time table }

          deptime := 0.25; { deployment time }
          ipts    := 2;

          IF deptime < 0.0
               deptime := 0.0;
               ipts := ipts + 2; { ipts = 4 }
               pt[1]   := 0.0;
               pt[2]   := deptime;
               pt[3]   := 0.00 + deptime;
               pt[4]   := 0.10 + deptime;
               pcds[1] := 0.0;
               pcds[2] := 0.0;
               pcds[3] := 0.3382;
               pcds[4] := 10.333;
               IF deptime <= 0.0
                    pt[1] := 0.0 + deptime;
                    pt[2] := 0.10 + deptime;
                    pcds[1] := 0.3382;
                    pcds[2] := 10.333;
               END IF;
          END IF;

          IF deptime > 0.0
               ipts := ipts + 2; { ipts = 4 }
               pt[1]   := 0.0;
               pt[2]   := deptime;
               pt[3]   := 0.00 + deptime;
               pt[4]   := 0.10 + deptime;
               pcds[1] := 0.0;
               pcds[2] := 0.0;
               pcds[3] := 0.3382;
               pcds[4] := 10.333;
          END IF;
```

```
{ convert EULER_ANGLES to di-rection cosines }

      psi     := 0.0;
      phi     := 0.0;
      rad     := 1.0/57.295;
      St.      := SIN (theta*rad);
      ct      := COS (theta*rad);
      sp      := SIN (psi*rad);
      cp      := COS (psi*rad);
      phi    := SIN (phi*rad);
      cphi    := COS (phi*rad);
      xe[3]   := -1.0 * alt;
      b[1,1]  := cp * ct;
      b[1,2]  := sp * ct;
      b[1,3]  := -1.0 * St.;
      b[2,1]  := -1.0 * sp * cphi + cp * St. * phi;
      b[2,2]  := cp * cphi + sp * St. * phi;
      b[2,3]  := ct * phi;
      b[3,1]  := sp * phi + cp * St. * cphi;
      b[3,2]  := -1.0 * cp * phi + sp * St. * cphi;
      b[3,3]  := ct * cphi;

END METHOD {ObjInit};

ASK METHOD jump;
BEGIN

     gees := 0.0;
     cds  := 0.0;
     t    := 0.0;
     gmax := 0.0;
     iend := 0;
     tar  := t;
     tend := tend - 0.1 * dt;
     alt  := -1.0 * xe[3]; { alt = 328.1 ft }
     h    := alt;
     test := 0;

     OUTPUT (tend, " ", hmin, " ", alt);
     OUTPUT (ue[1], " ", ue[3], " ", vwind[2]);

     {BEGIN TRAJECTORY LOOP}

     WHILE test = 0

          IF h > hmin

               g := gravCalc(h);

               densityCalc (h, rhoz, rho, sound);

               pb   := wb[1];
               qb   := wb[2];
               rb   := wb[3];
               ue1  := ue[1] - vwind[1];
               ue2  := ue[2] - vwind[2];
               ue3  := ue[3] - vwind[3];
               vp   := SQRT(POWER(ue1,2.0) + POWER(ue2,2.0) + POWER(ue3,2.0));
               vpo  := vp;
               mach := vp/sound;
               ub1  := b[1,1]*ue1 + b[1,2]*ue2 + b[1,3]*ue3;
               ub2  := b[2,1]*ue1 + b[2,2]*ue2 + b[2,3]*ue3;
               ub3  := b[3,1]*ue1 + b[3,2]*ue2 + b[3,3]*ue3;
               vp13 := SQRT(POWER(ub1,2.0) + POWER(ub3,2.0));

          { USE SIN(ALPHA) for ALPHA and COS(BETA) for BETA }

               IF vpo < 1.0E-06
                    vpo := 1.0E-06;
               END IF;

               sbeta := ub2 / vpo;
```

```
                     cbeta := vp13 / vpo;
                     beta  := sbeta;

                     IF vp13 < 1.0E-06
                          vp13 := 1.0E-06;
                     END IF;

                     salpha := ub3 / vp13;
                     calpha := ub1 / vp13;
                     alpha  := salpha;

                { AERODYNAMIC and BODY FORCES AND MOMENTS }

                { ISOLATED BODY AERODYNAMICS }

                     { AERO ROUTINE }

                     sac := salpha * calpha;
                     sas := salpha * ABS (salpha);
                     sbc := sbeta * cbeta;
                     sbs := sbeta * ABS(sbeta);
                     rad := cbar / (2.0*vpo);
                     cna2 := 0.0;
                     cnq  := 0.0;
                     cyb2 := 0.0;
                     cyr  := 0.0;
                     cmo  := 0.0;
                     cma2 := 0.0;
                     cnb2 := 0.0;

                     { FOREBODY XERO-LIFT DRAG COEFFIECIENT }

                     i  := 0;
                     loop := 0;

                     WHILE loop = 0
                          i  := i + 1;
                          ip := i + 1;
                          IF i = mpts
                               cao  := pcdf[2]; { when i = ipts }
                               loop := 1;
                          ELSE
                               IF mach <= pm[ip]
                                    cao  := pcdf[i]+(pcdf[ip]-pcdf[i])*(mach-
pm[i])/(pm[ip]-pm[i]);
                                    loop := 1;
                               END IF;
                          END IF;
                     END WHILE;

                     cn  := cna * sac + cna2 * sas + cnq * qb * rad;
                     cy  := cyb + sbc + cyb2 * sbs + cyr * rb * rad;
                     ca  := cao + caa2 * (1.0 - POWER(calpha,2.0) * POWER(cbeta,2.0));
                     csl := clo + clp * pb * rad;
                     cm  := cmo + cma * sac + cma2 * sas + cmq * qb * rad;
                     csn := cnb * sbc + cnb2 * sbs + cnr * rb * rad;

                     { PARACHUTE DRAG-AREA }

                     cds  := 0.0;
                     i    := 0;
                     loop := 0;

                     WHILE loop = 0
                          IF t < pt[1]
                               loop := 1;
                          ELSE
                               i  := i + 1;
                               ip := i + 1;
                               IF i = ipts
                                    cds := pcds[ipts];
                                    loop := 1;
                               ELSE
```

```
                                    IF t <= pt [ip]
                                         cds := pcds[i] + (pcds[ip]-pcds[i]) * (t-pt[i]) /
(pt[ip]-pt[i]);
                                         loop := 1;
                                    END IF;
                               END IF;
                          END IF;
                    END WHILE;

                    { END AERO ROUTINE }

                   q := 0.5 * rho * vp;
                    fpc := -1.0 * q * cds;
                    qs := 0.5 * rho * POWER (vp, 2.0) * sarea;
                    qsd := qs * cbar;
                    fb[1] := -1.0 * qs * ca + mass * g * b[1,3] + fpc * ub1;
                    fb[2] := qs * cy + mass * g * b[2,3] + fpc * ub2;
                    fb[3] := -1.0 * qs * cn + mass * g * b[3,3] + fpc * ub3;
                    mb[1] := qsd * csl;
                    mb[2] := qsd * cm + fpc * ub3 * (xbod - xcg);
                    mb[3] := qsd * csn - fpc * ub2 * (xbod - xcg);
                    gees := -1.0 * fb[1] / (mass * g);

                    IF ABS (gees) > ABS (gmax)
                         gmax := gees;
                    END IF;

                    IF iend = 0
                         IF t >= tar
                              tar := tar + dtpr;
                              h := -1.0 * xe[3];
                              vpe := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER
(ue3,2.0) );
                              qdyn := 0.5 * rho * POWER (vp,2.0);
                              bxy := SQRT ( POWER (b[1,1], 2.0) + POWER (b[1,2],2.0) );
                              theta := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                              alphad := 57.295 * ATAN2 ( salpha, calpha );
                              uxy := SQRT ( POWER (ue[1],2.0) + POWER (ue[2],2.0) );
                              gammad := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );
                              OUTPUT (t, " ", h, " ", xe[1], " ", xe[2], " ", vpe, " ",
vp, " ", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ", alphad, " ", cds);
                         END IF;

                         { EULER ROTATION FUNCTION FOR DIRECTION COSINE PROPOAGATION }

                         w2 := POWER(wb[1],2.0) + POWER(wb[2],2.0) + POWER(wb[3],2.0);
                         w := SQRT (w2);
                         coswt := COS (w*dt);
                         sinwt := SIN (w*dt);
                         coswtm := 1.0 - coswt;

                         IF w2 < 1.0E-12
                              w2 := 1.0E-12;
                              w := 1.0E-06;
                         END IF;

                         { ANGULAR MOMENTUM CROSS PRODUCT TERMS }

                         FOR k := 1 TO 3
                              hb [k] := in[k,1] * wb[1] + in[k,2] * wb[2] + in[k,3] *
wb[3];
                         END FOR;

                         FOR i := 1 TO 3
                              i1 := e[i+1];
                              i2 := e[i+2];
                              temp[i] := wb[i1] * hb[i2] - wb[i2] * hb[i1];
                         END FOR;

                         { FORCE RESOLUTION TO EULER SYSTEM                           }
                         { TRANSLATIONAL ACCELERATION AND DIRECTION COSINE ROTATION }

                         FOR i := 1 TO 3
```

```
                              fe[i] := fb[1] * b[1,i] + fb[2] * b[2,i] + fb[3] * b[3,i];
                              uedot[i] := fe[i] / mass;
                              FOR j := 1 TO 3
                                   bn[i,j] := b[i,j];
                                   j1 := e[j+1];
                                   j2 := e[j+2];
                                   bdot[i,j] := del[i,j]*coswt + wb[i]*wb[j]*coswtm/w2 + (
wb[j1]*del[i,j2] - wb[j2]*del[i,j1] )* sinwt/w;
                              END FOR;

                              { ANGULAR ACCELERATION IN BODY AXES }

                              wbdot[i] := jn[i,1]*(mb[1]-temp[1]) + jn[i,2]*(mb[2]-
temp[2]) + jn[i,3]*(mb[3]-temp[3]);
                         END FOR;

                         { INTEGRALS }

                         t := t + dt;

                         FOR i := 1 TO 3
                              xe[i] := xe[i] + dt*(ue[i]+0.5*dt*uedot[i]);
                              ue[i] := ue[i] + dt*uedot[i];
                              wb[i] := wb[i] + dt*wbdot[i];
                              FOR j := 1 TO 3
                                   b[i,j] := bdot[i,1]*bn[1,j] + bdot[i,2]*bn[2,j] +
bdot[i,3]*bn[3,j];
                              END FOR;
                         END FOR;

                         IF t >= tend
                              iend := 1;
                         ELSE
                              h := -1.0 * xe[3];
                         END IF;
                    ELSE { when iend = 1 }
                         h := -1.0 * xe[3];
                         vpe := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER (ue3,2.0)
);
                         qdyn := 0.5 * rho * POWER (vp,2.0);
                         bxy := SQRT ( POWER (b[1,1],2.0) + POWER (b[1,2],2.0) );
                         theta := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                         alphad := 57.295 * ATAN2 ( salpha, calpha );
                         uxy := SQRT ( POWER(ue[1],2.0) + POWER(ue[2],2.0) );
                         gammad := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );
                         OUTPUT (t, " ", h, " ", xe[1], " ", xe[2], " ", vpe, " ", vp, "
", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ", alphad, " ", cds);
                         test := 1;
                    END IF {iend};
               ELSE { when h <= hmin }
                    test := 1;
               END IF {hmin};

          END WHILE {test};

     END METHOD {jump};

END OBJECT {jumperObj};

END {IMPLEMENTATION} MODULE {jumperMod}.
```

APPENDIX C

Comparisons of Purvis 6-DOF and MODSIM 6-DOF State Variable Outputs

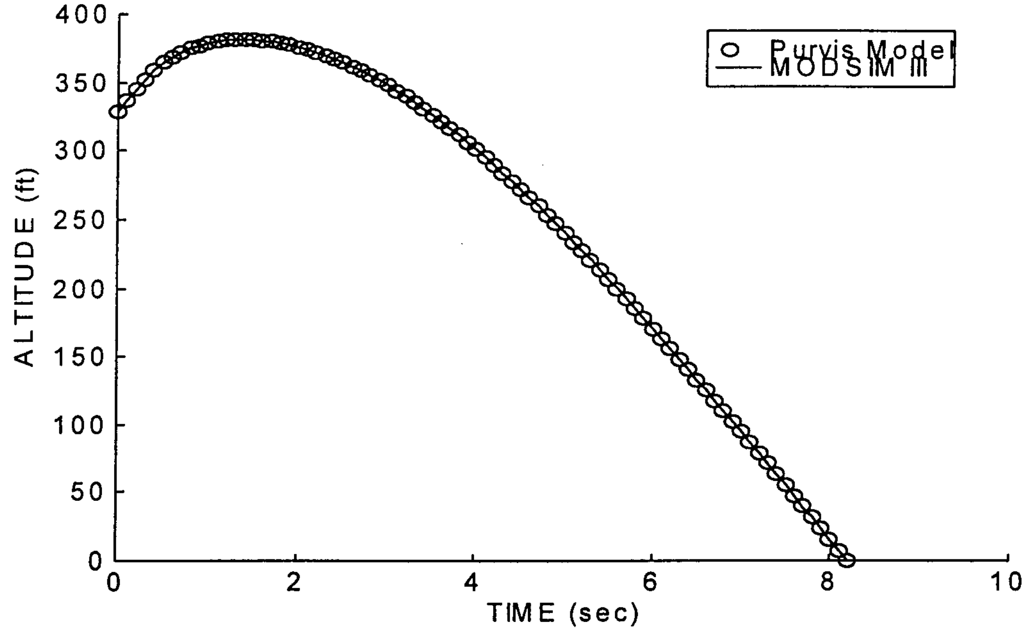

Figure 20. Altitude Vs. Time

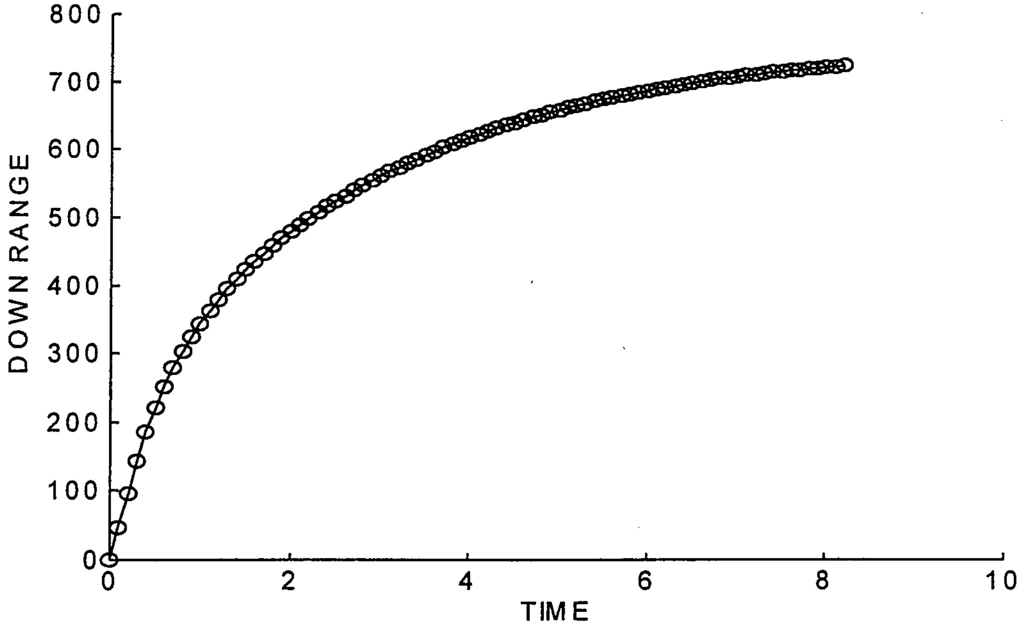

Figure 21. Down Range Vs. Time

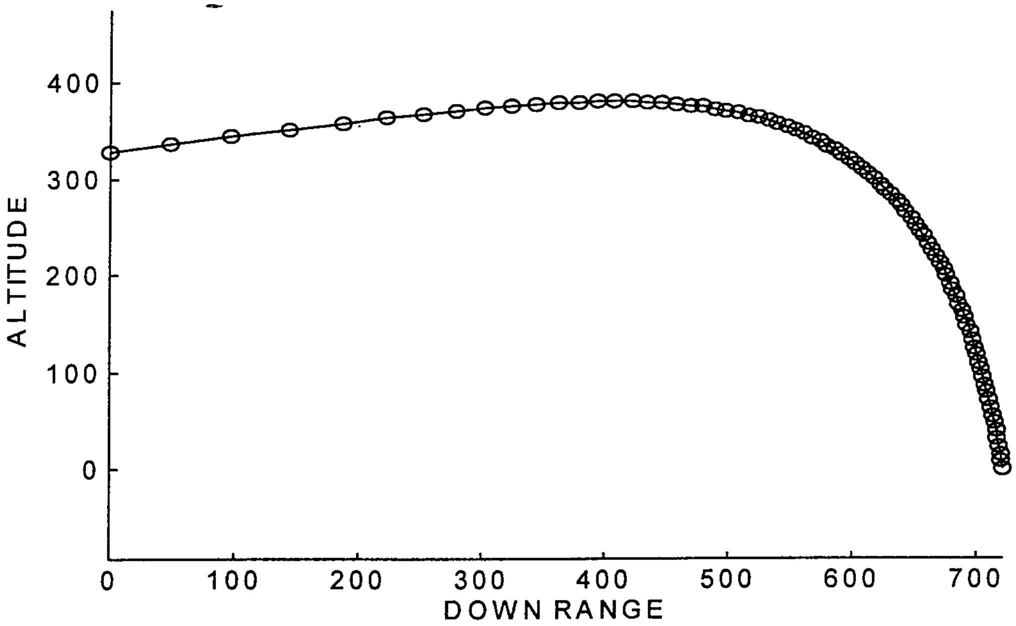

Figure 22. Altitude Vs. Down Range

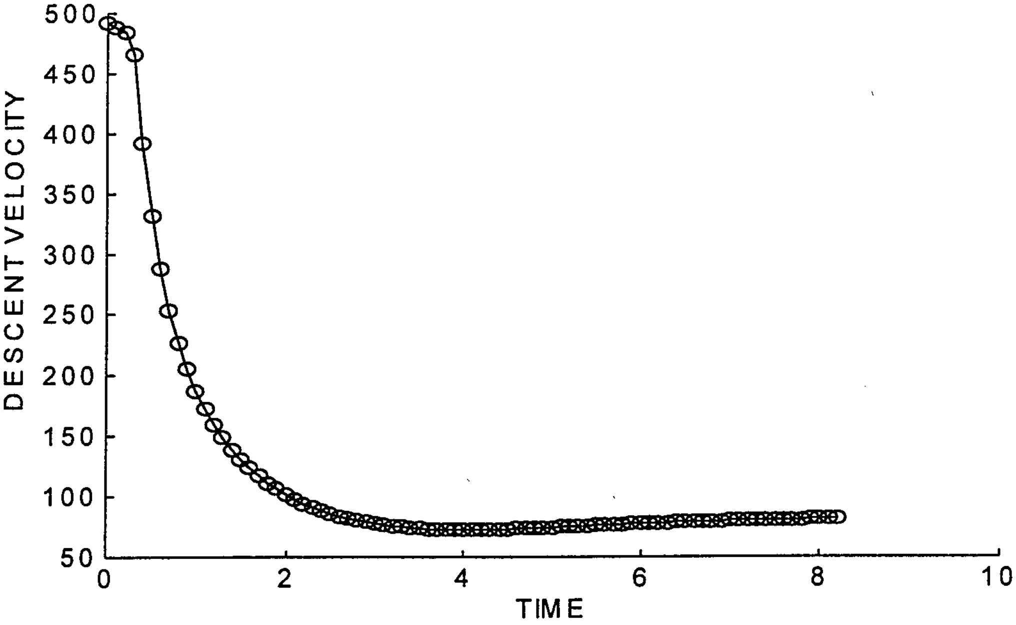

Figure 23. Descent Velocity Vs. Time

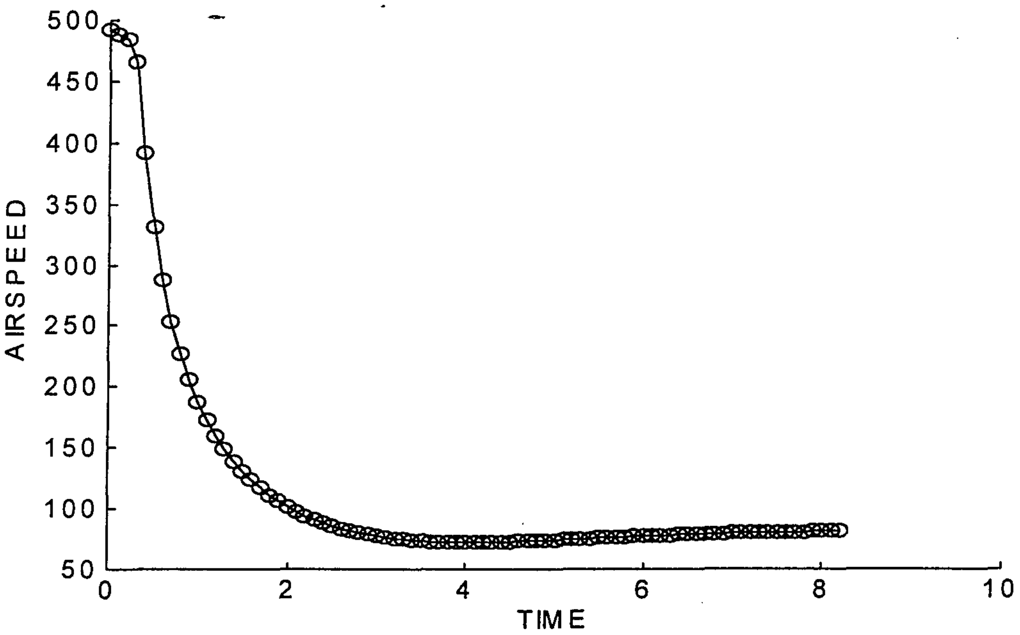

Figure 24. Airspeed Vs. Time

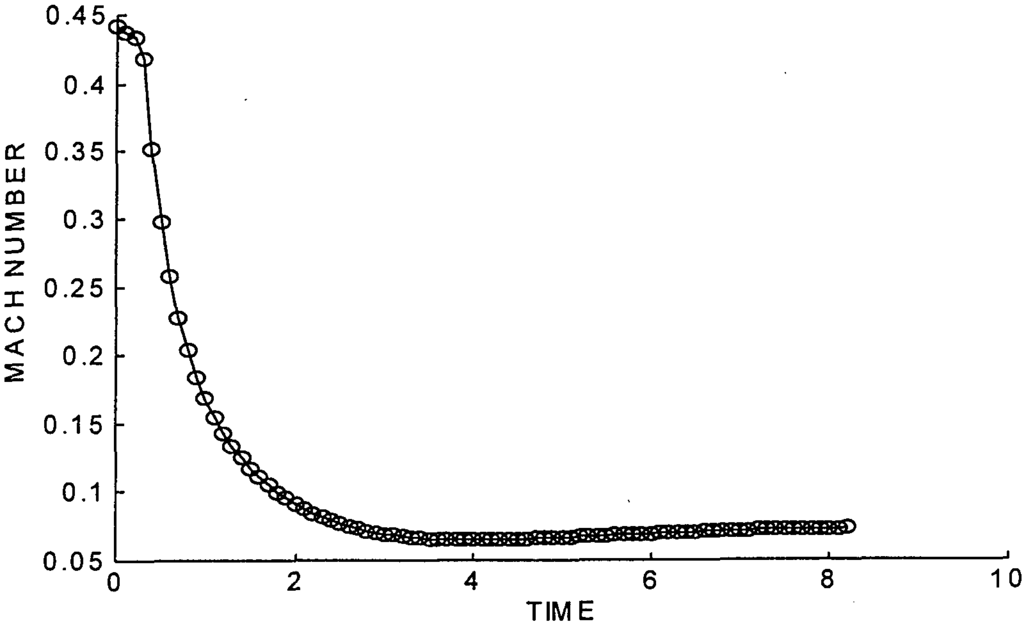

Figure 25. Mach Number Vs. Time

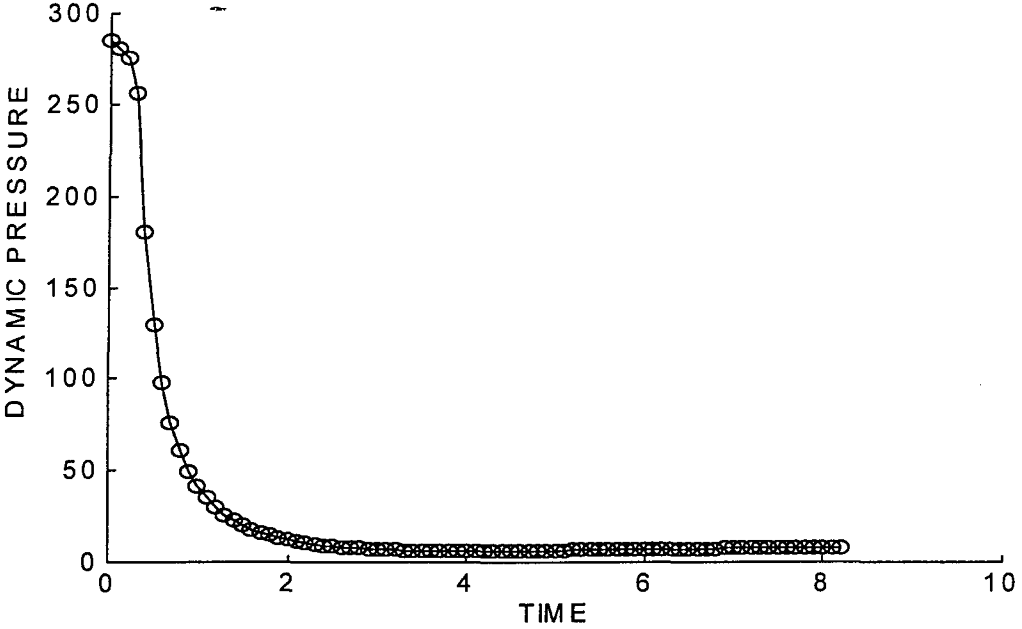

Figure 26. Dynamic Pressure Vs. Time

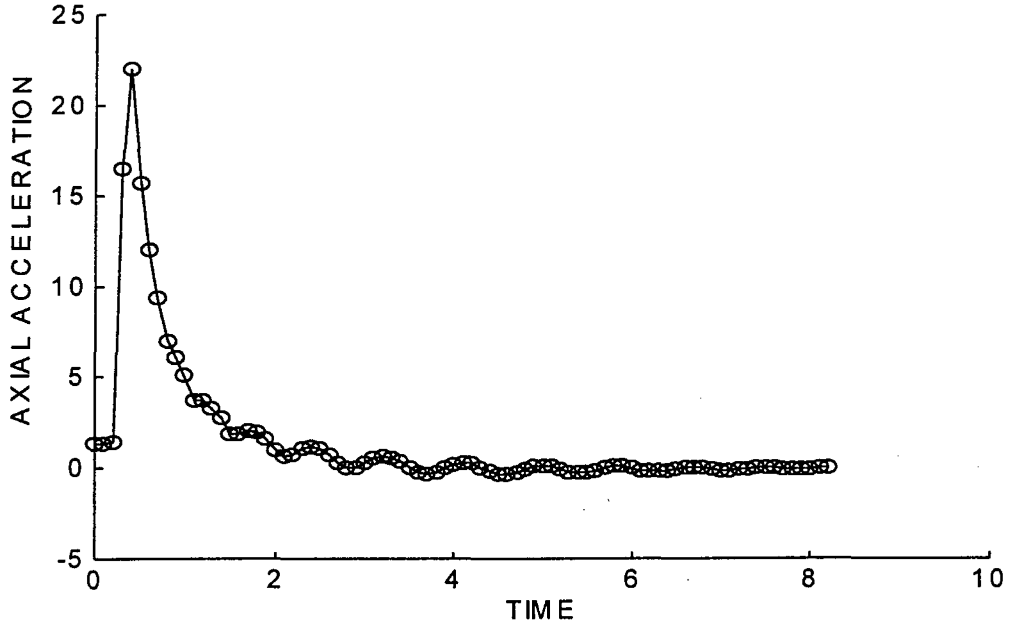

Figure 27. Axial Acceleration Vs. Time

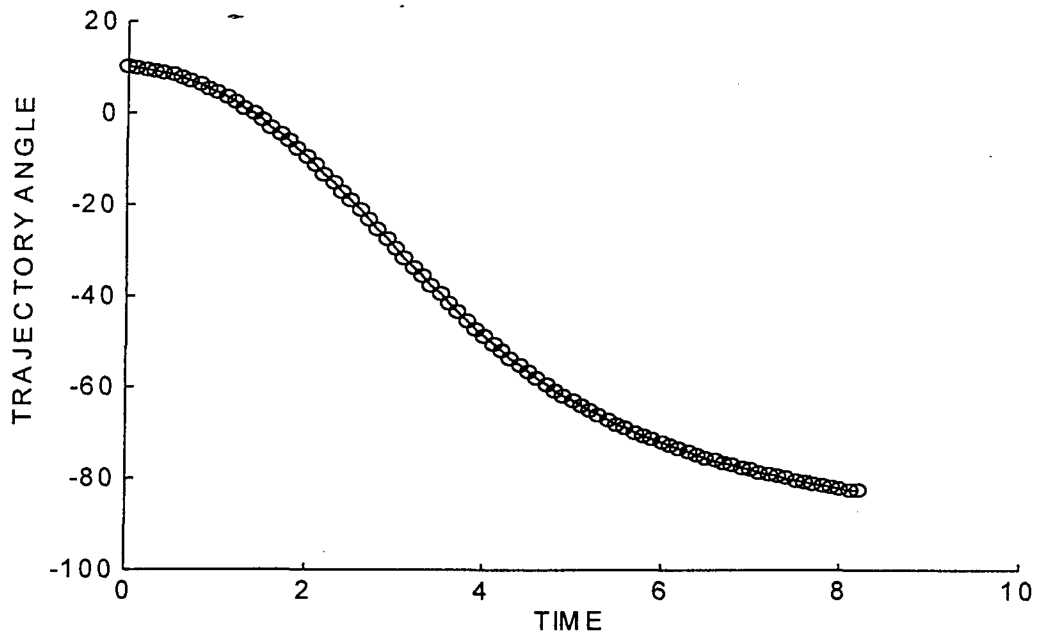

Figure 28. Trajectory Angle Vs. Time

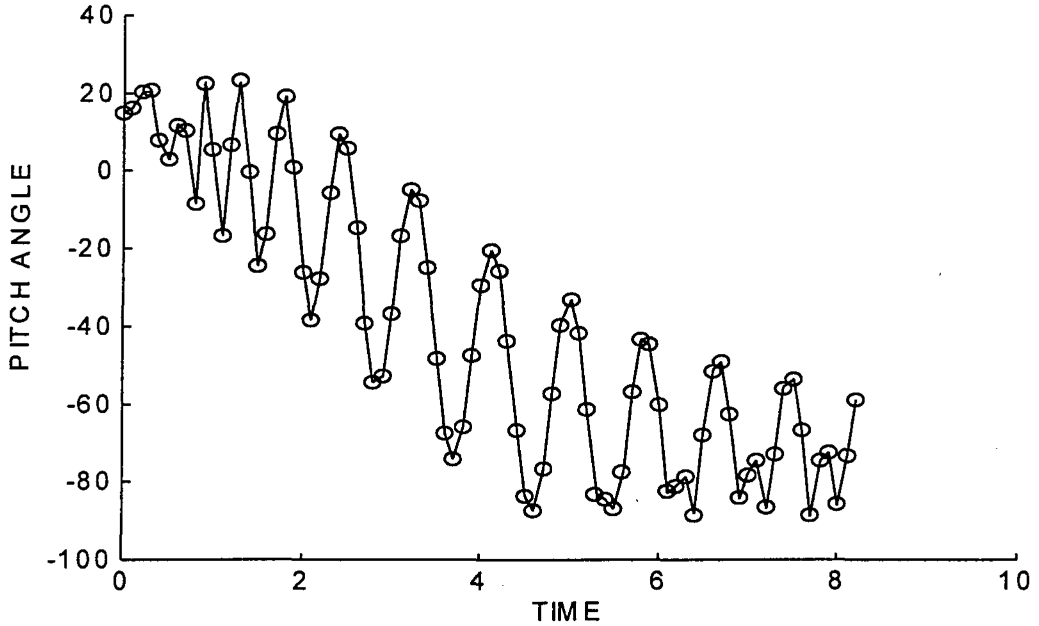

Figure 29. Pitch Angle Vs. Time

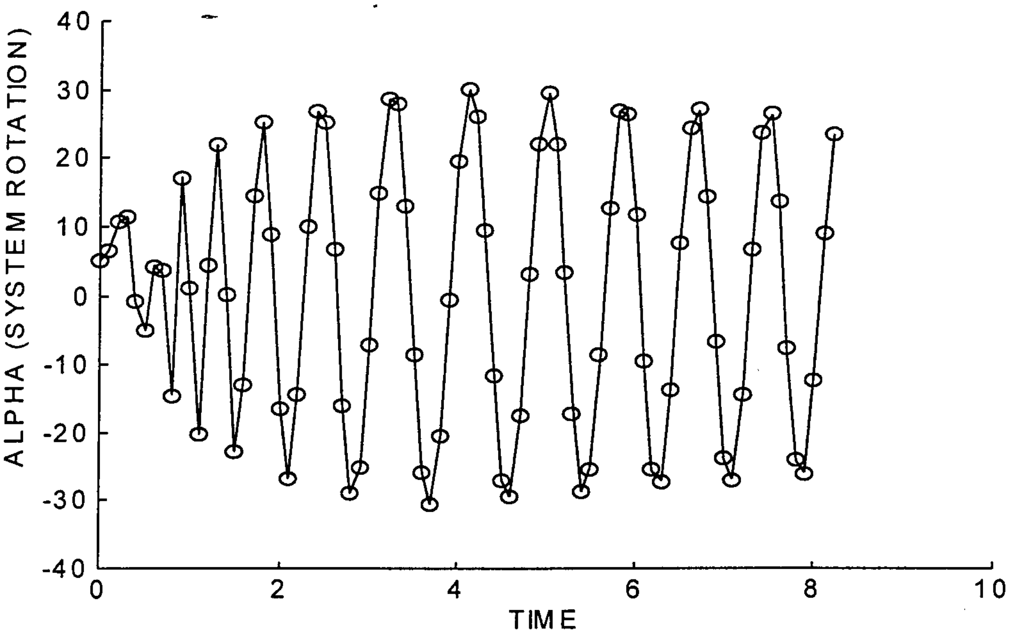

Figure 30. System Rotation Vs. Time

APPENDIX D
MODSIM III Paratrooper Object Modules

```
MAIN MODULE vortex;

FROM inputMod          IMPORT readData, disposeStreams;
FROM globalMod         IMPORT i, NumberofPlanes, nu, knotconv, initializeData, repeat;
FROM AirplaneMod       IMPORT C17Obj;
FROM VortexMod         IMPORT RightVortexObj, LeftVortexObj;
FROM MathMod           IMPORT pi;
FROM VortexControlMod  IMPORT Airdrop;
FROM SimMod            IMPORT ResetSimTime, StartSimulation;
FROM UtilMod           IMPORT DateTime;

VAR

BEGIN

{ ----- Start the input questions and set up the random seeds ----- }

     readData;
     initializeData;

     FOR repeat := 1 TO 50;

          ResetSimTime(0.0);

          { ----- Create the Vortex Control Object named Airdrop ----- }

          NEW (Airdrop);

          { ----- Schedule the first event to intiate the simulation ----- }

          TELL Airdrop TO Fly;

          StartSimulation;

          DISPOSE (Airdrop);

     END FOR;

     disposeStreams;

END {MAIN} MODULE {Vortex}.
```

```
DEFINITION MODULE globalMod;

FROM RandMod        IMPORT RandomObj;
FROM VortexMod      IMPORT RightVortexObj, LeftVortexObj;
FROM rightJumperMod IMPORT rightJumperObj;
FROM leftJumperMod  IMPORT leftJumperObj;

CONST

     re = 20855531.5;
     nu  = 0.0001654;
     knotconv = 1.69085;  {Converts knots to ft/sec}

TYPE

     eType      = ARRAY INTEGER OF INTEGER;
     delType    = ARRAY INTEGER, INTEGER OF REAL;
     matrixType = ARRAY INTEGER, INTEGER OF REAL;
     vectorType = ARRAY INTEGER OF REAL;

     encounterType = RECORD
          airplane : INTEGER;
          side     : STRING;
          position : INTEGER;
     END RECORD {encounterType};

     ElementPositionType = RECORD
          ElementPosNum : INTEGER;
          Intrail       : REAL;
          CrossTrack    : REAL;
     END RECORD;

     ElementGeometryType      =  ARRAY INTEGER OF ElementPositionType;

     FormationPositionType  = RECORD;
          PositonNumber   : INTEGER;
          Intrail         : REAL;
          CrossTrack      : REAL;
     END RECORD;

     FormationGeometryType   = ARRAY INTEGER OF FormationPositionType;

VAR

     NumberofPlanes    : INTEGER;
     PlanesPerElement  : INTEGER;
     NumberOfElements  : INTEGER;
     i, j, repeat      : INTEGER;

     ElementSpacing    : REAL;
     ElementGeometry   : ElementGeometryType;
     FormationGeometry : FormationGeometryType;

     CrossWind1        : REAL;
     CrossWind2        : REAL;
     CrossWind3        : REAL;
     ShearAlt1         : REAL;
     ShearAlt2         : REAL;
     StandDev1         : REAL;
     StandDev2         : REAL;
     StandDev3         : REAL;
     trailBox          : INTEGER;
     lateralBox        : INTEGER;
     altitudeBox       : INTEGER;
     HeadWind          : REAL;
     rho               : REAL;
     altitude          : REAL;
     vfk               : REAL;
     weight            : REAL;
     RunLength         : REAL;
     vs1               : REAL;
     vs2               : REAL;
     vs3               : REAL;
     rightJumper          : rightJumperObj;
     leftJumper           : leftJumperObj;
     e                    : eType;
     del                  : delType;
     seed1                : RandomObj;
     seed2                : RandomObj;
     seed3                : RandomObj;
     seed4                : RandomObj;
     windseed1            : RandomObj;
     windseed2            : RandomObj;
     windseed3            : RandomObj;
     trailseed            : RandomObj;
     lateralseed          : RandomObj;
     timeseed             : RandomObj;

PROCEDURE initializeData;

END {DEFINITION} MODULE {globalMod}.
```

```
IMPLEMENTATION MODULE globalMod;

FROM RandMod  IMPORT FetchSeed;
FROM inputMod IMPORT jumperseed;

PROCEDURE initializeData;

     BEGIN

          NEW (e, 1..5);
          NEW (del, 1..3, 1..3);
          NEW (seed1);
          NEW (seed2);
          NEW (seed3);
          NEW (seed4);
          NEW (windseed1);
          NEW (windseed2);
          NEW (windseed3);
          NEW (trailseed);
          NEW (lateralseed);
          NEW (timeseed);
          ASK seed1 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK seed2 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK seed3 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK seed4 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK windseed1 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK windseed2 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK windseed3 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK trailseed TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK lateralseed TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
          ASK timeseed TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));


          e[1] := 1;
          e[2] := 2;
          e[3] := 3;
          e[4] := 1;
          e[5] := 2;

          FOR i := 1 TO 3
               FOR j := 1 TO 3
                    IF i = j
                         del[i,j] := 1.0;
                    ELSE
                         del[i,j] := 0.0;
                    END IF;
               END FOR;
          END FOR;

     END PROCEDURE {initializeData};

END {IMPLEMENTATION} MODULE {globalMod}.
```

```
DEFINITION MODULE calcMod;

PROCEDURE gravCalc (IN a : REAL) : REAL;
PROCEDURE densityCalc (IN h, rhoz : REAL; OUT rho, sound : REAL);

END {DEFINITION} MODULE {calcMod}.
```

```
IMPLEMENTATION MODULE calcMod;

FROM MathMod IMPORT POWER, SIN, COS, SQRT, EXP;
FROM globalMod IMPORT re;

PROCEDURE gravCalc (IN a: REAL) : REAL;
     BEGIN
          RETURN 32.1741*POWER(re/(a+re), 2.0);
     END PROCEDURE {gravCalc};

PROCEDURE densityCalc (IN h, rhoz : REAL; OUT rho, sound : REAL);
     VAR

          t : REAL;

     BEGIN
          rho := rhoz * EXP(-1.0*h/23111.0 + 0.294 * SIN(h/28860.0) + 0.213 *
SIN(h/86580.0));

          IF h > 0.0
               t := 518.688 - (3.56616E-03)*h;
               sound :=  49.02118 * SQRT(t);
               IF h > 36152.0
                    sound := 968.08;
                    IF h > 82345.0
                         t := 254.988 + (1.64592E-03)*h;
                         sound := 49.02118 * SQRT(t);
                         IF h > 155348.0
                              sound := 1105.0;
                              IF h > 262448.0
                                   sound := 846.9;
                                   IF h > 299516.0
                                        t := -349.812 + 2.19456E-03*h
                                   END IF;
                              END IF;
                         END IF;
                    END IF;
               END IF;
          ELSE
               sound := 1116.44;
          END IF;

     END PROCEDURE {densityCalc};

END {IMPLEMENTATION} MODULE {calcMod}.
```

Note: Definition and Implementation Modules for right and left jumpers are the same.

```
DEFINITION MODULE rightJumperMod;

FROM IOMod             IMPORT StreamObj, FileUSeType(Output);
FROM globalMod         IMPORT eType, matrixType, vectorType, encounterType;
FROM VortexMod         IMPORT RightVortexObj, LeftVortexObj;
FROM VortexControlMod  IMPORT VortexControl;

TYPE

     rightJumperObj = OBJECT            fpc,
                                        g,
          i1,                           gammad,
          i2,                           gees,
          iend,                         gmax,
          ipts,                         h,
          ip,                           hmin,
          j1,                           mach,
          j2,                           mass,
          k,                            myTime,
          loop,                         myDrift,
          mpts,                         myDriftDirection,
          myNumber,                     pb,
          myPlane,                      phi,
          bigloop : INTEGER;            psi,
                                        q,
          alt,                          qb,
          alpha,                        qdyn,
          alphad,                       qs,
          beta,                         qsd,
          bxy,                          rad,
          ca,                           rb,
          cao,                          rho,
          caa2,                         rhoz,
          calpha,                       sac,
          cbar,                         sas,
          cbeta,                        sarea,
          cby2,                         salpha,
          cds,                          sbc,
          clo,                          sbeta,
          clp,                          sbs,
          cm,                           sinwt,
          cma,                          sound,
          cma2,                         sp,
          cmo,                          phi,
          cmq,                          St.,
          cn,                           t,
          cna,                          theta,
          cna2,                         tar,
          cnb,                          tpoll,
          cnb2,                         tdrift,
          cnq,                          ub1,
          cnr,                          ub2,
          coswt,                        ub3,
          coswtm,                       ue1,
          cp,                           ue2,
          cphi,                         ue3,
          csl,                          uxy,
          csn,                          vp,
          ct,                           vp13,
          cy,                           vpe,
          cyb,                          vpo,
          cyb2,                         w,
          cyr,                          w2,
          dens,                         weight,
          deptime,                      xbod,
          dt,                           xcg,
          dtpr,                         Xdrift,
          dtpoll,                       xlast,
          dtdrift,                      Ydrift     : REAL;

          slength   : REAL; {length of suspension lines}
          angle     : REAL; {the angle (in radians) which defines the "cone" of the
 suspension lines}
          dcglength : REAL; {distance from end of suspension lines to paratrooper c.g.}
          cweight   : REAL; {weight of canopy}
          sweight   : REAL; {weight of suspension lines}

          radius    : REAL;
          addedmass : REAL;
          distcm    : REAL;
          sysmass   : REAL;
          paymom    : REAL;
          distcan   : REAL;
          distline  : REAL;
          distpay   : REAL;

          addDrift  : BOOLEAN;

          pcds,
          pt,
          pcdf,
          pm    : vectorType; {1X2}

          xe,
          xs,
          xg,
          ue,
          wb,
          vwind,
          temp,
          fb,
          m,
          mb,
          fe,
          uedot,
          wbdot,
          hb     : vectorType; {1X3}

          in,
          jn,
          b,
          bn,
          bdot : matrixType; {3X3}

          lastRightLocation,
          lastLeftLocation : eType; {Dynamic Array}

          outfile : STRING;

          stream : StreamObj;

          encounter   : encounterType;

          ASK METHOD ObjInit;
          TELL METHOD jump;
          ASK METHOD initialize (IN stick   : INTEGER;
                                 IN myPlane : INTEGER);
          ASK METHOD pollVortices (IN vortexPlane : INTEGER);
          ASK METHOD changeDrift;
          ASK METHOD findDrift;

     END OBJECT {rightJumperObj};

END {DEFINITION} MODULE {rightJumperMod}.
```

```
IMPLEMENTATION MODULE rightJumperMod;

FROM MathMod          IMPORT EXP, SIN, COS, POWER, SQRT, ATAN2, TAN, pi;
FROM VortexMod        IMPORT RightVortexObj, LeftVortexObj;
FROM globalMod        IMPORT re, e, del, i, j, NumberofPlanes, seed1, seed2, seed3,
repeat;
FROM inputMod         IMPORT streamI, streamE, streamS, extension, printTrajectory;
FROM globalMod        IMPORT CrossWind1, CrossWind2, CrossWind3, ShearAlt1, ShearAlt2,
HeadWind, vs1, vs2, vs3;
FROM calcMod          IMPORT gravCalc, densityCalc;
FROM SimMod           IMPORT SimTime;
FROM VortexControlMod IMPORT VortexControl, Airdrop;

OBJECT rightJumperObj;

     ASK METHOD ObjInit;

          BEGIN
               NEW(pcdf, 1..2);
               NEW(pm  , 1..2);

               NEW(fb   , 1..3);
               NEW(fe   , 1..3);
               NEW(hb   , 1..3);
               NEW(mb   , 1..3);
               NEW(temp , 1..3);
               NEW(ue   , 1..3);
               NEW(uedot, 1..3);
               NEW(vwind, 1..3);
               NEW(wb   , 1..3);
               NEW(wbdot, 1..3);
               NEW(xe   , 1..3);
               NEW(xs   , 1..3);
               NEW(xg   , 1..3);

               NEW(pcds, 1..4);
               NEW(pt  , 1..4);

               NEW(in  , 1..3, 1..3);
               NEW(jn  , 1..3, 1..3);
               NEW(b   , 1..3, 1..3);
               NEW(bn  , 1..3, 1..3);
               NEW(bdot, 1..3, 1..3);

          { ----- system inertial properties ----- }

               { ----- parachute-payload system weight (lbs) = wight of jumper/gear +
weight of T-10C ----- }
               weight := seed3.Normal (247.0, 24.35);

               mass   := weight/32.17;
               xcg    := 2.0;           { forebody c.g. (ft) in the horizontal }
               xbod   := 6.0;           { forebody length (ft) in the vertical }

               FOR i := 1 TO 3
                    FOR j := 1 TO 3
                         jn[i,j] := 0.0;
                         in[i,j] := 0.0;
                    END {j} FOR;
               END {i} FOR;

          { ----- initial conditions ----- }

               FOR i := 1 TO 3
                    xe[i]    := 0.0; { (ft)  1: down range, 2: off range, 3: altitude loss
}
                    ue[i]    := 0.0; { (fps) 1: horizontal velocity, 2: lateral velocity,
3: ejection velocity positive down }
                    wb[i]    := 0.0; { ???
}
                    vwind[i] := 0.0; { (fps) 1: head (+) or tail (-) wind, 2: crosswind,
3: ????                          }
               END FOR;
               alt         := 0.0;       { altitude (ft)                                }
               hmin        := 0.0;       { ground level (ft)                            }
               ue[1]       := 0.0;       { horizontal velocity (fps)                    }
               ue[3]       := 0.0;       { ejection velocity, positive down             }
               theta       := 0.0;       { pitch angle (deg) nose up positive           }
               vwind[1]    := 0.0;       { head (+) or tail (-) wind (fps)              }
               vwind[2]    := 0.0;       { crosswind (fps)                              }
               dens        := 0.0;       { density (0 for standard atms) in slug/ft^3 }
               rhoz        := 0.002378;  { ????                                         }

               IF dens <> 0.0
                    rhoz := dens * EXP (alt/23111.0 - 0.295 * SIN(alt/28860.0) - 0.213 *
SIN(alt/86580.0))
               END IF;

          { ----- program constants ----- }

               dtpr     := 0.1;     { print interval (sec)
}
               dtpoll   := 0.5;     { poll vortex positions every 0.5 seconds
}
               dtdrift  := 5.0;     { change drift angle +/- 45.0 from current drift angle
every 10 seconds }

          { ----- forebody aerodynamic coefficients ----- }

               cbar   := 6.0;                { reference length (ft)                      }
               sarea  := POWER(0.5,2.0)*pi;  { reference area (ft^2)                      }
               cna    := 0.0;                { normal force cn-alpha (/rad)               }
               cyb    := 0.0;                { side force cy-beta (/rad)                  }
               caa2   := 0.0;                { axial force ca-alpha^2 (/rad^2)            }
               clo    := 0.0;                { roll torque coefficient (dimensionless) }
               clp    := 0.0;                { roll damping coefficient (/rad)            }
               cma    := -2.0;               { pitch moment cm-alpha (/rad)               }
               cmq    := -200.0;             { pitch damping (/rad)                       }
               cnb    := 0.0;                { yaw moment cn-beta (/rad)                  }
               cnr    := 0.0;                { yaw damping (/rad)                         }

          { ----- forebody drag versus mach number table ----- }

               mpts     := 2;
               pm[1]    := 0.00; { mach number }
               pm[2]    := 2.00;
               pcdf[1]  := 0.73+0.06*(360.0 - weight)/180.0; { drag coefficient }
               pcdf[2]  := 0.73+0.06*(360.0 - weight)/180.0;

          { ----- parachute drag-area versus time table ----- }

               deptime := 0.25; { deployment time }
               ipts    := 2;

               IF deptime > 0.0
                    ipts := ipts + 2; { ipts = 4 }
                    pt[1]    := 0.0;
                    pt[2]    := deptime;
                    pt[3]    := 0.00 + deptime;
                    pt[4]    := 2.80 + deptime;
                    pcds[1]  := 0.0;
                    pcds[2]  := 0.0;
                    pcds[3]  := 0.20;
                    pcds[4]  := 690.0;
               END IF;

          { ----- convert EULER ANGLES to direction cosines ----- }

               psi      := 0.0;
               phi      := 0.0;
               rad      := pi/180.0;
               St.       := SIN (theta*rad);
               ct       := COS (theta*rad);
               sp       := SIN (psi*rad);
               cp       := COS (psi*rad);
```

```
                 phi    := SIN (phi*rad);
                 cphi   := COS (phi*rad);
                 xe[3]  := -1.0 * alt;
                 b[1,1] := cp * ct;
                 b[1,2] := sp * ct;
                 b[1,3] := -1.0 * St.;
                 b[2,1] := -1.0 * sp * cphi + cp * St. * phi;
                 b[2,2] := cp * cphi + sp * St. * phi;
                 b[2,3] := ct * phi;
                 b[3,1] := sp * phi + cp * St. * cphi;
                 b[3,2] := -1.0 * cp * phi + sp * St. * cphi;
                 b[3,3] := ct * cphi;

          END {ASK} METHOD {ObjInit};

     TELL METHOD jump;

          BEGIN

               WHILE bigloop = 0

               WAIT DURATION dt;

               IF (-1.0*xe[3]) > hmin

                    g := gravCalc(h);
                    densityCalc (h, rhoz, rho, sound);

                    IF cds = 690.0
                         dt := 0.01;     { ----- ONCE CANOPY INFLATES, DECREASE TIME STEP
SIZE TO 0.01 ----- }
                    ELSE
                         dt := 0.001;    { ----- OTHERWISE, START WITH A SMALLER STEP SIZE
----- }
                    END IF;

                    radius     := SQRT(cds/pi);
                    addedmass  := rho*(4.0/3.0)*pi*POWER(radius,3.0);
                    distcm     :=
(32.17*addedmass*(slength*COS(angle)+(4.0/3.0)*(radius/pi)+dcglength)
                               +
cweight*(slength*COS(angle)+(4.0/3.0)*(radius/pi)+dcglength)
                               + sweight*(0.5*slength*COS(angle)+dcglength)) /
(32.17*addedmass+weight+sweight+cweight);
                    sysmass    := (weight+cweight+sweight)/32.17 + addedmass;
                    paymom     := (1.0/12.0)*mass*(3.0*POWER((0.5*xcg),2.0) +
POWER(xbod,2.0));
                    distcan    := slength*COS(angle) + (4.0/3.0)*(radius/pi) - distcm;
                    distline   := distcm - 0.5*slength*COS(angle);
                    distpay    := distcm;

                    in[1,1]    :=
(addedmass*((2.0/5.0)*POWER(radius,2.0)+POWER(distcan,2.0))+(cweight/32.17)*POWER(distcan,
2.0)
                               +
(sweight/32.17)*POWER(distline,2.0)+paymom+mass*POWER(distpay,2.0))/14.59;
                    in[2,2]    :=
(addedmass*((2.0/5.0)*POWER(radius,2.0)+POWER(distcan,2.0))+(cweight/32.17)*POWER(distcan,
2.0)
                               +
(sweight/32.17)*POWER(distline,2.0)+paymom+mass*POWER(distpay,2.0))/14.59;
                    in[3,3]    := ((2.0/5.0)*(cweight/32.17)*POWER(radius,2.0) +
(2.0/3.0)*(rho*(4.0/3.0)*pi*POWER(radius,3.0)
                                * POWER(radius,2.0)) + (0.5*mass))/14.59;

                    FOR i := 1 TO 3
                         jn[i,i] := 1.0 / in [i,i];
                    END FOR;

                    IF (-1.0*xe[3]) <= ShearAlt1
                         IF (-1.0*xe[3]) <= ShearAlt2
                              vwind[2] := vs3;
                         ELSE
```

```
                               vwind[2] := vs2;
                          END IF;
                     ELSE
                          vwind[2] := vs1;
                     END IF;

                     pb   := wb[1];
                     qb   := wb[2];
                     rb   := wb[3];
                     ue1  := ue[1] - vwind[1];
                     ue2  := ue[2] - vwind[2];
                     ue3  := ue[3] - vwind[3];
                     vp   := SQRT(POWER(ue1,2.0) + POWER(ue2,2.0) + POWER(ue3,2.0));
                     vpo  := vp;
                     mach := vp/sound;
                     ub1  := b[1,1]*ue1 + b[1,2]*ue2 + b[1,3]*ue3;
                     ub2  := b[2,1]*ue1 + b[2,2]*ue2 + b[2,3]*ue3;
                     ub3  := b[3,1]*ue1 + b[3,2]*ue2 + b[3,3]*ue3;
                     vp13 := SQRT(POWER(ub1,2.0) + POWER(ub3,2.0));

                { ----- USE SIN(ALPHA) for ALPHA and COS(BETA) for BETA ----- }

                     IF vpo < 1.0E-06
                          vpo := 1.0E-06;
                     END IF;

                     sbeta := ub2 / vpo;
                     cbeta := vp13 / vpo;
                     beta  := sbeta;

                     IF vp13 < 1.0E-06
                         vp13 := 1.0E-06;
                     END IF;

                     salpha := ub3 / vp13;
                     calpha := ub1 / vp13;
                     alpha  := salpha;

                { ----- AERODYNAMIC and BODY FORCES AND MOMENTS ----- }

                { ----- ISOLATED BODY AERODYNAMICS ----- }

                { ----- BEGIN AERO ROUTINE ----- }

                     sac  := salpha * calpha;
                     sas  := salpha * ABS (salpha);
                     sbc  := sbeta * cbeta;
                     sbs  := sbeta * ABS(sbeta);
                     rad  := cbar / (2.0*vpo);
                     cna2 := 0.0;
                     cnq  := 0.0;
                     cyb2 := 0.0;
                     cyr  := 0.0;
                     cmo  := 0.0;
                     cma2 := 0.0;
                     cnb2 := 0.0;

                     { ----- FOREBODY AERO-LIFT DRAG COEFFIECIENT ----- }

                     i    := 0;
                     loop := 0;
                     WHILE loop = 0 {WILL LOOP WHEN mach > pm[ip] UNTIL i = mpts}
                          i  := i + 1;
                          ip := i + 1;
                          IF i = mpts
                               cao  := pcdf[2]; { when i = ipts }
                               loop := 1;
                          ELSE
                               IF mach <= pm[ip]
                                    cao  := pcdf[i]+(pcdf[ip]-pcdf[i])*(mach-
pm[i])/(pm[ip]-pm[i]);
                                    loop := 1;
                               END IF;
                          END IF;
                          END WHILE;

                     cn  := cna * sac + cna2 * sas + cnq * qb * rad;
                     cy  := cyb + sbc + cyb2 * sbs + cyr * rb * rad;
                     ca  := cao + caa2 * (1.0 - POWER(calpha,2.0) * POWER(cbeta,2.0));
                     csl := clo + clp * pb * rad;
                     cm  := cmo + cma * sac + cma2 * sas + cmq * qb * rad;
                     csn := cnb * sbc + cnb2 * sbs + cnr * rb * rad;

                     { ----- PARACHUTE DRAG-AREA ----- }

                     cds  := 0.0;
                     i    := 0;
                     loop := 0;

                     { ----- THIS LOOP INFLATES THE PARACHUTE ----- }

                     WHILE loop = 0
                          IF t < pt[1]
                               loop := 1;
                          ELSE
                               i  := i + 1;
                               ip := i + 1;
                               IF i = ipts
                                    cds  := pcds[ipts];
                                    loop := 1;
                               ELSE
                                    IF t <= pt [ip]
                                         cds  := pcds[i] + (pcds[ip]-pcds[i]) * (t-pt[i]) /
(pt[ip]-pt[i]);

                                         loop := 1;
                                    END IF;
                               END IF;
                          END IF;
                     END WHILE;

                { ----- END AERO ROUTINE ----- }

                     q     := 0.5 * rho * vp;
                     fpc   := -1.0 * q * cds;
                     qs    := 0.5 * rho * POWER (vp, 2.0) * sarea;
                     qsd   := qs * cbar;
                     fb[1] := -1.0 * qs * ca + mass * g * b[1,3] + fpc * ub1;
                     fb[2] := qs * cy + mass * g * b[2,3] + fpc * ub2;
                     fb[3] := -1.0 * qs * cn + mass * g * b[3,3] + fpc * ub3;
                     mb[1] := qsd * csl;
                     mb[2] := qsd * cm + fpc * ub3 * (xbod - xcg);
                     mb[3] := qsd * csn - fpc * ub2 * (xbod - xcg);
                     gees  := -1.0 * fb[1] / (mass * g);

                     IF ABS (gees) > ABS (gmax)
                         gmax := gees;
                     END IF;

                     IF printTrajectory
                          IF t >= tar { THEN PRINT DATA }
                               tar := tar + dtpr;
                               h := -1.0 * xe[3];
                               vpe := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER
(ue3,2.0) );
                               qdyn := 0.5 * rho * POWER (vp,2.0);
                               bxy := SQRT ( POWER (b[1,1], 2.0) + POWER (b[1,2],2.0) );
                               theta := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                               alphad := 57.295 * ATAN2 ( salpha, calpha );
                               uxy := SQRT ( POWER (ue[1],2.0) + POWER (ue[2],2.0) );
                               gammad := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );
{
                               OUTPUT (myNumber, "R ", SimTime, " ", h, " ", xe[1], " ",
xe[2], " ", vpe, " ", vp, " ", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ",
alphad, " ", cds);
}
                               ASK stream TO WriteString (INTTOSTR(myNumber) + "R ");
                               ASK stream TO WriteString (REALTOSTR(SimTime) + " ");
                               ASK stream TO WriteString (REALTOSTR(h) + " ");
                               ASK stream TO WriteString (REALTOSTR(xe[1]) + " ");
                               ASK stream TO WriteString (REALTOSTR(xe[2]) + " ");
                               ASK stream TO WriteString (REALTOSTR(vpe) + " ");
                               ASK stream TO WriteString (REALTOSTR(vp) + " ");
                               ASK stream TO WriteString (REALTOSTR(mach) + " ");
                               ASK stream TO WriteString (REALTOSTR(qdyn) + " ");
                               ASK stream TO WriteString (REALTOSTR(gees) + " ");
                               ASK stream TO WriteString (REALTOSTR(gammad) + " ");
                               ASK stream TO WriteString (REALTOSTR(theta) + " ");
                               ASK stream TO WriteString (REALTOSTR(alphad) + " ");
                               ASK stream TO WriteString (REALTOSTR(cds));
                               ASK stream TO WriteLn;

                          END IF;
                     END IF;

                     { EULER ROTATION FUNCTION FOR DIRECTION COSINE PROPOAGATION }

                     w2 := POWER(wb[1],2.0) + POWER(wb[2],2.0) + POWER(wb[3],2.0);
                     w := SQRT (w2);
                     coswt := COS (w*dt);
                     sinwt := SIN (w*dt);
                     coswtm := 1.0 - coswt;

                     IF w2 < 1.0E-12
                          w2 := 1.0E-12;
                          w := 1.0E-06;
                     END IF;

                     { ANGULAR MOMENTUM CROSS PRODUCT TERMS }

                     FOR k := 1 TO 3
                          hb [k] := in[k,1] * wb[1] + in[k,2] * wb[2] + in[k,3] * wb[3];
                     END FOR;

                     FOR i := 1 TO 3
                           i1 := e[i+1];
                           i2 := e[i+2];
                           temp[i] := wb[i1] * hb[i2] - wb[i2] * hb[i1];
                     END FOR;

                     { FORCE RESOLUTION TO EULER SYSTEM                          }
                     { TRANSLATIONAL ACCELERATION AND DIRECTION COSINE ROTATION }

                     FOR i := 1 TO 3
                          fe[i] := fb[1] * b[1,i] + fb[2] * b[2,i] + fb[3] * b[3,i];
                          uedot[i] := fe[i] / mass;

                          FOR j := 1 TO 3
                               bn[i,j] := b[i,j];
                               j1 := e[j+1];
                               j2 := e[j+2];
                               bdot[i,j] := del[i,j]*coswt + wb[i]*wb[j]*coswtm/w2 + (
wb[j1]*del[i,j2] - wb[j2]*del[i,j1] )* sinwt/w;
                          END FOR;

                          { ANGULAR ACCELERATION IN BODY AXES }

                          wbdot[i] := jn[i,1]*(mb[1]-temp[1]) + jn[i,2]*(mb[2]-temp[2]) +
jn[i,3]*(mb[3]-temp[3]);

                     END FOR;

                     { ----- INTEGRALS ----- }

                     t := t + dt;
{                    t := SimTime - myTime;}

                     xlast := xe[1];

                     FOR i := 1 TO 3
```

```
                         xe[i] := xe[i] + dt*(ue[i]+0.5*dt*uedot[i]);
                         ue[i] := ue[i] + dt*uedot[i];
                         wb[i] := wb[i] + dt*wbdot[i];
                         FOR j := 1 TO 3
                              b[i,j] := bdot[i,1]*bn[1,j] + bdot[i,2]*bn[2,j] +
bdot[i,3]*bn[3,j];
                         END FOR;
                    END FOR;

                    { ----- INDUCE A DRIFT DIRECTION AND VELOCITY ON THE PARATROOP      ---
-- }

                         IF t >= tdrift
                              tdrift := t + dtdrift;
                              ASK SELF TO changeDrift;
                         END IF;

                         IF t >= 6.5
                              xe[1] := xe[1] + Xdrift*dt;
                              xe[2] := xe[2] + Ydrift*dt;
                         END IF;

                    IF t < 4.1
                              xe[2] := xe[2] - 9.25/4100.0;
                    END IF;

                    { ----- UPDATING MOVING AND GROUND COORDINATE SYSTEMS ----- }

                    xs[1] := Airdrop.Information[1].xg - xe[1];
                    xs[2] := xe[2];
                    xs[3] := xe[3];
                    xg[1] := Airdrop.Information[1].xg - xs[1];
                    xg[2] := xe[2];
                    xg[3] := xe[3];

                    { ----- POLL ALL VORTICES FOR MISSED DISTANCE ----- }

                    IF t >= tpoll
                         IF cds >= pcds[4]
                              tpoll := tpoll + dtpoll;
                              FOR i := 1 TO myPlane-1;
                                   ASK SELF TO pollVortices (i);
                              END FOR;
                         END IF;
                    END IF;

               ELSE { ----- when -xe[3] <= hmin, THEN PRINT DATA FOR LAST TIME ----- }

                    h        := -1.0 * xe[3];
                    vpe      := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER (ue3,2.0)
);
                    qdyn     := 0.5 * rho * POWER (vp,2.0);
                    bxy      := SQRT ( POWER (b[1,1],2.0) + POWER (b[1,2],2.0) );
                    theta    := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                    alphad   := 57.295 * ATAN2 ( salpha, calpha );
                    uxy      := SQRT ( POWER(ue[1],2.0) + POWER(ue[2],2.0) );
                    gammad   := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );

                    OUTPUT (myNumber, "R ", myPlane, " ", SimTime, " ", h, " ", xe[1], "
", xe[2], " ", vpe, " ", vp, " ", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, "
", alphad, " ", cds);

                         ASK streamS TO WriteString (INTTOSTR(repeat)    + " ");
                         ASK streamS TO WriteString (INTTOSTR(myNumber)  + "R ");
                         ASK streamS TO WriteString (INTTOSTR(myPlane)   + " ");
                         ASK streamS TO WriteString (REALTOSTR(SimTime)  + " ");
                         ASK streamS TO WriteString (REALTOSTR(h)        + " ");
                         ASK streamS TO WriteString (REALTOSTR(xe[1])    + " ");
                         ASK streamS TO WriteString (REALTOSTR(xe[2])    + " ");
                         ASK streamS TO WriteString (REALTOSTR(vpe)      + " ");
                         ASK streamS TO WriteString (REALTOSTR(vp)       + " ");
                         ASK streamS TO WriteString (REALTOSTR(mach)     + " ");
                         ASK streamS TO WriteString (REALTOSTR(qdyn)     + " ");
```

```
                     ASK streamS TO WriteString (REALTOSTR(gees)    + " ");
                     ASK streamS TO WriteString (REALTOSTR(gammad)  + " ");
                     ASK streamS TO WriteString (REALTOSTR(theta)   + " ");
                     ASK streamS TO WriteString (REALTOSTR(alphad)  + " ");
                     ASK streamS TO WriteString (REALTOSTR(cds));
                     ASK streamS TO WriteLn;

                IF printTrajectory

                          ASK stream TO WriteString (INTTOSTR(myNumber) + "R ");
                          ASK stream TO WriteString (REALTOSTR(SimTime) + " ");
                          ASK stream TO WriteString (REALTOSTR(h)       + " ");
                          ASK stream TO WriteString (REALTOSTR(xe[1])   + " ");
                          ASK stream TO WriteString (REALTOSTR(xe[2])   + " ");
                          ASK stream TO WriteString (REALTOSTR(vpe)     + " ");
                          ASK stream TO WriteString (REALTOSTR(vp)      + " ");
                          ASK stream TO WriteString (REALTOSTR(mach)    + " ");
                          ASK stream TO WriteString (REALTOSTR(qdyn)    + " ");
                          ASK stream TO WriteString (REALTOSTR(gees)    + " ");
                          ASK stream TO WriteString (REALTOSTR(gammad)  + " ");
                          ASK stream TO WriteString (REALTOSTR(theta)   + " ");
                          ASK stream TO WriteString (REALTOSTR(alphad)  + " ");
                          ASK stream TO WriteString (REALTOSTR(cds));
                          ASK stream TO WriteLn;

                END IF;

                bigloop := 1;

           END IF {hmin};

           END WAIT;

           END WHILE;

           IF printTrajectory
                ASK stream TO Close;
                DISPOSE (stream);
           END IF;

           DISPOSE (SELF);

      END {ASK} METHOD {jump};

ASK METHOD initialize (IN stick   : INTEGER;
                       IN Counter : INTEGER);

     BEGIN

          myPlane            := Counter;
          myNumber           := stick;
          gees               := 0.0;
          cds                := 0.0;
          myTime             := SimTime;
          t                  := SimTime-myTime;
          gmax               := 0.0;
          tar                := t;
          tpoll              := t;
          tdrift             := t + dtdrift;
          alt                := -1.0 * xe[3];
          h                  := alt;
          bigloop            := 0;
          myDrift          := seed1.UniformReal (0.0, 4.0);
          myDriftDirection   := seed2.UniformReal (0.0, 360.0);
          xe[1]              := Airdrop.Information[myPlane].xg;
          xlast              := xe[1];
          xe[2]              := Airdrop.Information[myPlane].yg + 9.25;
          alt                := Airdrop.Information[myPlane].altitude;
          xe[3]              := -1.0 * alt;
          ue[1]              := Airdrop.Information[myPlane].vf;

          vwind[1]           := HeadWind;  {FROM globalMod}
```

```
                NEW (lastRightLocation, 1..myPlane-1);
                NEW (lastLeftLocation,  1..myPlane-1);

                FOR i := 1 TO myPlane-1
                     lastRightLocation[i] := 1;
                     lastLeftLocation[i] := 1;
                END FOR;

                ASK SELF TO findDrift;

                IF printTrajectory
                     NEW (stream);
                     outfile := "RJ" + INTTOSTR(myPlane) + INTTOSTR(myNumber) + extension +
".mat";
                     ASK stream TO Open (outfile, Output);
                END IF;

                ASK streamI TO WriteString (INTTOSTR(repeat) + " ");
                ASK streamI TO WriteString (INTTOSTR(myPlane) + " ");
                ASK streamI TO WriteString (REALTOSTR(SimTime) + " ");
                ASK streamI TO WriteString (INTTOSTR(stick) + "R ");
                ASK streamI TO WriteString (REALTOSTR(weight) + " ");
                ASK streamI TO WriteString (REALTOSTR(xe[1]) + " ");
                ASK streamI TO WriteString (REALTOSTR(xe[2]) + " ");
                ASK streamI TO WriteString (REALTOSTR(alt) + " ");
                ASK streamI TO WriteLn;

          END METHOD {initialize};

     ASK METHOD pollVortices (IN vortexPlane : INTEGER);

          VAR

               x, xcord1, xcord2, vvx, vjx      : REAL;
               y, ycord1, ycord2, vvy, vjy      : REAL;
               z, zcord1, zcord2, vvz, vjz      : REAL;
               vjdistance, vvdistance, distance : REAL;
               projection                       : REAL;
               i, location                      : INTEGER;
               check                            : BOOLEAN;
               startRightSearch                 : INTEGER;
               startLeftSearch                  : INTEGER;

          BEGIN

               check := FALSE;
               location := 0;

               startRightSearch := lastRightLocation[vortexPlane];

               { ----- POLL RIGHT VORTEX ----- }

               FOR i := startRightSearch TO Airdrop.Information[vortexPlane].NumberOfSteps
                    IF location = 0
                         IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].xCord-xs[1])) <=
50.0
                              lastRightLocation[vortexPlane] := i;
                              IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].zCord+xs[3])) <=
50.0
                                   lastRightLocation[vortexPlane] := i;
                                   IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].yCord-xs[2])) <=
50.0
                                        check        := TRUE;
                                        location     := i;
                                        IF location =
Airdrop.Information[vortexPlane].NumberOfSteps
                                             check   := FALSE;
                                        END IF;
                                        lastRightLocation[vortexPlane] := i;
                                   ELSE
```

```
                                          location := i;
                                     END IF;
                                ELSE
                                     location := i;
                                END IF;
                           END IF;
                      ELSE
                           EXIT;
                      END IF;
                 END FOR;

                 IF check
                      xcord1 :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].xCord;
                      ycord1 :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].yCord;
                      zcord1 :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].zCord;
                      xcord2 :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].xCord;
                      ycord2 :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].yCord;
                      zcord2 :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].zCord;

                      vvx := xcord2-xcord1;
                      vvy := ycord2-ycord1;
                      vvz := zcord2-zcord1;

                      vjx := xs[1]-xcord1;
                      vjy := xs[2]-ycord1;
                      vjz := -1.0*xs[3]-zcord1;

                      vjdistance := SQRT(POWER(vjx,2.0)+POWER(vjy,2.0)+POWER(vjz,2.0));
                      vvdistance := SQRT(POWER(vvx,2.0)+POWER(vvy,2.0)+POWER(vvz,2.0));
                      projection := (vjx*vvx + vjy*vvy + vjz*vvz)/vvdistance;
                      distance   := SQRT(POWER(vjdistance,2.0)-POWER(projection,2.0));
                      IF distance <=
MAXOF(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].radius,
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].radius)

                          OUTPUT (myNumber, "R ", myPlane, " RV ", vortexPlane, " ", -
1.0*xe[3], " ", distance, " ", location, " ", SimTime);

                          ASK streamE TO WriteString (INTTOSTR(repeat) + " ");
                          ASK streamE TO WriteString (INTTOSTR(myNumber) + "R ");
                          ASK streamE TO WriteString (INTTOSTR(myPlane) + " RV ");
                          ASK streamE TO WriteString (INTTOSTR(vortexPlane) + " ");
                          ASK streamE TO WriteString (REALTOSTR(-1.0*xe[3]) + " ");
                          ASK streamE TO WriteString (REALTOSTR(distance) + " ");
                          ASK streamE TO WriteString (REALTOSTR(location) + " ");
                          ASK streamE TO WriteString (REALTOSTR(SimTime));
                          ASK streamE TO WriteLn;

                     END IF;

                END IF;

                check := FALSE;
                location := 0;

                startLeftSearch := lastLeftLocation[vortexPlane];

                { ----- POLL LEFT VORTEX ----- }

                FOR i := startLeftSearch TO Airdrop.Information[vortexPlane].NumberOfSteps
                     IF location = 0
                          IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].xCord-xs[1])) <=
50.0
                               lastLeftLocation[vortexPlane] := i;
```

```
                              IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].zCord+xs[3])) <=
50.0
                                   lastRightLocation[vortexPlane] := i;
                                   IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].yCord-xs[2])) <=
50.0
                                        check        := TRUE;
                                        location     := i;
                                        IF location =
Airdrop.Information[vortexPlane].NumberOfSteps
                                             check   := FALSE;
                                        END IF;
                                        lastRightLocation[vortexPlane] := i;
                                   ELSE
                                        location := i;
                                   END IF;
                              ELSE
                                   location := i;
                              END IF;
                         END IF;
                    ELSE
                         EXIT;
                    END IF;
               END FOR;

               IF check
                    xcord1 :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].xCord;
                    ycord1 :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].yCord;
                    zcord1 :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].zCord;
                    xcord2 :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].xCord;
                    ycord2 :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].yCord;
                    zcord2 :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].zCord;

                    vvx := xcord2-xcord1;
                    vvy := ycord2-ycord1;
                    vvz := zcord2-zcord1;

                    vjx := xs[1]-xcord1;
                    vjy := xs[2]-ycord1;
                    vjz := -1.0*xs[3]-zcord1;

                    vjdistance := SQRT(POWER(vjx,2.0)+POWER(vjy,2.0)+POWER(vjz,2.0));
                    vvdistance := SQRT(POWER(vvx,2.0)+POWER(vvy,2.0)+POWER(vvz,2.0));
                    projection := (vjx*vvx + vjy*vvy + vjz*vvz)/vvdistance;
                    distance   := SQRT(POWER(vjdistance,2.0)-POWER(projection,2.0));
                    IF distance <=
MAXOF(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].radius,
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].radius)

                        OUTPUT (myNumber, "R ", myPlane, " LV ", vortexPlane, " ", -
1.0*xe[3], " ", distance, " ", location, " ", SimTime);

                        ASK streamE TO WriteString (INTTOSTR(repeat) + " ");
                        ASK streamE TO WriteString (INTTOSTR(myNumber) + "R ");
                        ASK streamE TO WriteString (INTTOSTR(myPlane) + " LV ");
                        ASK streamE TO WriteString (INTTOSTR(vortexPlane) + " ");
                        ASK streamE TO WriteString (REALTOSTR(-1.0*xe[3]) + " ");
                        ASK streamE TO WriteString (REALTOSTR(distance) + " ");
                        ASK streamE TO WriteString (REALTOSTR(location) + " ");
                        ASK streamE TO WriteString (REALTOSTR(SimTime));
                        ASK streamE TO WriteLn;

                   END IF;

               END IF;
```

```
          END METHOD {pollVortices};

     ASK METHOD changeDrift;

          BEGIN

               myDrift          := seed1.UniformReal (0.0, 4.0);
               myDriftDirection := seed2.Normal (myDriftDirection, 2.8125);
               ASK SELF TO findDrift;

          END {ASK} METHOD {changeDrift};

     ASK METHOD findDrift;

          BEGIN

               Xdrift := myDrift * COS (myDriftDirection*pi/180.0);
               Ydrift := myDrift * SIN (myDriftDirection*pi/180.0);

          END {ASK} METHOD {findDrift};

END OBJECT {rightJumperObj};

END {IMPLEMENTATION} MODULE {rightJumperMod}.
```

APPENDIX E
Propagation Time Step Comparisons
(No Winds)

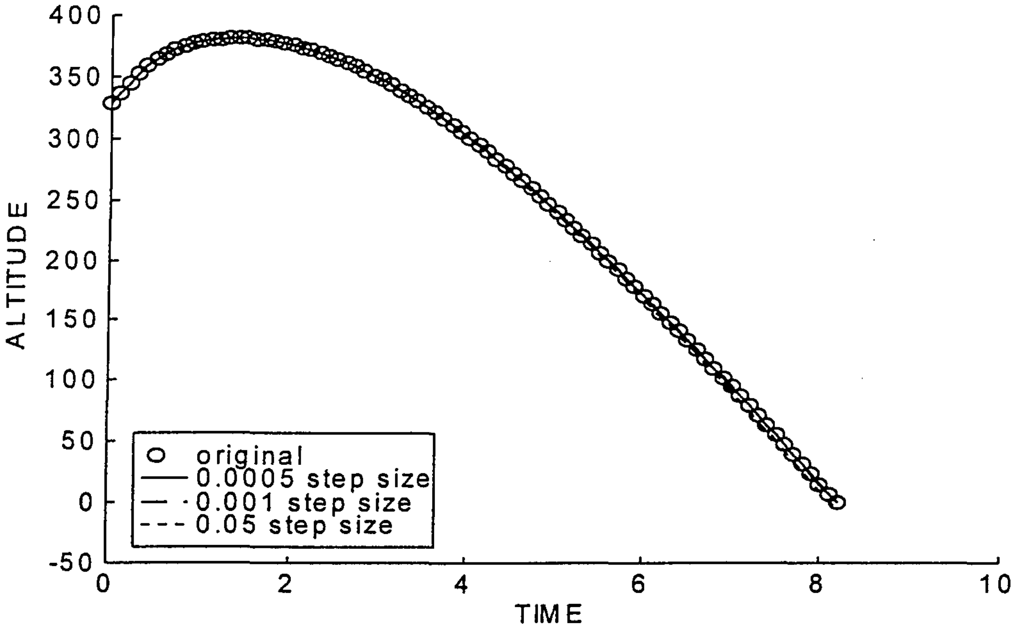

Figure 31. Altitude Vs. Time

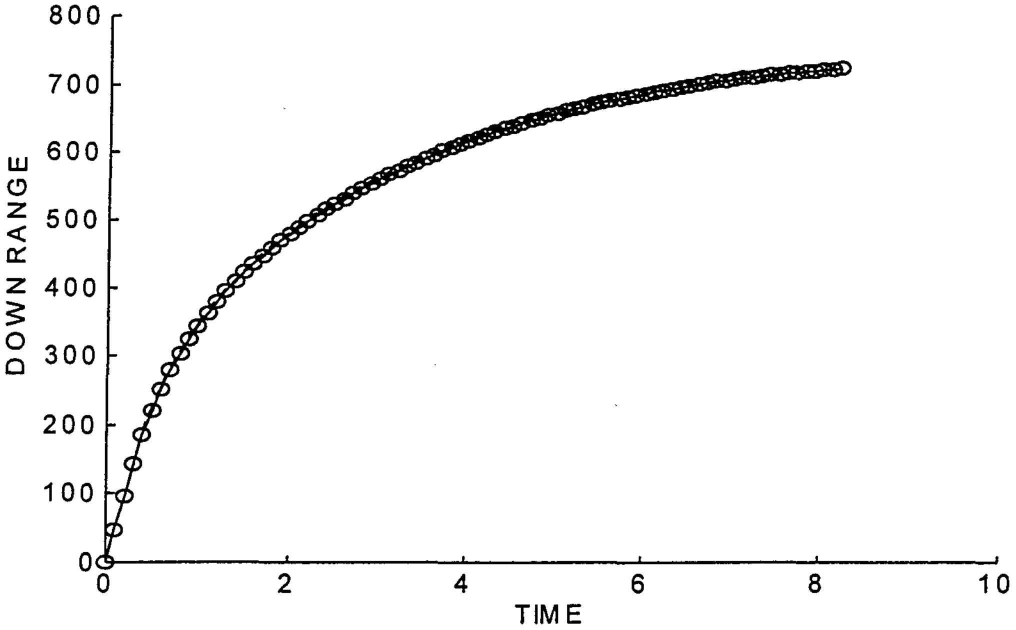

Figure 32. Down Range Vs. Time

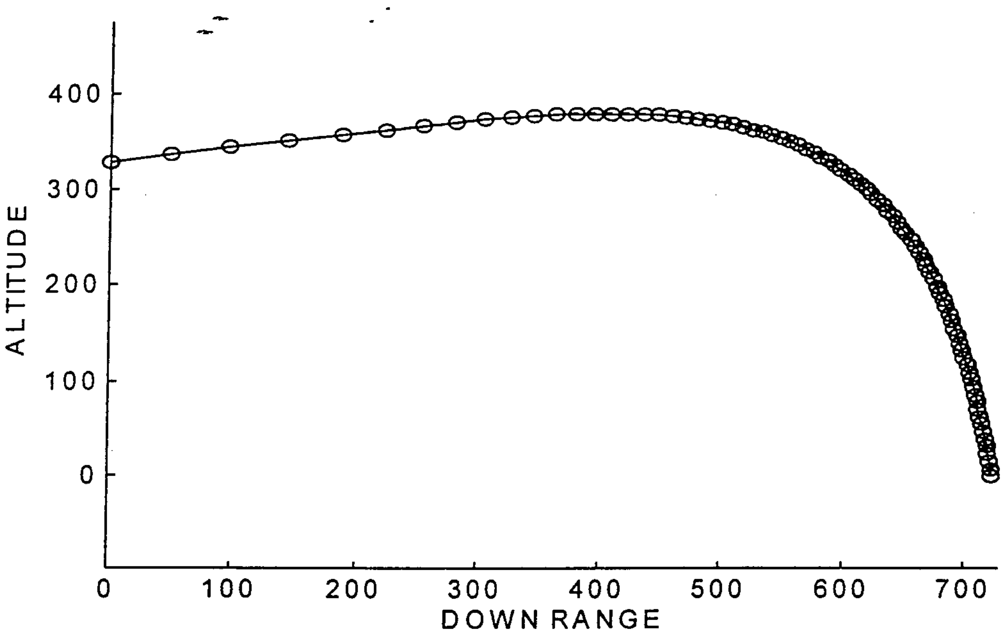

Figure 33. Altitude Vs. Down Range

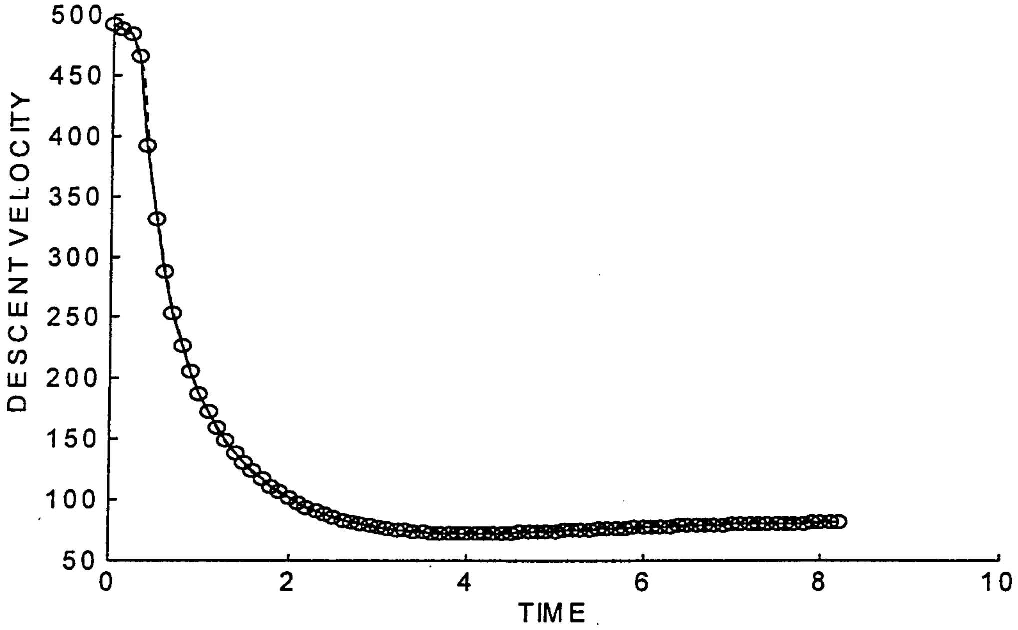

Figure 34. Descent Velocity Vs. Time

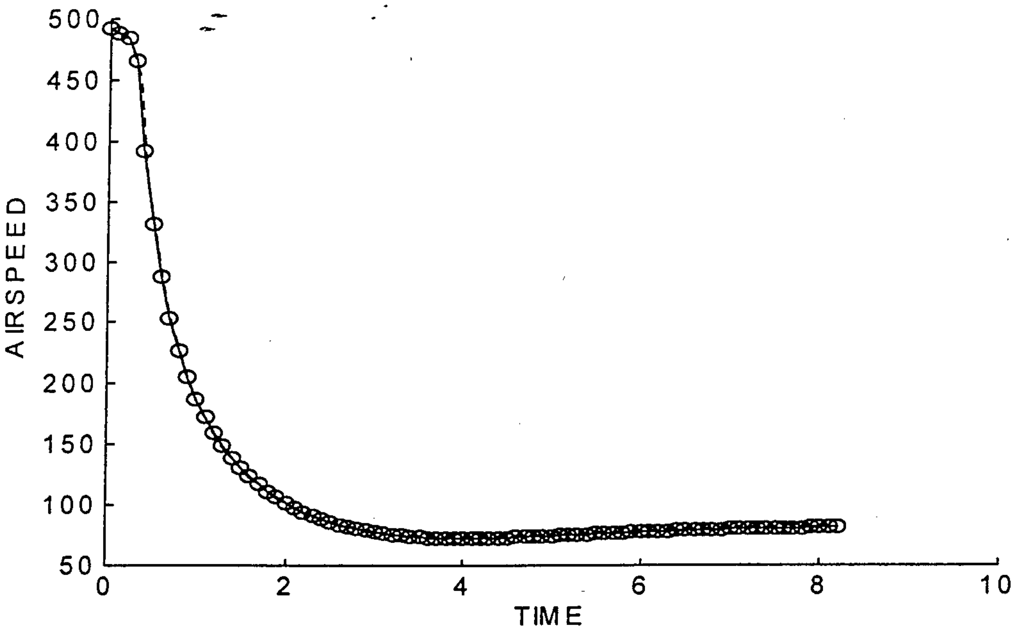

Figure 35. Airspeed Vs. Time

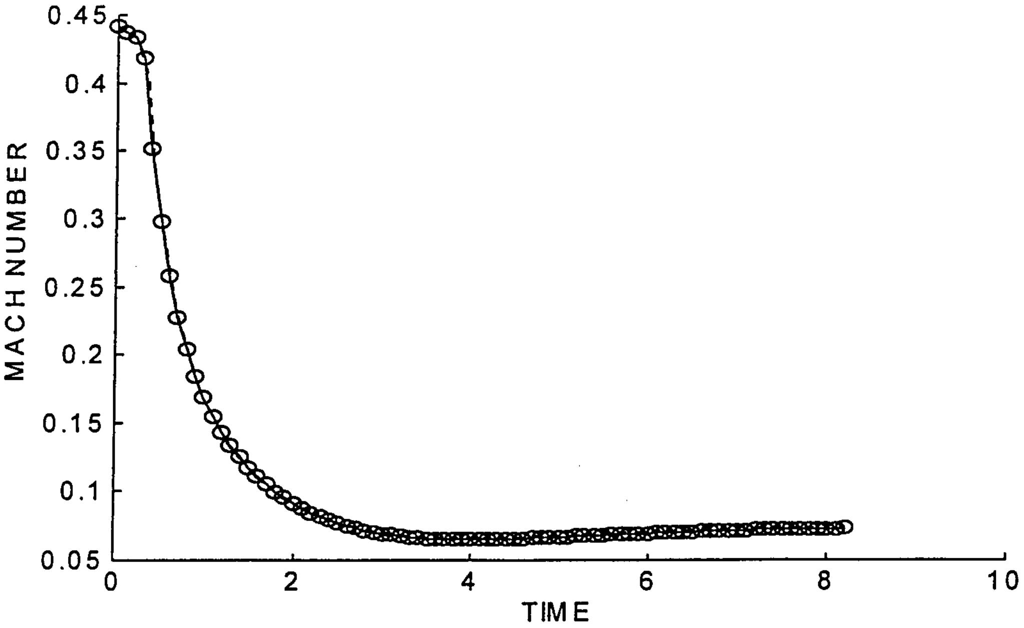

Figure 36. Mach Number Vs. Time

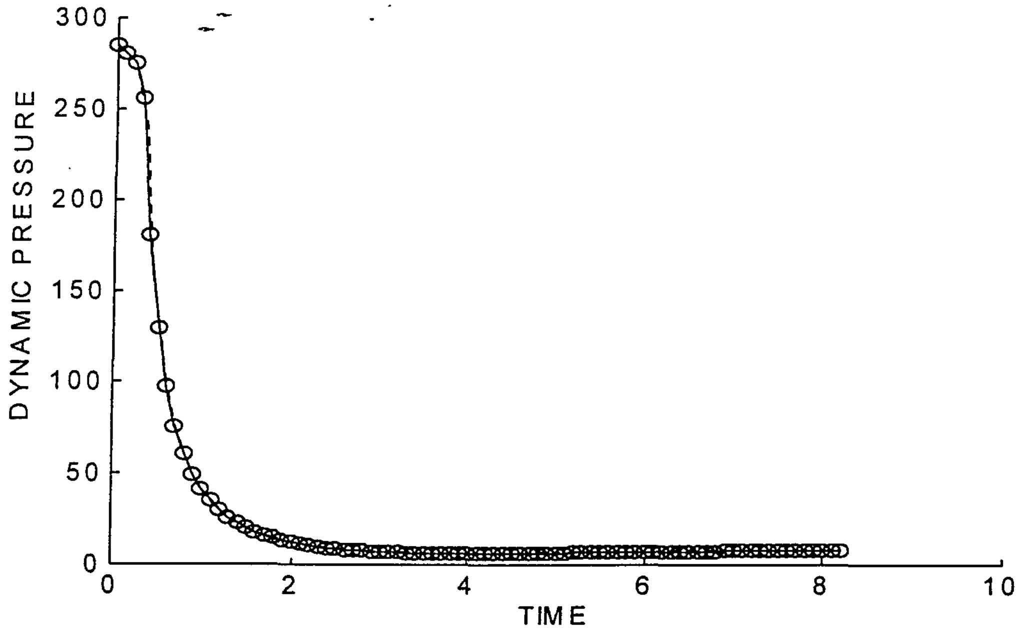

Figure 37. Dynamic Pressure Vs. Time

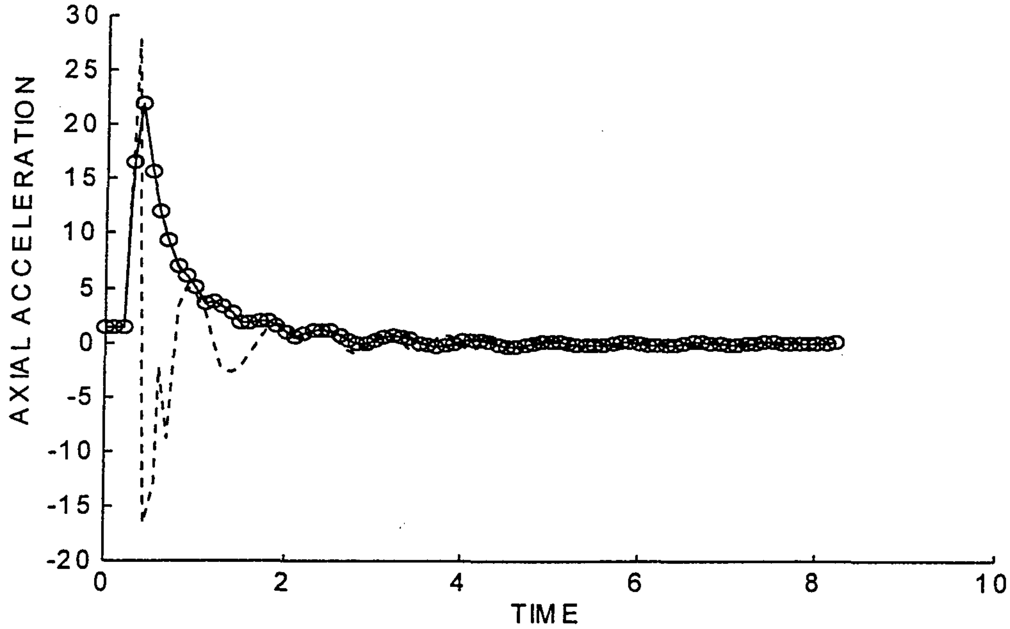

Figure 38. Axial Acceleration Vs. Time

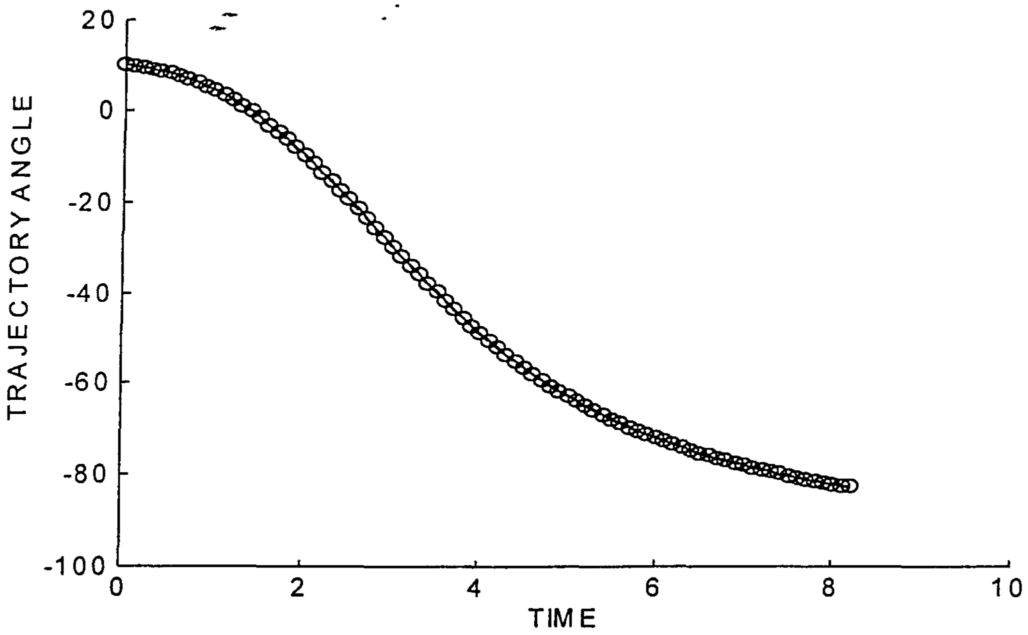

Figure 39. Trajectory Angle Vs. Time

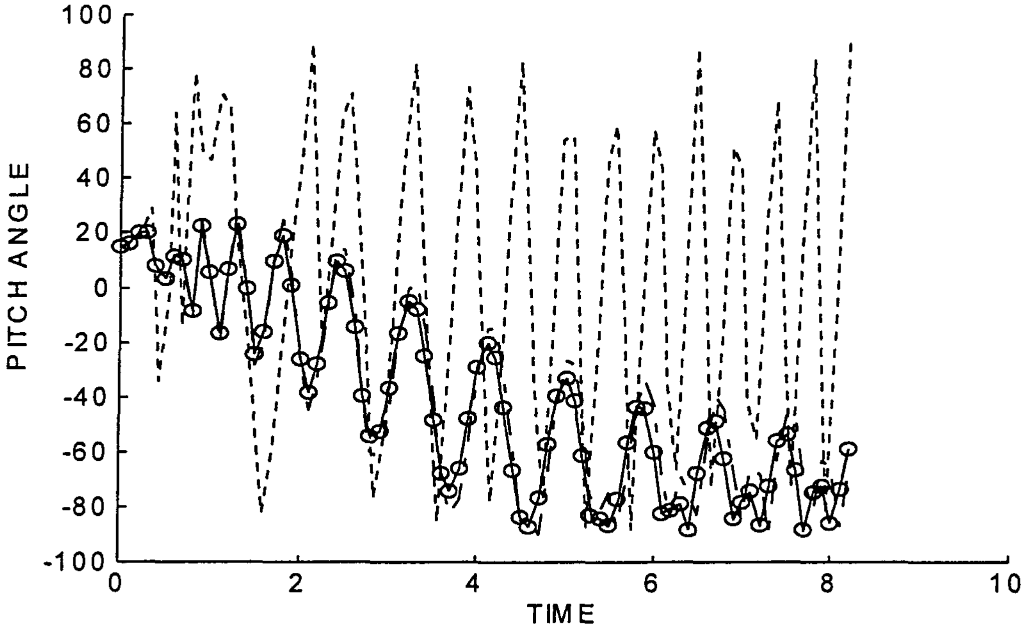

Figure 40. Pitch Angle Vs. Time

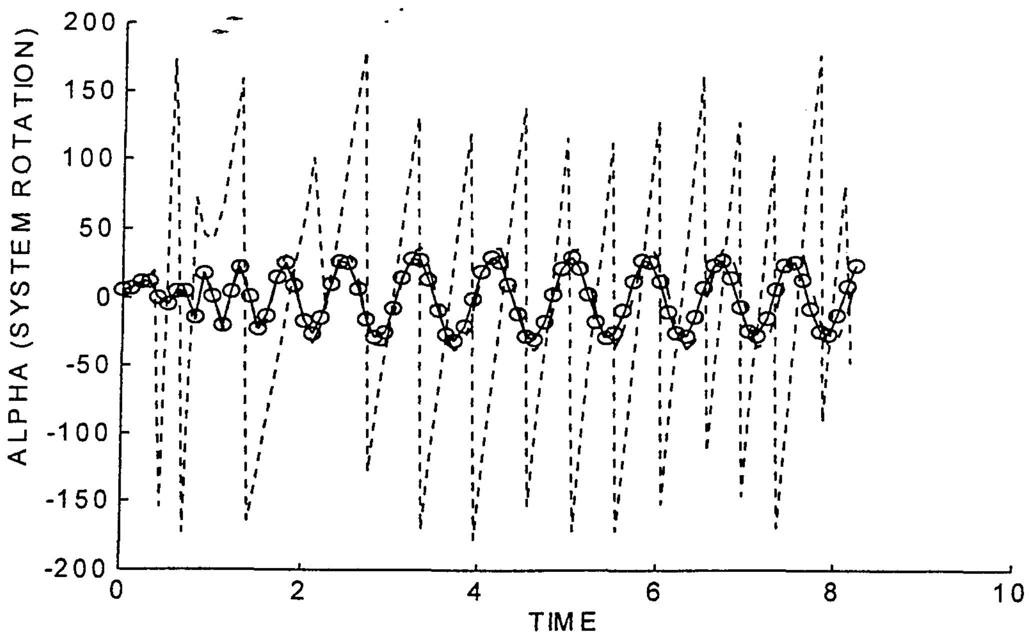

Figure 41. System Rotation Vs. Time

(Wind Conditions Present)

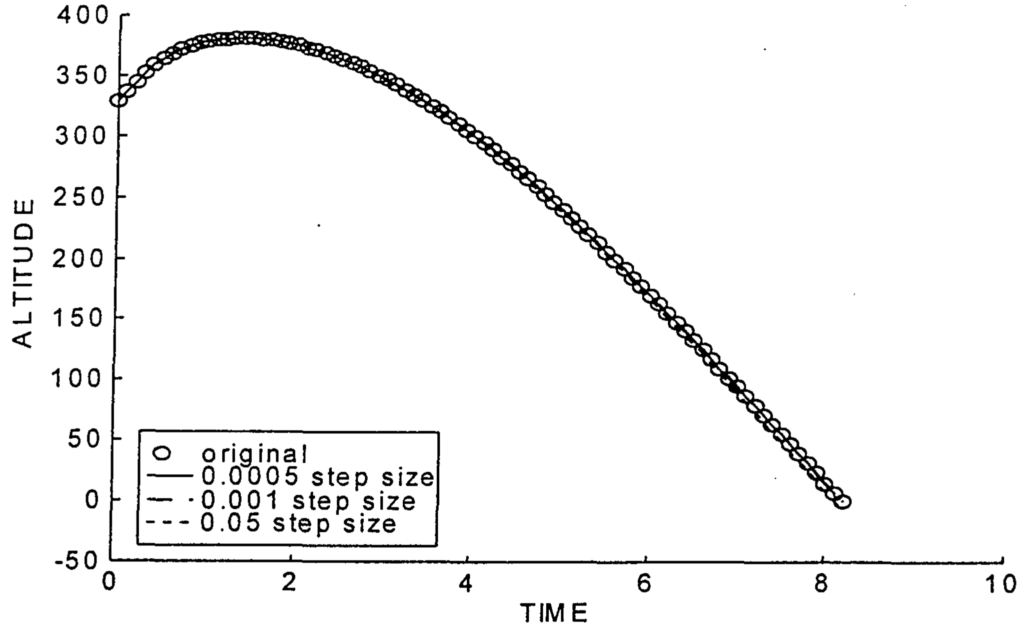

Figure 42. Altitude Vs. Time

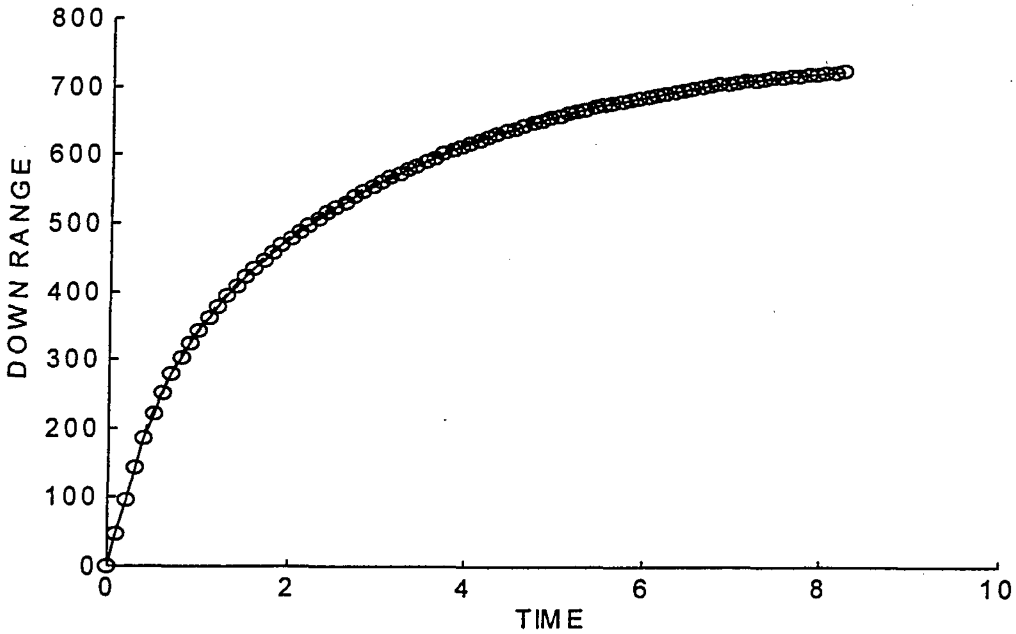

Figure 43. Down Range Vs. Time

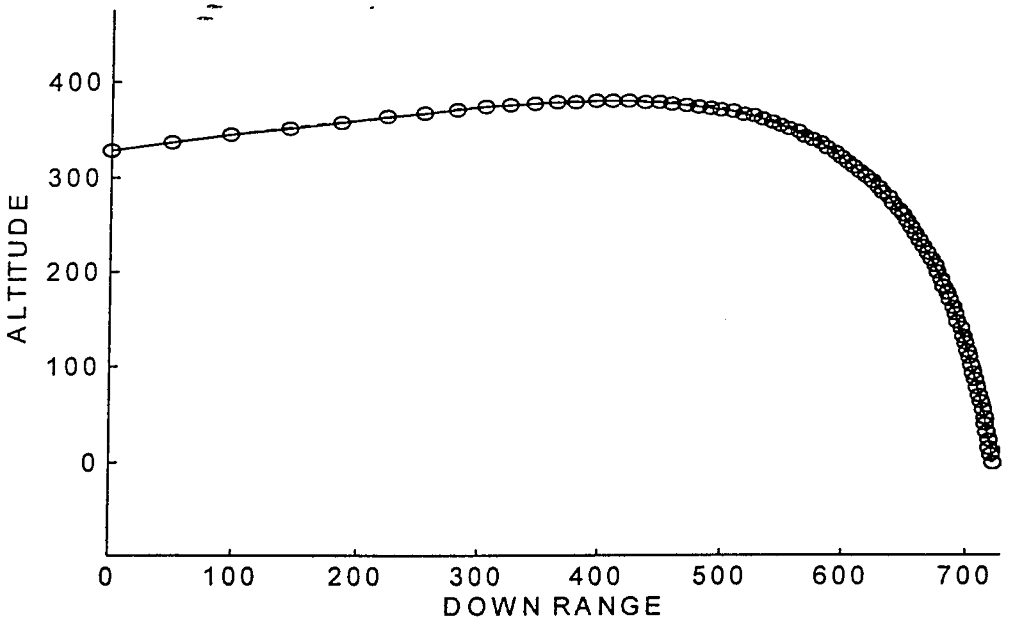

Figure 44. Altitude Vs. Down Range

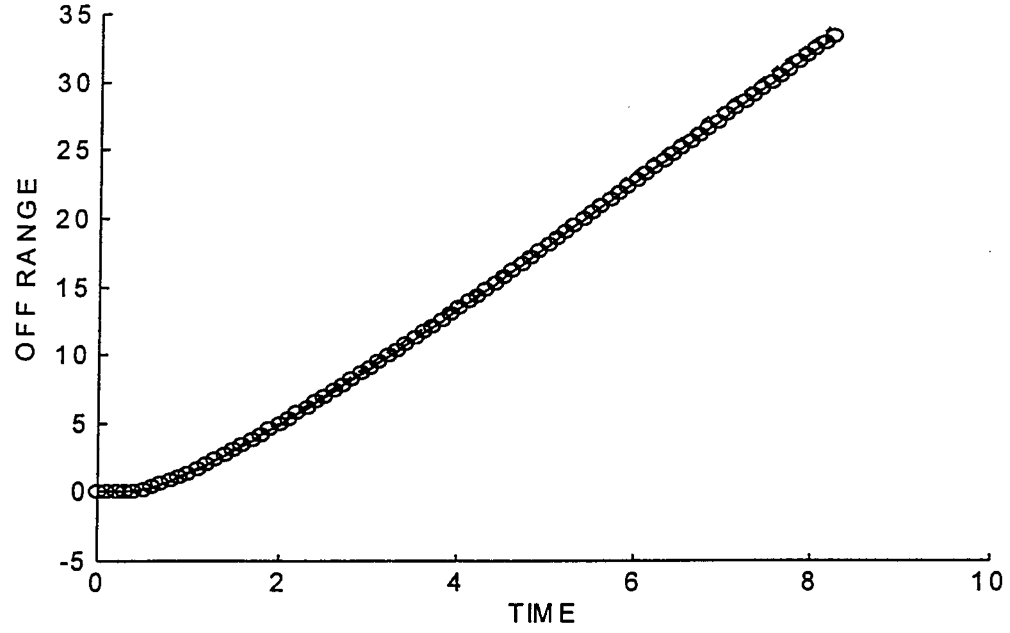

Figure 45. Off Range Vs. Time

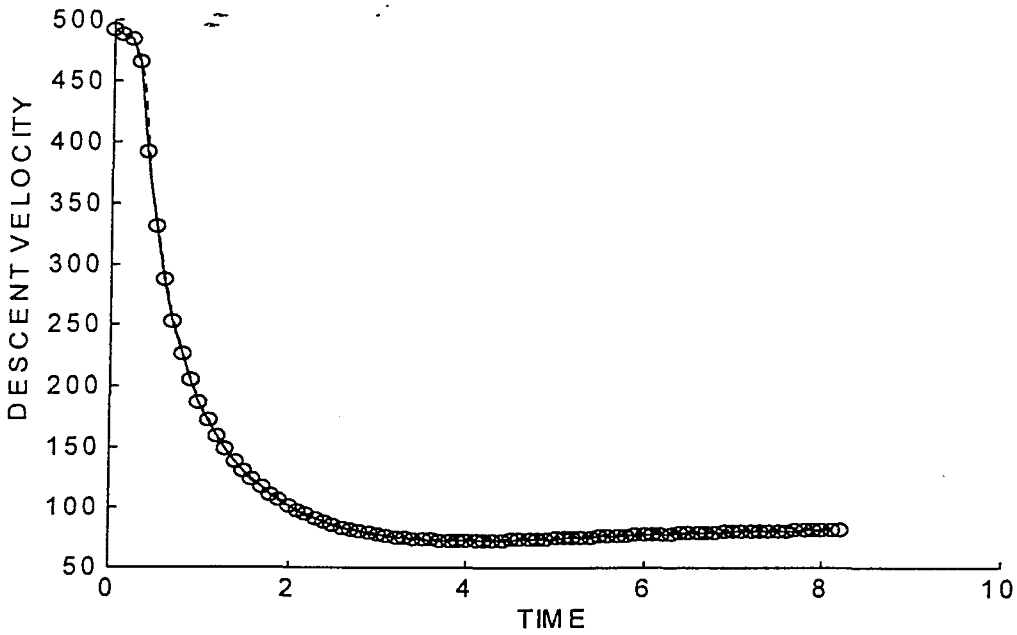

Figure 46. Descent Velocity Vs. Time

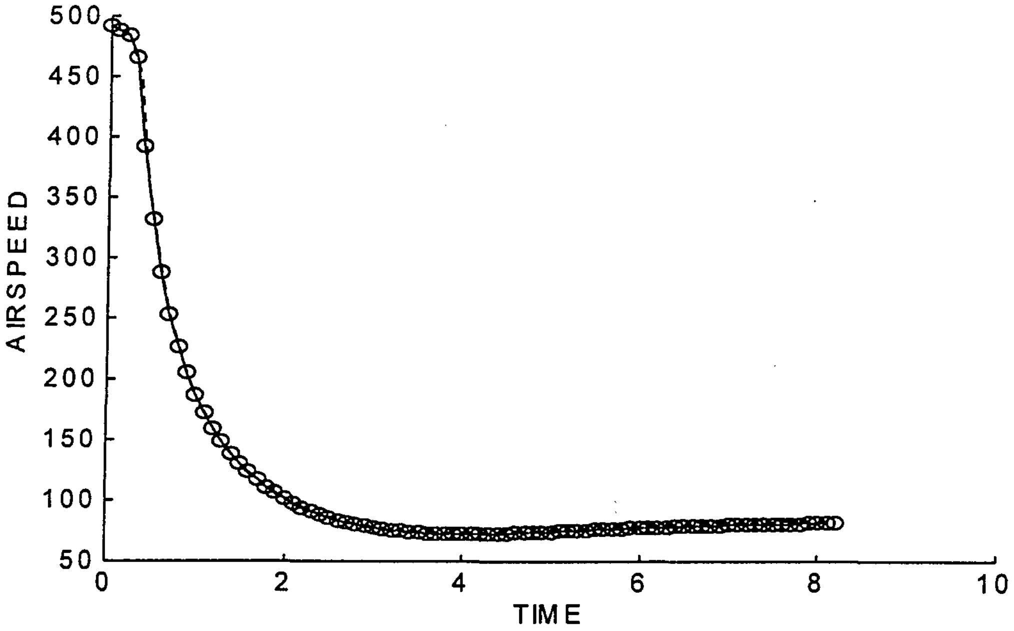

Figure 47. Airspeed Vs. Time

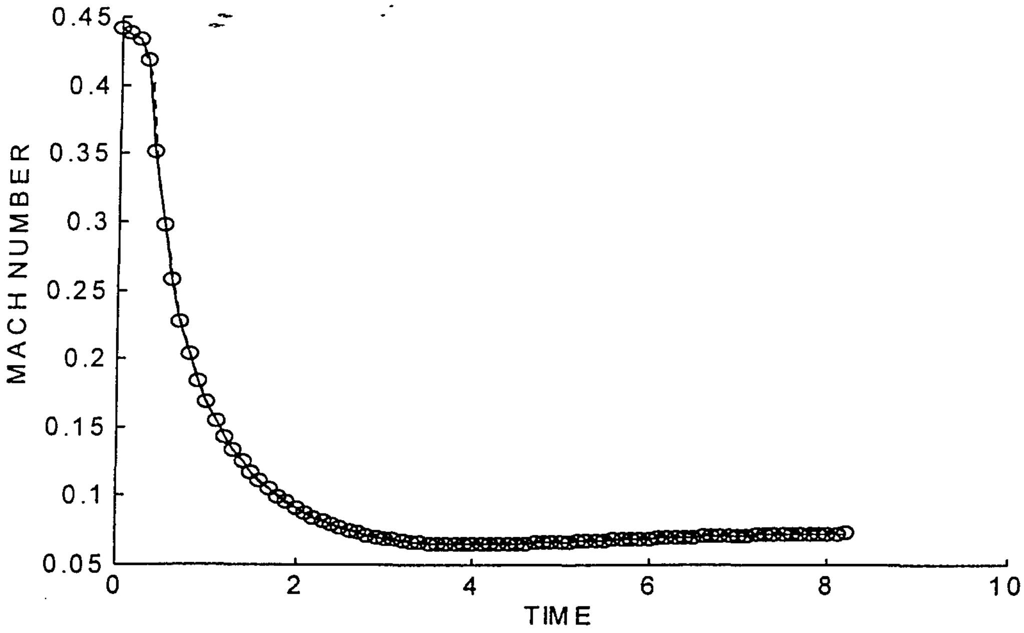

Figure 48. Mach Number Vs. Time

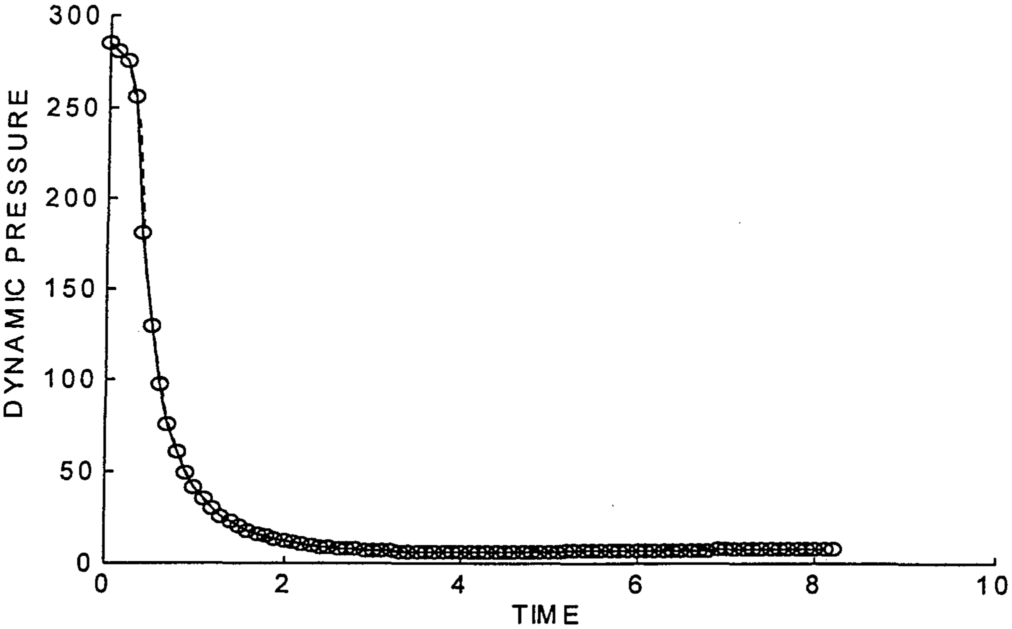

Figure 49. Dynamic Pressure Vs. Time

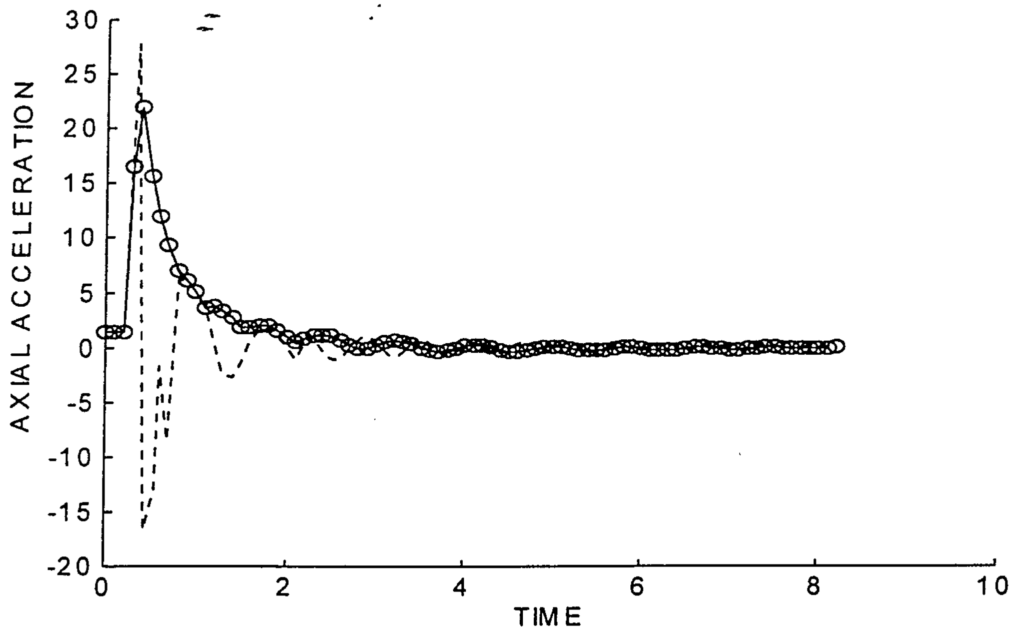

Figure 50. Axial Acceleration Vs. Time

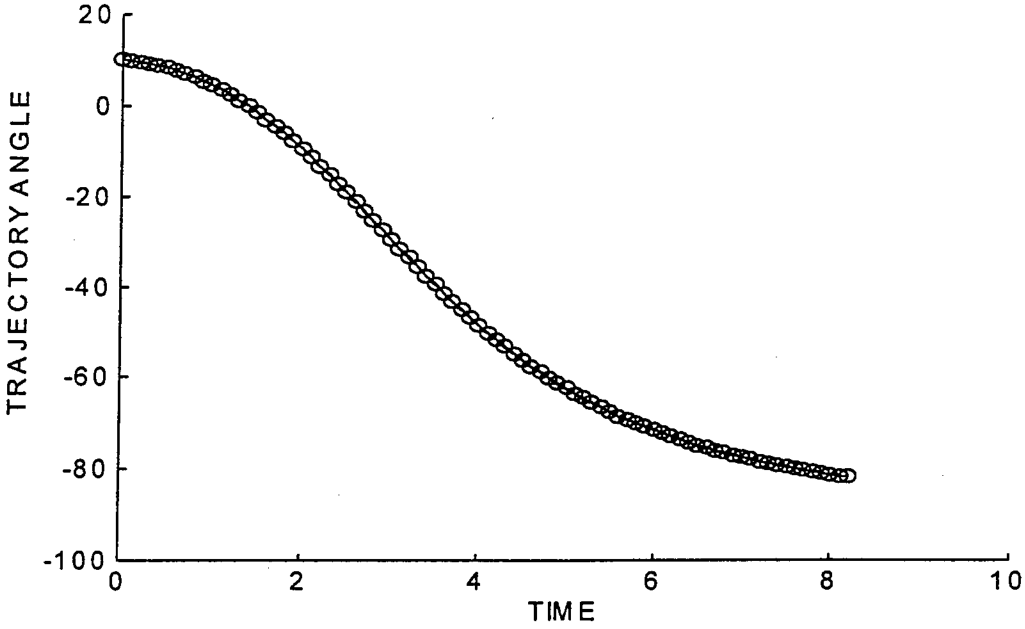

Figure 51. Trajectory Angle Vs. Time

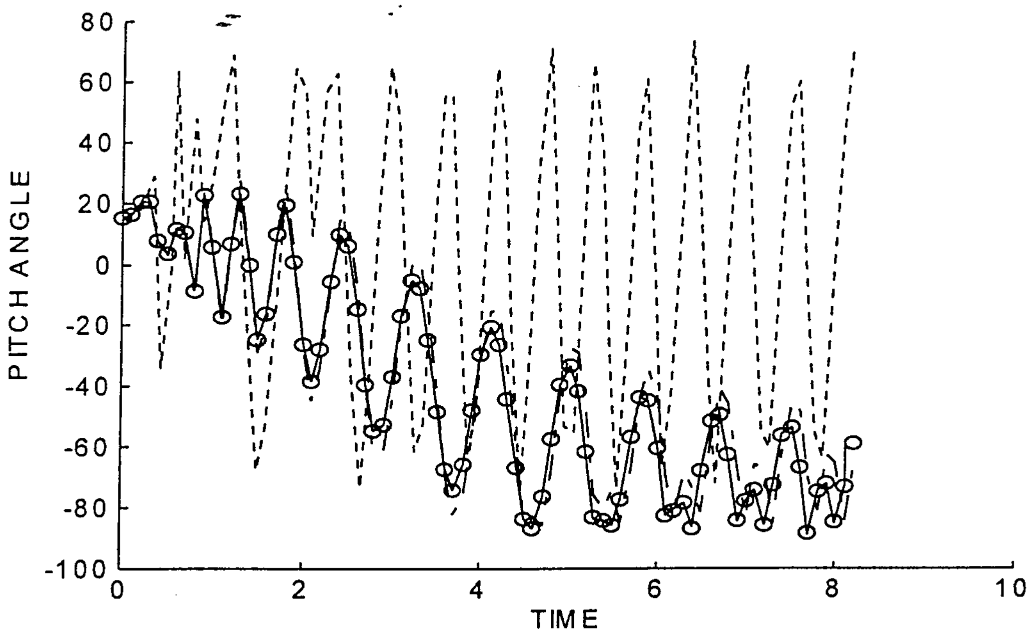

Figure 52. Pitch Angle Vs. Time

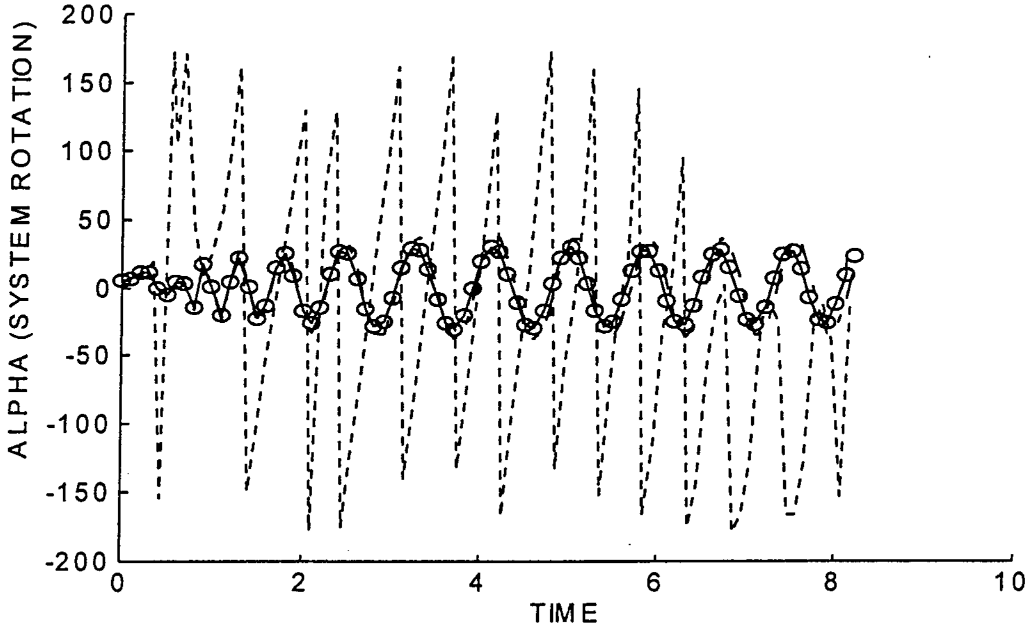

Figure 53. System Rotation Vs. Time

APPENDIX F
Experimental Design Analysis Results

Table 14
Design Point Description

Design Point	Seed Used	Trail Distance (ft)		Lateral Separation (ft)	
		Real	*Coded*	*Real*	*Coded*
1	1	15,000	-	0	-
2	2	15,000	-	500	+
3	3	32,000	+	0	-
4	4	32,000	+	500	+
5	5	23,500	0	250	0
6	6	15,000	-	250	0
7	7	23,500	0	0	-
8	8	32,000	+	250	0
9	9	23,500	0	500	+

Table 15
Design Point 1
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

		Company	
Run	**1**	**2**	**3**
1	430.0703	360.9418	389.4421
	66.9743	*35.0532*	*56.0624*
2	435.7381	359.4109	384.7579
	76.7285	*33.2316*	*54.573*
3	437.2077	361.274	384.6344
	73.434	*36.7194*	*54.731*
4	439.495	362.7578	389.312
	76.6081	*34.8662*	*55.036*
5	432.4826	362.8335	387.8803
	78.5673	*37.6427*	*57.4444*
6	440.9464	364.7747	389.0972
	74.4567	*33.7632*	*55.2675*
7	435.1913	360.851	385.1381
	71.6705	*37.2194*	*59.8104*
8	436.1233	361.5649	389.3502
	73.086	*32.6076*	*59.514*
9	439.5211	364.8641	387.8497
	74.6969	*34.6763*	*62.3549*
10	436.3754	366.1484	393.4379
	70.6358	*33.2577*	*60.258*
11	433.4462	366.547	388.3404
	75.7921	*26.8489*	*56.7347*
12	438.1389	362.4927	384.8852
	76.0421	*38.9077*	*61.7537*

Table 16
Design Point 2
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

		Company	
Run	**1**	**2**	**3**
1	503.6917	291.8136	320.3443
	95.4484	*83.1988*	*99.9413*
2	505.4644	286.7763	320.9853
	100.3061	*84.0816*	*99.838*
3	504.9273	289.429	313.3477
	97.159	*85.1705*	*96.2883*
4	506.7988	285.4877	320.8816
	102.3825	*87.3584*	*94.9966*
5	502.9726	280.5451	324.3242
	94.5522	*80.9113*	*100.4835*
6	504.9503	292.0325	319.6078
	95.2559	*82.4191*	*98.1254*
7	501.9642	286.1554	314.252
	100.7286	*86.4196*	*98.6861*
8	502.0507	287.9634	320.9301
	102.2936	*82.949*	*92.7337*
9	504.6824	288.4857	320.0993
	97.807	*85.9369*	*98.3397*
10	507.5064	290.7363	316.6711
	96.7031	*83.6097*	*94.6602*
11	507.2636	283.3885	321.9584
	97.0514	*81.6215*	*97.8721*
12	507.2985	284.5784	317.1065
	99.1181	*82.6224*	*90.1595*

Table 17
Design Point 3
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

		Company	
Run	**1**	**2**	**3**
1	430.4193	362.1975	395.0287
	76.6798	*33.6641*	*63.3109*
2	429.7015	364.5905	391.5208
	70.1104	*32.5386*	*57.8174*
3	428.6318	358.3023	395.2355
	75.4378	*33.2877*	*64.0831*
4	428.983	361.1818	397.8057
	74.1196	*29.0871*	*61.7333*
5	428.9359	358.9821	391.6958
	70.7156	*28.5781*	*60.588*
6	426.9134	364.7675	389.6335
	74.828	*31.3469*	*64.0662*
7	427.2351	361.5911	390.125
	67.914	*29.0319*	*59.5754*
8	424.3669	359.0497	390.1488
	75.1556	*32.9695*	*58.6665*
9	430.1527	365.4308	389.9997
	69.624	*26.3724*	*63.5268*
10	423.7409	360.0008	394.0922
	73.8034	*28.0813*	*63.9653*
11	428.9483	358.8766	394.8331
	74.7155	*30.6835*	*58.7379*
12	428.4258	361.8863	393.07
	72.8381	*32.1484*	*63.8151*

Table 18
Design Point 4
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

	Company		
Run	**1**	**2**	**3**
1	499.6217	284.3822	328.0279
	94.7887	*81.0878*	*106.4739*
2	496.7935	287.3618	330.9178
	99.7158	*82.0323*	*103.4148*
3	494.2892	287.2261	328.161
	98.6101	*86.2168*	*102.6164*
4	495.6269	283.0487	329.2329
	95.4149	*84.8408*	*98.2828*
5	497.7153	287.8541	324.6052
	98.7909	*83.7419*	*100.9202*
6	498.8645	285.3582	326.361
	94.7652	*86.7337*	*97.9337*
7	497.6243	287.7428	324.3128
	93.7862	*86.3785*	*102.5909*
8	502.3004	285.5206	326.4205
	95.6655	*87.4673*	*95.9542*
9	498.8575	284.9746	326.3337
	95.2053	*83.7642*	*101.6063*
10	496.2071	285.2436	327.2455
	93.6687	*84.3306*	*106.2513*
11	491.7247	284.9658	329.3919
	94.6025	*81.9371*	*104.9135*
12	496.7456	283.941	328.4583
	99.7924	*83.141*	*97.1479*

Table 19
Design Point 5
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

	Company		
Run	**1**	**2**	**3**
1	468.1302	319.4243	354.674
	79.034	*53.2432*	*72.7081*
2	502.6151	283.7802	318.1862
	63.0635	*43.202*	*77.1984*
3	467.1782	321.6672	354.2458
	80.9138	*50.5145*	*70.0561*
4	461.9555	322.3655	360.5623
	80.5971	*50.6149*	*73.8028*
5	463.5697	321.7715	356.1365
	77.5801	*48.3536*	*73.9633*
6	465.8508	323.945	360.0028
	74.9795	*48.0725*	*75.4596*
7	465.0878	322.6825	354.46
	77.3808	*51.0495*	*73.719*
8	470.5861	324.7836	352.4648
	79.7612	*53.0919*	*71.1524*
9	467.7505	320.5336	347.3068
	78.0351	*54.3631*	*74.4806*
10	464.2716	325.7187	355.7873
	79.9024	*57.1777*	*69.5746*
11	462.4232	325.5173	357.6986
	80.4521	*51.8559*	*71.1894*
12	462.7124	323.9769	352.3883
	79.1749	*51.9871*	*70.2033*

Table 20
Design Point 6
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

	Company		
Run	**1**	**2**	**3**
1	466.2106	321.3344	354.1209
	82.099	*55.28*	*73.5537*
2	465.632	322.7295	349.6821
	78.3881	*53.9951*	*76.4318*
3	471.0641	325.9871	351.8965
	76.9606	*53.4872*	*73.92*
4	464.0373	322.2092	356.4903
	80.1052	*52.102*	*69.5845*
5	469.5666	324.6656	351.3001
	80.8256	*51.5468*	*76.5468*
6	468.87	326.2134	352.3676
	78.3476	*52.0792*	*75.2512*
7	469.3715	319.1993	351.8846
	78.6952	*51.3488*	*74.4209*
8	462.1089	323.1526	352.1188
	74.1749	*50.8189*	*77.4046*
9	468.4109	322.657	347.4057
	81.6074	*52.308*	*70.4199*
10	475.9776	324.2565	357.0102
	81.2664	*52.4166*	*75.7948*
11	468.8593	324.9368	353.1134
	81.1118	*51.1617*	*70.5241*
12	468.9888	324.6072	349.1569
	77.6527	*52.3715*	*65.3615*

Table 21
Design Point 7
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

		Company	
Run	1	2	3
1	434.2541	358.2854	390.6165
	82.2195	*25.0726*	*59.428*
2	431.828	361.756	392.561
	74.7502	*31.3976*	*52.9896*
3	430.5695	355.67	388.312
	75.9724	*32.4983*	*58.6882*
4	429.3265	358.9588	392.4118
	75.4169	*28.8031*	*60.5126*
5	430.1651	363.8356	391.7858
	68.8505	*32.6825*	*59.0724*
6	436.1033	357.7043	392.7163
	68.4442	*34.3462*	*59.0649*
7	434.2491	362.9033	390.6143
	80.3527	*34.535*	*58.0623*
8	431.41	357.7596	391.1626
	72.5305	*33.9889*	*57.4189*
9	433.5167	356.8865	397.0389
	73.5515	*30.6631*	*57.9238*
10	434.4781	362.4084	389.9141
	73.1101	*31.4961*	*60.8326*
11	434.967	361.259	394.661
	77.0348	*32.9094*	*57.6345*
12	435.7988	360.1292	388.6504
	73.5451	*27.3131*	*57.2356*

Table 22
Design Point 8
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

	Company		
Run	**1**	**2**	**3**
1	459.4309	319.2125	355.7095
	72.5859	*50.0239*	*76.9929*
2	459.2828	325.5855	360.2764
	78.1941	*52.9487*	*77.9914*
3	465.1589	319.3877	362.3887
	80.4955	*51.5492*	*76.8498*
4	458.7631	321.2724	359.7524
	75.5444	*51.9716*	*76.7526*
5	459.2312	322.9782	357.3255
	76.3075	*48.9529*	*74.8719*
6	461.3052	321.6286	365.8863
	74.6898	*52.5543*	*76.4414*
7	457.8445	324.0579	358.9947
	81.1468	*51.3617*	*79.3221*
8	457.9037	322.0138	359.4987
	79.7287	*48.0619*	*75.4262*
9	466.3348	326.1008	360.3373
	78.7731	*49.9689*	*69.7964*
10	461.259	320.7537	361.4708
	79.2548	*46.7226*	*74.9443*
11	463.7841	321.9678	359.117
	76.6226	*50.8494*	*77.2573*
12	461.5772	322.0916	361.9966
	80.1016	*49.8423*	*79.7121*

Table 23
Design Point 9
Mean Distance From Assembly Areas
(Standard Deviation Italicized)

	Company		
Run	1	2	3
1	501.7365	290.1972	324.4814
	99.7392	*83.4302*	*100.7729*
2	499.0175	284.8344	321.6855
	94.1012	*80.6012*	*102.6073*
3	500.6175	290.4487	321.9107
	99.8688	*82.3488*	*101.3741*
4	499.6213	286.9653	324.1818
	97.2009	*85.7468*	*97.7782*
5	492.5589	291.6181	318.7303
	101.3205	*88.3113*	*100.0347*
6	498.8291	282.1408	319.6981
	95.0091	*88.7394*	*93.9041*
7	504.2666	282.5252	320.3021
	96.6847	*83.585*	*101.5246*
8	505.8378	290.7543	319.686
	97.7124	*84.4505*	*99.525*
9	500.845	288.8181	328.0465
	98.2773	*85.3541*	*101.6197*
10	496.5483	283.4779	322.9774
	100.1809	*86.3349*	*98.4959*
11	498.5302	286.4457	324.5565
	99.3958	*87.6416*	*98.5074*
12	501.3392	284.8207	322.2554
	95.3014	*86.069*	*106.6055*

Table 24
Summary of Results For
Mean Distance to Assembly Areas

			Company	
Design Point		**1**	**2**	**3**
	Overall Mean	436.228	362.8717	387.8438
1		*3.154*	*2.2571*	*2.6283*
(-,-)	Mean STD	74.0577	34.5662	57.795
		3.1681	*3.1314*	*2.8214*
	Overall Mean	504.9642	287.2827	319.209
2		*1.9993*	*3.4923*	*3.2287*
(-,+)	Mean STD	98.2338	83.8582	96.8437
		2.7011	*1.9889*	*3.1652*
	Overall Mean	428.0379	361.4047	392.7657
3		*2.1276*	*2.4866*	*2.641*
(+,-)	Mean STD	72.9951	30.6491	61.6572
		2.7497	*2.3736*	*2.442*
	Overall Mean	497.1976	285.635	327.4557
4		*2.6989*	*1.5706*	*1.961*
(+,+)	Mean STD	96.2339	84.306	101.5088
		2.3061	*2.0721*	*3.5443*
	Overall Mean	468.5109	319.6805	351.9928
5		*11.0566*	*11.4708*	*11.2221*
(0,0)	Mean STD	77.5729	51.1272	72.7923
		4.8712	*3.5312*	*2.3825*
	Overall Mean	468.2581	323.4957	352.2123
6		*3.5474*	*2.0236*	*2.791*
(-,0)	Mean STD	79.2695	52.4096	73.2678
		2.3322	*1.2823*	*3.5897*
	Overall Mean	433.0555	359.7963	391.7037
7		*2.3006*	*2.6265*	*2.4526*
(0,-)	Mean STD	74.6482	31.3088	58.2386
		4.0372	*2.9275*	*2.014*
	Overall Mean	460.9896	322.2542	360.2295
8		*2.8137*	*2.1514*	*2.5889*
(+,0)	Mean STD	77.7871	50.4006	76.3632
		2.633	*1.8532*	*2.5664*
	Overall Mean	499.979	286.9205	322.376
9		*3.4392*	*3.3919*	*2.6573*
(0,+)	Mean STD	97.8993	85.2177	100.2291
		2.2979	*2.4516*	*3.0755*

APPENDIX G

Dispersion Distribution Comparisons Down Length of DZ

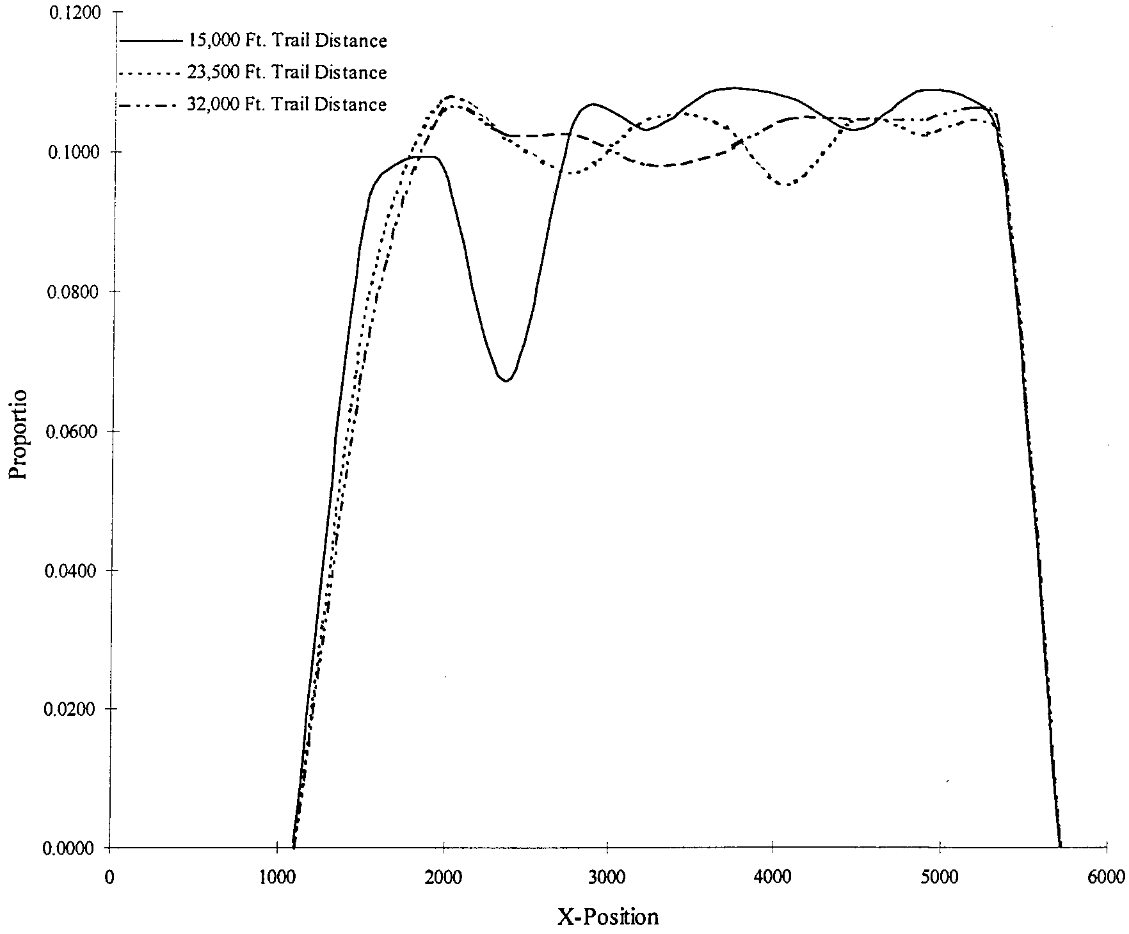

Figure 54. Dispersion Distribution Down DZ With No Lateral Separation

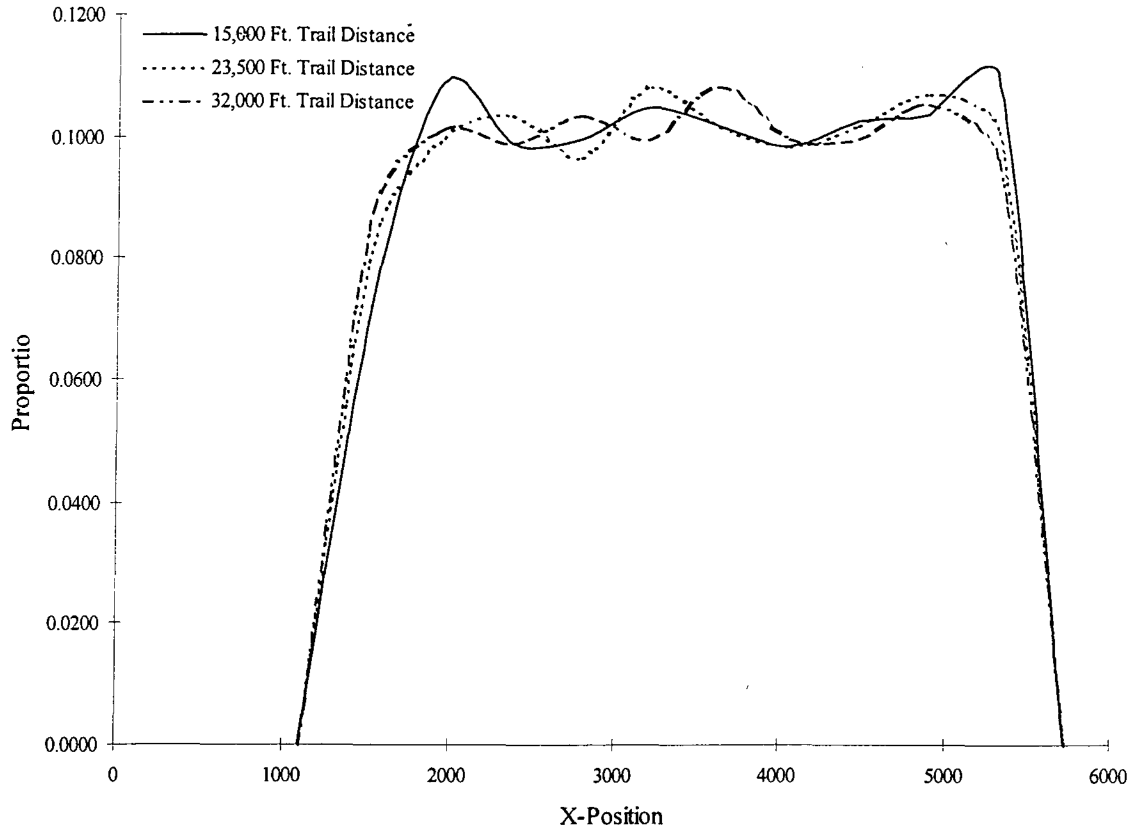

Figure 55. Dispersion Distribution Down DZ With Lateral Separation of 250 Feet

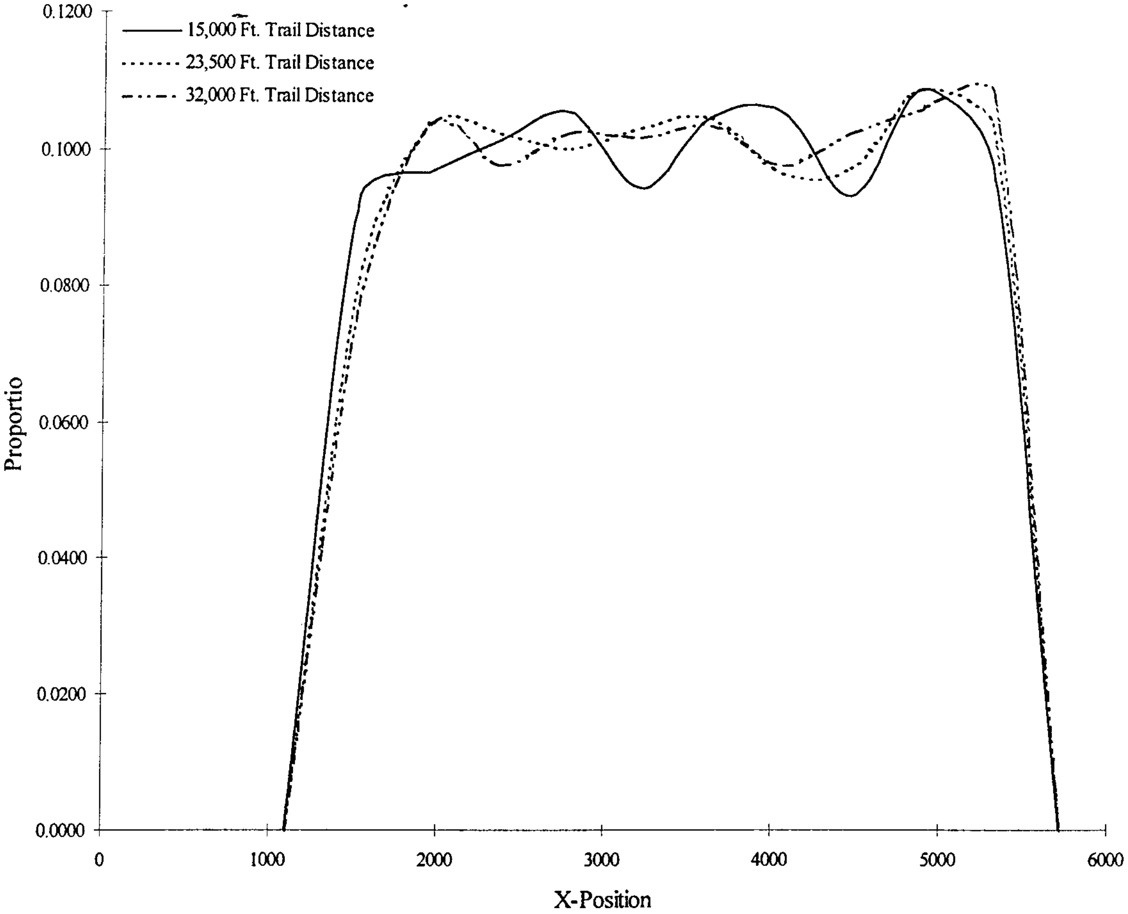

Figure 56. Dispersion Distribution Down DZ With Lateral Separation of 500 Feet

Bibliography

Banks, J., J. S. Carson, and B. L. Nelson. *Discrete-Event System Simulation.* New Jersey: Prentice Hall, 1996.

Benney, Richard. "3-DOF Round Parachute System Oscillation Model." Report to C-17 SPO. U.S. Army Natick Research, Development and Engineering Center, Natick MA. April 1996.

Benney, Richard. Aerospace Engineer, Natick Research Development and Engineering Center, Natick MA. Personal Interview. 25 November 1996.

Benney, Richard. Aerospace Engineer, Natick Research Development and Engineering Center, Natick MA. Personal Correspondence. 21 January 1997.

Blake, William. "Prediction of Paratrooper Vortex Encounters During Formation Airdrop." Report to C-17 SPO. Wright Laboratory, Wright-Patterson AFB, OH. June 1996.

Dassow, Daniel D. Senior Project Engineer, McDonnell Douglas Aerospace Advanced Systems & Technology - Phantom Works, St. Louis MO. Personal Interview. 9 February 1997.

Dassow, Daniel D. Senior Project Engineer, McDonnell Douglas Aerospace Advanced Systems & Technology - Phantom Works, St. Louis MO. Personal Correspondence. 11 February 1997.

Department of the Army. *82nd Airborne Division Airborne Standard Operating Procedure (ASOP), Volume II, Joint Airborne Operations Planning Guide.* Fort Bragg: HQ 82AD, 29 March 1985.

Doherr, Karl-Friedrich. "Nine-Degree-of-Freedom Simulation of Rotating Parachute System," *Journal of Aircraft*, 29: 774-781 (September-October 1992).

Hannon, S. M., J. A. Thomas, S. W. Henderson, and R Huffaker. "Windshear, Turbulence and Wake Vortex Characterization Using Pulsed Solid-State Coherent LIDAR," *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 2464, 94-102 (1995).

Heinrich, H. G., R. S. Noreen, and D. P. Saari. *Development of a Total Trajectory Simulation for Single Recovery Parachute Systems; Volume I: Analytical Model and Computer Design Rationale.* Contract DAAG17-72-C-0030. Natick MA: U.S. Army Natick Research, Development, and Engineering Center, December 1973 (AD-779527).

Huston, R. L. and J. W. Kamman, "On Parachutist Dynamics," *Journal of Biomechanics*, Vol. 14, No. 9, 645-652 (1981).

Johnson, D.J., "Operational Test and Evaluation of the Effects of C-130/C-141B Wake Vortices on the Drop Zone Environment," MAC Project 15-105-86, September 1988.

Johnson, David J., and Captain Jon K. Reynolds, "Operational Test and Evaluation of the Effects of C-5 Wake Vortices on the Drop Zone Environment," MAC Project 15-105-86-1, December 1988.

Jones, D. F., and P. E. Desmier, "Stabilization Parachutes: A Rigid Body Treatment," *American Journal of Physics*, vol. 55, no. 6, 538-544 (1987).

Klimack, William. Ph.D. Student, Air Force Institute of Technology, Wright Patterson AFB, OH. Personal Interview. 31 January 1997.

Knacke, T. W. *Parachute Recovery Systems: Design Manual.* Santa Barbara: Para Publishing, 1992.

Law, Averill M. and W. David Kelton. *Simulation Modeling and Analysis.* New York: McGraw-Hill Inc., 1991.

Martin, Grant and R. Peter Hypher. "Use of Simulation in the Study of Paradrop." *11th Annual Simulation Symposium*. Tampa: 1978.

Maydew, R.C and C.W. Peterson. *Design and Testing of High-Performance Parachutes.* Loughton, Essex: AGARD, 1991.

Natick Mobility Directorate. *T-10C Characteristics, Performance, and Trajectory Information.* Natick Research, Development, and Engineering Center, April 1996.

Natick Mobility Directorate. "Preliminary Proposal for Possible Methods to Determine Acceptable Wake Strength for Personnel Parachutes." Report to C-17 SPO. Natick Research, Development, and Engineering Center, April 1996.

Natick Mobility Directorate. "Preliminary Summary Report of Parachute Vortex Interaction Testing with C-141B and C-17A Aircraft". Report to C-17 SPO. Natick Research, Development, and Engineering Center, June 1996.

Neter, John and others. *Applied Linear Statistical Models.* Chicago: Irwin, 1996.

Noreen, R. A. and D. P. Saari. *Development of a Total Trajectory Simulation for Single Recovery Parachute Systems; Volume II: Calculation Procedures and Computer Program.* Contract DAAG17-72-C-0030. Natick MA: U.S. Army Natick Research, Development, and Engineering Center, December 1973 (AD-779528)

Petry, Hans J. *An Object Oriented Simulation of the C-17 Wingtip Vortices in the Airdrop Environment.* MS Thesis, AFIT/GOA/ENS//97M-13. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1997.

Purvis, J. *Six Degrees of Freedom Flight Simulation Directed Cosine-Euler Axes Method: A Single-Body Parachute Deceleration Program.* CCG-Course F12.01. Muenchen, 1987.

Seaman, Herbert. *Deceleration System Trajectory Equations.* AFFTC-TIM-75-5. Edwards AFB: Air Force Flight Test Center, April 1975.

Strang, Gilbert. *Linear Algebra and Its Application.* Fort Worth: Harcourt Brace Jovanovich College Publishers, 1988.

Strickland, James H. "An Approximate Method For Calculating Aircraft Downwash On Parachute Trajectories," *10th Annual Aerodynamic Decelerator Systems Technology Conference.* Cocoa Beach: AIAA Press, 1989 (AIAA Paper 89-0899).

Strickland, James H and Hiroshi Higuchi. "Parachute Aerodynamics: An Assessment of Prediction Capability," *Journal of Aircraft* 33: 241-252 (March-April 1996).

Tory, C. and R. Ayres, "Computer Model of a Fully Deployable Parachute," *Journal of Aircraft* 14: 675-679 (July 1977).

Wallace, Peter. *Report on Accuracy of Parachute Delivery of the Container Delivery System.* Report to Wright Laboratory. Natick Research, Development, and Engineering Center, 1996.

Watkins, John. Aerospace Engineer, Natick Research Development and Engineering Center, Natick MA. Personal Correspondence. 11 December 1996.

Watkins, John. Aerospace Engineer, Natick Research Development and Engineering Center, Natick MA. Personal Correspondence. 16 December 1996.

VITA

Captain Jose C. Belano III [redacted]. His family immigrated to the United States in 1973, and he was naturalized as a citizen in 1979. He graduated from Andrew P. Hill High School in 1988 and was appointed to the United States Air Force Academy, Colorado Springs, Colorado. He graduated with a Bachelor of Science in Operations Research in May 1992. He received his commission on 27 May 1992 the morning of his graduation from the Air Force Academy.

His first assignment was at Holloman AFB, assigned to the Fourth Space Warning Squadron as a space operations officer. In August 1995, he entered the School of Engineering, Air Force Institute of Technology.

[redacted]