Modeling and Analysis of Software Specifications for an Autonomous Aerial Vehicle

by

Hon Fai Vuong

B.S., Aerospace Engineering
UCLA, 1997

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999
[June 1999]

Author _____
Department of Aeronautics and Astronautics
May 13, 1999

Approved by _____
Mark R. Abramson
Group Leader of Decision Systems Group, Charles Stark Draper Laboratory
Thesis Supervisor

Certified by _____
Eric Feron
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by _____
Jaime Peraire
Department of Aeronautics and Astronautics
Chairman, Graduate Committee

Modeling and Analysis of Software Specifications for an Autonomous Aerial Vehicle

by

Hon Fai Vuong

Submitted to the Department of Aeronautics and Astronautics on
May 13, 1999 in partial fulfillment of the requirements for the
Degree of Master of Science in Aeronautics and Astronautics

Abstract

The present increase in the usage of software for controlling safety-critical systems drives the need for the development of software safety analysis techniques. This thesis describes the use of a state-machine based approach for modeling and analyzing software requirements specifications, for safety related properties, of the supervisory architecture of an autonomous aerial vehicle. The requirements specification model, developed using concepts from controls engineering, is in a tabular format and specifies the intended blackbox behavior of the system in terms of discrete events. The goal of modeling was to produce a set of specifications that clearly describe the desired system behavior for ease of human review. Analysis focused on identifying safety-related errors in design philosophy rather than implementation. This was done because software requirements errors (or high level design errors) that are allowed to propagate into later phases of the development process can result in tremendous costs and are often the most difficult to correct. Results show that the symbiotic nature of the modeling and analysis techniques was useful in improving software safety. Their application led to the identification and elimination of several hazardous conditions not found during traditional software unit and system testing.

Thesis Supervisor: Mark R. Abramson
Title: Group Leader of Decision Systems Group, Charles Stark Draper Laboratory

Thesis Supervisor: Eric Feron
Title: Assistant Professor, MIT Department of Aeronautics and Astronautics

Acknowledgements

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Draper Laboratory IR&D 927.

Hon Fai Vuong

I would first like to thank Mark Abramson for always providing me with the guidance and encouragement needed to get my work done. His quick responses and insightful comments were integral factors in completing this work on schedule. I would also like to thank Professor Eric Feron for introducing me to the area of software requirements analysis. He provided insight for evaluating the requirements model, and provided interesting suggestions for analysis.

In addition, my thanks goes out to Chris Sanders for helping me develop an solid understanding of the autonomous vehicle system, and for putting up with my incessant stream of questions. I would also like to thank Paul Debitetto for reviewing the specifications documentation on such short notice, and for providing excellent suggestions. I also owe my gratitude to Nancy Leveson for teaching me about state-based modeling of software requirements.

Finally, I would like to thank my family. Without their love and support throughout these past two years, I may not have endured. Their kind words of wisdom and understanding have always helped me put life into perspective. I would like to give a special thanks to my friends Emerson Quan and Rex Wu for making life at MIT bearable and, at times, even fun.

Hon Fai Vuong
May 1999

Table of Contents

Chapter 1

Problem Statement

This thesis will determine the practicality of modeling and analysis of software requirements specifications for a small autonomous vehicle system. The autonomous vehicle, shown in Figure 1.1, is a radio controlled (R/C) model helicopter, which will be used for military missions, including urban reconnaissance. Although the system has been in operation for several years, it was never subjected to an official safety analysis. This thesis takes the position that unsafe system behavior is the result of requirements errors or design errors. Thus, the vehicle system was an ideal candidate for ascertaining the usefulness of applying modeling and analysis principles to software requirements specifications to enhance their software safety properties. Although the actual system software has undergone extensive testing and debugging, and has several safety features, including the capability for an experienced safety pilot to manually override the autonomous controller, there is still cause for concern for potential accidents. Ultimately, the system must function without the safety pilot.

The behavior of the vehicle will be analyzed from the perspective of the high-level guidance software, known as the Flight Management System (FMS) along with its graphical user interface (GUI). The reasons for modeling these two subsystems are as follows.

- The FMS is one of the two main behavior-determining software components in the system
- The majority of the FMS software design is based on logic as opposed to numerical calculations. This fits perfectly with the chosen modeling technique.
- Convenience factor was also a consideration, since the FMS designer was willing to help in the study.

- The GUI was modeled because it was considered an extension of the FMS. A discussion of the FMS and its interaction with the other parts of the system will be presented in Chapter 4.

As will be discussed in later sections, requirements errors are difficult to catch during the implementation phases where the code has already been generated. Although a high-fidelity graphical simulation environment was created to run test missions with hardware in the loop before flying the physical vehicle, this was effective only in determining the behavior of the vehicle for specific mission scenarios. Any errors caught at this stage would most likely be implementation related, since the simulation used existing flight code. Even if requirements errors are caught at this stage the cost for correcting them may be tremendous. Even though the model is not fit for catching the errors quickly, the modeling technique used in this thesis is oriented towards identifying requirements errors by helping the system analyst with the systematic analysis of the system's design philosophy.

Figure 1.1 Autonomous Helicopter

Autonomous missions are planned ahead of time by the human operator, and executed with the operator acting as a mission monitor. Therefore, it was extremely important that the vehicle's behavior be completely safe and predictable. Surprises are especially undesirable in the case of military operations, since they can lead to severe

consequences, such as the failure of mission or loss of life/vehicle. Air vehicles pose a unique danger, since they will fall out of the sky if an accident occurs in flight. In addition, helicopter rotor blades can pose a special threat if someone approaches too closely and an unexpected startup occurs.

To the knowledge of the author, no literature has been produced regarding the modeling and analysis of software requirements specifications for autonomous controllers. There has been a growing interest in autonomous vehicles recently, and many research groups have focused on the design and architecture of the autonomy, but it is dangerous to fly these vehicles without the proper safety analysis performed. Many of these designs are related, so it is expected that the results generated in this thesis may be generalized to apply to other autonomous systems.

1.1 Research Objectives/Hypothesis

The focus of this thesis was to model and analyze the software requirements specifications for the supervisory architecture of the autonomous vehicle system for design flaws or requirements errors. Conceivably most of the errors, whether in design or implementation, have been found and corrected over the dozens of mission flown so far. Thus, there is a possibility that no faults exist in the system. The objective of this research was to identify and eliminate the remaining design faults (if any) in the system.

The modeling technique chosen was based on a state machine based method adopted from research done by Leveson [11]. The first task was to produce a state-based model of the requirements specifications of the FMS and the GUI. Then domain experts were asked to evaluate the readability and reviewability of the specifications model. The hypothesis was that the state-based specifications are easier for domain experts to review for design flaws than code, since specifications were produced at the level in between natural language and code.

When the specifications are written at this level, they are closer to natural language than code. Thus, it was hypothesized that they would be easier to read than code. The specifications at this level are executable, which means that they must be detailed enough to be executed like a computer program. Thus, the level of detail of these specifications will give the reviewers a clearer understanding of the system's

behavior, while minimizing the semantic separation between specifications and code. The ease of review was especially designed for improving the overall quality, safety and reliability of the system.

A further discussion of the modeling approach will be presented in Chapter 3. Once this model was completed, hazardous system states were identified, and various search techniques were applied to determine whether these hazardous states were reachable. Hazards identified in the model were simulated using the graphical simulation, and simulation verified hazards were presented to the designer for correction.

Additional objectives of research were to allow for the execution of the specifications in the form an animated state machine, and to improve the presentation of the initial state based specifications with helpful supplementary documentation. This was possible since many commercially available tools to simulate state-based models exist.

1.2 Control Systems Analogy

It is interesting to note the strong similarities in the modeling and analysis of requirements specifications to the modeling and analysis performed in state-space controls. Equations 1.1 and 1.2 form the basic structure of linear control theory [2].

$$\dot{\bar{x}} = A\,\bar{x} + B\,\bar{u} \qquad (1.1)$$

$$\bar{y} = C\,\bar{x} \qquad (1.2)$$

One obvious similarity is that both the control model and the software requirements model use the concept of state to describe the system. The states for the control system are presented in terms of the x vector in Equation 1.1, and they are described in terms of state variables for software requirements. In controls, the states usually take on continuous values, whereas they are discrete in software.

The x-dot vector indicates how the states of the system propagate as a result of the combination of the system's present state and received input. The input is modified by the "B" matrix, which can be compared to the manner in which software processes external input commands to determine its behavior. As will be discussed in Chapter 3, this is exactly how state machines operate.

Finally, an analogy can be drawn between the requirements specifications model and the "A" matrix. In controls, the "A" matrix is a model of the physical behavior of the controlled system in terms of its dynamics, just as the requirements specifications models the behavior of the resulting software system in terms of its discrete dynamics.

The purpose of making such an analogy is to motivate the use of concepts found in state space control to help in the analysis of software requirements. For example, the analysis of a control system for stability is similar to the analysis of requirements for safety.

1.3 Thesis Overview

Chapter 2 of this thesis will provide a motivational background for modeling the requirements specification of the Flight Management System by describing what determines software behavior, and how design errors can occur. Chapter 3 discusses the modeling approach taken to construct the specifications model. It will introduce state-machines as well as the interpretation of the completed model. Chapter 4 will be a presentation of the vehicle system along with a discussion on the relationship between the specifications and the actual vehicle components. Chapter 5 presents the methods of analyses and the results obtained. This will discuss the identification of hazardous states in the system as well as various techniques to search the state-space for these hazards. The chapter will also list the hazards found in the system and the steps taken to eliminate them. Chapter 6 concludes the thesis with some general comments about the usefulness of modeling and analysis of requirements specifications and proposes areas of further research.

Chapter 2

Motivation

2.1 Unpredictable Software Behavior and Accidents in Safety-Critical Systems

There is presently an increased usage of software in safety-critical systems. *Safety* refers to system operation free of catastrophic consequences. *Safety-critical software* is defined as any software that directly or indirectly contributes to the occurrence of a hazardous system state [13]. Nowadays, it is commonplace for humans to entrust their lives to safety-critical software. This is especially true of many modern software controllers, which monitor and control processes that are too complex for humans to manage. For example, software plays an important role in the control of aircraft, medical equipment and automobiles. Software failures in any of these systems can result in the loss of property and life.

Software complexity has evolved at an alarming rate since its first inception. The availability of powerful hardware, and demands for heavier software workloads by users fuels the evolution of software complexity, which shows no signs of slowing. On the one hand, increased software complexity implies increased software power, which in turn will allow humans to achieve more. On the other hand, increased complexity also translates into an increase in the complexity of software behavior. The more complicated the behavior of a software system, the less predictable its behavior will be. Unpredictable software may increase the chance of accidents.

2.2 Examples of Accidents or Unsafe Systems

An *accident* is defined as an undesired or unplanned, but potentially expected event that results in a specified level of loss [13]. There are several aspects to this statement. Accidents are *undesired* events, which imply that they are also *unplanned*, or *unintentional*. The term *accident* is sometimes used to imply that an event is unforeseen or unavoidable, which is untrue, since accidents can be expected but are unplanned. As

an example, car accidents due to drunk driving are unplanned, but they can certainly be expected to occur. This ties in with the research goal of identifying system hazards, which involves enumerating the events that can lead to accidents. Therefore, the definition suggests that accidents can be prevented or at least their consequences can be minimized.

Accidents result in a *specified level of loss*, which implies there must be some damage to life, property or environment. The level of loss to be specified as an accident is subjective. Damage may be immediate in the case of a crash, or it may be long term in the case of the leakage of hazardous material into an environment. However, appropriate safety measures can only be taken after accidents are defined.

Many accidents in the past have been attributed to the failure or improper design of safety-critical software. For example, the Therac-25, which is a computer-controlled radiation therapy machine, massively overdosed six people between 1985 and 1997. These patients suffered from severe radiation burns, and some injuries eventually led to fatalities. These accidents have been described as the worst in the 35-year history of medical accelerators [13]. These accidents were attributed to faulty software, which controlled many of the safety features of the Therac-25 design. Without going into detail, one specific design flaw relates to the capability of editing radiation dosages if they were incorrectly entered the first time. Repositioning the cursor over the incorrect value and typing over it allows the operator to edit the incorrectly entered parameter. Although the correction is reflected on the computer monitor, the Therac-25 software does not acknowledge the correction. Thus, in one session, high energy X-rays were accidentally administered to the patient instead of lower energy electron beams.

Another example of software related failure was the maiden flight of Ariane 5 on June 1996 [14]. About 40 seconds after the initiation of the flight sequence, the launch vehicle veered off its flight path, broke up and exploded. The official investigation showed that during the failure, the primary inertial reference system shut down due to a software exception, resulting in the on-board computer sending a diagnostic pattern to the booster nozzles. The nozzles responded by going into full deflection, thus forcing the vehicle to exceed its maximum angle of attack. The problem was that the inertial reference system used on Ariane 5 was the same one used on Ariane 4, which had totally

different requirements. This leads to the next section, which shall be a discussion of the contribution of requirements errors to software failures.

2.3 Requirements Specifications

Software is defined in terms of a *software requirements specification*, which is a document that embodies the desired behavior of the software. On large or lengthy software projects, a hierarchy of requirements may be produced. For example, the software requirements specification can be split into a *preliminary requirements specification* and a *detailed requirements specification*. The former document sets forth in very broad and qualitative terms the needs of the customer (often in the customer's vernacular). It asserts the goals the software application is to achieve as well as the constraints on the system. In contrast, the latter document sets forth in detailed quantitative terms (perhaps using software-engineering terminology) what the software is to do [1]. It should be noted that the terms *requirements* and *specifications* in this thesis are used interchangeably. However, *requirements* usually refer to higher-level goals and constraints as defined by the customer, and *specifications* are concrete expressions of requirements. Requirements express *what* the customer wants, and specifications express *how* to accomplish it. This process of going from *what* to *how* to actually implementing software is a complicated task, and is the source of many failures. This will be discussed further in the following sections.

2.3.1 Requirements Errors and Software Failures

It has been found that many software faults can be traced back to their requirements as the root cause. For example, Lutz reports that of the 387 significant software errors found during the integration and testing stages for the Voyager and Galileo spacecraft, only three were attributed to implementation errors. A vast majority of the remaining faults were due to requirements errors. Safety-related errors were found to occur because of discrepancies between the documented requirements and what was actually required for the safe functionality of the system, and a misunderstanding of the software interfaces with the remaining parts of the system [16].

It has now been generally recognized that errors introduced early in the lifecycle of software development process are the most difficult to detect and eliminate.

Moreover, correcting such errors during the final stages of development can cost many times more than correcting them at the beginning, since errors introduced this early become deeply entrenched in the system's design as the development process proceeds. A simplified specification for a copy-card dispenser is used to illustrate the problems of finding requirements errors during implementation.

> Suppose a copy-card dispenser is developed to work with a copier, that charges 10 cents a copy. The manufacturer wants to make the card dispenser versatile so users can deposit coins. The manufacturer also wants to guarantee the accuracy of the credit given on the card, so this becomes a requirement. A high-level requirement can be: *"The system shall credit the copy-card correctly in accordance with the amount deposited."* The interpretation of the specification may result in the design of a machine with no change dispenser. No mention of one was given in the specifications, and the assumption was that the system always gives the correct number of credits. Unfortunately, when 75 cents is deposited into the machine, the machine cannot credit the card with a fractional number of copies (7.5 copies), and thus fails to return the card (something similar happened to the author). This is an example of an incorrect set of requirements. If these were caught during implementation or during production, the consequences would be severe. In order to correct the error, redesign would include an addition of a change dispenser, a change holder, and new algorithms to calculate the amount of change to dispense. In the worst case scenario, the error could necessitate the whole system to be redesigned from scratch.

The requirements errors are usually not so egregious as to cause the redesign of the system. However, the difficult task of backtracking through the development process to the point where the fault was first introduced still remains.

2.3.2 Types of Requirements Errors

There are many ways to deal with requirements errors, but first the types of errors that occur in a software system must be understood. In contrast to hardware, where many failures come from fatigue or random errors, software failures are caused by logic or design errors. These errors come in two categories.

- The first type of error occurs due to the incorrect implementation of the specifications. This essentially means the code does not match the specified requirements.

- The second type of error stems from the incorrect requirements. Even though the code has been correctly implemented to match the requirements, the specified behavior is undesirable.

Both types of errors must be considered when attempting to increase software safety. The majority of software tools and techniques developed in software engineering deal exclusively with the first problem. These tools attempt to assure software *reliability*, which is the measure of the continuous delivery of proper service, as defined by the degree to which a system satisfies its functional requirements [1]. In other words, reliable software does exactly what it is told to do.

On the other hand, the software errors that appear in the second problem are embedded in the overall design philosophy. Therefore, any implementation of software, although in accordance with the requirements, will result in undesirable or unsafe behavior. There is sometimes confusion between software *safety* and *reliability*. A common misconception is that a reliable system is also safe. However, safety engineering has learned that highly reliable systems can be very dangerous, whereas it is possible for a system to be unreliable, yet very safe [9].

As an example, suppose there is a computer-controlled system that commands an actuator to turn the knob of a gas stove on and off. The system is highly reliable because it guarantees that gas will be vented whenever the user commands the system to turn on the stove. However, the system is unsafe because it does this regardless of whether the pilot light is on or off. An example of a highly unreliable system that is safe is a faulty light switch. The system is unreliable, because the light turns on only some of the time, but its failure mode is safe since failure merely implies that the light will remain off.

The misunderstanding of safety and reliability has led software engineers to apply hardware reliability methods to assure system safety. A common technique used in assuring reliability in hardware that has been incorrectly applied to software is the use of modular redundancy. It is possible that a physical component will fail either through manufacturing defects, fatigue, improper maintenance, or environmental effects. Therefore, backup systems are used in case of the failure of a primary system. Software engineering has made use of this principle by using what is known as *N-version programming*. N-version programming involves separate teams writing multiple versions

of the same software. Each of these versions is executed simultaneously, and the majority solution is used. Thus if there is a failure in one software component, the others are still available to provide a solution. Although the technique provides for ultra-high software reliability, it is ineffective for coping with design faults, since all versions of the software originate from the same set of requirements [9].

In cases where software satisfied the specifications with high reliability, accidents still occurred due to the following reasons.

- The requirements specified were unsafe, but correctly implemented. This is an *error of commission*, where the system has been designed to behave in an unsafe manner. For the gas stove controller example, an error of commission would be if the specifications state that the system will turn the gas on when the pilot light is out. This is an extremely simplified example, where the danger is obvious, but it is conceivable that a situation may arise in systems that are more complex where the danger is difficult to identify.

- The requirements neglected to specify a behavior necessary for system safety. This is an *error of omission* where the system encounters a situation for which it has no specified behavior. This might occur if the gas stove controller only has a specified behavior when it thinks the pilot light is on, whereas no behavior has been specified if the pilot light is off.

- The software contains behaviors unintended by the requirements. For example, if there is an additional requirement that the gas flow rate is maintained at a certain rate, the unintended behavior may be that the gas blowing out the pilot light.

The three types of requirements errors mentioned above arise as a product of the translation process involved in transforming the customer's original intent into design specifications and ultimately into implemented code.

2.3.3 Intent vs. Specification

One of the main problems with specifications is that they do not accurately express ideal system behavior, because it is difficult to do so. *Specification* is the act of stating explicitly the desired behavior of the system usually in terms of some language. The difficulty of specification comes from the translation of what is *intended* to what is

stated. The intended behavior for a software system is an abstract notion. It is more of an intuition or intangible object, which is difficult to express accurately and precisely. For example, a customer may desire a collision avoidance system for an air vehicle that will be able to avoid any obstacles found in its flight path. This is the customer's intent for the system's behavior. The customer has an abstract mental model of the ideal system, which, in his/her mind, can be designed in such a way such that it will respond appropriately under all conditions.

Unfortunately, software cannot be implemented with intention alone. The intent must be translated several times before it can be realized into code. Software will always behave exactly the way it was programmed, but not always the way it has been intended. For example, a reliable landing gear retraction system is desired by a customer and is built such that the landing gear predictably retracts every time it receives a command to do so. However, in the translation from customer intent, the landing gear may be designed so that it also retracts when commanded even if the aircraft is on the ground. This was clearly not the intention of the designer. This shows that the translation process can be fraught with errors. Figure 2.1 shows the translation process from intent into code.

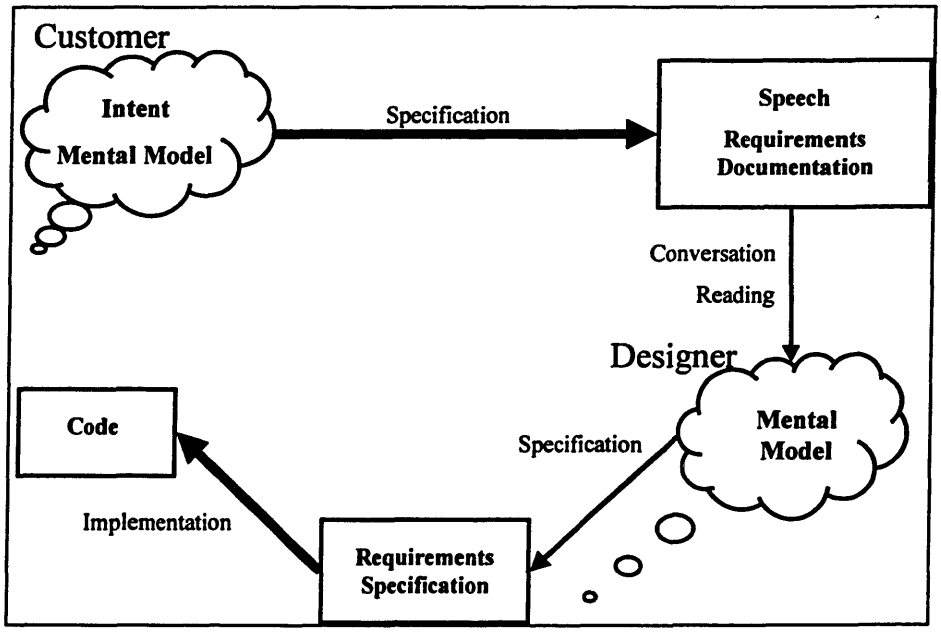


Figure 2.1 Translation Process from Customer Intent to Specifications

Most commonly, the first translation occurs when the customer contracts a software developer and expresses the desire for a software system with a particular set of behaviors. The customer has a mental model of how the software should and should not behave, and this model is translated into requirements through speech or through written documents. The customer tries to specify what is desired of the system in verbal or written terms that best describe the abstract notions of the mental model. Whenever an attempt is made to concretely state a thought, it is possible to make an error in the translation. Although customers know what they want, they may have trouble expressing themselves. The second possible point for the introduction of requirements errors is in the translation process from what is stated by the customer, to what is understood by the contractor. Words often carry different meanings to different people, so it is common for misunderstandings to occur [3].

Once the designer develops a mental model of what he/she thinks is desired by the customer, another translation from thought to written word must occur. The process of explicitly translating the designer's mental model into a requirements specification document also allows for errors. First, the aim of the specifications is to be complete, correct, and consistent (these properties will be discussed in the next section). Nevertheless, these properties are seldom achieved since requirements documents for most software systems of significance are large and difficult to maintain. Keeping all statements in the document clearly in mind is a daunting and often confusing task. For example, the possibility of writing down a requirement, which contradicts another, is common.

The problem is that in many situations, the designer may not even know what the system *should* do. To compound this problem, designers often neglect to specify system behavior under circumstances not conceived in their mental model. The system may perform perfectly under the implicit assumptions made in the mental model, but the intended behavior will diverge from the actual behavior when the system is subjected to real world circumstances.

Although software designers do not intend their software to behave dangerously, the actual behavior of the system can dramatically differ from the intent. There can be a

misinterpretation of the original intention, or the intentions were not compatible with what is found in the real world. For example, there is the story of astronauts who trained underwater to simulate the weightlessness experienced in space. Their task was to replace some screws in a satellite. In the simulation, the astronauts could perform this task satisfactorily. Unfortunately, in practice, it was found that the screws, which were contained in a small pouch, tended to drift out of the pouch in zero gravity. This example illustrates that abstractions can be wrong. Therefore, requirements errors can arise from a poor translation of valid intentions, or they can be caused by intentions being generated from an incorrect physical perspective.

The final translation process lies between requirements specifications and code, which is known as *implementation*. There are problems in translation here as well, although they are not as severe. The process of translating natural language into code takes place with several layers of refinement. Requirements are progressively refined from *what* the system is to do into *how* the system is to achieve the requirements. Only the customer can decide on whether a constraint has been met.

If the translation process is correct, the code can also be translated back to the original set of statements. The success of modeling and analysis depends on the fidelity of the model and the ability of the modeler. The fidelity of the model is gradually improved through an iterative process, which will be mentioned in the following chapter. This iterative process should aid analysts, who may initially have no understanding of the system, to familiarize themselves with it.

2.3.4 Desired Properties of Software Requirement Specifications

The ideal set of requirements should contain the following properties [7].

- Correctness
- Completeness
- Consistency

Correctness implies that the specifications accurately express the desired behavior of the software system. This entails that the specification will have the same interpretation by all who read it, such that it is interpreted in the *right* way.

Completeness is the property whereby the requirements have specified all behaviors of the system under all conditions based on the underlying assumptions about the system and it environment. Take the card dispenser case of Section 2.3.1 as an example. Its requirements were incomplete because no behavior was specified in the case where money for a non-integer number of copies is deposited in the machine. Incompleteness is a contributor to many requirements errors. Most requirements errors are from omission since it is difficult for humans to enumerate all of the possible cases that can occur during system operation. This difficulty is due to the nature of software.

Software operates based on discrete decisions. The input and output of a piece of software entirely depends on the cumulative effect of these decisions. The outcome of a piece of software can be radically changed depending which decisions were made during execution. In continuous systems, there is usually a smooth transition between changes in output. Discontinuity is usually the result of catastrophic breakdown in response to input beyond the operating range, such as a column buckling under extreme loads. It is possible to test a continuous system with only a finite number of tests and extrapolate the overall behavior of the system from the results. However, software cannot be tested in such a manner. Without continuity, tests on a selected sample do not provide confidence that neighboring cases will result in similar software behavior [4]. Enumeration of all test cases is possible if the number of cases is small. However, for larger systems, this may not be possible with a limited amount of resources.

Finally, in order to have *consistency*, none of the specifications in the system requirements can contradict other specifications in the requirements. This is a common occurrence since many requirements specifications are lengthy documents, where specifications for one component of the system can appear in several different places inconsistently. An inconsistency in the copy-card reader specifications might exist if one requirement specifies that change should be returned after the card has been dispensed, and another specifies that the card should be dispensed only after the change has been returned. Usually programming tasks are broken up into design groups. If one group designs the change returning subsystem according to the first requirement, and another group designs the card dispenser according to the second requirement, the software

behavior will be unpredictable. The system can crash and the customer might not obtain the card or any change.

Chapter 3

Modeling

A state-based approach was adopted in the modeling of the FMS. This method is based on work done by Leveson in the area of system safety research [11]. Although the application of these methods did not completely adhere to those developed by Leveson, they were found useful. This chapter begins with an introduction to the concepts around which the state-based model is based, and is followed by an introduction to state machines, which will provide the background necessary to understand the features of the modeling language. The chapter will conclude with discussion of the procedure used to model the Flight Management System.

3.1 Background on Modeling Technique

The modeling language used for the FMS comes from an approach for developing system and software specifications developed by Leveson called *intent specifications.* More information can be found in [8]. Intent specifications are hierarchical, and are broken up into five levels, each describing the system in terms of different sets of attributes or language. Each will be mentioned briefly.

- Level 1 (System Purpose) assists system engineers in their reasoning about system-level properties such as goals, constraints, hazards, priorities, and tradeoffs.
- Level 2 allows engineers to reason about the physical principles and laws upon which the design is based.
- Level 3 enhances reasoning about the logical design of the system as a whole and the interactions between components as well as the functional state without being distracted by implementation issues.

- Levels 4 and 5 provide the necessary information to reason about individual component design and implementation issues.

The level at which the FMS system specifications are written is the third level, also known as the Blackbox Level. In order to complete the process of the construction of intent specifications, the remaining levels or specifications also need to be modeled, since this research only models the blackbox behaviors of the components. A blackbox model of behavior permits statements and observations to be made only in terms of outputs and the inputs that stimulate or trigger those outputs. The model does not include any information about the internal design of the component itself, only its externally visible behavior. The overall system behavior is described by the combined behavior of the components, and the system design is modeled in terms of these component behavior models and the interaction and interfaces between the components [11].

The level of abstraction of the blackbox model is in between high-level requirements and low level implementation. The blackbox model reduces the semantic separation between the intent-level requirements and the implementation-level specifications, but does not actually include implementation details. Leveson states [10] that when low-level design information is included into a specification, it is much more difficult to validate. Just enough information should be included to adequately capture the behavior of the system of interest. For example, TCAS application experts who knew very little about computers and software were able to find faults in the requirements model for TCAS [11].

The blackbox model also makes it possible to mathematically verify the specifications for various desired properties. One such property is the consistency of the control model with the system goals and constraints. Another possibility is the application of formal correctness and robustness criteria to the specification model.

The specifications model is based upon concepts found in controls systems. Figure 3.1 illustrates this point. The controller reads the sensor data and uses this information, along with user inputs to formulate and issue a command to an actuator. The actuator manipulates the process in some way to achieve the overall goals while satisfying constraints on the way those goals can be achieved. In some cases, the controller does not directly issue the control commands. Rather, it provides the human

operator with necessary information, and processes the control commands issued by the human controller, who acts partly as a controller, and partly as an actuator.

All control software (and any controller in general) uses an internal model of the general behavior and current state of the process that it is controlling. This internal model may range from a very simple model including only a few variables to a more complex model. The model may be embedded in the control logic of an automated controller or in the mental model of a human controller, which is used to determine what control actions are needed. The model is updated and kept consistent with the actual system state through feedback. The automated controller also has a model of its interface with the human supervisors. This interface model, which contains the controls and displays, is important because it is the means by which the controller's model of the system and the operator's model are synchronized.

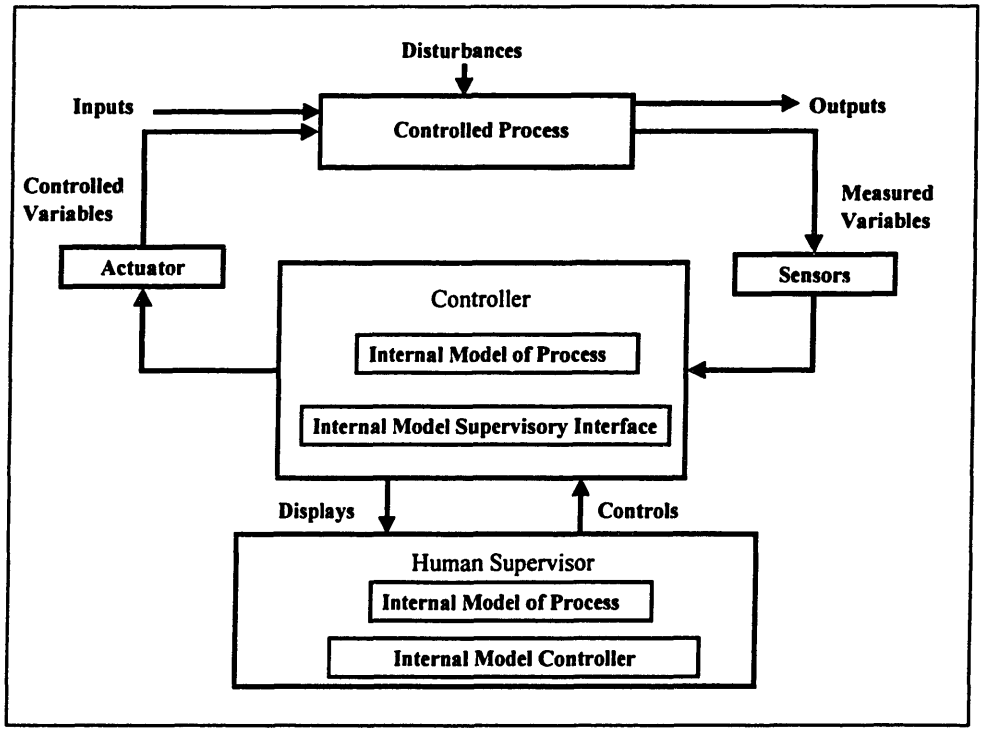


Figure 3.1 Control Loop Analogy of Blackbox Specifications Model

3.2 Background on State Machines

To further understand this state-based approach to defining specifications, an introduction to finite state machines is necessary. *Finite state machines* are structures that can be used to model many types of processes and systems. In general, all finite state machines have the following attributes in common: a finite set of *states*, an *initial state*, an *input alphabet*, and a set of *state transition functions* which define the transition from one state to another.

Several types of finite machines exist, but in the context of this research, the concentration will be on state machines that produce output. The formal definition of a finite-state machine with output will now be given [17].

> A finite-state machine $M = (S, I, O, f, g, s_0)$ consists of a finite set S of states, a finite input alphabet I, a finite output alphabet O, a transition function f that assigns to each state and input pair a new state, an output state, an output function g that assigns to each state and input pair an output, and an initial state s_0.

Before going any further, definitions will be given to clarify some of the terms used above. The definition of a *state* as used in the scope of this research is borrowed from that of controls engineering. The state of a dynamic system (such as a software system) is a set of quantities that describe the system, and can be used to completely determine the evolution of the system. The main difficulty with this definition is that the set of quantities used to define the system is not unique [2]. In many cases, the choice of variables used to represent the system is obvious, but in others, the choice in variables is by no means obvious.

The *input/output alphabet* is the set of well-defined inputs/outputs recognizable/transmittable by the state machine. The *transition function* is responsible for assigning one state to evolve into another once certain conditions have been satisfied. Most often these conditions consist of state and input pairs. The *output function* is responsible for sending output from the state machine if the conditions for sending it have been satisfied. Finally, an *initial state* is always necessary for the state machine to distinguish the state in which it commences execution.

3.2.1 Finite-State Machine Representations

State machines can be presented in the form of *state diagrams*, which are graphical representations of state machines using the notation of circles and arrows. Figure 3.2 depicts a sample state diagram of a water level monitoring system. The circles indicate the three states of the state machine. The single arrow indicates the initial state, and the connected arrows indicate the direction of the state transition. The labels on the arrows are the transition and output functions.

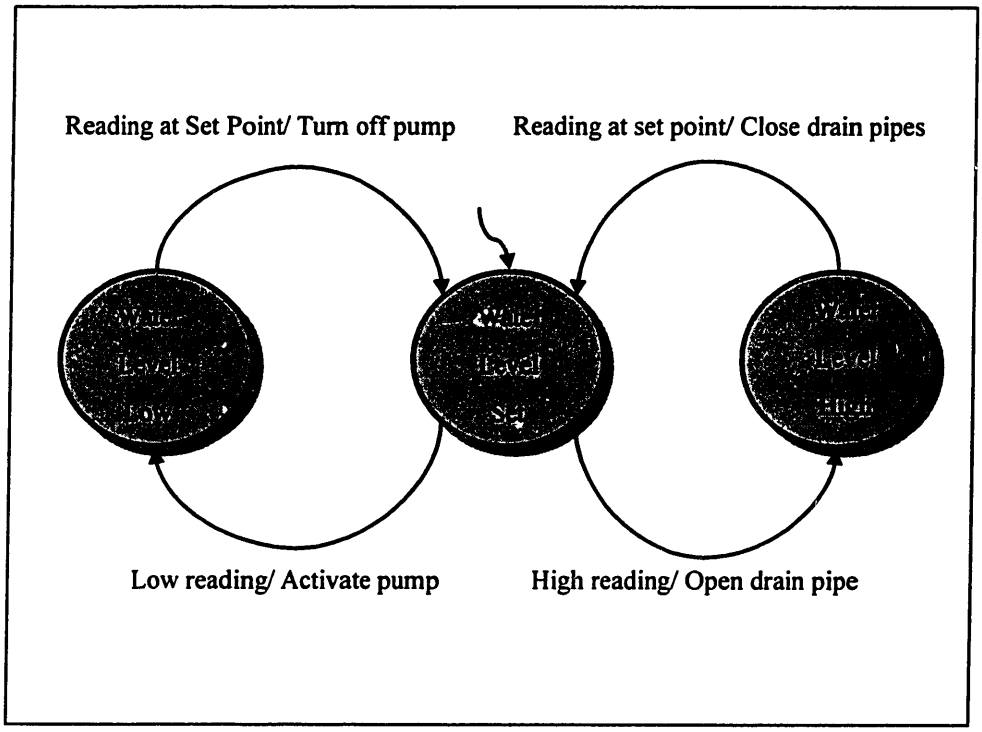


Figure 3.2 State Diagram for a Water Level Monitoring System

To take an example, the state will transition to WATER LEVEL LOW if the state machine is currently in WATER LEVEL SET and it receives a "Low Reading" input. As it makes this transition, the "Activate Pump" output action would be generated.

State diagrams are well suited for the structured modeling of the dynamics of a software system. This simple state diagram can easily be translated into code. It is just a matter of defining a few variables, initializing them and coding the logic using *if-then* statements.

Another way in which state machines have been represented is through the use of *state transition tables*. The specifications of the requirements for the Flight Management System have been generated in the form of state transition tables. More will be said about how state machines are represented using tabular format in Chapter 4.

3.2.2 Additional Notes on State-Machines

Unfortunately, as software systems become increasingly complicated the number of states and the number of transitions increase so quickly that state machines, as they have been presented thus far, start to lose their effectiveness. This is because the guarding conditions for state transitions become increasingly complicated, and they cannot all be shown on the state transition diagram.

> One way to avoid this problem is to use models that abstract away from all the states to a smaller number of higher-level states, from which the entire state machine can be generated. The complete "state space" [the entire set of possible states of a system] may never be generated (and it may be infeasible to do so), but many properties of the state space can be inferred from the higher-level model [13].

In the analysis of the Traffic Alert and Collision Avoidance System (TCAS) II, it was possible to reduce the state space from an estimated 10^{60} states to just a few hundred [11]. Collapsing the state space does not translate into neglecting essential states just for the sake of reducing the state space. It does mean that the state space should be minimized in such a way to adequately capture the behaviors of interest for analysis. Such an abstraction can be made, and essentially the main problem is the choice in state variables. It is not necessary or even possible, to completely describe the system such that it becomes entirely deterministic. This statement seems to contradict the definition of *state* as mentioned earlier. However, there should be an additional clause in the definition to address the problem of scope. The choice in state variable is very much a modeling problem. Basic engineering principles can be used to select the set of states to represent a system, or at least eliminate extraneous states in the model.

For example, in the Flight Management System, there was no reason for modeling the estimate of the x coordinate of the vehicle's position in terms of the actual field coordinates. This allowed the reduction from what could have been an infinite number of states (x = 0 ft, x = 20ft, x = 20.5 ft, x = 20.557893638 ft...), to only two states (x = in

position, x =out of position). The scope of the modeling encompassed only the behavior that the Flight Management System could affect. In terms of position, the FMS would only react under two conditions. Either the vehicle is in position, or the vehicle is out of position. If the modeling of the control algorithms of the vehicle were the focus, some other representation of the position may apply. The goal is to pick states appropriately and effectively for the particular application in mind.

Thus far, in the discussion of state machines, all states have been defined at the same level of abstraction. The FMS model has made use of the concept of multiple levels of states as well as the idea of orthogonal states.

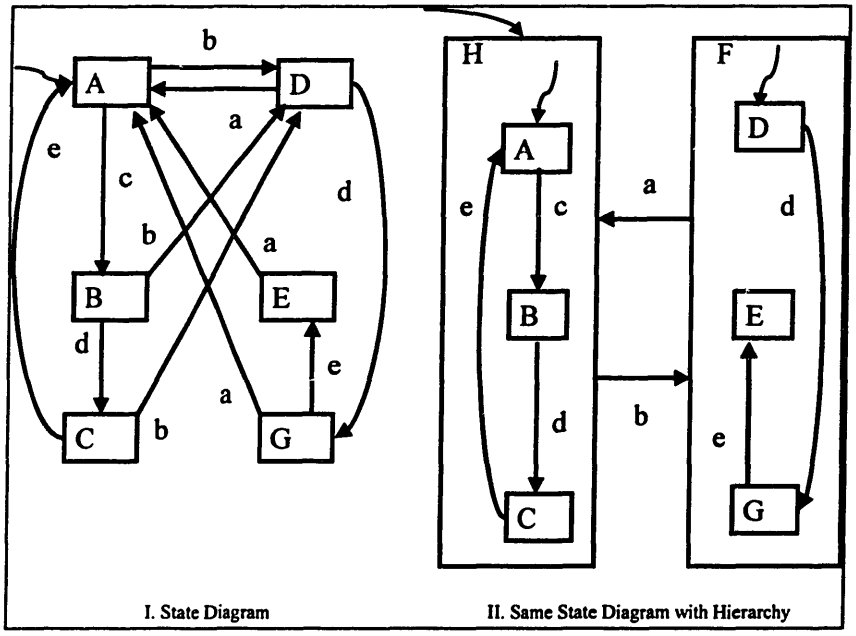


Figure 3.3 Hierarchical State Machine

Multiple levels of abstraction allow single states to contain other states. The higher level states are described as mutually exclusive behaviors of the system, and are referred to as *superstates* or *modes*. The lower level states are known as the *substates* of the superstate. These are relative terms, so it is possible for a substate at one level of abstraction to be a superstate for another level. In effect, each superstate contains a miniature state-machine within itself. The use of a hierarchical structure allows for a more concise way to express state transitions [11]. Figure 3.3 shows the advantages of specifying state transitions using hierarchy. The two diagrams in Figure 3.3 are

equivalent, but three separate arrows are necessary to specify the results of event b in the original state diagram while only one arrow is required with the use of a hierarchy.

Orthogonality of states allows the specification of concurrency, also known as AND-decomposition. Adopting such a convention allows for the partitioning of the state space into distinct pieces, which are independent of one another. It should be noted that orthogonal state decomposition is similar to that found in classical control systems decomposition, where the different independent degrees of freedom for a dynamic system are separated into *independent coordinates*. The advantages of orthogonality are that the state transitions are easier to read, and a lot of time is saved from not having to repeatedly write down a recurring state. Figure 3.4 shows two versions of the same state machine. This illustrates how much easier it is to understand a state transition diagram with orthogonality as opposed to one without it.

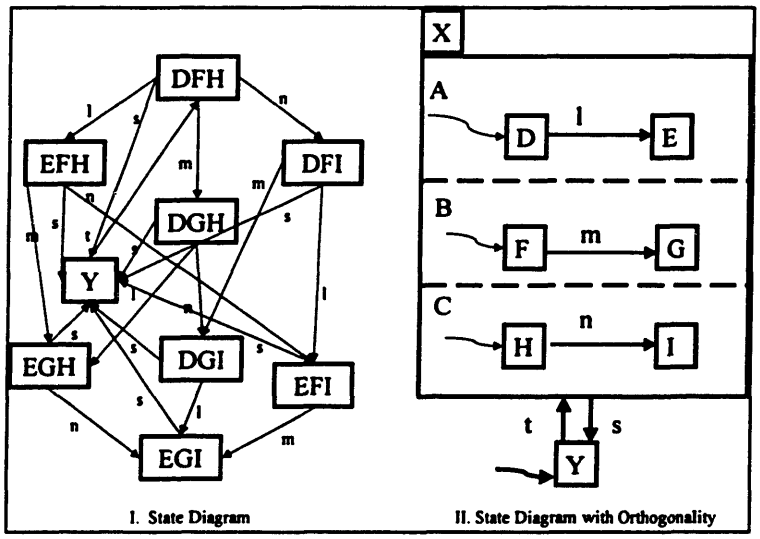


Figure 3.4 State Machine with Orthogonality

Part II of the figure shows that state X contains the orthogonal substates: A, B, and C. When the system is in state X, it is in states A, B **and** C at the same time. This is in contrast to the system either being in the mutually exclusive states D **or** E while in state A. The advantage is that only one arrow is needed to document the transition from X to Y as opposed to the eight arrows in Part I of the figure [5].

3.3 Introduction to Reading State-Based Requirements Specifications

This section gives an introduction on how to read the state-based specifications generated for the FMS. The specifications are broken up into two main components. The first component of the specifications is a graphical depiction of the state-machine architecture of the software specifications model, and the second component is the state transition tables.

3.3.1 State-Machine Diagrams

In general, multiple state machines of the system components would be strung together to form a model of the entire system. For this investigation, the model only contains the FMS and the GUI. The general features of these state diagrams will be presented using the FMS model in Appendix B.1.

The diagrams should capture all of the relevant input/output/controller-modes/states of the vehicle in the software system. Each of these will be shown on separate pages. The diagrams also explicitly indicate the possible values of the state variables and output messages.

Inputs into the state machine are divided into those that go through the GUI and those that go directly to the FMS. Each of the interface diagrams indicates the supervisory mode of the system. *Supervisory Mode* shows who (or what) is in control of the system at any time. For the Operational GUI interface diagram, the supervisor is the GCU Operator. The input space of the operator recognized by the Operational GUI is shown as a cascade of raised rectangular boxes, which contain the input commands. The approach was to make the graphical depiction of the input interface as intuitive as possible for reviewers to read. The layout of the menu items in the actual system is not required to look like the one shown in the diagram, but only to have the same menu structure.

An arrow located beneath a box indicates that a submenu will be presented as a result of depressing a button. In the case of the *File Menu* button, depressing the button will bring up a submenu containing the options: *Delete Waypoints/Show Engineering GUI/Quit*.

Flat rectangular boxes represent dialog boxes, which appear with additional input options. For example in the case of *Create a New Waypoint*, a dialog box appears, which contains additional checkbox options, represented by the circles. Textbox entry options are represented by dotted rectangular boxes. In the same example, the operator can choose either Waypoint Hover/Through/Land as the flight mode of the waypoint being created. The option of entering the waypoint's coordinates by clicking on the map, or entering them manually in MGRS coordinates must also be chosen. The altitude of the waypoint must be entered manually into the textbox. When all the information has been entered, a button to indicate that the entry is complete can be depressed to send the information to the FMS.

The Engineering GUI input interface is slightly different from the Operational GUI layout. The Engineering GUI is divided into mutually exclusive pages. There are four pages: GPS, Navigation, Waypoint, and Control. The raised boxes still represent buttons, and the dotted boxes still represent input parameters.

One additional feature not seen in the Operational GUI is the ability to input a list of waypoints. This list is shown under the *Change to Task List Mode* button. This is a way to represent the task list, which is an enumeration of the waypoints constituting the mission. The arrow indicates the current waypoint. The tasks listed below the current task are waypoints not yet executed. Waypoints can only be entered at the end of the task list. Waypoints are created using same parameters found in the Operational GUI, except for the addition of the parameter known as hover time, which is an indication of the amount of time the vehicle should hover at a waypoint. Hover time for hover waypoints was assumed to be preset in the Operational GUI.

Inputs that go directly to the FMS via the vehicle are shown in the lists on page 123. The list includes the name of the input message, as well as input parameters found in the message. For example, the *Raw Data* message contains information on the raw and actual compass data, and raw and actual sonar data.

Interface outputs of the state-machine are represented as boxes containing a list of output values as shown on pages 120, 121 and 123. For example, the output value for Communications Status from the Operational GUI, which would be sent to a computer screen, can either be UP or DOWN. Highlighted output values indicate the default output

value at start-up, so the Communications Status Panel would initially show communications to be DOWN. Continuous output values are indicated by empty boxes with the units of the output when appropriate. An additional feature in the output display of the Operational GUI is the map display panel. This contains a wealth of information about the vehicle's state, and mission status. The triangular icon represents the vehicle, and its position on the map display indicates the location of the vehicle on the field. The orientation of the vehicle is indicated by the pointing direction of the vehicle icon. The mission is represented on the map in terms of interconnected waypoints, where the current waypoint is highlighted. The Engineering GUI has a much less sophisticated map, which provide equivalent information.

The outputs of FMS to the vehicle are listed on page 124. The FMS model was limited to these outputs, which includes a variety of data request messages and guidance commands. However, a more detailed account of the FMS outputs will be given in the state transition tables.

The state-machine diagram also contains a vehicle model, which represents the FMS's view of the vehicle, as discussed in Section 2.1. The large boxes represent the name of the state variable. The possible discrete values taken by each state variable is listed in these boxes. The highlighted values represent the default value at start-up, and the connecting lines imply possible state transitions, which can occur when the appropriate triggering conditions are met. For example, the state variable known as Estimated Mode Confirmation Status either be in the state of ACKNOWLEDGED, or UNACKNOWLEDGED, where the latter value is indicated as the initial estimation of the state. The connecting line means that Estimated Mode Confirmation Status can transition from the state of UNACKNOWLEDGED to the state of ACKNOWLEDGED if the appropriate triggering events occur.

Finally, the *functional modes* of the FMS are also included in the diagram. These modes, as mentioned in Section 3.2.2, are treated similar to states, except they represent mutually exclusive behaviors of the FMS itself. There are several mode variables, which specify the higher level behavior of the FMS. Mutually exclusive modes are listed for each mode variable, but each of the mode variables are orthogonal as discussed previously. For example, FMS Guidance Mode can be in SINGLE MODE, which

prevents it from also being in TASK LIST MODE or RETURN HOME MODE. At the same time, the Guidance Initialization Status can have the value of READY.

It should also be mentioned that a state machine model of the task list was also added to the FMS specifications model to capture the discrete dynamics of the task list. One new feature is introduced in the task list model. Shadowed pages are used to indicate an array of tasks, which represent each of the tasks on the task list. The top task is Task [i], and the next is Task [i + 1]. Each individual task item contains information about the parameters associated with the task.

3.3.2 Modeling Language

Action phrases are part of the state-based modeling language, and are used to dictate the basic dynamics for state-machine model. Action phrases are indicated in bold throughout the specifications and used to specify the action or condition the state-machine must take or satisfy. These phrases were adapted for the FMS and based on work done by Leveson. The phrases used in these specifications are discussed below.

- **Receive...from..., Message Contents:** The first argument of **Receive...** indicates the name of the input message received. Possible messages are those listed in the state diagrams. For the purposes of research, this only accounts for messages received by the FMS via the GUI or directly from the vehicle. The second argument indicates the source of the message. For the system being modeled, this would be the vehicle or the GCU Operator. The third argument, for Message Contents, indicates the parameters and their values contained in the message. This is used to indicate that an input event has occurred, and is used as a triggering condition for initiating state transitions. It should also be noted that input/output messages in the specifications are indicated by quotations, and the "=>" symbol indicates a substate or additional information.

- **...in state...** This is used most often as a guarding condition for an event, such as a state transition or the issuing of an output message. The first argument indicates name of the state variable, and the second holds the state value. This phrase is used to express a required condition for the state variable. This statement can be evaluated to be true or false. An event occurs depending on the result of this evaluation. The value of a state variable is indicated by all capital letters.

- **Set...from state...to state...** This phrase is used to specify which of the state transitions is to occur if the triggering conditions are met. The first argument

is the name of the state variable. The second and third arguments are the current and destination states of that state variable.

- **Set...equal to...** This statement is used to change the estimated value of continuous input variables, where the first argument is the variable name, and the second is the new value of the variable.

- **Send...to...** This is used to indicate FMS commands to the vehicle. The first argument is the message name and accompanying message contents. The second argument indicates where to send the command. The model only includes FMS commands to the vehicle.

- **Timeout** is a function used to indicate that an expected event has not occurred within a specified time interval. The statement evaluates as true if a specified amount of time has elapsed, and the no event is detected. No times were used in the specification. Rather, the concept of a timeout was used to indicate that there should be a requirement on the timing constraint of events. The use of a timeout in the Guidance Message section will evaluate to be true if a *Mode Confirm Message* is not received within a certain time interval.

- **Prev** is used to indicate that the previous value, as opposed to the current value, of the state variable should be used in the evaluation of a triggering condition.

3.3.3 State Transition Tables

The modeling language chosen for the FMS uses a tabular notation to capture the events and state transition properties of the system. These charts are a mathematical description of the software dynamics, which are the same as the equations of motion for the physical system. Writing the software specifications in this form is equivalent to specifying the behavior of the system. These tables determine *when* state transitions occur, *how* the discrete states propagate, and *when* output messages should be sent.

State transition tables, as used to model the FMS, are decomposed into AND/OR tables, which is exactly how transitions were captured in the state transition diagram, but in a graphical form. However, due to the large number of transitions and the complicated triggering conditions for the FMS model, drawing the state transition diagram would be futile. Thus, the model was broken into the two parts: graphical, and tabular.

The table on page 127 will be used as an example to explain how to interpret the features of tabular specifications used for the FMS, and is repeated in Figure 3.5. The

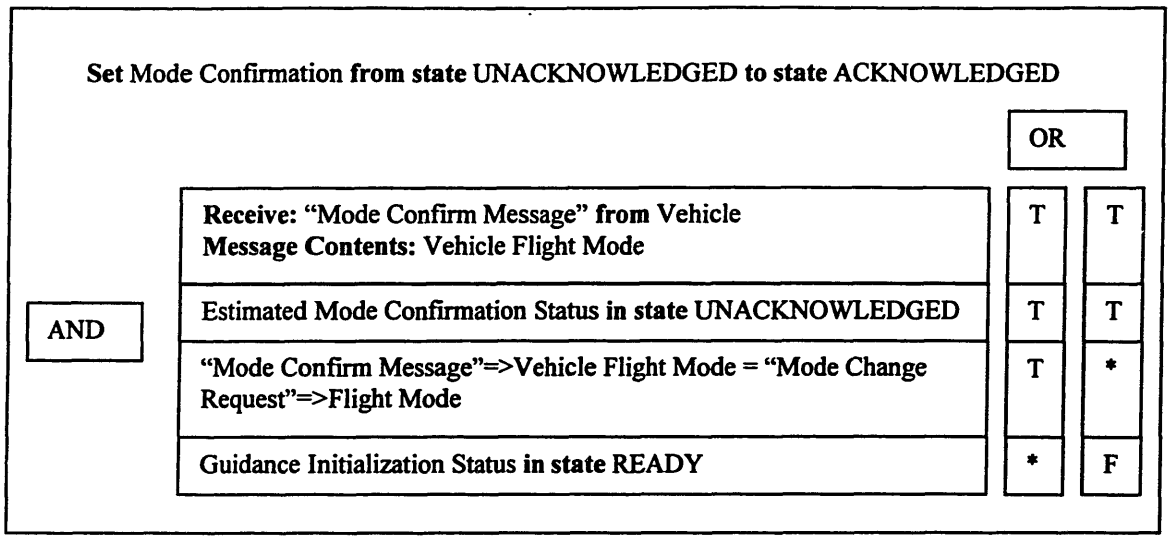


Figure 3.5 Tabular Specifications

state transition is located at the top of each table. In this case, the transition will involve the state variable Mode Confirmation. The transition specifies that Mode Confirmation will switch from the state UNACKNOWLEDGED to the new state ACKNOWLEDGED if the conditions listed below are satisfied. It should be mentioned that the specifications are continually being modified or corrected, therefore, some of the items mentioned here may not be completely coordinated with the specifications model in the appendices. For example, the state value UNACKNOWLEDGED was presented as NO ACKNOWLEDGE or NOT ACKNOWLEDGED. These are equivalent state values, which need to be corrected in the specifications for consistency. The OR above the two columns to the right indicates that the transition will occur if either one or the other column is completely satisfied. The columns are specified such that they are mutually exclusive events, so that both columns cannot be satisfied at the same time. Satisfaction of an entire column can be accomplished when the truth-values of the statements in each row match the value indicated in the corresponding row of a particular column. T/F/* correspond to True/False/Don't Care. The AND indicates that all the rows must be matched correctly.

An example will help to clarify this. Mode Confirmation transitions from UNACKNOWLEDGED to ACKNOWLEDGED if and only if **Column 1** (the conjunction of statements 1- 3) is true:

1. The statement: The FMS receives a *Mode Confirmation Message* evaluates to TRUE.
2. The statement: The state variable, Estimated Mode Confirmation Status, is currently in the state of UNACKNOWLEDGED evaluates to TRUE.
3. The statement: The Vehicle Flight Mode returned in *Mode Confirm Message* is the same as the requested flight mode is TRUE

or **Column 2** (the conjunction of statements 4-6) is true:

4. The statement: The FMS receives a *Mode Confirmation Message* evaluates to TRUE.
5. The statement: The Estimated Mode Confirmation Status is currently in an UNACKNOWLEDGED state evaluates to TRUE.
6. The statement: The Guidance Initialization Status is Ready evaluates to FALSE.

If neither of the two columns is true, then the transition does not occur. Notice in the second case, the FMS does not care if the vehicle flight mode returned was the same as the one that was requested. This is indicated by *. When Guidance Initialization Status (GIS) is in NOT READY, the vehicle is unable to send a *Mode Change Request.* Therefore, the second condition is unrelated, since it cannot be checked. When the GIS is in NOT READY, Requested Mode has no value against which to check the Vehicle Flight Mode.

3.3.4 Chart Characteristics

A chart refers to the complete set of tables associated with an output variable or state variable. There are three types of charts in the specification model. One chart type contains the output specifications, another contains specifications of state transitions, and the third contains functions, which will be described later. The chart type is listed at the top of each chart as a state variable, output variable, or function. For a special case in the FMS model, outputs and state transitions were bundled into one chart to reduce clutter.

A typical output specification for the FMS is shown on page 153 of Appendix B.2. Every output specification from the FMS has a *message destination*, which indicates where the message is to be sent. In this case, it is the vehicle. Output specifications also include a *comments* field to help reviewers understand the context in which the output

message is sent. In this case, there is a note that states that this specification should not be applied when the FMS enters Task List Mode from another Guidance Mode. The next feature of an output chart is the *triggering condition.* The triggering conditions are necessary conditions for an output message to be sent. The set of triggering conditions differs with each output message. Additional conditions, found in the *definition* section, determine the specifics of the message contents of the output. For example, in order to consider sending *Guidance Message*, the FMS must be in Tasklist Mode, and the Mode Confirmation Status must be in ACKNOWLEDGED. The *definition* of the output message determines the details of the output message. The name of the message to be sent, and the accompanying parameters are found directly above each individual condition set. Each set of conditions is mutually exclusive as well. Originally, all conditions for each output message variation were found on the same chart, but that quickly made it too large to fit on one page. Therefore, each of these was broken up into smaller tables, according to message content. This improved the readability of the specifications. In the example chosen, the first table specifies that the FMS should send a *Mode Request Message* containing Ground Mode as the commanded mode if the current mode is in Waypoint Land Mode, and Waypoint Land has been completed. In the second table, the command is Runup Mode, but the conditions have changed to the vehicle currently being in Ground Mode, and completing it.

Charts for specifying state transitions are similar to output charts. A typical chart state transition chart (in this case for the state variable named Guidance Mode), is illustrated in Appendix B.2, from page 138 to 141. The first section indicates restrictions on the possible values the state variable can take. In this case, the Estimated Flight Mode can take on the values of all of the flight modes. The next feature is a section that allows for comments. This is a useful section to give reviewers the necessary background to understand the specifications. This feature was also used to help negate some of the difficulties encountered with timing specifications. This was used to explicitly state which transitions were to be executed first in cases where this was ambiguous. Next, the *definition* section defines the conditions for making a state transition. The initial state of each state variable is indicated to satisfy the condition that the initial state of a state machine be deterministic. The initial value of Estimated Flight Mode is Ground Mode.

Finally, different state transitions are specified by different tables. As with the output tables, different state transitions of a particular state variable were decomposed into smaller, more readable tables. Additional notes indicate that the first table is applicable only during normal operation, and the second is applicable only at startup. The first transition is a way to generalize the specifications so that each individual flight mode would not have to be written out. The specification states that the Estimated Flight Mode should be set equal to the most recently Requested Flight Mode only if the conditions on in the first table are satisfied. The conditions in the second table, which are mutually exclusive from the first, indicate that the current value of Estimated Flight Mode should transition into the current state value of Vehicle Flight Mode instead.

The last chart type is the function. There are two types on functions. One is used to reduce clutter, by grouping commonly used conditions into a single statement, and the other is used to perform numerical calculations for FMS decision making. Functions located throughout the specifications will be indicated with the subscript F. The input arguments of the function are located within parentheses. The chart for a function has several parts to it. Every function has an *input argument list*, which indicates the values upon which the FMS should base its actions. There is also a section for comments. The *definition* section indicates the conditions or the calculations the function is to perform. The output of a function are either the Boolean values of true or false, as shown in Appendix B.2, page 133, or numerical values. This allows them to be placed directly into tables as guarding conditions for state transitions or sending output. For example, Waypoint Achieved_F is used to perform calculations necessary to determine whether the current waypoint has been completed. If all the conditions are met, the return value will be true. Otherwise, it is false. Distance to Waypoint_F is an example of a function that performs calculations.

3.4 Modeling Background

The objectives of modeling were to develop a model of the software requirements that would accurately reflect the intended behavior of the FMS, and use the model to analyze the requirements for design faults in the system. One of the primary reasons the FMS was chosen was that the designer of the system was willing to help in the modeling

process, and it seemed to be the best match for the modeling language that was selected. The FMS was the best candidate because its behaviors were related to logical decision making as opposed to numerical calculations.

Since only part of the system was modeled, assumptions had to be made about the behavior of the other systems interacting with the FMS. Since the FMS was modeled, it meant that the behaviors of the human operator and the vehicle could be assumed. In addition, when making these assumptions, the behaviors of interest were limited only to those that directly affected the decision processes of the FMS. To illustrate this point, the FMS does not care about the dynamical process of how the vehicle flies to each waypoint. However, information on whether a vehicle has reached a waypoint is very important to the decision processes of the FMS.

Specific behaviors of the GUI were included in the model because it is closely related with the FMS. Although there is very little logic in the GUI, there are some examples of the GUI refusing commands from the user because certain conditions were not satisfied. An example of this is that the operator is not allowed to issue an *Execute Mission* command until the GUI has successfully initialized the navigation filter. The *Execute Mission* menu item does not appear as a menu item until this condition has been satisfied. Normally, all the commands entered by the human operator pass through the GUI and go directly to the FMS. All information essential for the human operator is assumed to be sent directly from the vehicle through the FMS to the GUI output interface. The graphical information presented in the GUI was not specifically modeled, and was not the emphasis of this research. The GUI model exists primarily to illustrate the possible command signals that can be sent to the FMS by the human operator. The FMS clearly bases it decision making on these commands. A model of the displays would need to be developed if the focus of research were on the behavior of the human operator in reaction to the information presented in the displays.

A simple assumption about the vehicle's nominal behavior was used throughout the modeling process. The main assumptions are that the vehicle will respond to every request it receives from the FMS, and will execute every guidance command it receives from the FMS immediately. The vehicle is also assumed to send back a status message at

a set time interval. Malfunctions of the on-board system, which prevent the vehicle from executing a command or replying to requests, were not considered.

3.5 Modeling Approach

The approach to modeling was iterative in nature. Trial and error was a common method used in constructing the model. The FMS designer evaluated each attempt that was made to capture the behavior of the system. If errors in the specifications were found, they were corrected every iteration.

The first step taken in the modeling process was gathering information. There were several source of information and all of them were used. These included the code, the system designers, and various forms of documentation (unfortunately, an official requirements document was not available).

The very first piece of information examined was the code. It is possible to obtain a good understanding of the system by examining the code. Unfortunately, it takes an inordinate amount of time to develop a good understanding for the global behavior of the system. In addition, the code can provide incorrect or insufficient information for reaching the goals of this research. Errors introduced during implementation have been introduce into the code, so readers trying to determine intent just from code, could misinterpret the intent and be led astray. To put this into perspective, instead of finding flaws in the design, examining code will lead to the discovery of poor software implementation skills.

This observation should not discourage the examination of the code. Comments found in the code can be helpful in deciding why a particular form of implementation was chosen and it may give a hint about intent. Code was especially helpful, once the intended behavior of the system was well understood, and was used in the final stages of modeling. After working with the specifications for some time, the behavior of the vehicle was considered mostly in terms of code, which could then be compared to the mental model of the desired vehicle behavior. However, code is a poor alternative at the start of modeling, since it is difficult for humans to think about systems specifically in terms of variables and *if-then* statements. It takes tremendous effort in order to trace the flow of information through code. As a simple example, twenty *if-then-else* statements

yield over half a million execution paths which at one minute to test per path will take one year to complete [20]. Time can be better spent speaking with the designers of the system to learn, at least in informal terms, the intended behavior of the system. Getting up to speed on any project involves speaking to those with experience, and software requirements generation no different.

The best resource in generating requirements is the designers of the system. Theoretically, for the start of any project, a set of specifications has usually been written before any coding even begins. This was partially true in the case the FMS. Although no official requirements specification was documented, the specification did reside in the mind of the designer. The problem therefore was to elicit this mental model of the specifications from the designer. Designers have a solid understanding of the system, and this was especially true in the case of the FMS designer. Although the code was referenced, interviewing played a major role in the development of the specifications as well as the verification of the completed specifications. Initially the designer was the only one with an intuitive feel for the system, but eventually, after several conversations, the author was also able to develop this same intuition. The need for appropriate requirements elicitation is documented in [3]. Several approaches are used in eliciting requirements. Some the methods mentioned in the reference were introspection, interviewing, conversation, interaction, and questionnaires. Goguen [3] argues that conversation and informal social interaction is a more detailed and precise choice. Although not very structured, conversations with the designer were found the most useful in the development of the requirements for the FMS.

Experience from these conversations showed that the designer often presented behaviors of the system in terms of coded statements and conditions. This was the designers way of understanding the behavior of the system. Questions directed towards the desired behavior of the system were often answered by references to the code. If the answer did not reside in the code, the desired behavior was not immediately obvious.

The focus of the designer was on the physical realization of the system, whereas the focus of this thesis is on errors in intended behavior. Although the two objectives were slightly different, the designer provided a good anchor to keep the physically

realized system in the mind of the analyst. This prevented the analyst from analyzing a nonexistent design, which would be of no use to the project.

Finally, various other forms of documentation were read, which provided good background material on the vehicle, but did not help in some of the details in modeling. These included papers, memos and technical reports. The main reference was [18].

As the information was being collected, it was actively being incorporated into the mental model developing in the author's mind. Development of the mental model was aided by several methods: walk though, simulation, flight-testing.

The *walk-through* was a mental simulation technique commonly practiced by the analyst to learn new behaviors of the vehicle, and how they affect the behaviors already incorporated into the existing mental model. The walk-through consists of running missions mentally based on the most recent knowledge of the vehicle's behavior. This forced the analyst to consider the system and its operation. Walk-throughs were performed mentally by the analyst who took the FMS's view of the vehicle's behavior. Missions would be systematically executed, while the behavior of the vehicle would be determined as the mission was performed. The *walk-through* was especially helpful in quickly incorporating new states and determining new state transitions, as well as in checking that the model accurately defines the desired behavior of the system.

Simulation was another useful tool in helping to develop an understanding for the system. However, its use was discouraged, as with the code, because the implementation of the simulation could have corrupted the intent of the design. The difference between simulation and *mental walk-throughs* is that simulation uses implemented software behavior versus intended behavior. Simulation was helpful in that it presented a visualization of the vehicle in flight and another way to test the behavior.

Flight-testing helped develop another level of understanding. The problems with the design of the FMS would be apparent during testing on the field where the system is physically subjected to real environmental conditions. The benefits of flight testing in the identification of hazards are discussed in Chapter 5.

As mentioned earlier, modeling the specifications was one of trial and error. The sooner the modeling process begins, the more iterations can be performed. One of the main activities necessary in applying this modeling language to the FMS is the

identification of states. As mentioned above, the states of the model can be any set of parameters that adequately describe the FMS's view of the system. Identifying the states to describe the system in terms of *blackbox behavior* was difficult because the definition of *blackbox behavior* was not well understood. The initial attempt at modeling was a failure because the states chosen to represent the system were related to constructs found in the code. These states did not abstract enough of the behavior of the overall system. For example, in the first model, the flag that represented whether communications were active or down was used instead of the concept of communications status itself.

Another important point relates to the naming of the states. The use of the *underscore* symbol in the names of the state variables confused some readers, who were familiar with the modeling technique, but not with the FMS. They subsequently rejected these states as code related material. Slight modifications of the names of some state variables led to better results when reviewed again by the same people. Although these reviewers had a very little knowledge of the FMS, they were able to make sense of the specifications. The results of the review of these specifications emphasize the need for both content and presentation in order to promote better understanding.

States variables, used to model the estimated vehicle status, are usually physical parameters or conditions of the vehicle. For example, the initial choice of states included counter variables and flags, which did not capture the system's blackbox behavior. The choice of state is not limited to physical parameters. For example, Mode Confirmation Status, which is a measure of whether the vehicle received a guidance command, is not a physical parameter. Yet, it was modeled because it did capture information that could be seen by the FMS (blackbox), and is an important factor upon which the FMS bases its guidance decisions.

Input/output were the first items defined when the modeling process began. Of the various inputs and outputs of the FMS only the one's that affected the guidance sending capability of the system were modeled. The remaining I/O messages are either data requests to the vehicle, or data replies from the vehicle. The I/O were limited to the one's found in the state-machine diagrams of Appendix B.

Modification and revisions to model came with better understanding of the model.

However, the most difficult phase was starting the modeling process. The modeling of the states was difficult because initially, the modeling language was unfamiliar to the analyst, and the choice of states for the model was not unique, and there were an infinite number of ways to represent the system. The resulting set of specifications was kept because it reflects the behavior of the system most accurately, and clearly of all other versions.

Peer review was an important aspect of modeling. Without it, there was no feedback on the readability, usefulness, and correctness of the specifications. It provided a necessary check to the systems analyst. It is suggested to have a small group do the modeling so that the system is viewed from several perspectives. However, the modeling and analysis of the FMS specifications was entirely performed by one person, and the entire process took about a year's time. Fortunately, the system was small or else there would have been confusion due to the increase in the number of state transitions.

3.6 Human Review

Review of the completed specifications was performed by the FMS designer, and another individual who has worked extensively with the hardware components of the system, but had little familiarity with the FMS software. Results of the review by these domain experts found that the specifications were lacking in some respects. Although the tabular specifications were readable, they was not as reviewable as originally intended. Reviewers quickly developed an understanding of how to read the individual state transitions, and felt the specifications were clearer and easier to read than code. The hardware-oriented reviewer even found a system hazard (going to Runup Mode from Takeoff Mode) during the explanation of how to read the specifications. However, the consensus was that the overall structure and the interactions of the various components of the software system were less apparent. Thus, the introduction of additional diagrams and supporting documentation helped to fill in the overall deficiencies of the tabular specifications. Appendix A contains the additional documents supplied to improve the understanding of the behavior of the overall system architecture. These documents especially helped with the timing specifications of the state transitions.

Chapter 4

System Background

This chapter presents the autonomous vehicle system, which includes a qualitative discussion of the parts of the system that were modeled. Figure 4.1 is an illustration of the general architecture of the autonomous vehicle system. It is shown that the system can be broken up into three parts: the vehicle, the GCU, and the operator. The arrows indicate the flow of information, and the adjacent boxes indicated the type of communications used to transfer information. The full model is found in Appendix B.

4.1 Vehicle System

The vehicle is a two horsepower commercially available remote control helicopter, with a six-foot main rotor. Its intended use is for autonomous battlefield or urban reconnaissance. The vehicle houses a variety of on-board hardware systems including a computer processor, various sensors, and a small color camera as payload. The on-board computer runs both the on-board navigation and control systems software for the vehicle [18].

4.1.1 Navigation System

The navigation system provides an estimate of the vehicle's dynamic state to the FMS through the on board computer, which takes the navigation solution from the navigation filter and relays that information to the FMS as a vehicle "State Update" message. This information is used by the FMS to determine the next guidance command to send. The on-board computer also relays the initialization message to the navigation system, which is the only command the FMS can send to the navigation system.

The navigation filter also includes an estimate of its own health status in the vehicle state message. The status of the navigation filter is mainly affected by the quality of the sensor inputs coming into the system. The state of the navigation filter, in turn, can

affect the behavior of the FMS in sending guidance commands. When the navigation filter indicates NAV HEALTHY, the filter is running nominally, and all the sensor inputs are nominal. NAV STANDBY indicates the navigation filter is in the process of initialization and calibration. No outputs are sent to the controls system during this time. NAV DEGRADED indicates that the estimates have been degraded, but autonomous flight is still possible. Finally, NAV DOWN indicates that the navigation filter estimates are too inaccurate for autonomous flight. The navigation filter status is also relayed to the human operator via the GUI.

The navigation filter receives information from an array of sensors, and incorporates them in a Kalman filter, which integrates all the sensor inputs and calculates a navigation solution. The navigation solution includes information on the vehicle's position, altitude, attitude, and attitude rates.

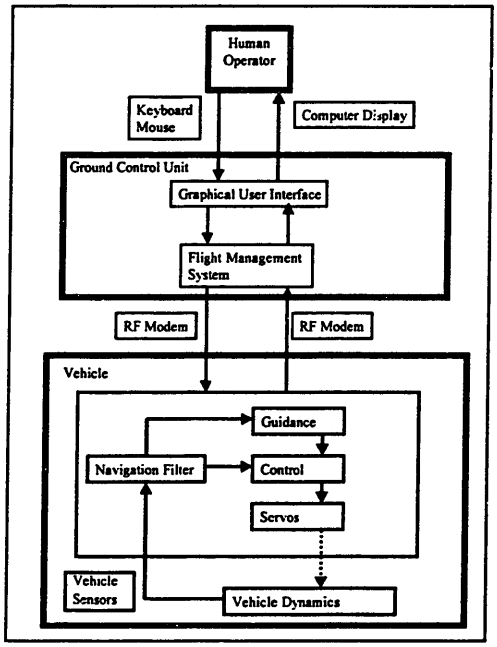


Figure 4.1 System Architecture

4.1.2 Vehicle Sensors

The sensor elements of the navigation system include a differential global positioning system (DGPS), an inertial measurement unit (IMU), a sonar altimeter, and a compass.

The differential GPS is an important component of the navigation system, which provides better accuracy than the traditional GPS [18]. There are two GPS receivers in the system, an on-board receiver, and a ground station receiver. The on-board receiver compares its measured position to the location of the ground station receiver, which is located at a known position. The known ground station receiver location can either be surveyed, in which it does an internal calculation to determine its latitude and longitude, or the coordinates of the ground station can be entered manually by the human operator. The result is a highly accurate low bandwidth navigation system. However, there are accuracy limitations to the DGPS system. Most limitations involve the loss of line of sight tracking from the GPS satellites, and interference from reflections of the signals being transmitted from the GPS satellites. The DGPS will revert to a regular GPS if differential comparison of signals between the ground station and the vehicle are not possible.

The differential GPS works in three modes. The first is GPS OK, where the standard deviations of the satellites are adequate for autonomous flight. The second mode is GPS DEGRADED, where the reported standard deviations and the number of satellites are not optimal, but still usable. Finally, in GPS DOWN, the reported standard deviations of the GPS solutions make the readings useless.

To compensate for some of the shortcoming of the DGPS system, the inertial measurement unit provides the navigation filter with a high bandwidth measurement of the dynamics as well as information on the rotational motion of the vehicle. The single GPS antenna on the vehicle is unable to sense rotational motion.

Additional sensors include a sonar altimeter and a compass. The altimeter is used to aid the system in providing a more accurate means for detecting the vehicle's altitude above ground level at low altitudes during autonomous landings. The compass improves accuracy of the navigation solution in terms of correcting for heading drift, which occur in other sensors.

4.1.3 Guidance and Control System

The FMS is the main guidance controller of the overall system; it is not located on-board the vehicle. However, there is an on-board guidance system, which plays a

small role in the translation of guidance commands issued from the FMS into commands recognized by the on-board controller. The appropriate waypoint commands are sent from the on-board guidance system to the control system after guidance commands from the FMS have been processed through the on-board guidance logic. Once the on-board guidance system receives a guidance command from the FMS, it will be responsible for executing that command to ensure that the vehicle reaches the commanded waypoint. The most recent waypoint commanded by the FMS is maintained by the on-board guidance system until a new command is received.

The control system of the vehicle takes the navigation solution and the translated guidance commands and generates control commands for the vehicle to the servos. There are four independent control loops on the vehicle, which are separated into roll, pitch, yaw and vertical. The control commands control the activity of the six servos on the vehicle, which affect the vehicle's motion. The control loop gains can be adjusted, and the loops can be independently opened or closed by the human operator via the GCU.

4.2 GCU Operator

The human operator, also known as the GCU Operator, plays an integral role in the operation of the vehicle, especially during the initialization phase of a mission. The GCU Operator is needed to power up the FMS and initialize the navigation system, including the DGPS, on the vehicle before a mission can be executed. The GCU Operator interacts with the FMS via the graphical user interface (GUI).

The GCU Operator has the highest level of supervisory control of the FMS. The primary function of the GCU Operator is to plan an autonomous mission, and program the mission into the FMS using the GUI. Missions are specified by the operator in terms of waypoints, which are plotted on a map display in the GUI.

Waypoints are an ordered set of points in space specified by the human operator. The vehicle will execute the mission on straight-line trajectories defined by these waypoints. The waypoints are executed according to their order of entry. A *waypoint* consists of the following parameters:

- Position in terms of field coordinates (x, y) in ft.

- Altitude above ground level in ft
- Commanded Flight Mode Hover Time (if Hover Mode is Specified)

These waypoints can be entered and executed individually or in the form of a list. The list of waypoints defines the mission to be flown by the FMS during autonomous mission execution. This list is referred to as the Task List, and the tasks (waypoints) are executed one at a time. The task list will be discussed in later sections.

A secondary function of the GCU operator is to monitor the status of the vehicle based on status updates displayed on the GUI, and take the appropriate actions in case of a failure.

At any point during mission execution, the GCU operator has the ability to directly command the vehicle via a joystick mounted on the GCU. The operator will be able to command the vehicle's position/altitude/heading while the low-level on-board controller maintains vehicle stability. This is known as Pilot Assist Mode, and will be discussed below.

The GCU Operator also controls the vehicle-mounted camera. A slide controller on the GCU allows the operator to zoom in, tilt, and pan the camera to focus on the objective to be reckoned.

4.3 Safety Pilot

The safety pilot is an experienced R/C helicopter pilot that works with the GCU operator during flight-testing to reduce the chances of accidents. The safety pilot commands an R/C transmitter, which can override commands given by the on-board control system. When this is done, the pilot is in complete control of the vehicle, and the vehicle reverts to a dumb machine. Overriding control commands will signal the FMS that the vehicle is in manual mode. Although the FMS will continue sending guidance commands, they will be ignored.

4.4 Ground Control Unit

The Ground Control Unit (GCU), shown in Figure 4.2, is a field-ready storage unit that contains a majority of ground systems hardware. These components include a laptop computer, a joystick and a payload controller. The FMS and GUI software run on

the laptop computer. The joystick allows the GCU Operator to directly command vehicle in Pilot Assist Mode, and the payload controller allows for the panning and zooming of the on-board camera. In addition, the GCU has a display, which shows real-time video fed from the vehicle.

The only communications link from the GCU to the vehicle consists of a radio modem. The communications link serves a number of purposes. First, the GPS receivers use it to determine the DGPS solution. Second, the communication provides a link for sending vehicle state updates and other on-board information to the FMS. The FMS also uplinks commands to the vehicle via the RF modem.

4.4.1 FMS/GUI Software

The FMS and GUI software specifications are the subjects of modeling in this research investigation, and will be discussed more in depth in the following sections. As mentioned in Chapter 1, the GUI is included in the model because it can be considered an extension of the FMS.

Although the GUI has some logic, the requirements model was generated from the standpoint of the FMS. From the perspective of the FMS, the GUI merely acts as an input/output device to the human operator, and so the model of the GUI was generated from that perspective. The GUI is included in the model to indicate the allowable inputs from the human operator to the FMS as well as to indicate the outputs from the FMS to the human operator.

The Flight Management System (FMS) is part of a hierarchical control system responsible for autonomous guidance of the vehicle [19] during mission execution. In addition, it serves as the only interface between the human operator and vehicle. The FMS sends a variety of commands to the vehicle, but the most important are the guidance commands.

4.4.2 Flight Management System Flight Modes

In addition to sending the position, heading and velocity commands that are sent to the control system, the FMS also sends *flight modes*. The *flight modes* are used to determine the overall behavior of the vehicle as it is executing a mission. The *flight modes are*: Ground Mode, Runup Mode, Takeoff Mode, Waypoint Hover Mode,

Waypoint Through Mode, Waypoint Land Mode, and Pilot Assist Mode [19]. The flight modes of the vehicle determine the actions of the on-board guidance system, which command the on-board controller.

Ground Mode is the flight mode the vehicle is in when the vehicle is on the ground before a mission has been executed. In Ground Mode, the controller will command a low throttle setting so the engine will idle, allowing the GCU operator a safe approach to the vehicle.

When in *Runup Mode*, the vehicle is commanded to increase its throttle setting gradually, while the rotor blade maintains a low pitch angle, resulting in negligible lift. This flight mode allows the engine to reach the Rpm's necessary for takeoff without actually performing the takeoff maneuver.

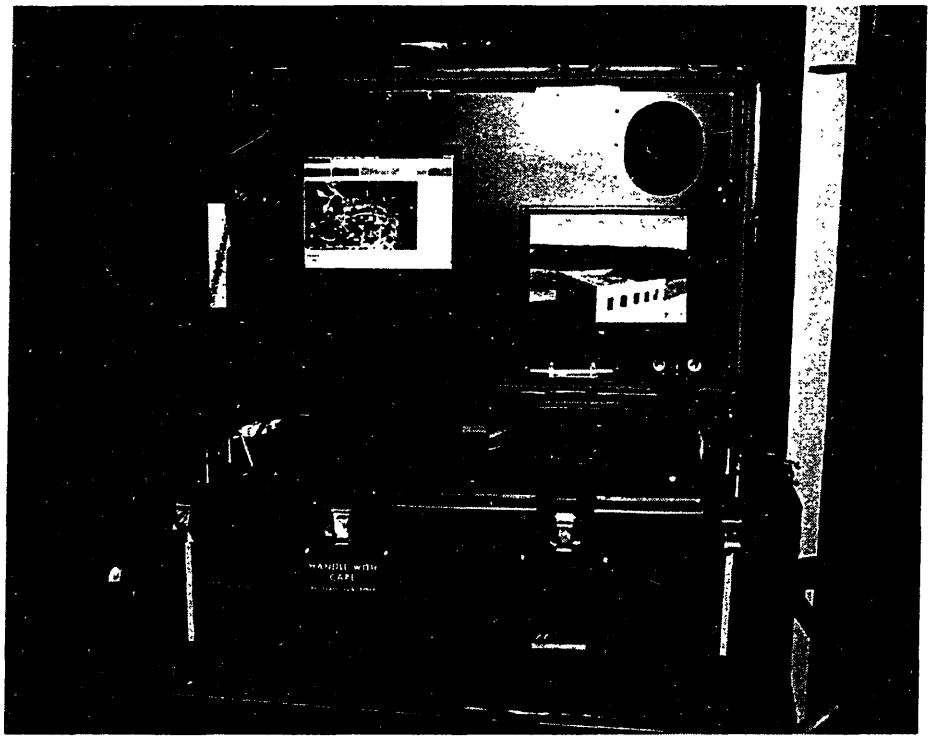


Figure 4.2 Ground Control Unit

In *Takeoff Mode*, the computer commands the same throttle setting as in Runup Mode, while collective pitch is gradually increased to produce lift. The result is the vehicle lifts off the ground and continues to ascend until it reaches the specified takeoff altitude. Once this is attained, it will maintain a hover. While the vehicle is close to the ground, the horizontal controller does not attempt to keep the vehicle level. The reason is

that any corrective motion could potentially result in the vehicle rolling over into the ground.

Waypoint Hover Mode commands the vehicle to travel to a specified position (x, y, z) and hover for a specified amount of time. This occurs in two stages. During the first stage, the vehicle travels to the specified location. Once there, the FMS will start a timer to ensure the hover has satisfied the required time.

Waypoint Through Mode is similar to *Waypoint Hover Mode*. In this case however, the vehicle is commanded to fly through the commanded waypoint without trying to reach the altitude specified. Only the horizontal position (x,y) is considered in this mode. The reason is that Waypoint Through Mode is a way to have the vehicle quickly cover distances, without having to worry about altitude.

Maneuvering in *Waypoint Land Mode* is decomposed into several phases. In the first phase, the FMS issues the command, which is received by the on-board guidance system. The on-board guidance takes commands, and instructs the control system to maneuver the vehicle to travel to the appointed landing site and hover there at the commanded landing altitude. The second phase of landing is the initial descent. During this phase, the vehicle descends from the landing altitude to a controller-specified altitude of 10 feet at a rate of 1 foot per second. Up to this point the on board computer is in total control of the landing sequence. As the vehicle reaches 2 feet above ground level, the FMS will give the signal for the vehicle to land. The control system then goes into an open loop descent in which the engine ramps down, resulting in a quick vertical descent to the ground.

Pilot Assist Mode allows the operator to directly control the vehicle's velocity by a joystick found in the GCU. This is similar to manual operation of the vehicle by the safety pilot, except in this case, the on-board controller plays an active role in stabilizing the vehicle. If no commands are issued by the joystick, the vehicle remains in a stable hover. Thus, the operator need only worry about commanding the vehicle to the desired location. Pilot Assist Mode was developed mainly for the purpose of fine tuning vehicle position for surveillance purposes.

4.4.3 *Flight Management System Guidance Modes*

The FMS operates in several modes. The guidance of the vehicle by the FMS will be managed differently as determined by the guidance mode. There are three major guidance modes to the FMS: Single Mode, Task List Mode, and Return Home Mode.

In *Single Mode*, the guidance commands sent by the FMS are under manual control by the human operator. The human can specify a flight mode and signal the FMS to send it to the vehicle. The FMS takes the commanded flight mode from the human operator, and transmits the command to the vehicle after the FMS logic has verified that the transition between the current mode and the requested mode is a valid one. After sending the new guidance command, the FMS is idle until a new guidance command is issued by the operator, or the guidance mode of the FMS is switched. In addition, when in Single Mode, the FMS does not check to see it the currently executing waypoint has been completed before sending a new one. As a note, the operator can fully access the entire set of flight modes in Single Mode via the Engineering GUI. *Pilot Assist* is the only flight mode that can be directly commanded via the Operational GUI. A discussion of the GUI will follow shortly.

Task List Mode is used for fully autonomous mission execution. The FMS processes the task list and updates, whether the vehicle has reached the commanded waypoint based on the vehicle's one Hz update rate. Once a waypoint has been completed, the FMS sends the next waypoint to be executed up to the vehicle. The task list is composed of only three of the seven flight modes mentioned previously: 1) Waypoint hover, 2) Waypoint through, 3) Waypoint Land. In Task List Mode, the FMS automatically executes the other flight modes in order to legally transition between the vehicle's current flight mode, and the flight modes on the task list.

For example, the vehicle must go through Runup Mode and Takeoff Mode before it is able to reach Waypoint Hover Mode. Therefore, if the next task on the task list is Waypoint Hover, and the vehicle is currently in Ground Mode, the FMS will automatically send Runup Mode to the vehicle. After Runup Mode has been achieved, the FMS will command the vehicle to takeoff. Finally, when Takeoff has been achieved, the FMS will send Waypoint Hover Mode to the vehicle. This example falls within the general scheme of autonomous mission execution for the FMS, which takes place in 5

stages: mode completion, mode determination, mode transition, mode request, and mode advance.

Mode Completion checks to see if the vehicle is ready to move on to the next flight mode by checking to see if it has completed it current waypoint. *Mode Determination* identifies the current waypoint of interest. Once this is done, the FMS must determine the intermediate flight modes to transition through in order to reach the waypoint of interest. This process is called *Mode Transition. Mode Request* is the actual issuing of the guidance command to the vehicle, and *Mode Advance* occurs when the vehicle receives the request and sends down an acknowledgement message. Once this occurs, the process resets.

Return Home Mode is specifically used to interrupt the current mission for whatever reason, and have the vehicle take immediate action to return to a designated location. The logic in *Return Home Mode* is similar to that found in Task list Mode.

4.4.4 Flight Management System Overview

The FMS software runs on the GCU computer and has three primary functions.

- Sending and receiving messages to and from the vehicle.
- Receiving commands from the GCU Operator via the GUI and processing them appropriately
- Monitoring the status of a mission and autonomously sending guidance commands to the helicopter [19].

The input commands from the vehicle are listed in Appendix B.1, page 123. The most significant input messages that the FMS receives from the vehicle are the *State Update Message* and the *Mode Confirmation Message*. The *State Update Message* comes down from the vehicle at a frequency of 1 Hz, and includes a variety of information on the vehicle's dynamic states such as position (x,y), altitude, and velocity. Additional information, which can also be found in State Update Message, is navigation health, time, and GPS status. This message is used by the FMS to determine whether a waypoint has been achieved, as well as for sending guidance commands in general.

Mode Confirmation Message is an output from the vehicle to the FMS. This is sent in response to guidance command to indicate that it has been received. The message includes within it the flight mode currently being executed by the vehicle, along with the commanded parameters it is currently trying to track. It informs the FMS that the guidance command send by the FMS has been received by the vehicle.

An additional message of interest is *GPS Message,* which contains detailed information on the status of the GPS system. This information is used to determine whether it is an appropriate time to initialize the navigation filter. The remaining messages related to control system parameters were not modeled in the specifications.

FMS output commands modeled are:

- *Mode Change Request*
- *Mode Request Message*
- *GPS Ground Station Coordinates*
- *Initialize Navigation Filter*
- *GPS Request Message*

Mode Request Message is issued automatically by the FMS as soon as it has been powered up. The purpose of this message is to ascertain the flight mode currently being executed by the vehicle. The FMS is restricted from issuing any guidance commands until it receives a reply from the vehicle in the form of a *Mode Confirmation Message.*

Mode Change Request is the guidance command issued by the FMS to the vehicle. It contains the commanded flight mode to which the vehicle is to transition along with command waypoint parameters.

GPS Ground Station Coordinates sends the initial coordinates of the ground station's GPS receiver to the vehicle in order for the navigation filter to be initialized.

Initialize Navigation Filter instructs the navigation filter to go into calibration mode so it can initialize to the current vehicle position.

GPS Request Message is triggered when the FMS receives the command to send an *Initialize Navigation Filter* message. The GPS solutions must be usable in order for the navigation filter to be properly initialized. Thus, the FMS will issue *GPS Request*

Message first, and will follow with *Initialize Navigation Filter* only when it receives a satisfactory reply from the GPS system.

The FMS updates its estimate of the vehicle's state through messages sent from the vehicle to the GCU via the RF modem at a rate of 1 Hz. As mentioned before, every control system has an internal model of the process being controlled. These take the form of the vehicle model found in the specifications. The model of the vehicle consists of the state variables that capture the estimated states of the vehicle. Each state is preceded with "Estimated", because it is the FMS's estimate of the vehicle's state, and is not necessarily the actual state. The state variable include the Estimated Flight Mode, Estimated Altitude, Estimated Navigation Filter Status, Estimated GPS Status, Estimated Position, Estimated Time, Estimated Velocity, Estimated Mode Confirmation Status, Estimated Control Loop Status, and Estimated Supervisory Mode. Detailed explanations of these state variables are found in Appendix A.1.

The FMS uses information from its internal model of the vehicle as well as commands received from the GCU Operator to determine the appropriate command to issue. Additional factors that affect the behavior of the FMS are the various functional modes of the FMS. The functional modes indicate mutually exclusive behaviors of the FMS. For example, the FMS Guidance Mode determines whether the FMS executes an autonomous mission (Task List Mode), or allows the human operator to send individual waypoints (Single Mode). The other functional modes of the FMS are Guidance Initialization Status, Navigation Initialization Status, and Communications Status. The Guidance Initialization Status of the FMS is a mode that determines whether the FMS has been enabled to send guidance commands. When the system first starts up, FMS is in an uninitialized mode, which prevents the FMS from processing any operator commands for sending guidance commands. This is shown in Appendix B.1, page 124. Navigation Initialization Status determines whether the GUI can accept an *Execute Mission* command from the operator. The navigation filter must have been previously initialized in order for the command to be accepted by the Operational GUI. Communications Status indicates whether the vehicle and the FMS are able to transfer and receive information to and from each other. Communications Status is not really a functional

mode, but it was included in the model as part of the system modes as opposed to a vehicle state because it seemed to be more of a system property than a vehicle property.

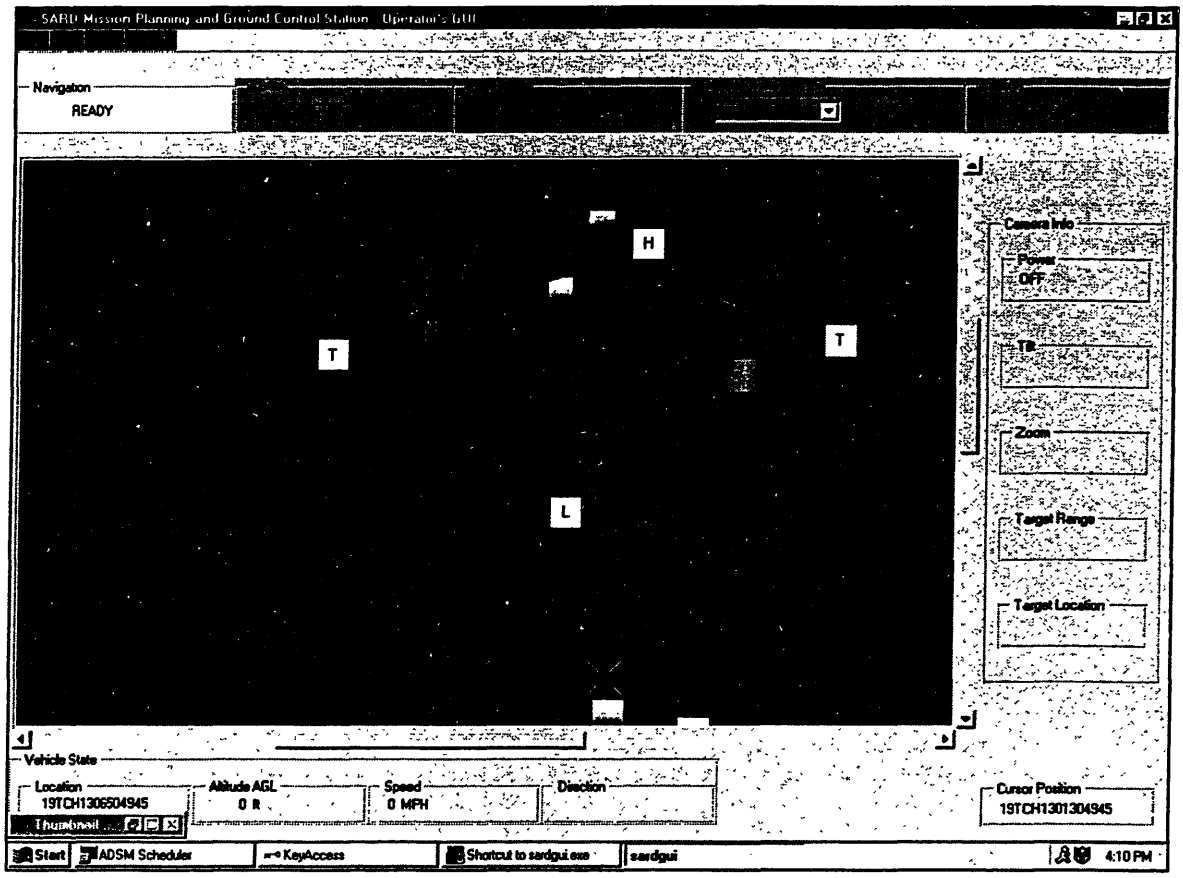


Figure 4.3 Operational GUI

The FMS specification also contains a model of the task list dynamics. The task list dynamics specify how tasks are to be added, how they are to be deleted, as well as the order in which tasks should be executed.

The commands sent to the vehicle include the output commands mentioned above as well as data request commands for the various systems on the vehicle. The specifications model of the FMS captures the logic involved in the determination of the correct command to send. These are the state transition tables of Appendix B

The commands the FMS receives from the GCU Operator will be discussed in the next section.

4.4.5 GUI

The GUI is a component located on the GCU, which is responsible for graphically displaying crucial information related to the guidance and control of the vehicle. The input interface between the GCU Operator and the GUI is with a keyboard and mouse as physical controllers. The GUI is divided into two separate displays. The primary display is known as the Operational GUI. This will be used as the only display during mission execution by customers, and is limited by the versatility of available guidance options. The secondary GUI, referred to as the Engineering GUI will be used mainly for flight testing of the vehicle. The Engineering GUI is more flexible than the Operational GUI, and is accessible through the Operational GUI as one of the options. The two GUI's display the similar information, although the engineering GUI includes more details and allows the GCU operator access to more diagnostic functions. In the discussions that follow refer to Appendix B.1 to understand how the GUI relates to the specifications.

The Operational GUI display is shown in Figure 4.3. There are four different menu items in the Operational GUI, each containing submenus, which allow the user to enter a number of different parameter values. The FILE menu has been simplified in this model to include only three options: *Delete Waypoints, Show Engineering GUI,* and *Quit.*

Delete Waypoints allows the user the remove all of the waypoints that have previously been entered and stored in the task list. *Show Engineering GUI* brings up the Engineering GUI as another display window. *Quit* serves to shut down the GUI and the FMS.

The EDIT menu in this model has been shown to only contain the *Create a New Waypoint* submenu. The GCU Operator uses this command to create new waypoints that can directly be incorporated into a mission. The GCU operator is prompted to enter the waypoint type and the relevant associated physical parameters, such as location and altitude, and hover time. The desired location of the waypoint can be specified using the mouse to point and click on the corresponding point on the map display, or it can be entered into the textbox in MGRS coordinates. The altitude is entered manually, and depending on the waypoint type involved, the hover time is also presented as an option. The *Entry Complete* button is used to signal the FMS that the information for the

waypoint has been completed. It is assumed for the purposes of this thesis that the FMS immediately incorporates the waypoint into the task list when it receives the signal.

The third menu item is the SYSTEM menu, which gives the human operator access to systems level commands. These include *Initialize System*, *Execute Plan, Pilot Assist,* and *Return Home*. *Initialize system* is a command used to signal the navigation filter to begin its initialization and calibration stage. There are two ways to do this. The GCU Operator can either enter the starting coordinates of the system manually, or the operator can instruct the navigation system to survey its coordinates based on readings from the GPS and automatically calculate its starting location.

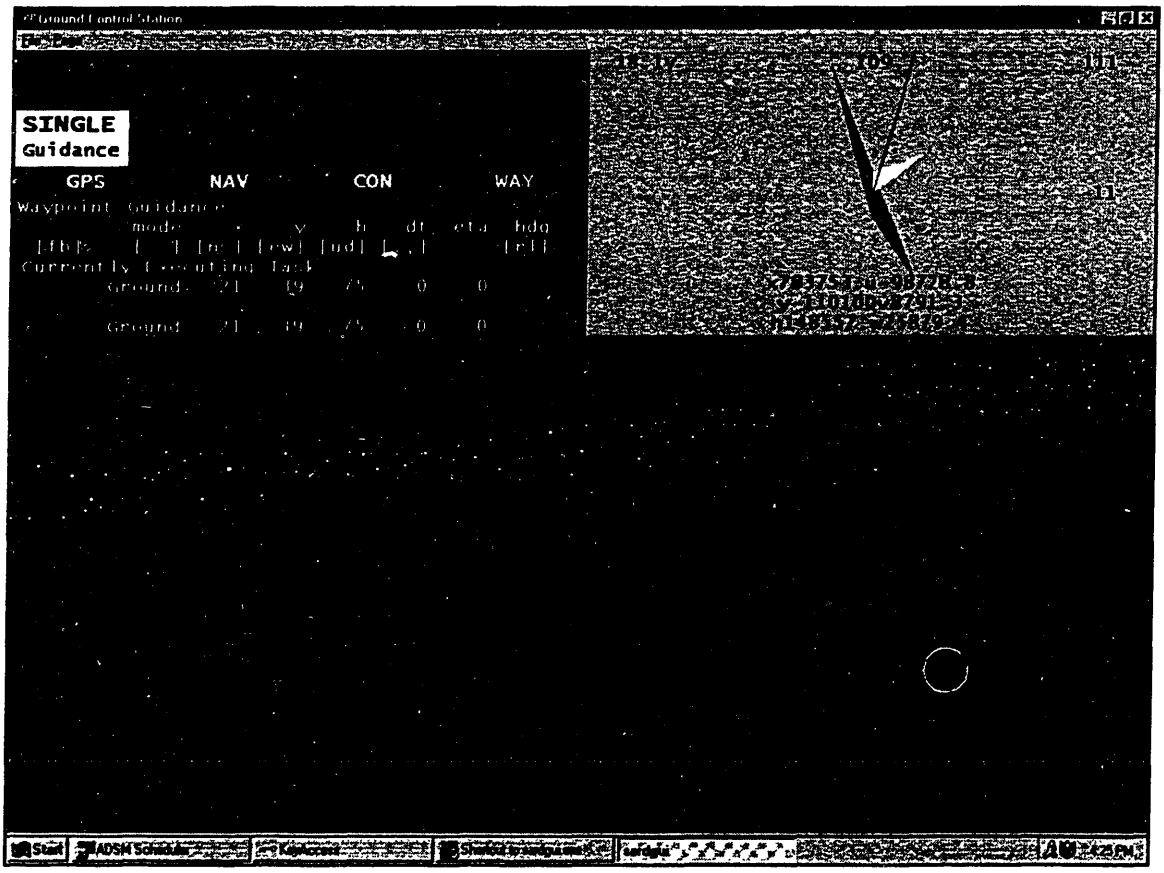


Figure 4.4 EngineeringGUI

Execute Plan instructs the FMS to execute the current mission (waypoints on the task list) that has been planned by the GCU Operator. *Pilot Assist* interrupts the current mission, and puts the GCU operator in manual control of the vehicle. The GCU Operator uses the joystick in the GCU to directly command vehicle velocities in the lateral,

longitudinal, and vertical directions. Heading can also be commanded with the joystick. The GCU operator only has partial control of the vehicle, since stability is still maintained by the on board control system. *Return Home* interrupts the current activity of the vehicle, and directs it to return immediately to the pre-appointed home location (currently the starting position). The GUI display consists of a map of the local area along with various status displays located throughout. The menu items just discussed are found at the top of the GUI. Starting with the main display, various maps can be preloaded into the GUI to display the area in which the vehicle is to exercise its mission. The vehicle is displayed as a triangular icon, where the head of the triangle indicates the heading of the vehicle. The mission flight trajectory is also overlaid on the map with the waypoints connected by the flight path the vehicle will follow. The waypoint type is indicated by the letter appearing in them H/T/L for Hover/Through/Landing. The home point it indicated by an "X".

A vehicle state panel, located at the bottom of the display, indicates the FMS's current estimate of the vehicle's state. The state estimates are the vehicle's coordinates, altitude above ground level, speed, and heading. Another slew of panels, which gives system-level information, is located on top of the map display. This panel indicates the status of various systems on-board the vehicle as well as additional health information for the vehicle. One indicator displays the health status of the navigation filter, which can take on the different values discussed previously in Section 4.1.1. The communications status indicator shows the communications status. There is an indicator of the current flight mode being executed by the FMS. Another indicator keeps track of the various times involved in a mission such as the time in the current flight mode, the total mission time, and the amount of time spent in hover. The last panel shows the battery status.

The sample display of the Engineering GUI is shown in Figure 4.4. It is similar to the makeup of the Operational GUI, but is not as menu driven. Instead of menu items, the various features found in the Engineering GUI are located on four separate pages: GPS page, Navigation page, Control page, and Waypoint page.

The GPS page allows the operator to enter commands related to the GPS system. The three commands modeled in the specifications model are:

- *Program GPS Cards*

- *Upload Settings*
- *Request GPS Data.*

Program GPS Cards allows the human operator to startup the GPS system in preparation for flight. *Upload Field Settings* allows the operator to initialize the GPS system with pre-existing field coordinates as the starting location. Finally, *Request GPS Data* sends a request through the FMS to the vehicle for raw GPS information such as position, altitude, and the GPS solution status. The displays on this page include the numerical values of latitude, longitude, altitude, as well as their standard deviations calculated by the GPS system. It also gives information about the number of GPS satellites being tracked, the solution status, as well as the quality of the solution.

The Navigation page allows the operator to initialize the navigation filter, and request raw compass and sonar data. The initialization coordinates are entered into textboxes.

The Waypoint page gives the human operator control of the guidance commands sent to the vehicle. On Waypoint page, the active page of Figure 4.4, the operator can switch between guidance modes, such as from Single Mode to Task List Mode. The *Switch to Single Mode* button will signal the FMS to command the vehicle into the flight mode indicated in the editable line under the button. The parameters found here are the same as those found in the *Create a New Waypoint* option in the Operational GUI. The *Switch to Task List Mode* button allows the human operator to signal the FMS to go into Task List Mode. The list of waypoints directly below the button is the task list. The task list is executed in order from top to bottom, and the current task is indicated with an arrow. New waypoints always appear at the end of the task list.

The last page is the Control page, which pertains to commands given by the FMS to the on-board control system. The commands included in this model were *Record Trims*, *Request Trims*, *Open/Close Control Loops*, *Adjust Gains* and *Upload Gains and Switches*. A note should be made here, that although the commands appear in the state diagrams of the specifications, no state transition specifications have been written for them. These commands were not included in hazard analysis either. It was assumed that the human operator would not try to access any of these commands once all the

parameters have been correctly set. *Record* and *Request Trims,* respectively, allows the operator to record the current trim settings in the control system, and request the trim settings to be displayed on the GUI. *Open/Close Control Loops* allows the user to deactivate any of the four independent control loops in the on board system: pitch, roll, yaw, vertical. *Adjust Gain* allows the operator to change the gains associated with each of the control loops to alter the flight dynamics of the vehicle. *Upload Gains and Switches* relays these changes to the on-board control system.

The Engineering GUI also contains a rudimentary map display where a vehicle icon indicates the heading and position of the vehicle. Another graphical display gives the GCU operator a graphical means to determine the velocities, attitudes, and the attitude rates of the vehicle. This is shown in Figure 4.4. As the triangular figure in the upper right hand corner. A summary display, to the upper left, gives flight critical information such as communications status, GPS status, navigation health, guidance mode, and flight mode.

Chapter 5

Analysis and Results

This chapter will explore methods in the exploration and of the specifications model. Some of these techniques will be applied specifically to perform hazard analysis of the requirements specifications. Hazard analysis is performed by determining whether a hazardous state can be reached from a safe state. Table 5.1 summarizes the results of the hazard analysis, which will be discussed at length in later sections. First, hazardous states for the autonomous vehicle system need to be defined.

Section Reference	Hazard	State Space Description of Hazards	Reachability
3.1.1	Loss of Lift While in Flight	Estimated Vehicle GuidanceMode **in state** GROUND MODE and Estimated Altitude **in state** $\geq$ 5ft. Estimated Vehicle Flight Mode **in state** RUNUP MODE and Estimated Altitude **in state** $\geq$ 5ft.	YES, Eng. GUI
3.1.2	Loss of Navigation System While in Flight	Estimated Navigation Filter Status **in state** STANDBY and Estimated Altitude **in state** $\geq$ 5 ft.	YES, Eng. GUI
3.1.3	Low Altitude Maneuvering	Estimated Vehicle Flight Mode **in state** WAYPOINT HOVER MODE and Estimated Altitude **in state** $\leq$ 5ft. Estimated Vehicle Flight Mode **in state** WAYPOINT THROUGH MODE and Estimated Altitude **in state** $\leq$ 5ft. Estimated Vehicle Flight Mode **in state** PILOT ASSIST MODE and Estimated Altitude **in state** $\leq$ 5ft.	YES, Eng. GUI YES, OPERATIONAL GUI Comm Goes Down
3.1.4	Injury from Rotor Blades	Estimated Flight Mode **in state** RUNUP and Vehicle **in state** OFF	YES, Eng. GUI and OPERATIONAL GUI
3.1.5	Negative Altitude Flight	Estimated Altitude **in state** $\leq$ 0 ft.	NO
3.1.5	Unexpected Takeoff	Flight Manager Guidance Mode **in state** RETURN HOME MODE and Estimated Vehicle Flight Mode **in state** GROUND MODE and Vehicle is at the home position.	YES, OPERATIONAL GUI
3.1.5	Landing at Wrong Location	Estimated Velocities U, V, W **in state** $\leq$ 1.0 fps and Estimated Altitude **in state** $\leq$ 1.0 ft and Estimated Vehicle Position **in state** NOT IN POSITION	YES, Low Likelihood

Table 5.1 Summary of Hazard Analysis

5.1 Hazards

The chosen state-based model allows for the explicit definition of hazards in terms of *hazardous states*. A *hazardous state* is defined as a set of conditions of a system that, together with other conditions in the environment of the system, will lead inevitably to an accident [13].

Identification of whether the FMS specifications contained any hazardous states begins with the definition of potential system hazards. As mentioned in Chapter 2 accidents are the result of faulty software specifications. However, accidents are physical occurrences. In reality, it is the combination of software and physical circumstances that lead to accidents. Therefore describing a hazard just in terms of the component's software parameters is not practical for hazard identification. Instead, hazards should also be defined in terms of physical parameters or unsafe physical conditions. Once the hazardous state has been identified, the task becomes one of tracing the software specifications to prove or disprove whether they lead the physical system into the hazardous condition. The success of this task depends on the familiarity of the analyst's enthusiasm for working with system requirements [15]. The novelty of the FMS presented some difficulty, but several factors aided in hazard definition.

Conversations with the FMS designer proved to be very useful in defining system hazards. In these conversations, the designer voiced concern with several system behaviors, which ultimately contributed to the development of the list of hazardous states. The designer was unsure of the desired behavior of the system in some cases. However, there was a strong conviction as to the types of behaviors the system was not to exhibit. These undesirable behaviors were translated into state-space descriptions of hazardous states. For example, the designer defined an FMS controlled maneuver into the ground as undesirable. This lead to defining possible crash situations as hazards.

The designer's experience with operating the vehicle and the GCU was a valuable resource in hazard definition. Many of the missions flown by the vehicle in simulation and during flight-testing were conducted by the designer. The advantage was that the actual behavior of the system could be directly compared to the designer's intent. In general, the designer of the system is not always the one who does system testing. System hazards defined from a designer's perspective would be biased towards problems

noted during design. This would potentially exclude problems actually encountered in the field. This is a detriment for the hazard identification process since the physical situations encountered by the system may not be known by the designer. On the other hand, hazards from a testing standpoint would involve problems encountered on the field with little insight into what in the design might be causing the problem. For the FMS, both perspectives were integrated into one engineer, so problems encountered on the field were presented with some indication of design faults.

Some hazards of particular concern were identified on the field. Many of these problems were not identified or even considered hazards when the system originally underwent testing. The hazards were the result of *unplanned operator behavior* or *extreme conditions*. *Unplanned operator behaviors* are assumed to be actions taken by the operator that were not designed for in the system, which can lead to accidents. These actions are assumed to be improbable since the design is based on the assumption that operators would have enough system comprehension to preclude these actions. Therefore, safeguards were not developed to defend against these actions. Experienced operators have developed an implicit understanding of how the system operates, and it is improbable for them to take these actions. However, in many situations such as customer demonstrations, the FMS is operated by inexperienced people who are unfamiliar with, and have just the minimal amount of knowledge necessary to operate the system.

During one particular customer demonstration, an inexperienced operator executed a mission much quicker than expected. In very little time, the operator had planned a mission and sent the *Execute Mission* command to the vehicle. Unfortunately, the on-board controller received this command while the vehicle's engine was off. This caused the on board guidance system to transition into Runup Mode, at the same time another person was approaching the vehicle to power up the engine. Recall that Runup Mode causes the main rotor to spin up to takeoff Rpm. The person approaching the vehicle to start the engine could have been severely injured from the rotor blades. Fortunately, an experienced operator, who was monitoring the trainee, noticed the problem and corrected it, preventing anyone from being injured. This event subsequently defined led to the definition of a hazard.

It should be noted that many of the hazards due to unplanned behaviors occurred within the context of the Engineering GUI, because this GUI allows the operator more freedom to send commands to the vehicle. Operators who use the Engineering GUI should have an extensive understanding of how the guidance system operates. Less experienced operators should use the Operational GUI, which is easier to use and limits the commands the operator can send.

Extreme conditions are physical conditions assumed highly unlikely to occur. As an example, landing at the wrong waypoint as shown in the last entry of Table 5.1 requires a combination of highly improbable events. However, the possibility of these events occurring merits investigation.

The designer was not the only source used in defining system hazards. Designers who have worked with a system for a long time tend to be caught up in the details of the system. This tendency may cause them to overlook problems. Some hazards could only be identified with the introduction of a fresh perspective on the system.

The process of modeling allowed an outside analyst to develop an understanding and appreciation for the system. This helped the analyst to uncover hazards not identified by the designer. One such example is the third entry of Table 5.1, maneuvering at low altitudes. Some of these hazards were identified through questions that appeared during the modeling process. Many questions involved how to model certain aspects of the system. These lines of questioning led to the discovery of hazard number two in Table 5.1. The point is that modeling and analysis can occur concurrently in practice. It is often a good approach to take.

An example of analysis and modeling occurring together, was the identification of a hazard during a *mental walk through* of the system's discrete dynamics. It was found that the vehicle could be commanded via the Engineering GUI into Task List Mode even when the navigation filter had not been initialized. The system would switch from Ground Mode to Runup Mode, but the vehicle would not takeoff. This would have caused confusion for the operator who might have assumed that the navigation filter had already been initialized, since the FMS was able to switch into Task List Mode. This hazard is not mentioned further in this thesis, but should be further pursued.

Another illustration of analysis and modeling occurring at the same time led to the discovery that the FMS could send flight mode commands to the vehicle even if the navigation filter was down. This is potentially a dangerous event because if the vehicle were to track these commands, it would have no sense of direction since the navigation filter is down. Fortunately, it was discovered that there were checks in the on-board control system, which would prevent the system from following the guidance commands until the navigation filter had been successfully initialized. Although the hazard was identified in error, the objective of this example was to encourage analysis during the modeling process, and to show that there was a symbiosis between the two. Modeling helped to identify hazards, while analysis helped to find errors in modeling. Indeed, this was how many of the FMS hazards were identified.

5.2 Hazardous States Defined

The major hazard of concern was crashing the vehicle. The goal here was to ensure that software would not be a contributor to crashing the vehicle. Within the context of the FMS, crashing can occur in several ways. The only behaviors of interest were those affected by the FMS. For example, mid-air collisions were not considered, since the vehicle was not equipped with a collision avoidance system [18]. Mainly collisions with the ground were of interest. However crashing was not the only concern. The following subsections describe the hazards that have been defined so far for the system.

5.2.1 Loss of Lift While in Flight

The concern here was voiced by the designer regarding the loss of lift in flight, which would subsequently result in a crash. There are two possible states described by the model that results in this situation. Part of the state description would have to be physical. The vehicle must be in flight when the loss of lift occurs, since no damage will occur if the vehicle is grounded. It is postulated that if the vehicle were to drop more than 5 feet, considerable damage can occur. Therefore, part of the state space description of the hazard will be that the altitude be above 5 feet.

According to the discussion of the flight modes in Chapter 4, it is known that two flight modes result in the loss of lift: Ground Mode, and Runup Mode. Therefore, the

conjunction of these state variables defines the loss of lift while in flight. The two hazardous states are depicted in Figure 5.1.

Figure 5.1 State Space Description of the Loss of Lift in Flight

5.2.2 Loss of Navigation System While in Flight

Initializing the navigation filter results in the system going into a calibration, or STANDBY MODE, during which navigation output is not sent to the vehicle's control system. This is not dangerous while the vehicle is on the ground, but if the vehicle were in the air, the control system would send servo command to stabilize the vehicle without input from the navigation system. This is similar to the Ariane 5 example found in Chapter 2 and will inevitably lead to a crash. This hazardous state is show in Figure 5.2.

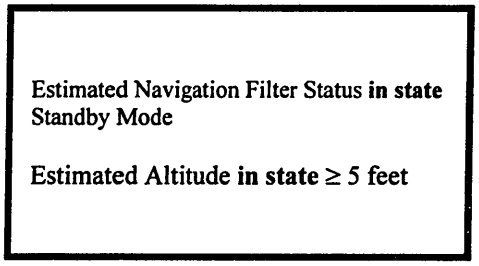


Figure 5.2 State Space Description of the Loss of Navigation System in Flight

5.2.3 Maneuvering at Low Altitudes

This hazardous condition was implicitly defined by the designer, but was not explicitly stated by the designer during conversation or caught during the modeling process. This hazard manifested itself from comments buried in the code. The comment reads:

If the helicopter is in WPNT_LAND_MODE and above 5.0 feet, switch directly into the next tasklist mode. Otherwise follow the proper sequence in gcsModetransition ().

This comment (and a conversation with the designer about the 20-foot altitude limitation in Pilot Assist Mode) prompted the examination of whether the vehicle was ever allowed to maneuver below 5 feet. As stated in the section on landing, in Chapter 3, the last leg of the landing maneuver is an open loop drop.

The designer did not wish to use closed loop control during that portion of the landing sequence because this could cause the vehicle to collide with the ground. The same requirements apply for low altitude maneuvering. For example, when the vehicle banks to head towards a waypoint, it may lose some altitude. This is acceptable when the vehicle is in the air, but it could be disastrous when the vehicle is close to the ground, since it increases the chances of the vehicle rolling over [19]. Thus, the hazard is defined in part by specifying an altitude limit under which the vehicle must not be commanded to maneuver. The five-foot limit is used. The remainder of the hazard description is in defining the flight modes involved in commanding vehicle maneuvers. From Chapter 4, these are Waypoint Hover Mode, Waypoint Through Mode, and Pilot Assist Mode. Figure 5.3 illustrates the three hazardous states that describe this hazard.

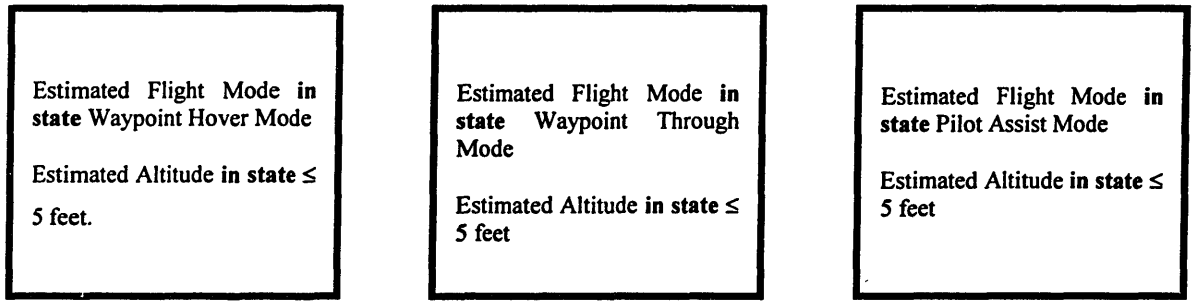


Figure 5.3 State Space Description of Low Altitude Maneuvering

5.2.4 Injury From Rotor Blades

Injury from the rotor blades was determined to be a possible hazard if the engine were to spin up to high Rpm if a person were nearby. The vehicle's engine requires an operator to approach manually to start it. If the vehicle were already in a mode requiring a high engine throttle setting, such as Runup Mode, injury can occur. Figure 5.4 shows

the hazardous state. It should be noted that the state variable, Engine, is not currently in the model of the FMS, but should be included.

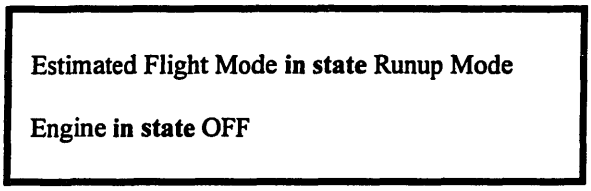


Figure 5.4 State Space Description of Unexpected Spin Up of Main Rotor Blades

5.2.5 Other Undesirable Behaviors

Three other behaviors were identified as undesirable, but not necessarily hazardous. These are mentioned just to make a note that these behaviors exist in the system. One undesirable characteristic is the capability of the human operator to enter negative altitudes as a waypoint parameter. The FMS/GUI system was not designed to ensure that the altitudes entered are above a certain height. The problem is obvious, in that it is possible to command the vehicle into a collision with the ground if the operator is not careful. Fortunately, there is logic in the control system that prevents the vehicle from tracking a negative altitude command. However, this anomaly can confuse the operator who sees that the commanded altitude is not being met.

Another undesirable system characteristic is that it is possible for the system to command a landing at the wrong location. The specifications for landing are found in the function Waypoint Achieved$_F$ as:

- Estimated Velocities U, V, W **in state** $\leq$ 1.0 foot per second.
- Estimated Altitude **in state** $\leq$ 1.0 feet.

The specifications do not place conditions on the vehicle's position. Theoretically, the above conditions can be satisfied with proper wind conditions, and erroneous sonar readings. For example, if the wind conditions were enough to null the velocities of the vehicle, and a bird flies closely under the sonar at the same time, the FMS may consider

landing to be completed. It would then send the vehicle into Ground Mode. If the vehicle is close to the ground when this occurs, the most that will happen is the vehicle lands in the wrong place. If this happens when the vehicle is in the air, traveling to the landing site, it may result in a crash. This was not further considered because of the low probability of these events occurring.

Another unexpected behavior of the system is that it allows the human operator to send the vehicle into Return Home Mode, even when the vehicle is already in the home position. The specifications do not require a check on the vehicle's position. The human operator may accidentally hit the Return Home command in the GUI if there is an inaccurate mental model that the vehicle should not return home if it is already home. Common sense dictates that if the vehicle were in the home position, it should not takeoff. This assumption is similar to the situation with the landing gear example, mentioned in Chapter 2, where there was no expectation for the gear to be raised while the vehicle was on the ground.

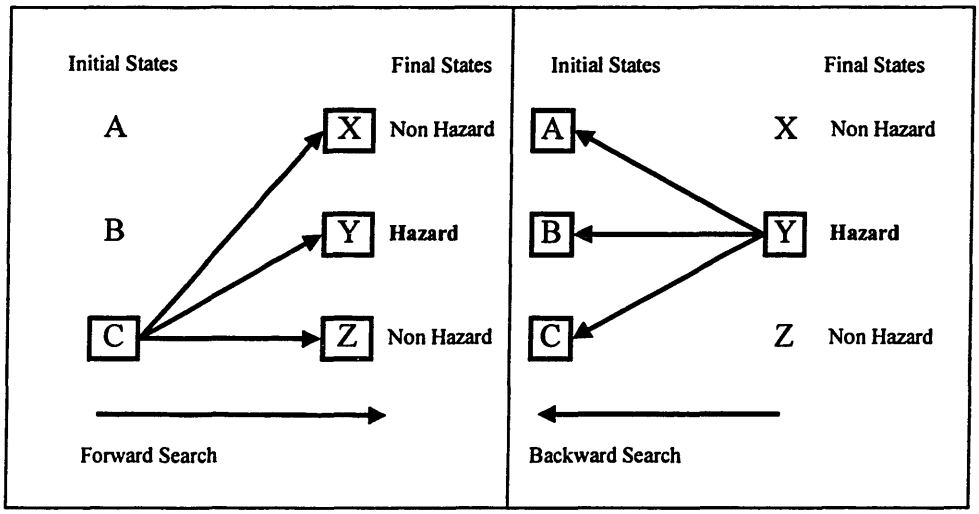


Figure 5.5 Forward and Backward Search

5.3 Hazard Analysis

Analysis of the specifications lead to the confirmation that some of these hazardous states existed in the system. The analysis of the FMS software specifications consists of several state-space search techniques. Initially, the search techniques recommended by Leveson were to be automated. However, the analysis done during this

thesis was done by hand, but can be automated with further effort. Suggestions for doing so will be presented as future work in Chapter 6. The techniques for hazard analysis, as will be explained in the following sections.

5.3.1 Backward/Forward Analysis

Backward and Forward Analyses are complementary methods that can be used to trace requirements errors when the structure of the requirements is temporal and consists of events and conditions [13]. Figure 5.5 illustrates the mechanics of Forward and Backward Analysis respectively. These concepts are similar to dynamic programming concepts, which suggests that dynamic programming may be one way to automate the search of the state space for hazards.

Forward Analysis starts with the system in an allowable initial state and propagates the state forward to identify the possible hazardous states that result, as well as the chain of events that lead to them. Forward analysis is also known as inductive search. Forward Analysis is used mostly as a technique to trace the consequences of an initiating event. An example would be to determine how the loss of communications would affect the performance of the FMS. Tracing forward will lead to a generation of a large number of states. Identifying all reachable states from an initial state may be impossible to do. Thus, forward analysis is limited to a small number of events. This technique was seldom used in the analysis of the FMS.

Backward Analysis begins with the system in the hazardous state and propagates the state backward to identify the chain of events and the potential initial conditions leading to the hazard's realization. This is also known as a deductive search, since the analysis reconstructs the initiating events leading to the hazard or accident. Backward analysis was used most in the analysis of the FMS. With backward search, a system can be proven not to contain the hazard, if all states leading to that hazard are known to be unreachable. The system contains the hazard if it is possible to find some preceding state, which is known to be reachable. Unfortunately, backward analysis of the FMS specifications led quickly to a state explosion, after which it was very difficult to keep track of all the possible branches that were created. However, there are methods in operations research, such as *branch and bound* and *integer programming,* that may help

in dealing with the state explosion problem. Several of the hazards identified in the system were found using a loose mental application of this backward analysis. This includes Maneuvering at Low Altitudes, and going into Ground Mode and Runup Mode in flight.

5.4 Results

This section will discuss whether any of the hazardous states identified in Section 5.2 are reachable. The results of analysis showed that they were all reachable in practice. Each of these hazards will be explained in detail.

5.4.1 Loss of Lift in Flight

It was found that the Engineering GUI allows the human operator to command either Ground Mode or Runup Mode while the vehicle is still in flight. There are a number of initial conditions that will allow the system to transition into the hazardous states. The dynamics of the state transition is found on page 144, but is illustrated in Figure 5.6

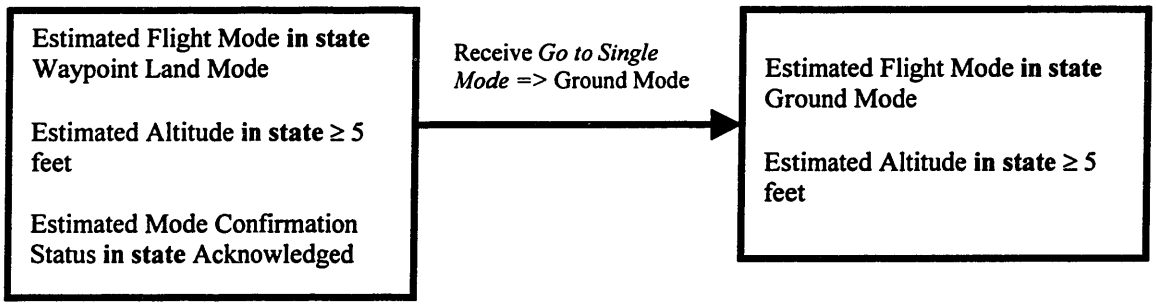


Figure 5.6 Hazardous Transition into Ground Mode While in Flight

This figure shows that if the vehicle is in Waypoint Land Mode, it is possible for it to transition into Ground Mode. The transition is triggered by the human operator who must issue the *Go to Single Mode* command for the vehicle to go into Ground Mode.

Described physically, the vehicle heads towards the landing site at an altitude of 120 feet in the air. The waypoint land command has obviously been acknowledged in this case, so the Mode Confirmation Status condition has also been met. The remaining event, where the human operator sends a command to the FMS to signal the vehicle into

Ground Mode can occur due to carelessness or by accident. Therefore, the flight mode transition can occur, so the hazard is realizable. In most cases, the Engineering GUI is operated by an experienced operator, who understands the behavior of the system. This event is highly improbable to occur, but possible.

Runup Mode presents the same conditions except the vehicle must be taking off in order for the command to be processed. Thus, a crash is imminent when the command occurs when the vehicle is at its standard takeoff altitude of 50 feet. Figure 5.7 depicts a possible state transition resulting in this hazardous state.

It should be noted that these features were intentionally introduced to allow for easy access to Ground Mode and Runup Mode during flight-testing. This allows the human operator to quickly land the vehicle when it is thought to be safe without having to wait for the automated landing conditions to be satisfied. Runup Mode gives the operator the ability to abort a takeoff. For experienced operators these would not be considered system hazards, because they know enough about the system to avoid sending wrong commands. From a safety standpoint, however, this is a serious hazard, which could potentially lead to the destruction of the vehicle, especially if the GUI is operated by an untrained operator.

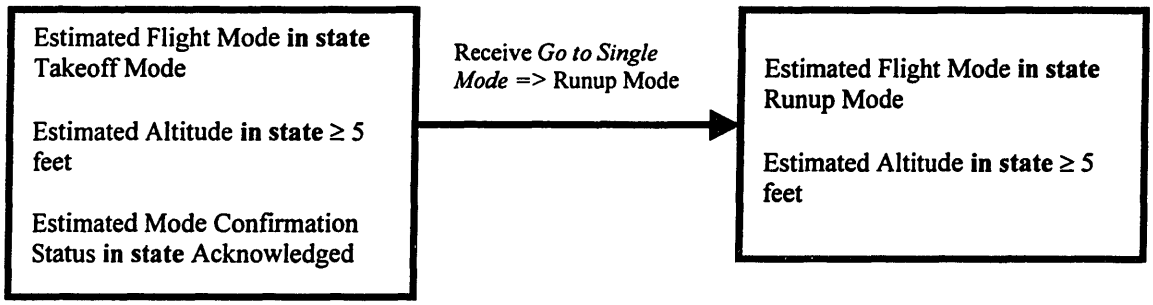


Figure 5.7 Hazardous Transition into Runup Mode While in Flight

This was one of the hazards identified through *backward analysis*, by examining the model to determine what previous state could have led to Ground Mode or Runup Mode. From the specifications, it can easily be seen that the Ground Mode can only be reached by Waypoint Land Mode and Runup Mode. Runup implies the vehicle is on the ground, so that branch was not explored. Tracing back a step with to Waypoint Land Mode showed two possibilities for transition to occur. Either the FMS checks to

see that the landing maneuver is complete before sending the Ground Mode command or the operator can send it directly via the Engineering GUI. The conditions used by the FMS prevent the vehicle from transitioning into Ground Mode in the air, so the alternative was to investigate the operator command. It was found that there were no altitude restrictions for sending this command. Thus, it was identified that this command could be sent while the vehicle is in flight.

5.4.2 Loss of Navigation Filter While in Flight

Loss of the navigation filter was a reachable hazardous state in one operational version of the FMS. This is depicted in Figure 5.8, which illustrates a possible transition into this hazardous state. Initially, no condition was required for the vehicle to be in Ground Mode before the initialize command would be issued. The hazard was identified in a question as to whether the navigation filter could be initialized in the air, during a conversation with the designer. The designer confirmed that this was possible, which led to the identification of the reachability of this hazard. Since its discovery, an additional condition has been added, a consequence of confirming the reachability of this hazard, to the specifications to ensure that the vehicle is not in the air before the *Initialize Navigation Filter* command is sent. Again, this may not constitute a hazard from the perspective of an experienced operator. However, an inexperienced operator may be tempted to hit the initialize navigation filter button when the display panel shows that the navigation reading is in a downgraded status.

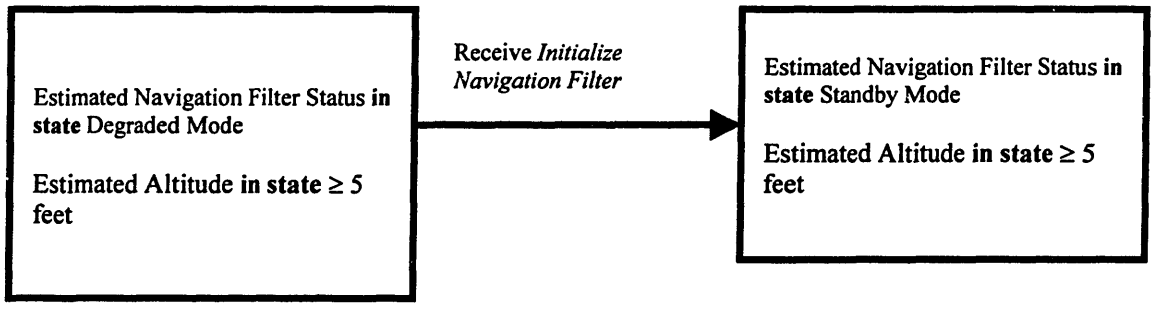


Figure 5.8 Hazardous State Transition Resulting in Loss of Navigation

5.4.3 Low Altitude Maneuvering

One of the three hazardous states for low altitude maneuvering was found to be reachable during an attempt to reorganize the specifications to make them more presentable. Concern about whether or not to include Communications Status as a state variable prompted the question as to why the FMS does have a behavior specified when communications is down. Communications going down does not inherently cause a major problem. However, with the appropriate conditions, the vehicle can transition into one of the hazardous state defined as the vehicle being in Pilot Assist Mode at low altitude. An example of the thought process that led to the confirming the reachability of this hazardous state involved mental applications of both forward and backward analysis is given below. The first segment uses *forward analysis*, to determine the propagation of state due to the failure of the communications link.

> *What happens when communications status goes down? The vehicle continues to head towards or keeps its current waypoint, and the FMS continues to send the new command until it receives a mode confirmation message from the vehicle. The timeout counter determines how often the FMS sends the command. Until a mode confirmation is received, no other guidance commands can be sent. If communications comes back up, the new command should be received by the vehicle, and a confirmation should be sent back to the FMS.*

The second segment used backward analysis to determine whether the hazardous state is reachable from a known state.

> *How can the vehicle end up flying at a low altitude? Pilot Assist Mode allows the operator to manually command the velocity, and maybe allow the vehicle to be commanded towards the ground. This would be bad. How can we get to Pilot Assist Mode when the vehicle is close to the ground? Waypoint Land Mode is the only flight mode that commands the vehicle to head towards the ground. What happens if the vehicle is landing, and the operator decides not to land, but to go into Pilot Assist Mode instead?*
>
> *That is an allowable transition, and it is only unsafe when the vehicle is very close to the ground. How can the vehicle get close to the ground in waypoint land mode and still transition into Pilot Assist Mode? What if communications goes down exactly when the Pilot Assist Command is issued, so that the vehicle never receives it? The vehicle would continue to land. The FMS would continue to send the Pilot Assist Command. If communications is reestablished when the vehicle is below 5 feet in altitude, it will receive the command to go into Pilot Assist Mode, and execute it. The vehicle will be in Pilot Assist Mode close to the ground.*

The conditions leading to the transition into this hazardous state describes the classical problem of state confusion, where the FMS thinks the system is in a state it is

not. State confusion results in the controller (the FMS in this case) sending commands based on incorrect assumptions. Leveson states [13]:

> Hazards and accidents can result from mismatches between the software view of the process and the actual state of the process - that is, the model of the process used by the software gets out of synch with the real process. For example, the software does not think the tank is full and therefore does not stop the flow into the tank, or it does not know that the plane is on the ground and raises the landing gear.

Under nominal conditions, the FMS would not allow the state transition to occur unless the vehicle is at an altitude above 5 feet. This condition was satisfied when the Pilot Assist Command was initially sent. However, the FMS has a timeout event that triggers the command to be sent continually, at a certain time interval, until it receives a mode confirmation from the vehicle. This is done blindly without regard for the state of the vehicle.

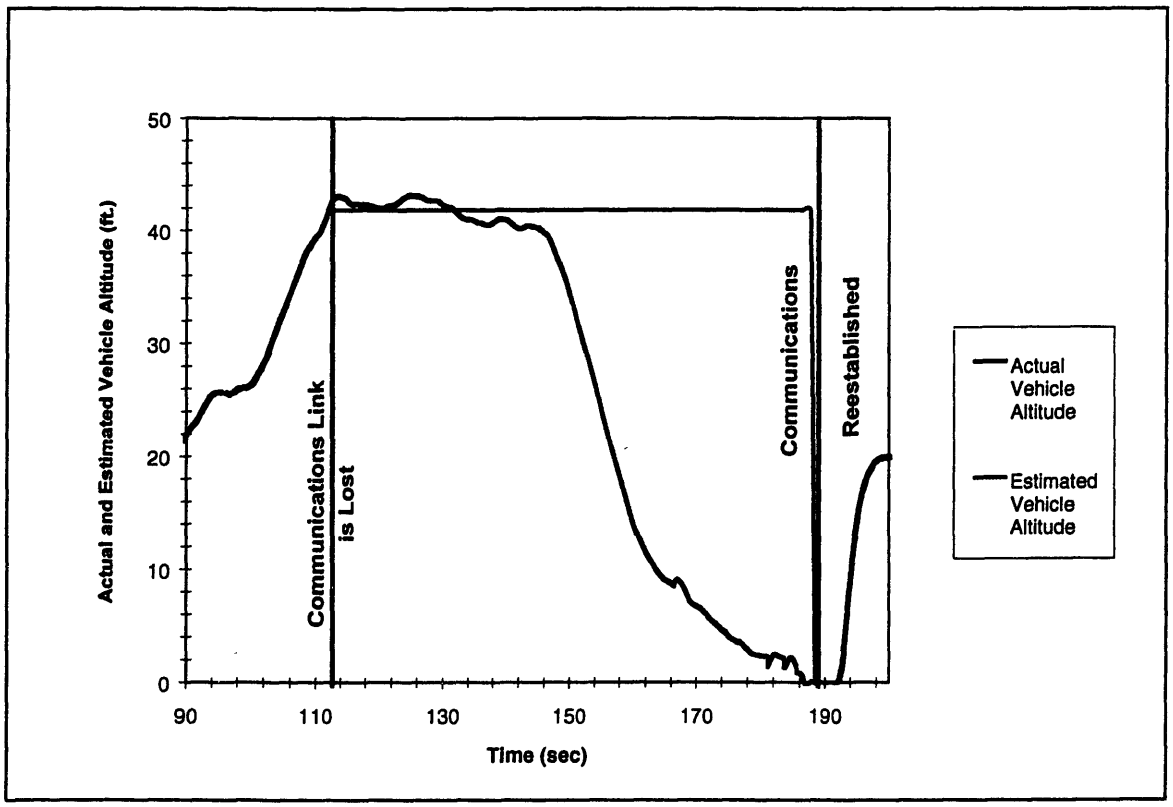


Figure 5.9 Results of a simulation to show FMS state confusion when communications goes down. The estimated and actual altitudes are out of synch, which ultimately leads to a transition into the hazardous state

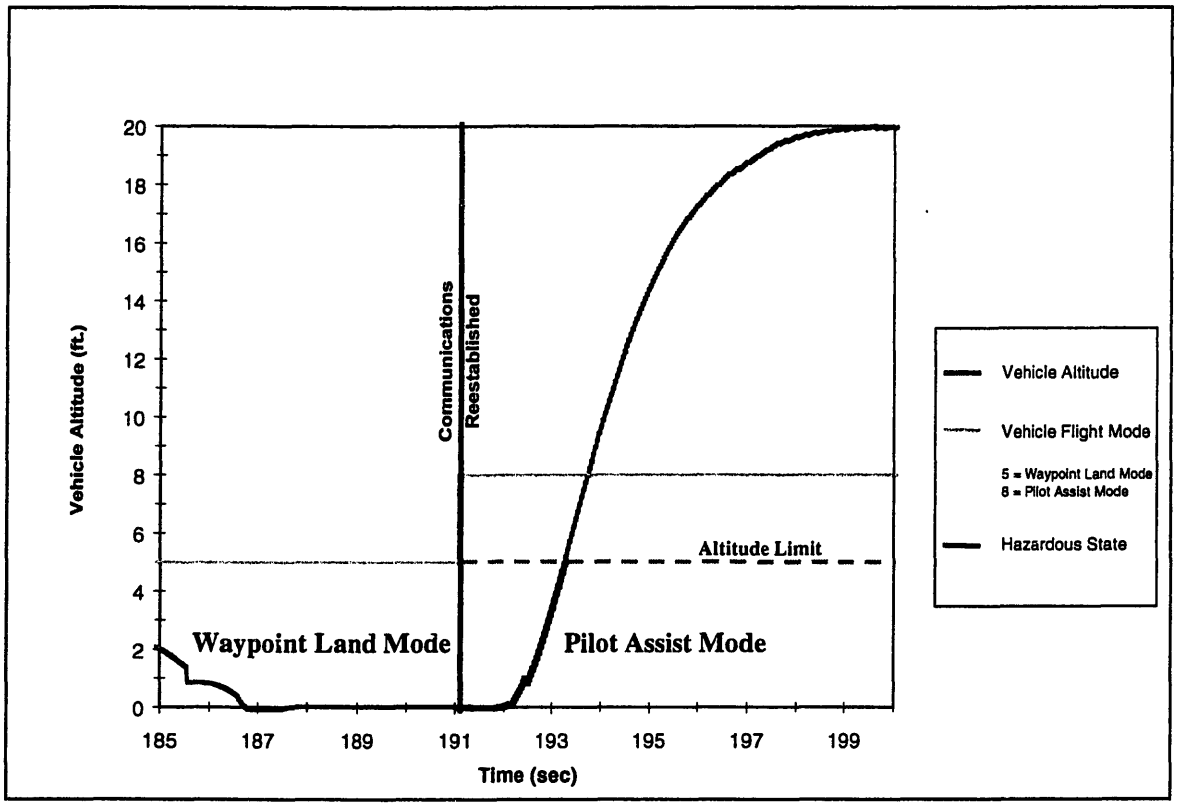


Figure 5.10 Magnified Plot of Figure 5.9 indicates the hazardous state is reachable, and is shown in red. The state transition between Waypoint land Mode and Pilot Assist Mode is possible due to fluctuating status of communications.

Figures 5.9 and 5.10 are result from using the graphical vehicle simulation. Figure 5.9 illustrates the possibility of state confusion when communications between the vehicle and the FMS is lost. This is because the FMS no longer receives state updates. As shown, before communications is lost the estimated altitude is equivalent to the actual altitude. As soon as communications is lost, the FMS maintains its altitude estimate of 42 feet, as the vehicle continues to land. During this time, it is possible for the operator to command Pilot Assist Mode. However, the vehicle will not receive this message, since communications is down.

Figure 5.10 is a magnification of Figure 5.9, and it depicts the hazardous state transition that occurs if Pilot Assist Mode is issued after communications goes down. As shown, the vehicle is already on the ground when communications returns. The FMS is continually sending the Pilot Assist command since it has not received a mode confirmation from the vehicle. The vehicle receives this new guidance command when

communications is reestablished, and the vehicle responds by executing Pilot Assist Mode while still on the ground.

This is an illustration of incompleteness in the FMS specifications, which extends all the way back to incomplete intent. The intended behavior of the system under the circumstances mentioned above were not specified, and that is why the problem exists.

The two other hazardous states describing the hazard of low altitude maneuvering were also found to be reachable. The transitions into Waypoint Hover or Waypoint Through Mode at low altitudes can be triggered by the operator, since there is no altitude restriction on the transition from Takeoff Mode into these two modes. Figure 5.11 shows the hazardous state transitions.

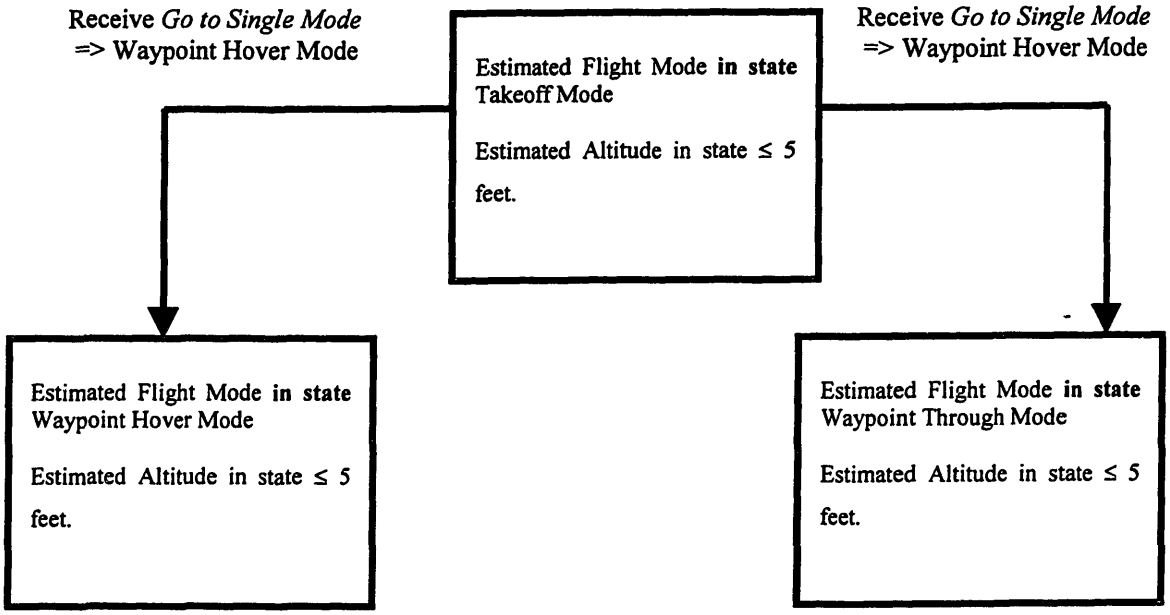


Figure 5.11 Hazardous State Transitions Resulting in Low Altitude Maneuvering

5.4.4 Injury From Rotor Blades

This specification error was not found during analysis, but during a vehicle demonstration. An overly zealous operator, who had very little prior training, executed a mission while the vehicle's engine was off. As mentioned above, the accident was prevented by an observant operator. No conditions are listed in the specifications to ensure that the vehicle's engine is on before a mission can be executed, because the

vehicle is not equipped with sensors to do so. This is an example of incompleteness in the specifications. However, in hindsight, the vehicle's engine status should have been modeled since it is an important physical parameter. The problem is that the engine and the on-board computer can be turned on separately. In this situation, the computer was turned on first, and was already in Runup Mode. Therefore, it the operator were to start the engine at that time, it would have resulted in operator injury.

5.4.5 Improvements

Some of the suggested improvements for solving these problems will be discussed. Additional conditions should be added to the GUI or the FMS to prevent a human operator from inadvertently sending the vehicle into a guaranteed disaster, as in the case with Ground and Runup Mode. The added specification could be a limitation on the altitude before issuing the *Mode Change* command.

- Estimated Altitude **in state** $\leq$ 1 foot.

This is considered a safe height from which to drop. Anything above this can cause the vehicle to roll over. Even an experienced operator would not want to send the vehicle into Ground Mode or Runup Mode above this height.

A condition that was added onto the GUI to prevent the vehicle from sending a navigation filter initialization message was:

- Estimated Flight Mode **in state** Ground Mode

This specification will work because the vehicle will most likely be on the ground in Ground Mode, so initializing the navigation filter then will not cause any control problems. In addition, the vehicle cannot transition out of Ground Mode unless the Navigation Filter has been initialized. Therefore, it is consistent with the current specifications. If the vehicle is in the air and in Ground Mode, initializing the navigation filter is not the main concern, since the vehicle will shortly drop out the sky.

Maneuvering at low altitudes, at least for the specific hazard identified, can be corrected by clearing the FMS of the current flight mode command when communications goes down. Doing this in conjunction with requesting an update of the vehicle's flight mode and sending appropriate commands in response the to reply will correct the problem.

Executing a mission when the engine is off can be prevented with the addition of a tachometer, a sensor to give a reading of the engine rpm's. There would need to be an additional specification on the engine's speed before a mission can be executed. An example specification would include this additional condition.

- Engine **in state** ON

5.5 Summary

This chapter presented some of the concepts involved in hazard analysis for the FMS specification. It was emphasized that modeling and analysis have a symbiotic relationship, so they should be performed concurrently. Hazardous states were described physically, and were identified through conversations with the designer and through experience gained from modeling. The hazardous states were manually found reachable through techniques better suited for automation. Some of the behaviors identified were not considered hazards, but merely undesirable. However, they merit further investigation.

Chapter 6

Conclusions

This chapter will conclude the thesis with comments about lessons learned during the modeling and analysis processes. It includes a discussion on the practicality of modeling requirements, such as the amount of effort involved and the difficulties encountered. Some suggestions for improving these processes will also be presented. The chapter will conclude with suggestions on topics for further investigation.

6.1 Lessons Learned

The practicality of modeling and analyzing software requirements specifications has been illustrated in this thesis by showing that the FMS safety characteristics can be improved by applying these techniques. The merit of these techniques can be confirmed in two respects.

First, it transformed the analyst, who initially had no knowledge of the autonomous system, into a system expert. It took time for the analyst to develop familiarity with the system, as well as an understanding of the modeling technique. Indeed, modeling and analysis of software requirements specifications are cumbersome and time-consuming processes, which will lengthen the overall software development time. However, by taking the time to go through the modeling process, system engineers will gain the insight necessary to be successful in performing safety analysis. This intuition will help engineers pinpoint differences between actual system behavior, and intended system behavior. Thus, the importance of modeling is not necessarily the resulting model, but the benefits gained through exercising the modeling process. Towards the conclusion of modeling the FMS, the author knew the system almost as well as the designer.

It was also found that speaking informally with the designer improved understanding of the system more effectively than reading documentation. Through

interaction, additional information about the system was obtained, which would not have been found by reviewing documentation. The point is better communication results in better requirements.

However, just a good understanding of the system may not be enough to adequately perform a hazard analysis. The model gives analysts a tool with which a systematic approach can be applied. For example, the model allowed for the formal definition of hazards in terms of states. This is particularly important in the case of large software systems with multiple components. The ability to define hazards is a necessary characteristic of the model if hazard analysis is to be automated.

The second point is that the analysis of the specifications model helped to uncover dormant hazards in the system. Although the autonomous vehicle had undergone extensive flight-testing and simulation, and had been operating with no major safety violations for 3 years, existing requirements errors, which might not have been found until too late, were still identified. These latent hazards, which hide in the system, are dangerous. When errors do not occur for a long time, it is difficult for operators to avoid assuming that the system is fault-free and completely predictable. The result can be carelessness and complacency, which lead to accidents as in the case of the Therac-25 story from Chapter 2.

6.1.1 Difficulties Encountered

The difficulties encountered during the modeling and analysis efforts were mostly related to the analyst's unfamiliarity with the modeling technique. One of the main problems was the notion of timing in the specifications. An introduction of a dummy state variable to help in the specifications would have been useful in the model. However, an effort was made not to include internal state variables in the model, which includes extraneous timing variables. The specifications for the timing of state transitions with the same triggering conditions were especially difficult to write. Finally, the option of placing notes in the comment section of the charts was used to help to indicate timing specifications.

Another problem was identifying the states of the system. The model is not unique, and therefore a large number of state variable combinations could be used to

describe the same system. The approach was to describe the model entirely using externally visible variables. There was initially some confusion on the difference between externally visible states and internal variables, which initially led to the generation of an incorrect model. The choice in states is up to the modeler. There is no guarantee that the states chosen entirely capture the behavior of the FMS, so state completeness can be a problem when building the model. On the other hand, if too many states are chosen, the modeling and analysis processes can become intractable.

Hazard identification was another source of difficulty. There are no strict rules for determining the hazards in a system, so a great deal depends on the experience of the domain experts. The definition of hazardous states was derived from modeling the requirements, others resulted from brainstorming, and still others resulted from events that actually took place during field-testing. It is recommended that hazard identification be done as a group effort. There will be more breadth in the ideas generated, as opposed to doing it on an individual basis where there is a tendency to explore a single hazard in depth before moving on to the next one.

6.1.2 Applicability

Although the modeling technique was applied with success on the FMS, a few qualifications should be stated. In general, state-based modeling approach can be used for any software system. From consensus however, it is believed that the model works best when used with systems that involve discrete decisions as opposed to numerical calculations. It would not be the ideal choice to use as a model for the control algorithms on-board the vehicle. Control algorithms base their behavior on a very fine grating of numerical values, almost akin to continuous values. In this case, the number of states using the same type of model would be enormous. If it were really desired to use to state-based modeling approach however, the recommendation would be to introduce the results of the control calculations as external inputs into the discrete model.

Recall that in Chapter 2 it was stated that analyzing discrete systems is more difficult than continuous systems. This is an illustration of such a case. The control system dynamics can be modeled as equations, where mathematical theory has been developed specifically for their analysis. The main message is that a modeling notation

or technique should be chosen such that it is appropriate for the particular system of interest. Certain models work better with certain types of systems.

In addition, it should be noted that an abstract systems model might not properly reflect the real world. For example, incorrect assumptions and/or inaccurate information about the system can invalidate the model. This means there is no guarantee that the model will say anything about the properties of a software system.

Finally, human ability and knowledge are crucial factors to the success of modeling and analysis of software specifications. The better the analyst understand the system, the better the specifications will be. In a case study by Lubars, it was found that the most successful requirements developers are those that show intense conviction and passion about their view of the system. They live and breathe the system. One person even commented that he had dreams about his system at night [15].

6.2 Future Work

The research performed here does not guarantee that all of the hazards have been eliminated from the system, although this exercise is a step in the right direction.

Further research in analyzing state-based models should include a development of automated tools to help determine whether state-based software specifications are consistent and complete. Leveson is currently developing SpecTRM (Specifications Tools and Requirement Methodology), a software application suite, explicitly designed for use with state-based specifications [12]. This will not only perform consistency and completeness checks, which are lacking for the FMS, but it can also perform a variety of other analyses as well as execute the specifications.

Automated hazard analysis would also improve the process. Some independent ideas generated during research to implement automated analysis of hazards include mixed integer programming [6] and discrete event simulation.

Mixed integer programming (MIP) is a well-established method of performing system optimization based on a certain objective function. It is conceivable that the hazard identification process can be defined in terms of an optimization problem. It may be possible to express the hazardous state as a well-defined linear objective function, and have the state transitions be the constraints on the system. If it can be characterized in

such a manner, there are well-established tools to solve the optimization problem. This is one remedy to the state explosion problem encountered when performing hazard reachability by hand. MIP should be practical even for searching a very large state space.

Another possibility is to develop a discrete event simulator in an environment such as the Matlab Stateflow toolbox. This would essentially be an attempt to find hazards by enumerating all possible states. The simulation can be programmed in such a way as to halt if a hazardous state has been reached. The state transitions can be recorded during the simulation so the sequence of events leading to the hazardous state can be traced.

Additional developments to improve the modeling and analysis processes should include techniques in requirements elicitation. This will help to reduce the number of errors in the translation process from the customer to the designer. General guidelines for determining hazardous states should also be developed.

More work can be done to improve some of the documents generated during this study. Based on the result of the review, it may be necessary to rethink the presentation of the specifications. For example, the natural language specifications as presented in Appendix A would benefit from reorganization. This document was written to illustrate how high level requirements can be translated into more detailed specifications. Indeed, to complete the entire requirements verification and validation process, a completed set of specifications would need to be generated. This would not just include software specifications, but also hardware and system performance specifications. Additional documentation would include intermediate levels of specifications, in between natural language specifications, and the state-space model, as well as additional levels between the state-space model and the code. Each level would be a refinement of the previous level.

A suggested experiment to determine whether the state-based specifications are more reviewable than code is to have two experimental groups. One group is given just the code, while the other is given the specifications model. Each group is asked to find system hazards from those two views of the FMS. The experiment can compare the two documents to determine which is more readable, and which helps in the identification of requirements errors and system hazards.

To completely perform safety analysis on the system, the other software components should also be included in the system model. This includes modeling the behavior of the flight controller, as well as adding more details to the model of the GUI. As the model become more complex, it may uncover more hazards. This is because system level hazards differ from component level hazards. In component level hazards, assumptions have been made about the other subsystems interacting with it, and they are not modeled specifically. However, by including in the model the interaction between the subsystems, a whole new level of complexity is introduced, and this may result in hazards that would not be found by just looking at the affect of one component. That is why the whole system must be modeled.

Finally, many changes have occurred throughout the development of the specifications. It was decided that for the benefit of research, only one version of the software system would be modeled. To truly capture the system's behavior, the model must be updated. For example, the idea of adding new flight modes for aggressive maneuvers has been discussed. The problem is where to fit these modes into the system, and whether they will compromise system safety. These new modes can be incorporated into the current model of the software specifications, and then analyzed again for hazards.

References

[1] Bryan, William L., Siegel, Stanley G. *Software Product Assurance: Techniques for Reducing Software Risk*, Elsevier, New York, 1987.

[2] Friedland, Bernard. *Control System Design: An Introduction to State-Space Methods*, McGraw-Hill, Inc., New York, NY, 1986.

[3] Goguen, Joseph A., Linde, C. Techniques for Requirements Elicitation. In: *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, San Diego, CA, March 1993.

[4] Hamlet, Richard. Testing for Trustworthiness. In J.P. Jacky and D. Schuler, editors, *Directions and Implications of Advanced Computing*, Ablex Publishing Company, 1989.

[5] Harel, David. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 1987.

[6] Hillier, Frederick S., Lieberman, G.J. *Introduction to Operations Research*, McGraw- Hill, Inc., New York, NY, 1995.

[7] Jaffe, Matthew S. *Completeness, Robustness, and Safety of Real-Time Requirements Specification.* PhD thesis, University of California, Irvine, CA, 1988.

[8] Leveson, N.G. Intent Specifications: An Approach to Building Human-Centered Specifications. *IEEE Transactions on Software Engineering*, to appear in 1999.

[9] Leveson, Nancy G. High-Pressure Steam Engines and Computer Software. In: *Proceedings of the International Conference on Software Engineering*, Melbourne, Australia, May 1992.

[10] Leveson, Nancy G., Heimdahl, M., Hildreth, H., Reese, J. Requirements Specification for Process-Control Systems. *IEEE Transactions Of Software Engineering*, March 1994.

[11] Leveson, Nancy G., Heimdahl, M., Hildreth, H., Reese, J., Ortega, R. *TCAS II System Requirements Specification*, Draft, December 1992.

[12] Leveson, Nancy G., Reese, J., Heimdahl, M. SpectTrM: A CAD System for Digital Automation. *Digital Avionics Conference*, October 1998.

[13] Leveson, Nancy. *Safeware: System Safety and Computers,* Addison-Wesley Publishing Company, Inc., Reading, MA, 1995.

[14] Lions, J.L. Ariane 5 Flight 501 Failure Report, Paris, July 1996.

[15] Lubars, Mitch, Potts C., Richter, C. A Review of the State of the Practice in Requirements Modeling. In: *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, San Diego, CA, March 1993.

[16] Lutz, Robyn R. Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. In: *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, San Diego, CA, March 1993.

[17] Rosen, Kenneth H. *Discrete Mathematics and Its Applications*, McGraw-Hill, Inc., New York, NY, 1994.

[18] Sanders, Christopher P. *Real-time Avoidance for Autonomous Air Vehicles.* SM thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, January 1998.

[19] Sanders, Christopher P., DiBitteto, Feron, E., Vuong, H., Leveson, N. A Hierarchical Control of Small Autonomous Helicopters. *37th IEEE Conference on Decision and Control*, Tampa, FL, December 1998.

[20] Wingate, Dr. Guy, Smith, M., Lucas, P. Assuring Confidence in Pharmaceutical Software. In: *Proceedings of the Twelfth Annual CSR Workshop on Safety and Reliability of Software Based Systems*, Bruges, Belgium, September 1995.

Appendix A Supplementa l Documentation

This appendix contains an example of supplemental documents supplied to reviewers to aid in the understanding of the FMS as a whole because the original state-base specifications were insufficient for providing this information. The first document is an example of a natural language specification for the FMS. This document begins with the definition of system goals, and continues with high level requirements. The requirements are then decomposed further until they express the same specifications in natural language as in the state-based model. Hyperlinks were added to the electronic version of the natural language requirements document so reviewers can easily locate the appropriate tables pertaining to a specification.

The second piece of supplemental information is a flowchart, which indicates the overall timing of the state transitions listed in the state-based specifications. Although incomplete, it was found helpful to reviewers for deciding the ordering of the state transitions.

A.1 Example Natural Language Software Requirements Specifications for the FMS

The following is the set of natural language specifications generated for the FMS. Definitions of terms that are indicated in bold throughout the specifications may be found at the end of Appendix A.1.

Flight Management System Functional Goals

The goal of this system is to provide a robust autonomous mission management capability for a small autonomous helicopter.

1) Provide battlefield ready system for tactical reconnaissance.
2) Provide a system with easy to use graphical user interface, allowing for effective operation with minimal training.
3) Provide manual mission planning and monitoring capabilities.
4) Provide fully autonomous mission execution.

High-Level Functional Requirements (Constraints)

1) The FMS shall be a ground based software application capable of being run from a laptop computer.
   a) Commands to the FMS by the GCU Operator will be made via the mouse pointer, and the laptop keyboard.
   b) The FMS shall be the only communications channel between the GCU Operator and the vehicle.
   c) The FMS shall be able to interpret vehicle input data as useful information.
2) The FMS shall provide the capability to autonomously execute a planned mission for the vehicle.
   a) A mission shall be composed of the vehicle following a flight trajectory defined by waypoints.
      i) *Waypoints* are defined as points in space (x, y, z) to which the vehicle is commanded to fly.
      ii) Waypoints shall be specified according to spatial coordinates and flight mode.
         (1) A *flight mode* defines the physical behavior of the vehicle.
         (2) Waypoints can take on the attributes of a limited number of flight modes (Hover, Through, and Land).
         (3) The remaining flight modes are necessary to facilitate the transition of the physical state of the vehicle to the desired waypoint.
            (a) There shall be 7 flight modes that can be commanded by the FMS. This is consistent with the on-board guidance system of the vehicle.
               (i) The first flight mode shall be *Ground Mode.*
                  1. Ground Mode allows the GCU Operator or the FMS to throttle down the engine and set the collective pitch to an appropriately low value, resulting in the vehicle not producing any appreciable lift. This mode will mainly be used to command the vehicle to remain on the ground, at the both beginning and end of a mission.
                  2. The FMS shall assume the vehicle is in Ground Mode when the FMS is first started up. This is verified via communication with the vehicle.
               (ii) The next flight mode shall be *Runup Mode.* Runup Mode allows the GCU Operator, or the FMS to ramp-up the throttle setting, but maintaining the low collective setting, so the vehicle can achieve takeoff Rpm's without actually taking off.
               (iii) The next flight mode shall be *Takeoff Mode.*
                  1. Takeoff Mode commands the vehicle to gradually increase the collective pitch and throttle so that enough lift is created for the vehicle to be lifted off the ground.
                  2. The vehicle shall continue to ascend in Takeoff Mode until a pre-specified altitude has been reached.

3. The vehicle shall hover at that point for a pre-specified amount of time.

(iv) The next flight mode shall be *Waypoint Hover Mode.*

1. Waypoint Hover Mode shall command the vehicle to stop, and hover at a commanded point. This shall provide the vehicle-mounted camera with a stable platform for reconnaissance.
2. Waypoint Hover Mode shall be a valid flight mode to be used as a waypoint on the Task List.
3. Waypoint Hover Mode shall include a parameter known as Hover Time, which commands the period of time the vehicle should hover at the commanded waypoint.

(v) The next flight mode shall be *Waypoint Through Mode.*

1. Waypoint Through Mode shall command the vehicle to fly through the commanded waypoint, on the way to the next waypoint.
2. Waypoint Through shall be a valid flight mode to be used as a waypoint on the Task List.

(vi) The next flight mode is *Waypoint Land Mode.*

1. Waypoint Land Mode shall command the vehicle to fly to the location specified for landing and descend.
2. Waypoint Land Mode shall be a valid flight mode to be used as a waypoint on the Task List.
3. Waypoint Land shall provide a safe method of returning the vehicle to the ground.

(vii) The final flight mode is *Pilot Assist Mode.*

1. Pilot Assist Mode shall provide the GCU Operator with an addition degree of freedom to help in reconnaissance. Pilot Assist allows the operator to command the velocities of the vehicle, and thus the position, in the x- and y-directions, up and down, and in heading. This shall allow the camera to obtain better views if the vehicle is oriented poorly with respect to the target.
2. Pilot Assist Mode shall be the only single flight mode directly accessible through the Operational GUI.

b) The FMS shall have Three Flight Manager Modes.

i) Guidance Modes shall be used to determine how the FMS will command the vehicle during flight. The first Guidance Mode shall allow for fully autonomous mission execution and shall direct the transitions between flight modes. The second Guidance Mode shall allow for manual mission execution by the GCU Operator, with less restriction on the flight mode transitions. The last mode shall be used as a method to return the vehicle to a secure location quickly during an emergency.

ii) **Task List Mode** shall provide the FMS with the capability to execute preplanned autonomous missions.

(1) Missions shall be implemented in the form of ordered waypoints on the **Task List.**

(a) The **Task List** shall only allow waypoint entries of the flight mode type (Waypoint Hover, Waypoint Through, and Waypoint Land), although it will still be necessary to execute the other flight modes to achieve the waypoint on the list.

(b) Initially, there shall be no waypoints on the Task List, and the **Total Number of Tasks** defaults to zero.

(c) The **Total Number of Tasks** shall be incremented by one if the FMS receives **"Create a New Waypoint => Entry Complete"** from the GCU Operator. State Transitions2.doc - NumTasks Table 1/Col. 1

(d) The most recent tasks entered on the **Task List** shall be listed at the end of the list. State Transitions2.doc - Tasklist Table 1/Col. 1

(e) The **Total Number of Tasks** shall be reset to zero if FMS receives **"Delete Waypoints"** from the GCU Operator. State Transitions2.doc - NumTasks Table 2/Col. 1

(f) The **Total Number of Tasks** shall be reset to zero if FMS receives **"Quit"** from the GCU Operator. State Transitions2.doc - NumTasks Table 2/Col. 2

(g) All waypoints on the **Task List** shall be removed if the FMS receives **"Delete Waypoints"** from the GCU Operator. State Transitions2.doc - Tasklist Table 2/Col. 2

(h) All waypoints on the **Task List** shall be removed if the FMS **"Quit"** from the GCU Operator. State Transitions2.doc - Tasklist Table 2/Col. 1

(i) The **Current Task Index** shall indicate the current waypoint on the **Task List** being executed.

(j) The **Current Task Index** shall be incremented by one if the current waypoint on the **Task List** has been achieved, and there are more tasks remaining. State Transitions2.doc - Tasklist Table 3/Col. 1

(k) Entering a new task onto an empty **Task List** in **Task List Mode** shall result in the task being automatically executed if the currently executing task has been achieved. State Transitions2.doc - Tasklist Table 3/Col. 2

(l) The **Current Task Index** shall be incremented by one if the current **Task List** waypoint being executed is Waypoint Land, and the waypoint has been achieved. This is to ensure that the vehicle will not execute Waypoint Land again if **Task List Mode** is commanded again. State Transitions2.doc - Tasklist Table 3/Col. 3

(2) **Task List Mode** shall possess the capability to determine whether a waypoint has been achieved. The next waypoint on the list will not be sent to the vehicle until the currently executing waypoint is complete.

(a) The FMS shall consider Ground Mode complete only if the Navigation Filter is in state HEALTHY and indicates that the vehicle is on the ground and not moving. State Transitions2.doc - WptAchvd Table 1/Col. 1. This indicates that the system has been initialized, and ready for flight.

(b) The FMS shall consider Runup Mode to be complete only if the Navigation Filter is in state HEALTHY, and the vehicle is on the ground and not moving, and the vehicle has been in Runup for more than 40 seconds. State Transitions2.doc - WptAchvd Table 2/Col. 1. This allows the main rotor enough time to achieve takeoff Rpm's.

(c) The FMS shall consider Takeoff Mode to be complete only if the vehicle is within the **Radius of Capture** of the default Takeoff point and has been hovering within that radius for a pre-specified amount of time. State Transitions2.doc - WptAchvd Table 3/Col. 1

(d) The FMS shall consider Waypoint Hover Mode to be complete only if the vehicle is within the **Radius of Capture** of the commanded hover waypoint and has been hovering within that radius for the specified hover time. State Transitions2.doc - WptAchvd Table 4/Col. 1

(e) The FMS shall consider Waypoint Through Mode to be complete only if the vehicle is within the **Radius of Capture** of the commanded waypoint. In this case, the Radius of Capture is defined as a cylinder surrounding the commanded waypoint. The vehicle is considered to have reached the waypoint if it is within the cylinder. State Transitions2.doc - WptAchvd2 Table 1/Col. 1 The reason for the cylinder is that since this is a through waypoint, it would be physically demanding for the vehicle to attempt flying exactly through the commanded points especially if the altitude between them differ significantly.

(f) The FMS shall consider Waypoint Land Mode to be complete only if the vehicle is below 1ft in altitude and has velocities less than 0.1 ft/s. State Transitions2.doc - WptAchvd2 Table 2/Col. 1

(3) When the FMS is in **Task List Mode**, the FMS shall command the necessary flight modes to achieve the tasks on the list. The following will determine the type of flight mode sent by the FMS when it is already in Task List Mode, and the currently executing flight mode has been completed.

(a) If the currently executing waypoint is Waypoint Land Mode, and the vehicle has achieved landing, then the FMS shall send a **"Mode Change Request =>Ground Mode"** to the vehicle. State Transitions2.doc - InTL Table 1/Col. 1

(b) If the currently executing flight mode is Ground Mode, and the waypoint is completed, then the FMS shall send a **"Mode Change Request =>Runup Mode"** to the vehicle. State Transitions2.doc - InTL Table 2/Col. 1

(c) If the currently executing flight mode is Runup Mode, and the waypoint is completed, then the FMS shall send a **"Mode Change Request =>Takeoff Mode"** to the vehicle. State Transitions2.doc - InTL2 Table 1/Col. 1

(d) If the currently executing flight mode is Takeoff/Hover/Through Mode, and the waypoint is completed, and the next waypoint has Waypoint Hover Mode as the flight mode, the FMS shall send a **"Mode Change Request =>Waypoint Hover Mode"** to the vehicle. State Transitions2.doc - InTL2 Table 2/Col. 1, State Transitions2.doc - InTL2 Table 2/Col. 2, State Transitions2.doc - InTL2 Table 2/Col. 3

(e) If the currently executing flight mode is Takeoff/Hover/Through Mode, and the waypoint is completed, and the next waypoint has Waypoint Through Mode as the flight mode, the FMS shall send a **"Mode Change Request =>Waypoint Through Mode"** to the vehicle. State Transitions2.doc - InTL3 Table 1/Col. 1, State Transitions2.doc - InTL3 Table 1/Col. 2, State Transitions2.doc - InTL3 Table 1/Col. 3

(f) If the currently executing flight mode is Takeoff/Hover/Through Mode, and the waypoint is completed, and the next waypoint has Waypoint Land Mode as the flight mode, the FMS shall send a **"Mode Change Request =>Waypoint Land Mode"** to the vehicle. State Transitions2.doc - InTL3 Table 2/Col. 1, State Transitions2.doc - InTL3 Table 2/Col. 2, State Transitions2.doc - InTL3 Table 2/Col. 3

iii) Single Mode: This Guidance Mode provides the GCU Operator with the capability to command individual flight modes subject to some constraints on the mode transitions. Single Mode shall allow the GCU Operator to command all seven flight modes for manual mission execution.

(1) Single Mode execution is necessary to allow for more flexibility during flight-testing.

(2) Flexibility shall not override the need for system safety.

(a) The system must not be allowed to go into Ground Mode or Runup Mode when executing Waypoint Hover Mode or Waypoint Through Mode.

(b) The system must not be allowed to go into Ground Mode when executing a takeoff flight mode.

(3) Single Mode execution shall not be made available in the primary GUI of the FMS. Use of single flight modes requires extensive knowledge of the FMS, and thus will not be available in the Operational GUI, whose main purpose is to make the system easy to use.

(4) Single Mode shall be the default Guidance Mode.

iv) Return Home Mode: This mode shall result in the FMS automatically commanding the flight mode transitions necessary for the vehicle to return to a pre-specified home location as quickly as possible.

(1) The Return Home location shall be the initial point of Takeoff.

(2) The Return Home location shall not be available until the vehicle has taken off. This is to avoid the having the vehicle go into Return Home Mode while at the home location.

(3) **Return Home Mode** shall possess the capability to determine whether a waypoint has been achieved. The next waypoint on the list will not be sent to the vehicle until the currently executing waypoint is complete. (Same as Task List Mode)

(4) When the FMS is in **Return Home Mode**, the FMS shall command the necessary flight modes to achieve the tasks on the list. The following will determine the type of flight mode sent by the FMS when it is already in Return Home Mode, and the currently executing flight mode has been completed.

(a) If the currently executing waypoint is Waypoint Land Mode, and the vehicle has achieved landing, then the FMS shall send a **"Mode Change Request =>Ground Mode"** to the vehicle. State Transitions2.doc - InRH Table 1/Col. 1

(b) If the currently executing flight mode is Runup Mode, and the waypoint is completed, then the FMS shall send a **"Mode Change Request =>Takeoff Mode"** to the vehicle. State Transitions2.doc - InRH Table 2/Col. 1

(c) If the currently executing flight mode is Takeoff Mode, and the waypoint is completed the FMS shall send a **"Mode Change Request =>Waypoint Land Mode"** to the vehicle. State Transitions2.doc - InRH Table 3/Col. 1

v) The FMS shall provide the capability to seamlessly transition from one Guidance Mode to another.

(1) The FMS shall transition to **Single Mode** if the FMS completes a Waypoint Land Mode in **Task List Mode.** State Transitions2.doc - GuidMode2 Table 2/Col. 2

(2) The FMS shall transition to **Single Mode** if the FMS completes a Waypoint Land Mode in **Return Home Mode.** State Transitions2.doc - GuidMode2 Table 2/Col. 1

(3) The FMS must not be allowed to transition from **Task List Mode** to **Task List Mode.**

(4) Switching Into Task List Mode

(a) Upon receipt of a **"Change to Task List Mode"** from the GCU Operator via the Engineering GUI, the FMS shall determine whether the previous command has been received.

(i) If the vehicle has acknowledged the previous command and there are tasks on the **Task List,** the FMS shall switch into **Task List Mode** from another **Guidance Mode** if the previous waypoint command was Waypoint Land Mode and the vehicle is estimated to be 5ft above ground level. State Transitions2.doc - GuidMode Col. 3

(ii) If the vehicle has acknowledged the previous command and the previous task was not waypoint land, the FMS shall switch into **Task List Mode** from another **Guidance Mode** if there are tasks on the **Task List.** State Transitions2.doc - GuidMode1 Col. 4

(b) Upon receipt of a **"Execute Plan"** from the GCU Operator via the Operational GUI, the FMS shall determine whether or not the previous command has been received.

(i) If the vehicle has acknowledged the previous command and there are tasks on the **Task List,** the FMS shall switch into **Task List Mode** from another **Guidance Mode** if the previous waypoint command was Waypoint Land Mode and the vehicle is estimated to be 5ft above ground level. State Transitions2.doc - GuidMode Col. 1

(ii) If the vehicle has acknowledged the previous command and the previous task was not waypoint land, the FMS shall switch into **Task List Mode** from another **Guidance Mode** if there are tasks on the **Task List.** State Transitions2.doc - GuidMode1 Col. 2

(5) The FMS must not be allowed to transition from **Return Home Mode** to **Return Home Mode.**

(6) Switching Into Return Home Mode

(a) Upon receipt of a **"Return Home"** from the GCU Operator via the Operational GUI, the FMS shall determine whether the previous command has been received.

(i) If the vehicle has acknowledged the previous command, and it was not waypoint land, the FMS shall switch to **Return Home Mode** from another **Guidance Mode.** State Transitions2.doc - GuidMode2 Table 1/Col. 2

(ii) If the vehicle has acknowledged the previous command and the previous task was waypoint land, the FMS shall switch into **Return Home Mode** from another **Guidance Mode** only if the estimated vehicle altitude is 5ft above ground level. State Transitions2.doc - GuidMode2 Table 1/Col. 1

(7) Switching Into Single Mode

(a) Upon receipt of a **"Change to Single Mode"** message from the GCU Operator via the Engineering GUI, the FMS shall determine whether or not the previous command has been received. If the previous command was received:

(i) And if **Estimated Flight Mode** is Runup Mode, the FMS shall switch into Single Mode only if the currently commanded flight mode is Ground Mode. State Transitions2.doc - GuidMode2 Table 3/Col. 1

(ii) And if **Estimated Flight Mode** is Waypoint Land Mode, the FMS shall switch into Single Mode only if the currently commanded flight mode is Ground Mode. State Transitions2.doc - GuidMode2 Table 3/Col. 2

(iii) And if **Estimated Flight Mode** is Runup Mode, the FMS shall switch into Single Mode only if the currently commanded flight mode is Takeoff Mode. State Transitions2.doc - GuidMode2 Table 3/Col. 3

(iv) And if **Estimated Flight Mode** is Ground Mode, the FMS shall switch into Single Mode only if the currently commanded flight mode is Runup Mode. State Transitions2.doc - GuidMode3 Table 1/Col. 1

(v) And if **Estimated Flight Mode** is Takeoff Mode, the FMS shall switch into Single Mode only if the currently commanded flight mode is Runup Mode. State Transitions2.doc - GuidMode3 Table 1/Col. 2

(vi) And if **Estimated Flight Mode** is not Waypoint Land or Ground or Runup Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Waypoint Hover Mode. State Transitions2.doc - GuidMode3 Table 2/Col. 1

(vii) And if **Estimated Flight Mode** is not Waypoint Land or Ground or Runup Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Waypoint Through Mode. State Transitions2.doc - GuidMode3 Table 2/Col. 2

(viii) And if **Estimated Flight Mode** is not Waypoint Land or Ground or Runup Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Waypoint Land Mode. State Transitions2.doc - GuidMode3 Table 2/Col. 3

(ix) And if **Estimated Flight Mode** is Waypoint Hover Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Pilot Assist Mode. State Transitions2.doc - GuidMode4 Table 1/Col. 1

(x) And if **Estimated Flight Mode** is Waypoint Through Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Pilot Assist Mode. State Transitions2.doc - GuidMode4 Table 1/Col. 2

(xi) And if **Estimated Flight Mode** is Waypoint Land Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Pilot Assist Mode and the vehicle is estimated to be above 5ft above ground level. State Transitions2.doc - GuidMode4 Table 1/Col. 3

(b) Upon receipt of a **"Pilot Assist"** message from the GCU Operator via the Operational GUI, the FMS shall determine whether or not the previous command has been received. **"Pilot Assist"** shall result in **"Change To Single Mode =>Flight Mode"** to contain Pilot Assist Mode as the commanded mode. If the previous command was received:

(i) And if **Estimated Flight Mode** is Waypoint Hover Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Pilot Assist Mode. State Transitions2.doc - GuidMode4 Table 2/Col. 1

(ii) And if **Estimated Flight Mode** is Waypoint Through Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Pilot Assist Mode. State Transitions2.doc - GuidMode4 Table 2/Col. 2

(iii) And if **Estimated Flight Mode** is Waypoint Land Mode, the FMS shall switch into Single Mode if the currently commanded flight mode is Pilot Assist Mode. State Transitions2.doc - GuidMode4 Table 2/Col. 3

c) The FMS shall determine waypoint commands based on 3 main parameters: GCU Operator commands, vehicle flight mode, and FMS Guidance Modes.

i) The FMS shall make use of GCU Operator commands to generate waypoint commands.

(1) **Guidance Initialization Status (GIS)**

(a) **GIS** in state NOT READY shall indicate that FMS is in *Startup*. **GIS** shall default in state NOT READY.

(i) *Startup* shall be defined as the initial state of the FMS when it has just started running.

(ii) Until **GIS** is in state READY, the FMS shall not acknowledge GCU Operator waypoint commands.

(iii) At *Startup*, FMS is assumed to have no information about the current flight mode of the vehicle.

(b) **GIS** shall transition from state NOT READY to READY whenever a **"Mode Confirm Message"** is received from the vehicle. State Transitions2.doc - GuideInit Col. 1

(c) **GIS** in state READY shall indicate that FMS is in *Normal Operating Mode.*

(d) **GIS** shall be the last state variable to transition out of *Startup Mode.* Otherwise all other transitions that depend on the system being in *Startup Mode* will not occur.

(2) **Estimated Mode Confirmation Status (EMCS)**

(a) **EMCS** must be tracked to determine whether the vehicle has received either a **"Mode Change Request"** or a **"Mode Request Message"** issued by the FMS.

(i) **EMCS** in state UNACKNOWLEDGED shall indicate the vehicle has not received the **"Mode Change Request"** or the **"Mode Request Message"**. **EMCS** defaults in state UNACKNOWLEDGED.

(ii) **EMCS** in state ACKNOWLEDGED shall indicate the vehicle has confirmed receipt of the **"Mode Change Request"** or the **"Mode Request Message"**.

(iii) The FMS shall issue no waypoint commands or allow the GCU Operator to command new waypoints until it obtains the most recent vehicle flight mode. Thus **EMCS** must be in state ACKNOWLEDGED before a new waypoint command can be sent.

(b) **EMCS** must reset to state UNACKNOWLEDGED just before issuing a **"Mode Change Request "** to the vehicle. State Transitions2.doc - Unacknowleged

(c) At *Startup*, **EMCS** shall transition from UNACKNOWLEDGED to ACKNOWLEDGED if the FMS receives a **"Mode Confirmation Message"**. State Transitions2.doc - ConfirmStatus Col. 2

(d) In *Normal Operation*, **EMCS** shall transition from UNACKNOWLEDGED to ACKNOWLEDGED only if the FMS receives a **"Mode Confirmation Message"** and the vehicle flight mode returned is the same as the one commanded. State Transitions2.doc - ConfirmStatus Col. 1

ii) The FMS shall make use of the estimated state of the vehicle including the flight mode to generate waypoint commands.

(1) The FMS must always have an accurate estimate of the current flight mode of the vehicle.

(a) At *Startup*, the FMS shall assume that the **Estimated Flight Mode** is Ground Mode until a **"Mode Confirm Message"** has been received from the vehicle with a flight mode update.

(b) Startup

(i) At *Startup*, the FMS automatically sends a **"Mode Request Message"** to the vehicle. State Transitions2.doc - ModeRequest Col. 2

(ii) At *Startup*, the FMS must reissue the **"Mode Request Message"** in the event of a timeout of the **"Mode Confirm Message"**. State Transitions2.doc - ModeRequest Col. 1

(iii) At *Startup*, a **"Mode Confirm Message"** shall result in the update of the **Estimated Flight Mode** from Ground Mode to the flight mode received from the vehicle. State Transitions2.doc - EstMode Table 2/Col. 1

(c) *Normal Operation*

(i) During *Normal Operation*, a receipt of a **"Mode Confirm Message"** shall result in the update of the **Estimated Flight Mode** to the most recent **"Mode Change Request =>Flight Mode"** if and only if the vehicle flight mode returned matches the **"Mode Change Request"** sent by the FMS.

iii) The Guidance Mode shall determine that manner in which waypoint commands are issued by the FMS.

(1) Refer to 2.b.v.

iv) Determining waypoint commands can be decomposed into 3 stages.

(1) Incorporate vehicle input data.

(2) Handle operator inputs.

(3) Send appropriate waypoint commands.

(a) The FMS shall issue waypoint commands with consideration to the active **Guidance Mode** and the GCU Operator commands.
(b) Each **Guidance Mode** has a set of guarding conditions, which shall determine whether the FMS should continue the process of sending out a new waypoint command.
(c) Guarding Conditions for sending a **Single Mode** waypoint:
  (i) The FMS receives **"Change To Single Mode"** from the GCU Operator via the Engineering GUI.
  (ii) The vehicle has acknowledged the previous waypoint command.
(d) If the Guarding Conditions for **Single Mode Waypoints** are satisfied then:
  (i) If the **Estimated Flight Mode** is Runup Mode, and the **"Change To Single Mode => Flight Mode"** is Ground Mode then the FMS shall issue a **"Mode Change Request => Ground Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange1 Table 1/Col. 1
  (ii) If the **Estimated Flight Mode** is Waypoint Land Mode, and the **"Change To Single Mode => Flight Mode"** is Ground Mode then the FMS shall issue a **"Mode Change Request => Ground Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange1 Table 1/Col. 2
  (iii) If the **Estimated Flight Mode** is Ground Mode, and the **"Change To Single Mode => Flight Mode"** is Runup Mode then the FMS shall issue a **"Mode Change Request => Runup Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange2 Table 1/Col. 2
  (iv) If the **Estimated Flight Mode** is Takeoff Mode, and the **"Change To Single Mode => Flight Mode"** is Runup Mode then the FMS shall issue a **"Mode Change Request => Runup Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange2 Table 1/Col. 1
  (v) If the **Estimated Flight Mode** is Runup Mode, and the **"Change To Single Mode => Flight Mode"** is Takeoff Mode then the FMS shall issue a **"Mode Change Request => Takeoff Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange2 Table 2/Col. 1
  (vi) If the **Estimated Flight Mode** is not Ground, not Runup, not Waypoint Land Mode, and the **"Change To Single Mode => Flight Mode"** is Waypoint Hover Mode then the FMS shall issue a **"Mode Change Request => Waypoint Hover Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange2 Table 3/Col. 1
  (vii) If the **Estimated Flight Mode** is not Ground, not Runup, not Waypoint Land Mode, and the **"Change To Single Mode => Flight Mode"** is Waypoint Through Mode then the FMS shall issue a **"Mode Change Request => Waypoint Through Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange3 Table 1/Col. 1
  (viii) If the **Estimated Flight Mode** is not Ground, not Runup, and the **"Change To Single Mode => Flight Mode"** is Waypoint Land Mode then the FMS shall issue a **"Mode Change Request => Waypoint Land Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange3 Table 2/Col. 1
  (ix) If the **Estimated Flight Mode** is Waypoint Hover Mode, and the **"Change To Single Mode => Flight Mode"** is Pilot Assist Mode then the FMS shall issue a **"Mode Change Request => Pilot Assist Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange4 Col. 1

(x) If the **Estimated Flight Mode** is Waypoint Through Mode, and the **"Change To Single Mode => Flight Mode"** is Pilot Assist Mode then the FMS shall issue a **"Mode Change Request => Pilot Assist Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange4 Col. 2

(xi) If the **Estimated Flight Mode** is Waypoint Land Mode and the vehicle is estimated to be above 5ft above ground level and the **"Change To Single Mode => Flight Mode"** is Pilot Assist Mode then the FMS shall issue a **"Mode Change Request => Pilot Assist Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc – ModeChange4 Col. 3

(e) Guarding Conditions for sending **Pilot Assist** via commands through the Operational GUI

(i) The FMS receives **"Pilot Assist"** from the GCU Operator via the Operational GUI.

(ii) The previous waypoint command has been acknowledged.

(f) If all guarding conditions for sending Pilot Assist via the Operational GUI are satisfied then:

(i) If the **Estimated Flight Mode** is Waypoint Hover Mode, the FMS shall issue a **"Mode Change Request => Pilot Assist Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange5 Table 1/Col. 1

(ii) If the **Estimated Flight Mode** is Waypoint Through Mode, the FMS shall issue a **"Mode Change Request => Pilot Assist Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange5 Table 1/Col. 2

(iii) If the **Estimated Flight Mode** is Waypoint Land Mode and the vehicle is above 5 ft above ground level, the FMS shall issue a **"Mode Change Request => Pilot Assist Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange5 Table 1/Col. 3

(g) Guarding Conditions for sending **Task List** waypoint commands.

(i) Condition Set 1 State Transitions2.doc - ModeChange6 Trigger Col. 4

1. The FMS receives **"Change To Task List Mode"** from the GCU Operator via the Engineering GUI.
2. The previous waypoint command has been acknowledged.
3. The **Estimated Flight Mode** is not in Waypoint Land Mode.
4. There are tasks on the **Task List.**

(ii) Condition Set 2 State Transitions2.doc - ModeChange6 Trigger Col. 3

1. The FMS receives **"Change To Task List Mode"** from the GCU Operator via the Engineering GUI.
2. The previous waypoint command has been acknowledged.
3. The **Estimated Flight Mode** is in Waypoint Land Mode, and the vehicle is not in the altitude range of 1 to 5 ft above ground level.
4. There are tasks on the **Task List.**

(iii) Condition Set 3 State Transitions2.doc - ModeChange6 Trigger Col. 2

1. The FMS receives **"Execute Plan"** from the Operational GUI.
2. The previous command has been acknowledged.
3. The **Estimated Flight Mode** not in Waypoint Land Mode.
4. There are tasks on the **Task List.**

(iv) Condition Set 4 State Transitions2.doc - ModeChange6 Trigger Col. 1

1. The FMS receives **"Execute Plan"** from the GCU Operator via the Engineering GUI.
2. The previous waypoint command has been acknowledged.
3. The **Estimated Flight Mode** is in Waypoint Land Mode, and the vehicle is not in the altitude range of 1 to 5 ft above ground level.

4. There are tasks on the **Task List.**

(h) If Condition Set 2 or 4 for **Task List Mode** is completely satisfied, then:

(i) If the **Estimated Flight Mode** is Waypoint Land Mode, the vehicle is below 5 ft, and the current task on the **Task List** is Waypoint Hover/Through/Land Mode, then the FMS shall issue a **"Mode Change Request => Ground Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange6 Table 1/Col. 1, State Transitions2.doc - ModeChange6 Table 1/Col. 2, State Transitions2.doc - ModeChange6 Table 1/Col. 3

(ii) If the **Estimated Flight Mode** is Waypoint Land Mode, the vehicle is above 5 ft, and the current task on the **Task List** is Waypoint Hover/Through/Land Mode, then the FMS shall issue a **"Mode Change Request => Waypoint Hover Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange7 Table 3/Col. 1, State Transitions2.doc - ModeChange8 Table 1/Col. 1, State Transitions2.doc - ModeChange8 Table 2/Col. 1

(i) If Condition Set 1 or 3 for **Task List Mode** is completely satisfied, then:

(i) If the **Estimated Flight Mode** is Ground Mode, and the current task on the **Task List** is Waypoint Hover/Through/Land Mode, then the FMS shall issue a **"Mode Change Request => Runup Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange7 Table 1/Col. 1, State Transitions2.doc - ModeChange7 Table 1/Col. 2, State Transitions2.doc - ModeChange7 Table 1/Col. 3

(ii) If the **Estimated Flight Mode** is Runup Mode, and the current task on the **Task List** is Waypoint Hover/Through/Land Mode, then the FMS shall issue a **"Mode Change Request => Takeoff Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange7 Table 2/Col. 1, State Transitions2.doc - ModeChange7 Table 2/Col. 2, State Transitions2.doc - ModeChange7 Table 2/Col. 3

(iii) If the **Estimated Flight Mode** is not in Ground, Runup, or Waypoint Land Mode and the current task on the **Task List** is Waypoint Hover Mode, then the FMS shall issue a **"Mode Change Request => Waypoint Hover/Through/Land Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange7 Table 3/Col. 2, State Transitions2.doc - ModeChange8 Table 1/Col. 2, State Transitions2.doc - ModeChange8 Table 2/Col. 2

(j) Guarding Conditions for sending **Return Home** waypoint commands.

(i) Condition Set 1 State Transitions2.doc - ModeChange9 Trigger Col. 1

1. The FMS receives **"Return Home"** from the GCU Operator via the Operational GUI.
2. The previous waypoint command has been acknowledged.
3. The **Estimate Flight Mode** is not in Waypoint Land Mode.

(ii) Condition Set 2 State Transitions2.doc - ModeChange9 Trigger Col. 2

1. The FMS receives **"Return Home"** from the GCU Operator via the Operational GUI.
2. The previous waypoint command has been acknowledged.
3. The **Estimate Flight Mode** is in Waypoint Land Mode, and the vehicle is not in the altitude range of 1to 5 ft above ground level.

(k) If Condition Set 1 **Return Home Mode** is completely satisfied, then:

(i) If the **Estimated Flight Mode** is in Ground Mode, the FMS shall issue a **"Mode Change Request => Runup Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange9 Table 1/Col. 1

(ii) If the **Estimated Flight Mode** is in Runup Mode, the FMS shall issue a **"Mode Change Request => Takeoff Mode"** to the vehicle and reset **Estimated Mode**

Confirmation Status to UNACKNOWLEDGED. State Transitions2.doc - ModeChange9 Table 2/Col. 1

(iii) If the **Estimated Flight Mode** is in not Ground, Runup, or Waypoint Land Mode, the FMS shall issue a **"Mode Change Request => Waypoint Land Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange10 Table 1/Col. 1

(l) If Condition Set 2 **Return Home Mode** is completely satisfied, then:

(i) If the **Estimated Flight Mode** is in Waypoint Land Mode and above 5 ft above ground level, the FMS shall issue a **"Mode Change Request => Waypoint Land Mode"** to the vehicle and reset **Estimated Mode Confirmation Status** to UNACKNOWLEDGED. State Transitions2.doc - ModeChange10 Table 1/Col. 2

(ii) In the event of a timeout of the **"Mode Confirm Message"** the FMS shall reissue the previous **"Mode Change Request"**. State Transitions2.doc - ModeChange10 Table 2/Col. 1

3) The FMS shall be robust to communications dropouts.
   a) The communications link shall be assumed down if the FMS does not receive any messages from the vehicle within a pre-specified number of cycles of the FMS software. State Transitions2.doc - Communications Table 1/Col 1
   b) The communications status shall be reset to active whenever a new message is received by the FMS. State Transitions2.doc - Communications Table 2/Col 1
4) The FMS shall have an accurate estimate of the vehicle's dynamic states. These estimates shall be used to determine whether the vehicle is within the **Radius of Capture**. They will also be used to determine if the vehicle is on the ground.
   a) The FMS must be able to work make use of information from the Navigation Filter on board the vehicle. The Navigation Filter uses sensor inputs limited to an IMU, Sonar Altimeter, Compass, and Differential GPS that are then integrated into a navigation solution via Kalman Filtering.
   b) The Navigation Filter must be initialized before the vehicle can execute a mission
      i) The FMS shall update its estimate of the **Navigation Filter Status** to the state received in the **"State Update"** message from the vehicle. State Transitions2.doc - VehState Table 8/Col. 1
      ii) At *Startup*, the Navigation Filter Status is assumed to be in state NOT READY until a **"State Update"** message is received.
      iii) The FMS shall not be allowed to send a command to the vehicle to transition out of Ground Mode until **Navigation Filter Status** is HEALTHY.
      iv) **Navigation Filter Status** in state STANDBY shall indicate the vehicle is in the act of initializing the navigation filter.
      v) The FMS can initialize the Navigation Filter using coordinates entered manually by the GCU Operator, or automatically be commanding the GPS ground station into surveying its own coordinates.
         (1) The FMS must send a **"GPS Ground Station Coordinates"** message upon receipt of an **"Upload Field Settings"** message from the GCU Operator via the Engineering GUI. The **"GPS Ground Station Coordinates"** message contains the GCU Operator's manually entered ground station location estimate. State Transitions2.doc - FieldCoords Col. 1
         (2) The FMS shall send a **"GPS Ground Station Coordinates"** message only if the **Estimated Flight Mode** is in ground Mode, and the FMS receives an **"Initialize System"** message from the GCU Operator via the Operational GUI. State Transitions2.doc - FieldCoords Col. 2
         (3) The FMS shall send an **"Initialize Navigation Filter"** message to the vehicle only if the **Estimated Vehicle Flight Mode** is in Ground Mode, and the FMS receives an **"Initialize Navigation"** message from the GCU Operator via the Engineering GUI. State Transitions2.doc - InitNav Col. 2
         (4) The FMS shall send an **"Initialize Navigation Filter"** message to the vehicle only if the **Estimated Vehicle Flight Mode** is in Ground Mode, the **Navigation Filter Status** is DOWN, the **GPS Status** is OKAY, and the FMS receives an **"Initialize System"**

message from the GCU Operator via the Operational GUI. State Transitions2.doc - InitNav Col. 1

(a) In order to receive an estimate of the **GPS Status**, the FMS must issue a **"GPS Request Message"** to the vehicle.

(b) The FMS shall issue a **"GPS Request Message"** to the vehicle if it receives a **"GPS Request Data"** message from the GCU Operator via the Engineering GUI.

(c) The FMS shall issue a **"GPS Request Message"** to the vehicle prior to issuing an **"Initialize Navigation Filter"** message when it receives an **"Initialize System"** message from the GCU Operator via the Operational GUI. In addition the **Navigation Filter Status** must be DOWN, and the **Estimated Flight Mode** must be in Ground Mode.

(d) The FMS shall reissue a **"GPS Request Message"** to the vehicle if there is a timeout of the **"GPS Message"**. In addition, the **Navigation Filter Status** must be DOWN, and the **Estimated Flight Mode** must be in Ground Mode.

(e) The **Navigation Filter Status** must be DOWN and the **Estimated Flight Mode** must be in Ground Mode, to initialize the system since it is assumed that the if the navigation is not DOWN, and the flight mode is not Ground, the vehicle can be airborne. Initializing the Navigation Filter in the air is a hazard. See Safety Requirements.

c) The FMS must be able to maintain high level control of the vehicle with dynamic state inputs limited to position (x, y, altitude), and velocity (u, v, w).

i) The FMS must be capable of maintaining high level control of the vehicle with position and velocity inputs being received from the vehicle at a rate of 1 Hz.

ii) The FMS shall update its estimate of vehicle position upon receipt of the **"State Update"** message from the vehicle. State Transitions2.doc - VehState Table 1, 2, 3/Col. 1

iii) The FMS shall update its estimate of vehicle velocity upon receipt of the **"State Update"** message from the vehicle. State Transitions2.doc - VehState Table 4, 5, 6/Col. 1

d) The FMS shall be capable of operating with sub-optimal sensor inputs.

e) The FMS shall be temporally in synch with the vehicle, so that waypoints can be updated in a timely manner.

i) The FMS shall update its estimate of the current time via the **"State Update"** message from the vehicle. State Transitions2.doc - VehState Table 7/Col. 1

ii) The FMS shall make use of the **Estimated Hover Start Time** to determine whether the vehicle has hovered for the commanded amount of time in the specified position in Waypoint Hover Mode.

(1) The FMS shall update the **Estimated Hover Start Time** to the most recent time update if the vehicle is in Waypoint Hover Mode, and the position estimate is within the **Radius of Capture**. State Transitions2.doc - HvrTime Table 1/Col. 1

(2) The FMS shall update the **Estimated Hover Start Time** only once for each Waypoint Hover Mode.

(3) The difference between the current time estimate, and the **Estimated Hover Start Time** shall be the amount of time spent hovering.

iii) The FMS shall make use of the **Estimated Mode Start Time** to determine when Runup Mode has been completed and to determine the time in spent thus far in the current mode.

(1) At *Startup*, a **"Mode Confirm Message"** shall result in the **Estimated Mode Start Time** to be updated to the most recent time update. State Transitions2.doc - ModeStart Table 1/Col. 2

(2) During *Normal Operation*, the FMS shall update the **Estimated Mode Start Time** only if the FMS receives a **"Mode Confirmation Message"** and the vehicle flight mode returned is the same as the one commanded. . State Transitions2.doc - ModeStart Table 1/Col 1

(3) The FMS shall update the **Estimated Hover Start Time** only once for each receipt of a **"Mode Confirm Message"**.

(4) The difference between the current time estimate, and the **Estimated Mode Start Time** shall be the amount spent in the current flight mode.

5) The FMS shall provide timely vehicle state information to the GCU Operator via the GUI.

6) The Operational GUI must be consistent in presenting information with the Engineering GUI.
7) The FMS shall transmit Ground Station joystick data to the vehicle when in Pilot Assist Mode.
8) The FMS shall transmit camera control commands to the on-board camera.
9) The FMS shall provide the GCU Operator with the capability to adjust the on-board controller parameters (gains, open/close control loops)

Operational/Engineering GUI Requirements

1) The Operational GUI and Engineering GUI shall be consistent with one another.
2) The Operational GUI and Engineering GUI shall display Ground Mode as the default Estimated Flight Mode on the monitor.
3) The Operational GUI and Engineering GUI shall update its display information in a timely manner.
4) The displays shall provide the GCU Operator with the most recent update of the vehicle's state.
5) Initially Communications Status shall be displayed as DOWN.
6) The Navigation Filter Health shall be displayed as in STANDBY when the navigation filter is being initialized.
7) The "Execute Plan" command must not initially be selectable. It shall be made selectable only if the navigation filter has been initialized successfully.
8) The "Pilot Assist" command must not initially be selectable. It shall be made selectable only if the navigation filter has been initialized successfully.
9) The "Return Home" command must not initially be selectable. It shall be made selectable only if the navigation filter has been initialized successfully.
10) The GUI shall display the flight path of the vehicle from waypoint to waypoint.

System Limitations

1) The FMS does not possess a collision avoidance system.
   a) The FMS does not possess the capability to avoid obstacles in its flight path. Sensor information is only used to determine position and velocity of the vehicle.
2) The FMS does not make use of vehicle resources to command the vehicle.
   a) Missions planned on the Operational or Engineering GUI will not reflect the amount of resources that will be consumed for a specified mission. It is possible to create an unrealizable mission.
   b) During mission execution, the FMS does not make use of remaining resources to determine whether or not to continue the mission if resources are low.
3) The FMS does not make use of terrain information. Altitude is determined as the difference between the GPS Ground Station altitude to the vehicle altitude.
   a) The FMS assumes a flat earth model. Attempts to fly over uneven terrain can potentially result in collisions into terrain.
4) The GUI cannot coordinate map graphics to physical coordinates online.
   a) Each map must be matched up with the actual coordinates of the mission region in order for the GUI to accurately reflect the position of the vehicle on the map.

Safety Requirements

1) The FMS must not be allowed the capability to command a lateral maneuver below 5 ft above ground level.
   a) Maneuvering at low altitude increase the change of the vehicle performing a rollover.
   b) The FMS must have an accurate estimate of the current flight mode being executed by the vehicle.
2) The FMS must not command a flight mode resulting in the loss of lift while above 3 ft above ground level.
   a) Dropping from an altitude above 3 ft. above ground level can cause impact damage to the vehicle.
   b) The FMS must have an accurate estimate of the current flight mode being executed by the vehicle.

3) The FMS must not allow for the navigation filter to be initialized while above 3 ft above ground level.
   a) Initialization of the navigation filter results in the Kalman Filter issuing random navigation solutions. This can potentially be disastrous if the vehicle is in the air attempting to maintain stability with these navigation solutions which do not correctly reflect the state of the vehicle
4) The FMS must not command a flight mode that cannot be executed by the vehicle.
   a) This could potentially cause the FMS to be stuck continually commanding the unrealizable flight mode.
5) The FMS not allow the GCU Operator to execute a mission until the vehicle computer is Operational.
   a) Executing a mission before the vehicle computer is on can result in human injury. The flight crew responsible for working on the vehicle can be injured from the rotor blades if a mission has been executed, and they go to physically turn on the vehicle computer.
6) The FMS must not cause of contributed to a controlled maneuver of the vehicle into terrain.
   a) The GUI must not allow negative altitude entries.
      i) The GUI must ensure that the negative altitude entries are not allowed, otherwise controlled flight into terrain can occur.
7) The FMS must not cause or contribute to the loss of control of the vehicle.

Definitions

Current Task Index: The task number of the task that is being executed in **Task List Mode.** The first task is associated with the index number of 1.

Estimate Flight Mode: This is the most recent flight mode received by the FMS from the vehicle through a **"Mode Confirm Message".** This will be used to update waypoints and to determine which mode transitions are valid. The default estimate at *Startup*, is Ground Mode.

Estimate Mode Start Time: This is an indicator of when the current flight mode was initiated. The Mode Start Time shall be used to determine whether Runup Mode has been completed, and shall also be used to determine the amount of time that has been spent in the current mode to be displayed in the GUI.

Estimated Hover Start Time: This is an indicator of the time hover has actually commenced when the vehicle is within the **Radius of Capture** of the hover waypoint. This is used to determine whether Hover Mode has been completed.

Estimated Mode Confirmation Status: This is a state variable used as an estimate of whether the vehicle has acknowledged the current waypoint command. **MCS** can take on two states ACKNOWLEDGED, or UNACKNOWLEDGED. ACKNOWLEDGED indicates the vehicle has received the waypoint command. UNACKNOWLEDGED indicates the vehicle has not received the waypoint command.

GPS Status: This is a quality indicator of the GPS updates. The possible states for GPS Status are GPS OK, GPS Degraded, and GPS Down. GPS OK indicates that the GPS readings received by the navigation filter are good enough to be used for determining the Navigation Filter Solution. GPS Degraded indicates that the readings are of lower quality, but are still useful. GPS Down indicates that the quality of the GPS reading is too poor to use.

Guidance Initialization Status: This is a mode of the FMS. At startup, the GIS is in the state of NOT READY. This implies that the FMS has not received a **"Mode Confirm Message"** from the vehicle yet. When the FMS is in this state, no waypoint commands can be issued until a **"Mode Confirm Message"** is received, at which time the GIS will transition to the state of READY.

Mode Change Request: This is an output message from the FMS to the vehicle. This message commands a change in the current flight mode of the vehicle and the associated parameters such as position, hover time, etc.

Mode Confirm Message: This is an input message sent to the FMS by the vehicle in response to a "Mode Change Request" or a "Mode Request Message". The message contains the current vehicle flight mode.

Mode Request Message: This is an output message send by the FMS to the vehicle at *Startup* to get an estimate of the current flight mode of the vehicle.

Navigation Filter Status: This is an indication of the relative validity of the navigation solution. There are 5 states: NOT READY, STANDBY, READY, HEALTHY, DERGRADED, DOWN. Standby indicates that the navigation filter is in the act of initialization. The default state is NOT READY. HEALTHY indicates that the sensor input going in to the Kalman Filter are all working nominally. DEGRADED indicates that some of the sensor inputs coming into the Kalman Filter have errors. This includes the GPS, IMU, Compass and the SONAR. DOWN indicates that the Navigation Filter solution is useless.

Pilot Assist: This is a Single Flight Mode that gives the GCU Operator direct control of the vehicle's velocities through a joystick located on the GCU.

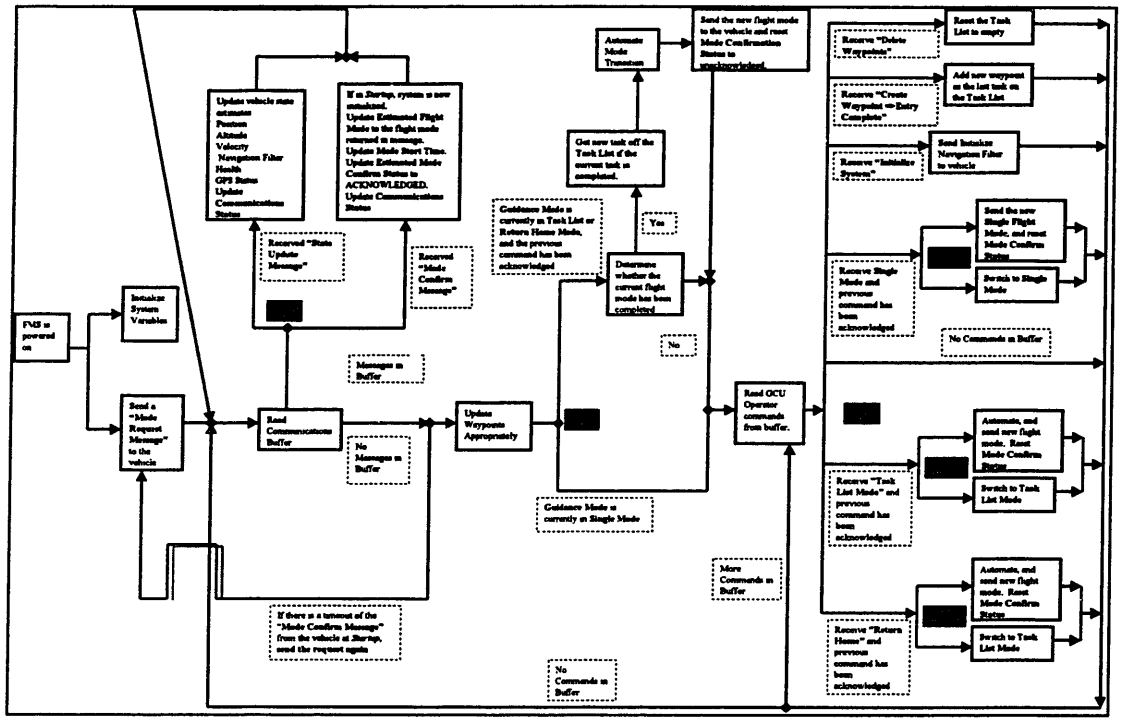
Radius of Capture: An imaginary spherical region surrounding the waypoint of interest. If the vehicle falls within this region, the FMS considers the vehicle to have reached the commanded waypoint.

State Update: This is a message sent from the vehicle to the FMS at a rate of 1 Hz. The message provides the FMS with essential information from the vehicle. The message contains the current position (x, y, altitude), and the current velocity (u, v, w) estimates of the vehicle. This message also includes an estimate of the Navigation Filter Health, GPS Status, and Sonar Status.

Task List: The Task List is a list of waypoints defined by the GCU Operator to be executed in sequential order. This is the autonomous mission's flight trajectory. The Task List can only contain waypoints of the flight mode type (Waypoint Hover, Waypoint Through, and Waypoint Land).

Total Number of Tasks: The sum of the number of tasks currently on the **Task List.**

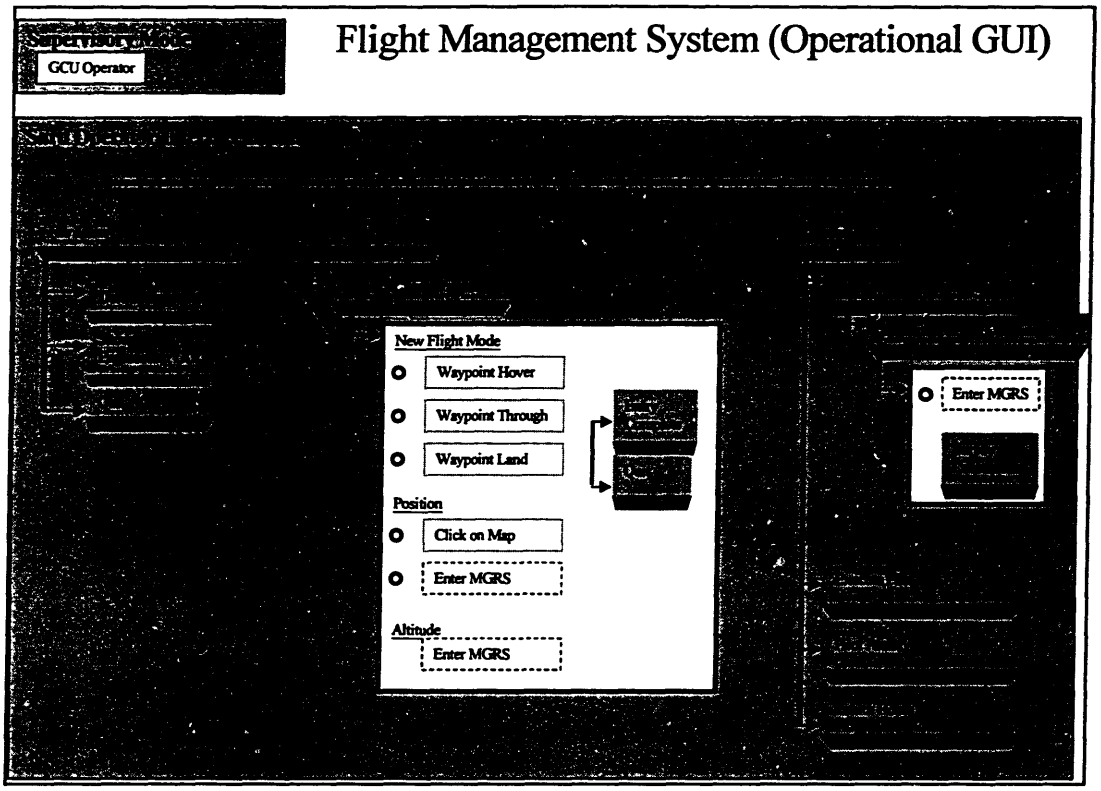
A.2 Supplemental Flowchart for FMS Specifications

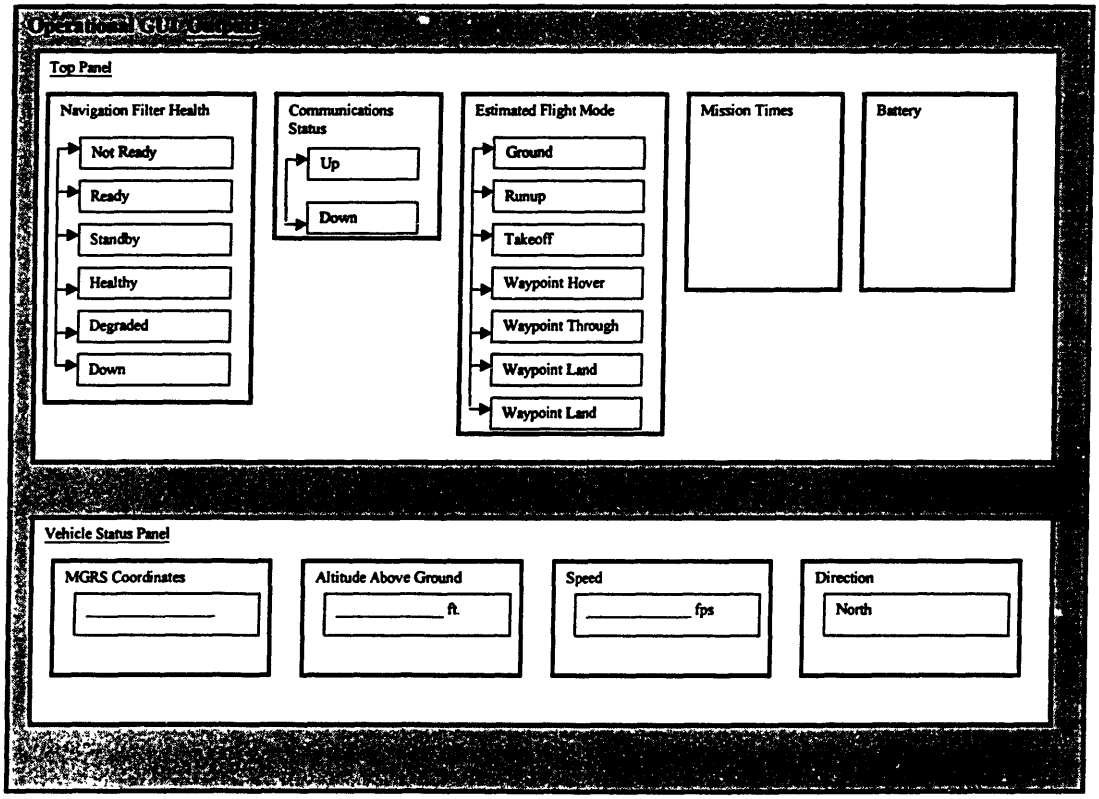


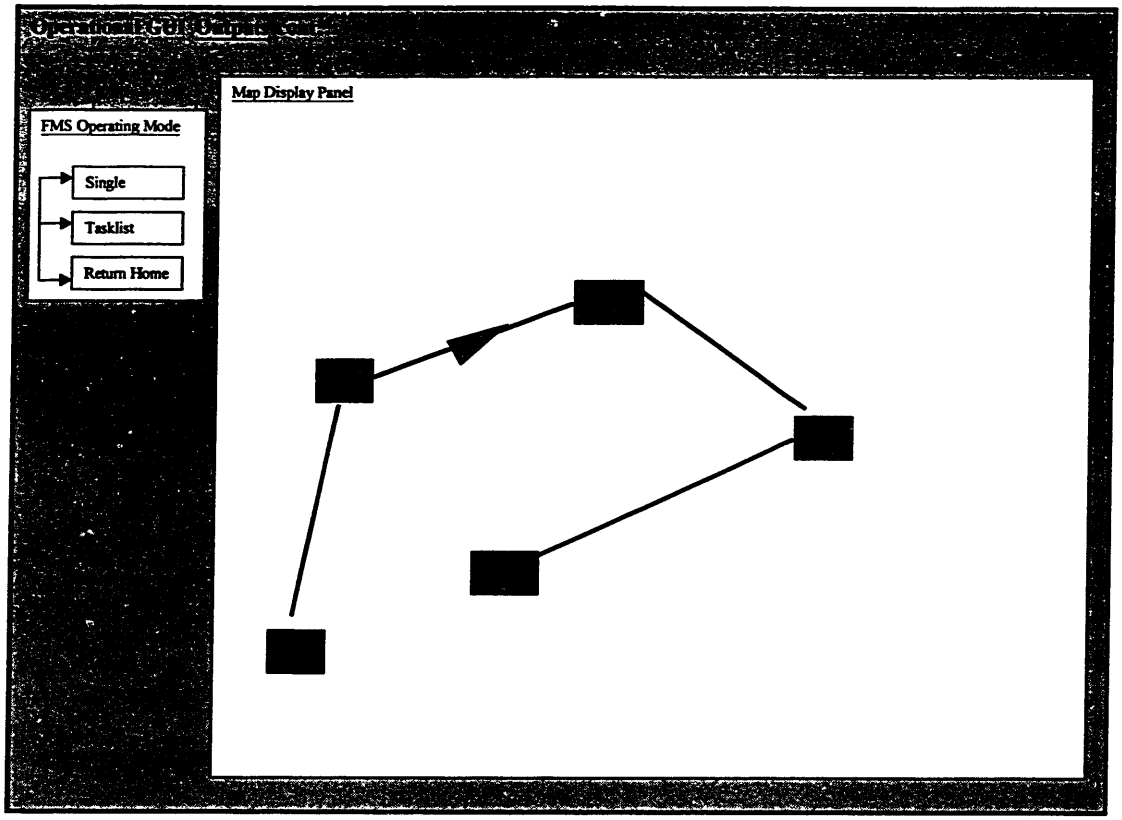
Appendix B State Based Software Requirements Specifications for the FMS

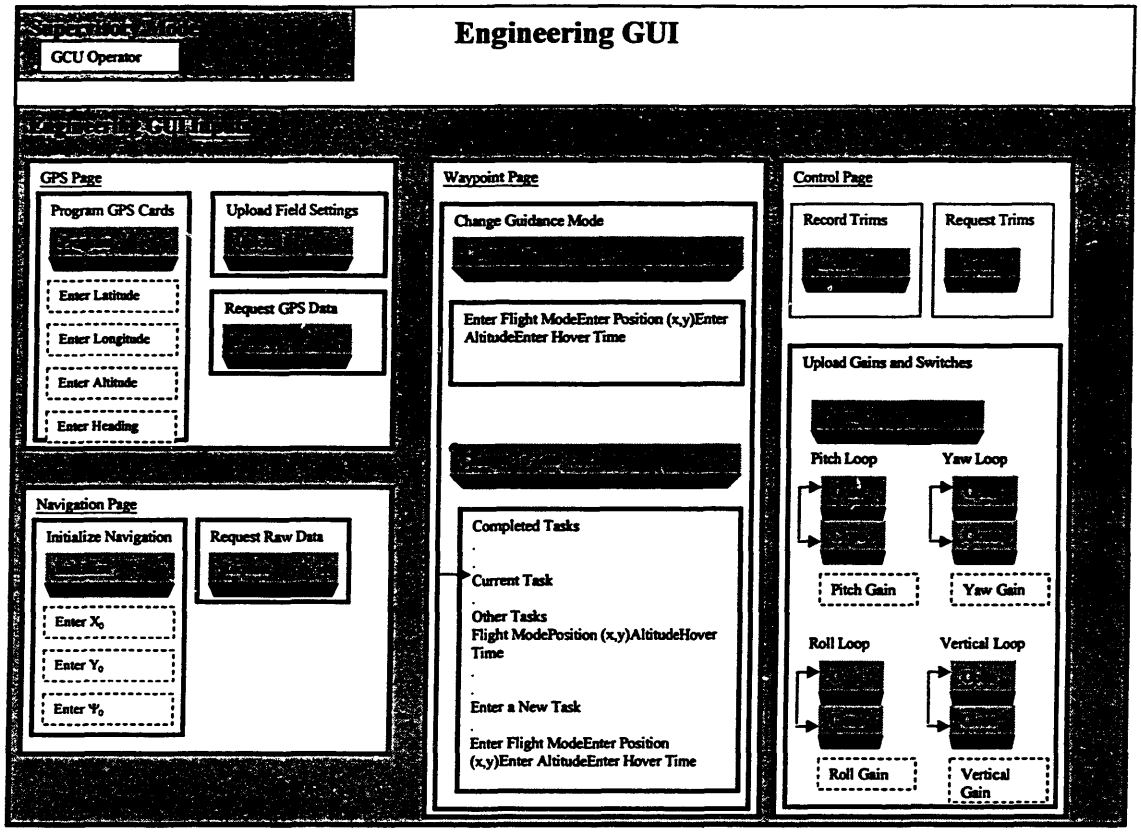
This appendix contains the state-based specifications generated for the FMS during the course of research. The first section presents the State-Machine Diagrams for the specifications model, and the second section presents the related State Transition Tables.

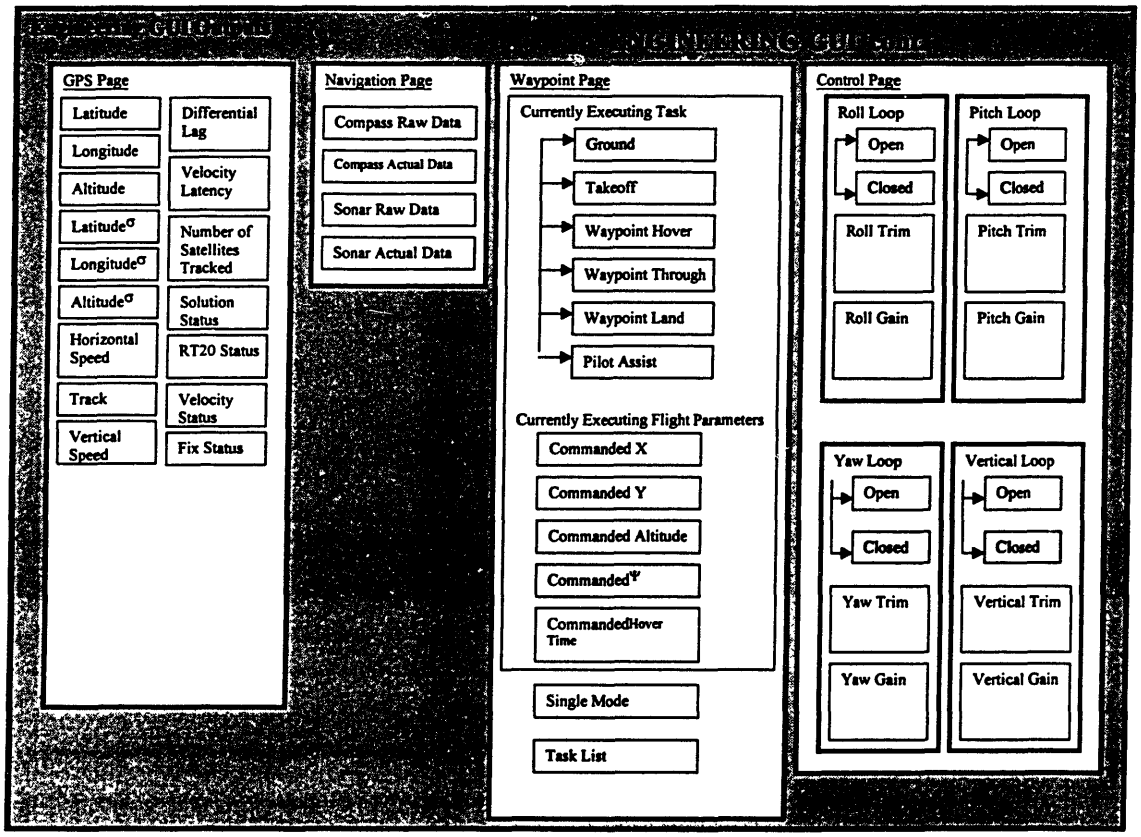
B.1 State-Machine Diagrams for the FMS Software Specifications

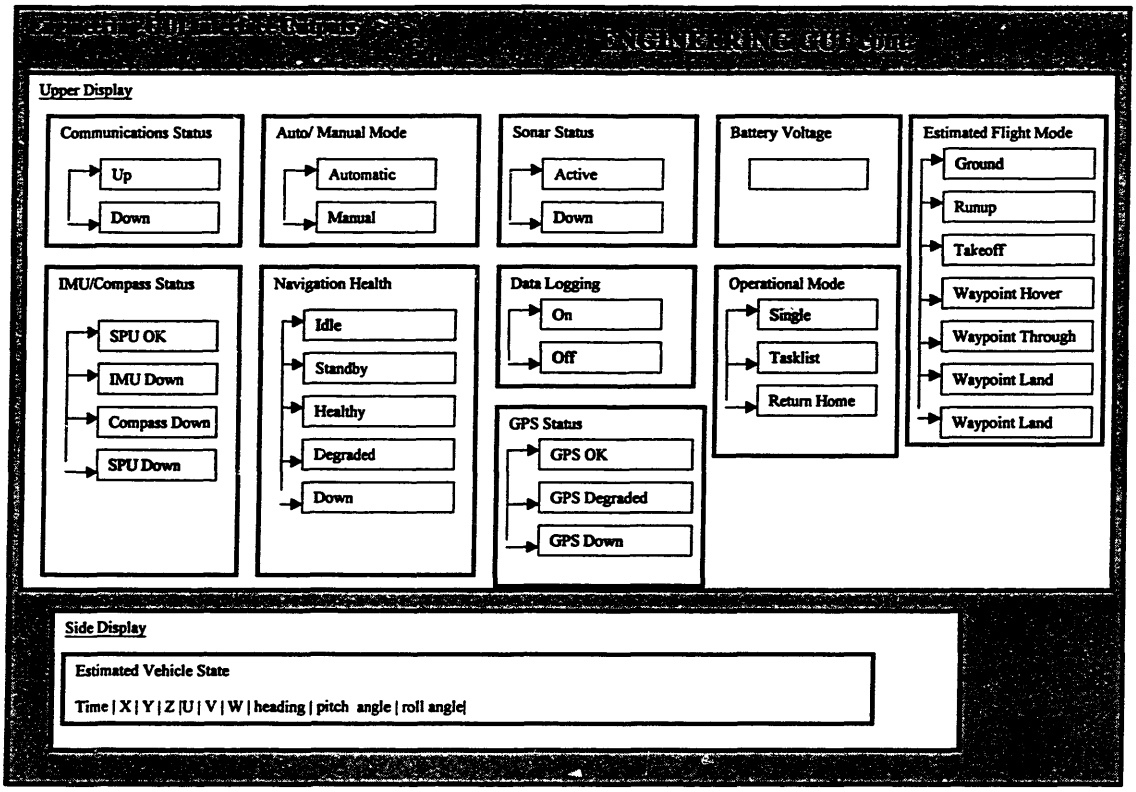


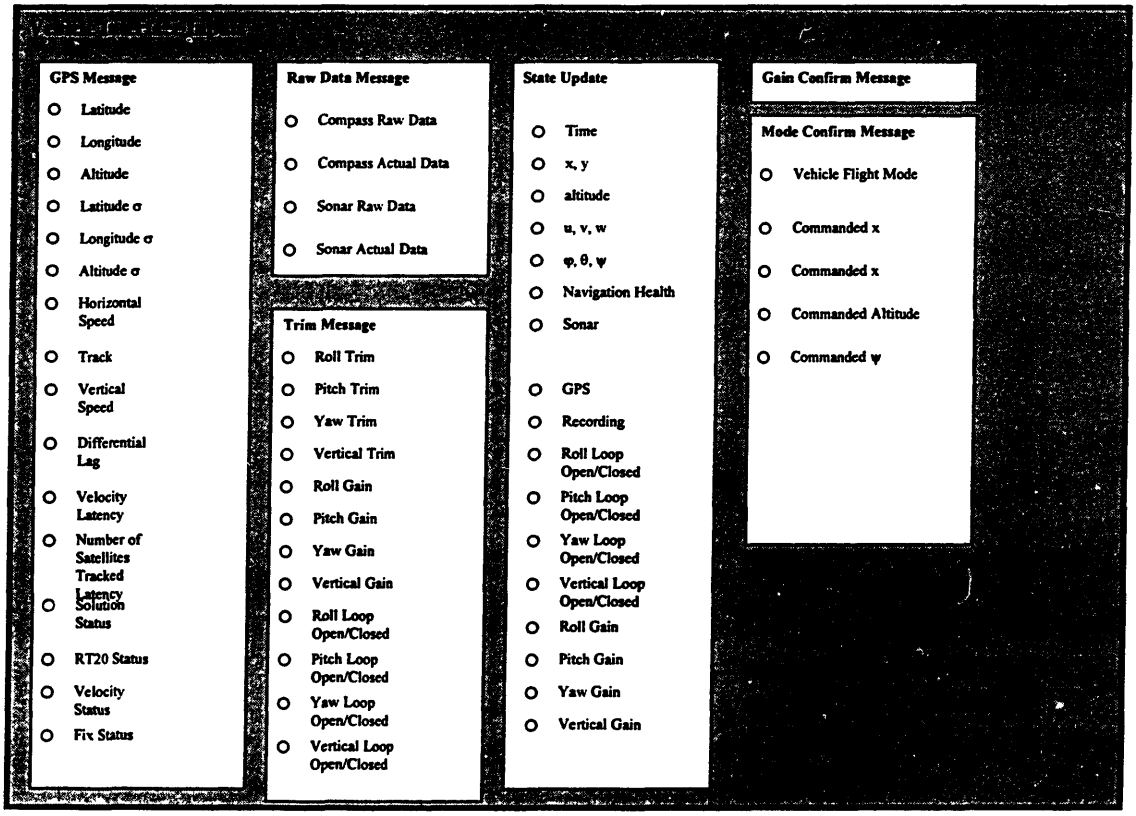


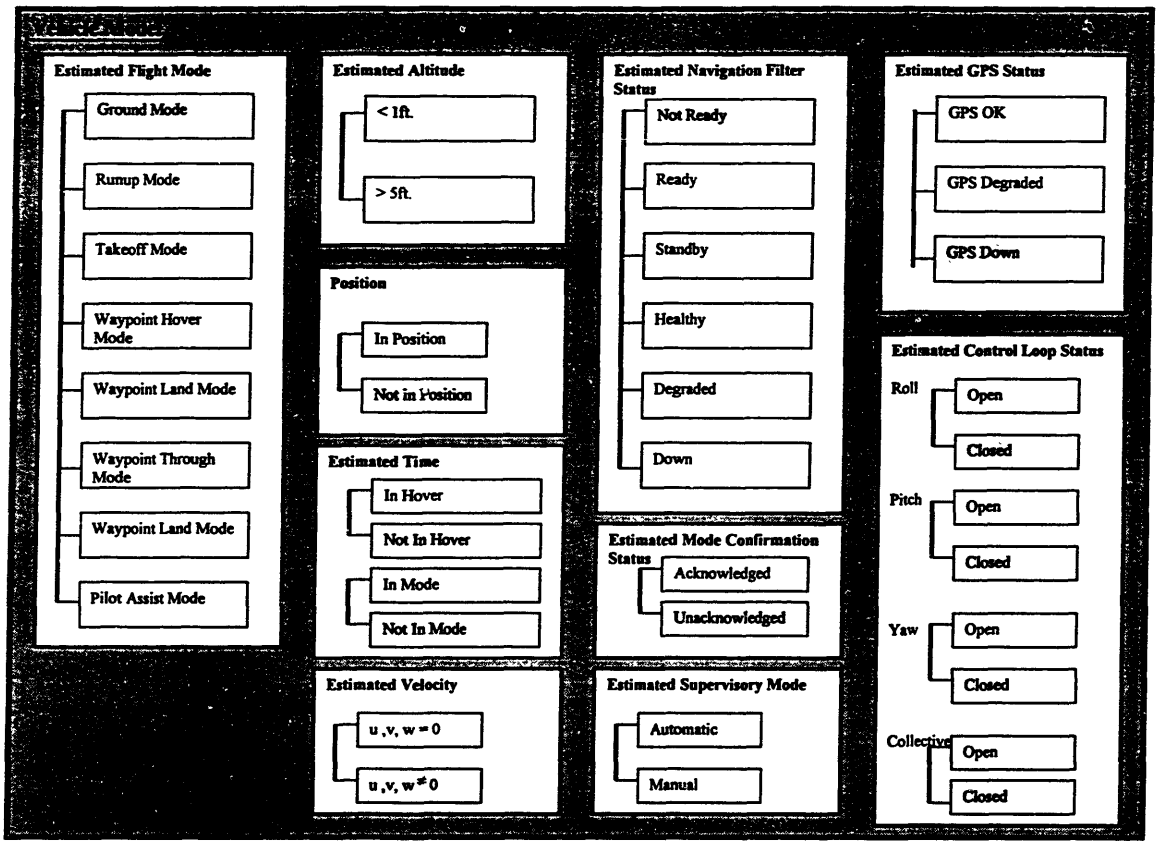

Map Display Panel
FMS Operating Mode
Single
Tasklist
Return Home

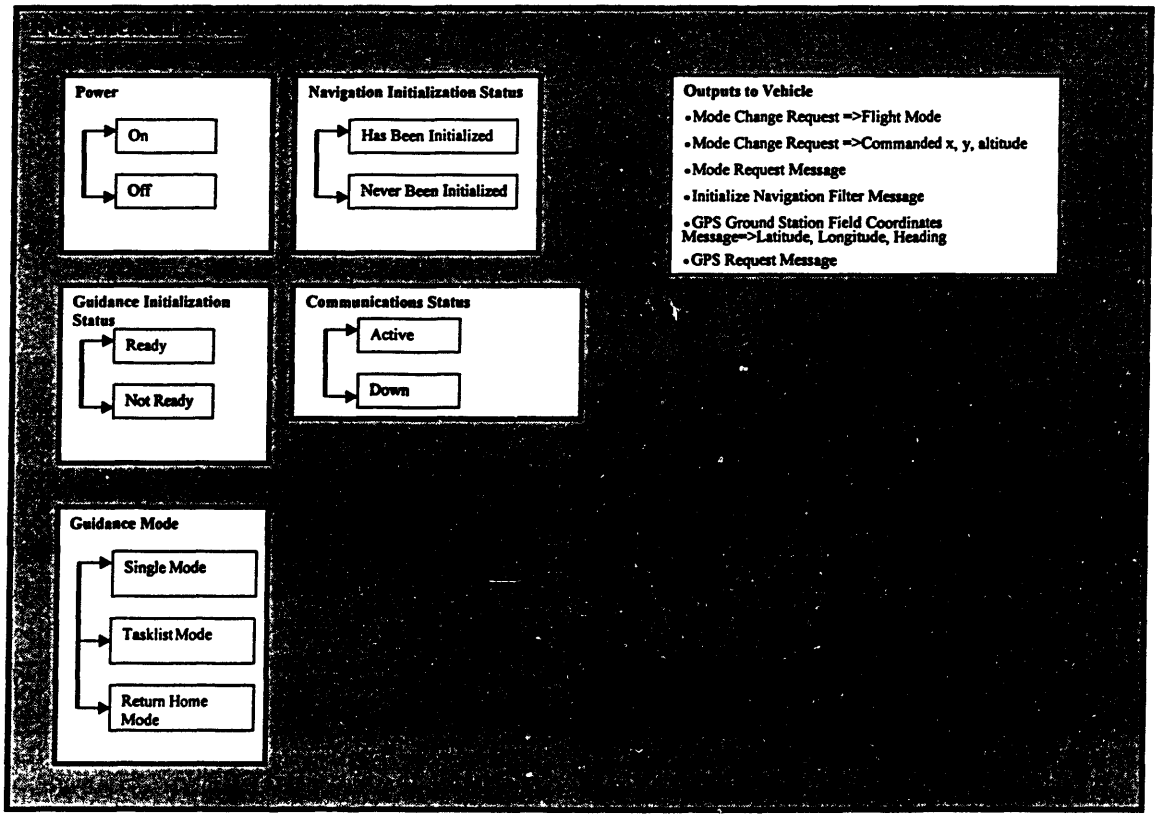

Engineering GUI
GCU Operator
GPS Page
Program GPS Cards
Upload Field Settings
Request GPS Data
Enter Latitude
Enter Longitude
Enter Altitude
Enter Heading
Navigation Page
Initialize Navigation
Request Raw Data
Enter X_0
Enter Y_0
Enter Ψ_0
Waypoint Page
Change Guidance Mode
Enter Flight ModeEnter Position (x,y)Enter AltitudeEnter Hover Time
Completed Tasks
Current Task
Other Tasks
Flight ModePosition (x,y)AltitudeHover Time
Enter a New Task
Enter Flight ModeEnter Position (x,y)Enter AltitudeEnter Hover Time
Control Page
Record Trims
Request Trims
Upload Gains and Switches
Pitch Loop
Yaw Loop
Pitch Gain
Yaw Gain
Roll Loop
Vertical Loop
Roll Gain
Vertical Gain

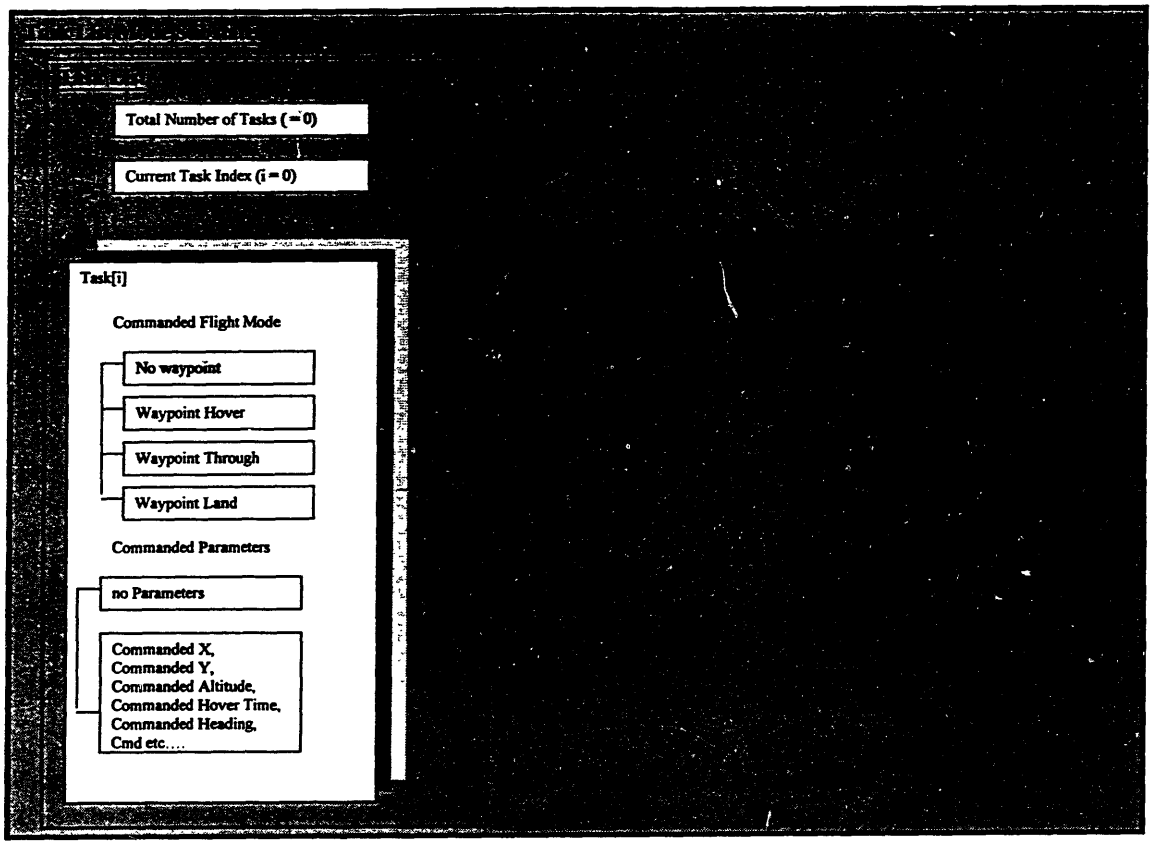

GPS Page
Latitude
Longitude
Altitude
Latitude$^{\sigma}$
Longitude$^{\sigma}$
Altitude$^{\sigma}$
Horizontal Speed
Track
Vertical Speed
Differential Lag
Velocity Latency
Number of Satellites Tracked
Solution Status
RT20 Status
Velocity Status
Fix Status
Navigation Page
Compass Raw Data
Compass Actual Data
Sonar Raw Data
Sonar Actual Data
Waypoint Page
Currently Executing Task
Ground
Takeoff
Waypoint Hover
Waypoint Through
Waypoint Land
Pilot Assist
Currently Executing Flight Parameters
Commanded X
Commanded Y
Commanded Altitude
Commanded$^{\Psi}$
CommandedHover Time
Single Mode
Task List
Control Page
Roll Loop
Open
Closed
Roll Trim
Roll Gain
Pitch Loop
Open
Closed
Pitch Trim
Pitch Gain
Yaw Loop
Open
Closed
Yaw Trim
Yaw Gain
Vertical Loop
Open
Closed
Vertical Trim
Vertical Gain


Upper Display
Communications Status
Up
Down
Auto/ Manual Mode
Automatic
Manual
Sonar Status
Active
Down
Battery Voltage
Estimated Flight Mode
Ground
Runup
Takeoff
Waypoint Hover
Waypoint Through
Waypoint Land
Waypoint Land
IMU/Compass Status
SPU OK
IMU Down
Compass Down
SPU Down
Navigation Health
Idle
Standby
Healthy
Degraded
Down
Data Logging
On
Off
Operational Mode
Single
Tasklist
Return Home
GPS Status
GPS OK
GPS Degraded
GPS Down
Side Display
Estimated Vehicle State
Time | X | Y | Z |U | V | W | heading | pitch angle | roll angle|


GPS Message
Latitude
Longitude
Altitude
Latitude σ
Longitude σ
Altitude σ
Horizontal Speed
Track
Vertical Speed
Differential Lag
Velocity Latency
Number of Satellites Tracked
Latency Solution Status
RT20 Status
Velocity Status
Fix Status
Raw Data Message
Compass Raw Data
Compass Actual Data
Sonar Raw Data
Sonar Actual Data
Trim Message
Roll Trim
Pitch Trim
Yaw Trim
Vertical Trim
Roll Gain
Pitch Gain
Yaw Gain
Vertical Gain
Roll Loop Open/Closed
Pitch Loop Open/Closed
Yaw Loop Open/Closed
Vertical Loop Open/Closed
State Update
Time
x, y
altitude
u, v, w
φ, θ, ψ
Navigation Health
Sonar
GPS
Recording
Roll Loop Open/Closed
Pitch Loop Open/Closed
Yaw Loop Open/Closed
Vertical Loop Open/Closed
Roll Gain
Pitch Gain
Yaw Gain
Vertical Gain
Gain Confirm Message
Mode Confirm Message
Vehicle Flight Mode
Commanded x
Commanded x
Commanded Altitude
Commanded ψ


Estimated Flight Mode
Ground Mode
Rumup Mode
Takeoff Mode
Waypoint Hover Mode
Waypoint Land Mode
Waypoint Through Mode
Waypoint Land Mode
Pilot Assist Mode
Estimated Altitude
< 1ft.
> 5ft.
Position
In Position
Not in Position
Estimated Time
In Hover
Not In Hover
In Mode
Not In Mode
Estimated Velocity
u ,v, w = 0
u ,v, w ≠ 0
Estimated Navigation Filter Status
Not Ready
Ready
Standby
Healthy
Degraded
Down
Estimated Mode Confirmation Status
Acknowledged
Unacknowledged
Estimated Supervisory Mode
Automatic
Manual
Estimated GPS Status
GPS OK
GPS Degraded
GPS Down
Estimated Control Loop Status
Roll
Open
Closed
Pitch
Open
Closed
Yaw
Open
Closed
Collective
Open
Closed


Power
On
Off
Navigation Initialization Status
Has Been Initialized
Never Been Initialized
Outputs to Vehicle
•Mode Change Request =>Flight Mode
•Mode Change Request =>Commanded x, y, altitude
•Mode Request Message
•Initialize Navigation Filter Message
•GPS Ground Station Field Coordinates Message=>Latitude, Longitude, Heading
•GPS Request Message
Guidance Initialization Status
Ready
Not Ready
Communications Status
Active
Down
Guidance Mode
Single Mode
Tasklist Mode
Return Home Mode


Total Number of Tasks (=0)
Current Task Index (i=0)
Task[i]
Commanded Flight Mode
No waypoint
Waypoint Hover
Waypoint Through
Waypoint Land
Commanded Parameters
no Parameters
Commanded X,
Commanded Y,
Commanded Altitude,
Commanded Hover Time,
Commanded Heading,
Cmd etc....

B.2 State Transition Tables for the FMS Software Specifications

Output

Mode Request Message

Destination: On Board Guidance

Comments: Request Vehicle Flight Mode from Vehicle

Triggering Event

	Timeout Event	*Initial Startup Event*
Flight Manager Power **in state** ON	T	T
Previously (Flight Manager Power **in state** OFF)	*	T
"Mode Confirm Message" **Timeout**	T	F
Guidance Initialization Status **in state** READY	F	T

Output Action => **Send** "Mode Request Message" **to** Vehicle

State Variable

Estimated Mode Confirmation Status

Possible Values: ACKNOWLEDGED, UNAKNOWLEDGED

Description: This is the Flight Management System's estimate of whether the vehicle has received the Flight Manager's "Mode Change Request" (*normal operation*), or whether the vehicle has received the mode request message sent by the flight manager (*at start-up*). The FMS is restricted from sending further mode change requests until an acknowledgement is received.

Definition

Initially Mode Confirmation Status **in state** UNACKNOWLEDGED

Set Mode Confirmation **from state** NO ACKNOWLEDGE **to state** ACKNOWLEDGE

	Normal Operation	*Initial Startup*
Receive: "Mode Confirm Message" **from** Vehicle **Message Contents:** Vehicle Flight Mode	T	T
Estimated Mode Confirmation Status **in state** UNACKNOWLEDGED	T	T
"Mode Confirm Message"=>Vehicle Flight Mode = "Mode Change Request"=>Flight Mode	T	*
Guidance Initialization Status **in state** READY	*	F

Set Mode Confirmation **from state** ACKNOWLEDGE **to state** NO ACKNOWLEDGE

See **Output**: Guidance Message for the state transition conditions for this transition

State Variable

Estimated Mode Start Time

Possible Values: Real

Description: This keeps track of the time at which the helicopter has commenced operating at the current flight mode. It should be triggered as soon as the helicopter receives a mode confirm message from the vehicle matching what was sent out..

Definition

Set Estimated Mode Start Time **equal to** Estimated Time

	Normal Operation	*Initial Startup*
Receive: "Mode Confirm Message" **from** Vehicle **Message Contents:** Vehicle Flight Mode	T	T
"Mode Confirm Message"=>Vehicle Flight Mode = "Mode Change Request"=>Flight Mode	T	*
Guidance Initialization Status **in state** READY	T	F
Estimated Mode Confirmation Status **in state** UNACKNOWLEDGED	T	T

State Variable

Estimated Flight Mode

Possible Values: GROUND MODE, RUNUP MODE, TAKEOFF MODE, WAYPOINT HOVER MODE, WAYPOINT THROUGH MODE, WAYPOINT LAND MODE, PILOT ASSIST MODE

Description: Keeps track of the current estimated flight mode of the vehicle. During normal operation, the Flight Manager will update the estimated mode when the vehicle's current flight mode is equal to the commanded flight mode. At start up, the Flight Manager will update whenever it receives a mode confirm message.

Definition

Initially in **state** GROUND MODE

Normal Operation

Set Estimated Flight Mode **equal to** Mode Change Request=>Flight Mode

Receive "Mode Confirm Message" **from** Vehicle **Message Contents:** Vehicle Flight Mode	T
"Mode Confirm Message"=>Vehicle Flight Mode = "Mode Change Request"=>Flight Mode	T
Guidance Initialization Status **in state** READY	T
Estimated Mode Confirmation Status **in state** UNACKNOWLEDGED	T

Initial Startup

Set Estimated Flight Mode **equal to** Mode Confirm Message=>Vehicle Flight Mode

Receive "Mode Confirm Message" **from** Vehicle **Message Contents:** Vehicle Flight Mode	T
Guidance Initialization Status **in state** READY	F
Estimated Mode Confirmation Status **in state** UNACKNOWLEDGED	T

State Variable

Guidance Initialization Status

Possible Values: READY, NOT READY

Description: The FM automatically sends a Mode Request Message on start up. Once the FMS receives a Mode Confirm message from the vehicle, it will be considered initialized, and ready for the GCU Operator to perform tasks.

Special Restrictions: This state transition is triggered when the system receives its first mode confirm message, but this message also triggers other state transitions with the additional requirement that the Guidance Initialization Status is UNINITIALIZED. Therefore Guidance Initialization Status must not make a transition before the states: Estimated Start Mode Time, Mode Confirmation, and Estimated Flight Mode have made their transitions.

Definition

Initially Guidance Initialization Status **in state** NOT READY

Set Guidance Initialization Status **from state** NOT READY **to state** READY

Receive "Mode Confirm Message" **from** Vehicle	T
Estimated Mode Confirmation Status **in state** UNACKNOWLEDGED	T
Guidance Initialization Status **in state** NOT READY	T

State Variable

Communications Status

Possible Values: ACTIVE, DOWN

Description: The Communication System must receive a new signal after *X* cycles of the program. If not it will be considered to be malfunctioning

Definition

Initially Communications Status **in state** DOWN

Set Communications Status **from state** ACTIVE **to state** DOWN

Receive "END MESSAGE" **from** Vehicle within past *X* cycles	F

Set Communications Status **from state** DOWN **to state** ACTIVE

Receive "END MESSAGE" **from** Vehicle	T

State Variable

Flight Management System

↳ Dynamic States: Estimated x, Estimated y, Estimated Altitude, Estimated u, Estimated v, Estimated w, Estimated Time

Possible Values: Real

Description: These are the estimated physical states of the helicopter as seen by the Flight Manager. These are updated at 2.5 Hz when the helicopter is on.

Definition

Set Estimated x **equal to** x

Receive "State Update" **from** Vehicle **Message Contents:** x, y, Altitude, Time, u, v, w, Navigation Health	T

Set Estimated y **equal to** y

Receive "State Update" **from** Vehicle	T

Set Estimated Altitude **equal to** Altitude

Receive "State Update" **from** Vehicle	T

Set Estimated u **equal to** u

Receive "State Update" **from** Vehicle	T

Set Estimated v **equal to** v

Receive "State Update" **from** Vehicle	T

Set Estimated w **equal to** w

Receive "State Update" **from** Vehicle	T

Set Estimated Time **equal to** Time

Receive "State Update" **from** Vehicle	T

Set Estimated Navigation Health **equal to** Navigation Health

Receive "State Update" **from** Vehicle	T

Function

Distance to Waypoint

Inputs to this Function: Estimated Flight Mode, Estimated x, Estimated y, Estimated Altitude, Commanded x, Commanded y, Commanded Altitude

Comment: This function returns the distance the vehicle is from the current waypoint it is headed for. This is used by Waypoint Achieved$_F$ to test if a waypoint has been reached.

Definition

Estimated Flight Mode in state TAKEOFF MODE	T

= ABSOLUTE VALUE (Estimated Altitude –Commanded Altitude)

Estimated Flight Mode in state WAYPOINT THROUGH	T

= SQUARE ROOT (SQUARE (Estimated x – Commanded x) + SQUARE (Estimated y – Commanded y))

Estimated Flight Mode in state WAYPOINT HOVER	T

= SQUARE ROOT (SQUARE (Estimated x – Commanded x) + SQUARE (Estimated y – Commanded y) + SQUARE (Estimated Altitude – Commanded Altitude))

Function

Waypoint Achieved

Inputs to this Function: Estimated Flight Mode, Estimated Time, Estimated x, Estimated y, Estimated Altitude, Estimated u, Estimated v, Estimated w, Estimated Hover Start Time, Estimated Mode Start Time, Estimated Navigation Health

Definition

Estimated Flight Mode in state GROUND MODE — T

= { Estimated Navigation Filter Status in state HEALTHY → TRUE ; else → FALSE

Estimated Flight Mode in state RUNUP MODE — T

= { Estimated Navigation Filter Status in state HEALTHY and Estimated Time - Estimated Time in Mode ≥ 30 sec. → TRUE ; else → FALSE

Estimated Flight Mode in state TAKEOFF MODE — T

= { Estimated Time – Estimated Hover Start Time ≥ Commanded Hover Time and Distance To $Waypoint_F$ < Radius of Capture → TRUE ; else → FALSE

Estimated Flight Mode in state WAYPOINT HOVER MODE — T

= { Estimated Time – Estimated Hover Start Time ≥ Commanded Hover Time and Distance to $Waypoint_F$ < Radius of Capture → TRUE ; else → FALSE

Function

Waypoint Achieved (continued)

Definition

Estimated Flight Mode in state WAYPOINT THROUGH MODE		T
-	Distance to $Waypoint_F$ < Radius of Capture	TRUE
	else	FALSE
Estimated Flight Mode in state WAYPOINT LAND MODE		T
-	Estimated Altitude < 1.0 ft. Estimated u < 1.0 ft/s Estimated v < 1.0 ft/s Estimated w < 1.0 ft/s	TRUE
	else	FALSE

State Variable

Estimated Hover Start Time

Possible Values: Real

Description: This keeps track of the time at which the vehicle commenced hovering.

Definition

Set Estimated Hover Start Time **equal to** Estimated Time

Estimated Flight Mode **in state** WAYPOINT HOVER MODE	T
Distance to $Waypoint_F$ < Radius of Capture	T

Output

GPS Ground Station Field Coordinates Message

Destination: Vehicle

Comments: Uplinks Field Reference Position (Ground GPS) to Vehicle

Triggering Event

	Eng	Op
Receive "Upload Field Settings" **from** GCU Operator	T	F
Estimated Flight Mode **in state** GROUND MODE	*	T
Receive "Initialize System" **from** GCU Operator	F	T
Navigation Filter Status **in state** DOWN	*	T

Output Action => **Send** "GPS Ground Station Field Coordinates Message (*Reference Field Latitude, Longitude, Altitude, Ψ, Variation)* " **to** Vehicle

Output

Initialize Navigation Filter Message

Destination: Vehicle

Comments: Signals Navigation Filter to Initialize Its Position (x, y), and Heading

Triggering Event	OP GUI	Eng GUI
Receive "Initialize Navigation" **from** GCU Operator ENG	F	T
Estimated Flight Mode **in state** GROUND MODE	T	T
Receive "Initialize System" **from** GCU Operator OP	T	F
Navigation Filter Status **in state** DOWN	T	*
Estimated GPS Status **in state** GPS OK	T	*

Output Action => **Send** "Initialize Navigation Filter Message (X_0, Y_0, Ψ_0)" to Vehicle

Output

GPS Request Message

Destination: Vehicle

Comments: Request GPS Data

Triggering Event

	Timeout	Eng GUI	OP GUI
Receive "Request GPS Data" from GCU Operator ENG	F	T	F
Estimated Flight Mode in state GROUND MODE	T	*	T
Receive "Initialize System" from GCU Operator OP	*	F	T
Navigation Filter Status in state DOWN	T	*	T
Timeout (MESSAGE GPS Solution Status = 0 & GPS Fix Status >1)	T	*	*

Output Action => Send "GPS Request Message" to Vehicle

State Variable

Estimated GPS Status

Possible Values: GPS OK, GPS DEGRADED, GPS DOWN

Description: This is the Flight Manager's estimate of the GPS Status on board the vehicle

Definition

Initially in state GPS DOWN

Set Estimated GPS Status to "GPS Message"

Receive "GPS MESSAGE" from Vehicle	T

State Variable

Flight Management System

Guidance Mode

Possible Values: TASKLIST , SINGLE, RETURN

Description: These are considered the major modes of the operation for the Flight Manager. It will behave differently depending on the mode it is in.

Definition

Initially Guidance Mode **in state** SINGLE

Set Operating Mode **to** TASK LIST

	OP GUI		*Eng. GUI*	
Receive "Go To Task List Mode" **from** GCU Operator	F	F	T	T
Receive "Execute Plan" **from** GCU Operator	T	T	F	F
Estimated Mode Confirmation Status **in state** ACKNOWLEDGED	T	T	T	T
Estimated Guidance Mode **in state** WAYPOINT LAND MODE	T	F	T	F
Estimated Altitude < 1 ft. or Estimated Altitude > 5ft.	T	•	T	•
Guidance Mode **in state** TASK LIST	F	F	F	F
Total Number of Tasks **in** Task List > 0	T	T	T	T
Navigation Initialization Status **in state** HAS BEEN INITIALIZED	T	T	•	•

Set Guidance Mode to RETURN

Receive "Return Home" **from** GCU Operator OP	T	T
Mode Confirmation Status **in state** ACKNOWLEDGED	T	T
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	F
Altitude < 1 ft. or Altitude > 5ft.	T	*
Navigation Initialization Status **in state** HAS BEEN INITIALIZED	T	T

Set Guidance Mode to SINGLE

Guidance Mode **in state** RETURN	T	F
Guidance Mode **in state** TASK LIST	F	T
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	T
Waypoint Achieved$_F$ (WAYPOINT LAND MODE)	T	T

Set Guidance Mode to SINGLE

Receive "Change To Single Mode" **from** GCU Operator ENG.	T	T	T
Mode Confirmation Status **in state** ACKNOWLEDGED	T	T	T
"Change To Single Mode=>Flight Mode" **in state** GROUND MODE	T	T	F
"Change To Single Mode=>Flight Mode" **in state** TAKEOFF MODE	F	F	T
Estimated Flight Mode **in state** RUNUP MODE	T	F	T
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F	T	F

Set Guidance Mode to SINGLE

Receive "Change To Single Mode" from GCU Operator ENG.	T	T
Mode Confirmation Status in state ACKNOWLEDGED	T	T
"Change To Single Mode=>Flight Mode" in state RUNUP MODE	T	T
Estimated Flight Mode in state GROUND MODE	T	F
Estimated Flight Mode in state TAKEOFF MODE	F	T

Set Guidance Mode to SINGLE

Receive "Change To Single Mode" from GCU Operator ENG.	T	T	T
Mode Confirmation Status in state ACKNOWLEDGED	T	T	T
"Change To Single Mode=>Flight Mode" in state WAYPOINT HOVER MODE	T	F	F
"Change To Single Mode=>Flight Mode" in state WAYPOINT THROUGH MODE	F	T	F
"Change To Single Mode=>Flight Mode" in state WAYPOINT LAND MODE	F	F	T
Estimated Flight Mode in state GROUND MODE	F	F	F
Estimated Flight Mode in state RUNUP MODE	F	F	F
Estimated Flight Mode in state WAYPOINT LAND MODE	F	F	*

Set Guidance Mode **to** SINGLE

	ENG. GUI		
Receive "Change To Single Mode" **from** GCU Operator ENG.	T	T	T
Mode Confirmation Status **in state** ACKNOWLEDGED	T	T	T
"Change To Single Mode=>Flight Mode" in **state** PILOT ASSIST MODE	T	T	T
Estimated Flight Mode **in state** WAYPOINT HOVER MODE	T	F	F
Estimated Flight Mode **in state** WAYPOINT THROUGH MODE	F	T	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F	F	T
Altitude > 5.0 ft.	*	F	T

Set Guidance Mode **to** SINGLE

	OP GUI		
Receive "Pilot Assist" **from** GCU Operator OP	T	T	T
Mode Confirmation Status **in state** ACKNOWLEDGED	T	T	T
"Change To Single Mode=>Flight Mode" in **state** PILOT ASSIST MODE	T	T	T
Estimated Flight Mode **in state** WAYPOINT HOVER MODE	T	F	F
Estimated Flight Mode **in state** WAYPOINT THROUGH MODE	F	T	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F	F	T
Altitude > 5.0 ft.	*	F	T
Navigation Initialization Status **in state** HAS BEEN INITIALIZED	T	T	T

State Variable

Flight Management System

↳ Total Number of Tasks

Possible Values: Integer

Description: This number represents the total number of tasks in the Task List.

Definition

Initially Total Number of Tasks **equal to** 0

Set Total Number of Tasks **to** (Total Number of Tasks + 1)

Receive "Create Waypoint" **from** GCU Operator	T

Set Total Number of Tasks **to** 0

Receive "Delete Waypoints" **from** GCU Operator	F	T
Receive "Quit" **from** GCU Operator	T	F

State Variable

Task List=>Task [i]

Possible Values: WAYPOINT HOVER MODE, WAYPOINT THROUGH MODE, WAYPOINT LAND MODE
Commanded Parameters

Description: This is the list of tasks entered by the user prior to and during the execution of a mission. There is also a condition on

Definition

Initially i = 0
Initially Task [0]=>Commanded Flight Mode**equal to** NO WAYPOINT
Initially Task [0]=>Commanded Parameters **equal to** NO PARAMETERS

Set Task List=>Task [Total Number of Tasks]=>Commanded Flight Mode**to** "New Flight Mode"
Set Task List=>Task [Total Number of Tasks]=>Commanded Parameters **to** "Position", "Altitude"

Receive "Create a New Waypoint=>Entry Complete" **from**GCU Operator **Message Contents**: New Flight Mode, Position, and Altitude	T

Clear all contents in Task List, and reset Task List Parameters to default values

Receive "Delete Waypoints" **from** GCU Operator	F	T
Receive "Quit" **from** GCU Operator	T	F

Set Current Task Index **to** Current Task Index+1

Receive "Create Waypoint" **from** GCU Operator	T	F	F
Current Task Index = 0 (Empty Task List)	T	F	F
Guidance Mode **in state** TASK LIST	T	T	T
Estimated Flight Mode = Waypoint Land	*	*	T
Waypoint Achieved$_F$ (Estimated Flight Mode) = TRUE	T	T	T
Estimated Flight Mode = Task List=>Task [i]	*	T	T
Current Task Index +1 $\leq$ Total Number of Tasks	T	T	F

Output

Mode Change Request=>Flight Mode (Output)
Mode Change Request=>Commanded Parameters (Output)
Mode Confirmation Status (State Variable)
Estimated Mode Confirmation Status

Destination: Vehicle

Possible Values: GROUND MODE, RUNUP MODE, TAKEOFF MODE, WAYPOINT HOVER, WAYPOINT THROUGH MODE, WAYPOINT LAND MODE, PILOT ASSIST MODE
Commanded Flight Parameters

Comments: Uplinks Guidance Commands to Vehicle

Special Comments: Mode Confirmation Status, a state variable, has been included with the output message, Mode Change Request, because, the conditions for triggering this output message are the exactly the same conditions for some state transitions to occur for these two particular variables. To save space, and reduce complexity they have been grouped together.

Triggering Event

Receive "Change To Single Mode" **from** GCU Operator **Message Contents**: Flight Mode, Position, Altitude, Hover Time ENG.	T
Mode Confirmation Status **in state** ACKNOWLEDGE	T

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: GROUND MODE

Set Estimated Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = GROUND MODE	T	T
Estimated Flight Mode **in state** RUNUP MODE	F	T
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	F

Send "Mode Change Request" **to** Vehicle
Message Contents: RUNUP MODE

Set Estimated Mode Confirmation Status **from** ACKNOWLEDGED**to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = RUNUP MODE	T	T
Estimated Flight Mode **in state** GROUND MODE	F	T
Estimated Flight Mode **in state** TAKEOFF MODE	T	F

Send "Mode Change Request" **to** Vehicle
Message Contents: TAKEOFF MODE, Commanded Altitude, Commanded Hover Time

Set Estimated Mode Confirmation Status **from** ACKNOWLEDGED**to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = TAKEOFF MODE	T
Estimated Flight Mode **in state** RUNUP MODE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT HOVER MODE, Commanded x, Commanded y, Commanded Altitude, Commanded Hover Time, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity

Set Mode Confirmation Status **from**ACKNOWLEDGED **to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = WAYPOINT HOVER MODE	T
Estimated Flight Mode **in state** RUNUP MODE	F
Estimated Flight Mode **in state** GROUND MODE	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT THROUGH MODE Commanded x, Commanded y, Commanded Altitude, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity, Groundspeed Mode, XNext, YNext, Radius of Capture

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = WAYPOINT THROUGH MODE	T
Estimated Flight Mode **in state** RUNUP MODE	F
Estimated Flight Mode **in state** GROUND MODE	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT LAND MODE Commanded x, Commanded y, Commanded Altitude, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity, Radius of Capture

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = WAYPOINT LAND MODE	T
Estimated Flight Mode **in state** RUNUP MODE	F
Estimated Flight Mode **in state** GROUND MODE	F

Send "Mode Change Request" **to** Vehicle
Message Contents: PILOT ASSIST MODE: u_stick, v_stick, w_stick, r_stick, planarDetent, vertDetent, yawdetent, PosHold, min_alt, StickMap, umax, umin, vmax, vmin, zdot_max, zdot_min, rmax, rmin

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

"Change To Single Mode=>Flight Mode" = PILOT ASSIST MODE	T	T	T
Estimated Flight Mode **in state** WAYPOINT HOVER MODE	T	F	F
Estimated Flight Mode **in state** WAYPOINT THROUGH MODE	F	T	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F	F	T
Estimated Altitude > 5.0 ft.	*	*	T

Output

Guidance Message5

Destination: Vehicle

Triggering Event

Receive "Pilot Assist" **from** GCU Operator OP	T
Navigation Initialization Status **in state** HAS BEEN INITIALIZED	T
Mode Confirmation Status **in state** ACKNOWLEDGED	T

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: PILOT ASSIST MODE u_stick, v_stick, w_stick, r_stick, planarDetent, vertDetent, yawdetent, PosHold, min_alt, StickMap, umax, umin, vmax, vmin, zdot_max, zdot_min, rmax, rmin

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** WAYPOINT HOVER MODE	T	F	F
Estimated Flight Mode **in state** WAYPOINT THROUGH MODE	F	T	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	F	F	T
Estimated Altitude > 5.0 ft.	*	*	T

Output

Guidance Message4

Destination: Vehicle

Comments: Uplinks Guidance Commands to Vehicle. ***Use these conditions when the Flight Manager enters Task List Mode from some other Guidance Mode.***

Triggering Event	OP		ENG	
Receive "Change To Task List Mode" **from** GCU Operator ENG.	F	F	T	T
Receive "Execute Plan" **from** GCU Operator OP	T	T	F	F
Mode Confirmation Status **in state** ACKNOWLEDGED	T	T	T	T
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	F	T	F
Estimated Altitude < 1 ft. or Estimated Altitude > 5ft.	T	*	T	*
Guidance Mode **in state** TASK LIST	F	F	F	F
Total Number of Tasks	T	T	T	T
Navigation Initialization Status **in state** HAS BEEN INITIALIZED	T	T	*	*

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: GROUND MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	T	T
Estimated Altitude > 5.0 ft.	F	F	F
Task [i]=>Commanded Flight Mode **in state** WAYPOINT HOVER MODE	T	F	F
Task [i]=>Commanded Flight Mode **in state** WAYPOINT THROUGH MODE	F	T	F
Task [i]=>Commanded Flight Mode **in state** WAYPOINT LAND MODE	F	F	T

Send "Mode Change Request" **to** Vehicle
Message Contents: RUNUP MODE

Set Mode Confirmation Status **from**ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** GROUND MODE	T	T	T
Desired Task **in state** WAYPOINT HOVER MODE	T	F	F
Desired Task **in state** WAYPOINT THROUGH MODE	F	T	F
Desired Task **in state** WAYPOINT LAND MODE	F	F	T

Send "Mode Change Request" **to** Vehicle
Message Contents: TAKEOFF MODE, Commanded Altitude, Commanded Hover Time

Set Mode Confirmation Status **from**ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** RUNUP MODE	T	T	T
Task [i]=>Commanded Flight Mode **in state** WAYPOINT HOVER MODE	T	F	F
Task [i]=>Commanded Flight Mode **in state** WAYPOINT THROUGH MODE	F	T	F
Task [i]=>Commanded Flight Mode **in state** WAYPOINT LAND MODE	F	F	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT HOVER MODE, Commanded x, Commanded y, Commanded Altitude, Commanded Hover Time, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity

Set Mode Confirmation Status **from**ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** GROUND MODE	F	F
Estimated Flight Mode **in state** RUNUP MODE	F	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	F
Estimated Altitude > 5.0 ft.	T	*
Task [i]=>Commanded Flight Mode **in state** WAYPOINT HOVER MODE	T	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT THROUGH MODE Commanded x, Commanded y, Commanded Altitude, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity, Groundspeed Mode, XNext, YNext, Radius of Capture

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** GROUND MODE	F	F
Estimated Flight Mode **in state** RUNUP MODE	F	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	F
Estimated Altitude > 5.0 ft.	T	*
Task [i]=>Commanded Flight Mode **in state** WAYPOINT THROUGH MODE	T	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT LAND MODE Commanded x, Commanded y, Commanded Altitude, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity, Radius of Capture

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** GROUND MODE	F	F
Estimated Flight Mode **in state** RUNUP MODE	F	F
Estimated Flight Mode **in state** WAYPOINT LAND MODE	T	F
Estimated Altitude > 5.0 ft.	T	*
Task [i]=>Commanded Flight Mode **in state** WAYPOINT LAND MODE	T	T

Output

Guidance Message5

Destination: Vehicle

Comments: Uplinks Guidance Commands to Vehicle. Use these conditions when the Flight Manager enters Return Mode fro some other Guidance Mode.

Triggering Event

Receive "Return Home" **from** GCU Operator OP	T	T
Mode Confirmation Statu **in state** ACKNOWLEDGED	T	T
Estimated Flight Mod **in state** WAYPOINT LAND MODE	F	T
Estimated Altitude < 1 ft. or Estimated Altitude > 5 ft.	*	T
Navigation Initialization Statu **in state** HAS BEEN INITIALIZED	T	T

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: RUNUP MODE

Set Mode Confirmation Statu **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mod **in state** GROUND MODE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: TAKEOFF MODE, Commanded Altitude, Commanded Hover Time

Set Mode Confirmation Statu **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mod **in state** RUNUP MODE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT LAND MODE Commanded x, Commanded y, Commanded Altitude =30 ft.

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** WAYPOINT LAND MODE	F	T
Estimated Altitude > 5 ft.	*	T
Estimated Flight Mode **in state** GROUND MODE	F	F
Estimated Flight Mode **in state** RUNUP MODE	F	F

Output

Guidance Message6

Destination: Vehicle

Comments: Uplinks Guidance Commands to Vehicle. Applies during normal operation.

Triggering Event

Timeout MODE CONFIRM MESSAGE	T
Guidance Initialization Status in **state** READY	T

Definition

= Guidance Message *(send the message again)*

Output

Guidance Message7

Destination: Vehicle

Comments: Uplinks Guidance Commands to Vehicle. These conditions apply only when the Flight Manager is already in Task List Mode. Ignore this when the Flight Manager first enters Task List Mode from another Guidance Mode.

Triggering Event

Mode Confirmation Status **in state** ACKNOWLEDGED	T
Guidance Mode **in state** TASK LIST	T

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: GROUND MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** WAYPOINT LAND MODE	T
Waypoint Achieved$_{\downarrow}$ (WAYPOINT LAND MODE) = TRUE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: RUNUP MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** GROUND MODE	T
Waypoint Achieved$_{\downarrow}$ (GROUND MODE) = TRUE	T

Send "Mode Change Request" to Vehicle
Message Contents: TAKEOFF MODE, Default Altitude, Default Hover Time

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** RUNUP MODE	T
Waypoint Achieved$_F$ (RUNUP MODE) = TRUE	T

Send "Mode Change Request" to Vehicle
Message Contents: WAYPOINT HOVER MODE, Commanded x, Commanded y, Commanded Altitude, Commanded Hover Time, Heading Mode, Ψ, Airspeed Mode, Nominal Velocity

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Task [i]=>Commanded Flight Mode **in state** WAYPOINT HOVER MODE	T	T	T
Estimated Flight Mode **in state** TAKEOFF	T	F	F
Waypoint Achieved$_F$ (TAKEOFF MODE) = TRUE	T	F	F
Estimated Flight Mode **in state** WAYPOINT HOVER MODE	F	T	F
Waypoint Achieved$_F$ (WAYPOINT HOVER MODE) = TRUE	F	T	F
Estimated Flight Mode **in state** WAYPOINT THROUGH MODE	F	F	T
Waypoint Achieved$_F$ (WAYPOINT THROUGH MODE) = TRUE	F	F	T

Output

Guidance Message

Destination: Vehicle

Comments: Uplinks Guidance Commands to Vehicle. These conditions apply only when the Flight Manager is already in Return Mode. Ignore this when the Flight Manager first enters Return Mode from another Guidance Mode.

Triggering Event

Mode Confirmation Status **in state** ACKNOWLEDGED	T
Guidance Mode **in state** RETURN	T

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: GROUND MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** WAYPOINT LAND MODE	T
Waypoint Achieved$_F$ (WAYPOINT LAND MODE) = TRUE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: TAKEOFF MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** RUNUP MODE	T
Waypoint Achieved$_F$ (RUNUP MODE) = TRUE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT LAND MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **in state** TAKEOFF MODE	T
Waypoint Achieved$_F$ (TAKEOFF MODE) = TRUE	T

Output

Guidance Message

Destination: Vehicle

Comments: Uplinks Guidance Commands to Vehicle. These conditions apply only when the Flight Manager is already in Return Mode. Ignore this when the Flight Manager first enters Return Mode from another Guidance Mode.

Triggering Event

Mode Confirmation Status **in state** ACKNOWLEDGED	T
Guidance Mode **in state** RETURN	T

Definition

Send "Mode Change Request" **to** Vehicle
Message Contents: GROUND MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **instate** WAYPOINT LAND MODE	T
Waypoint Achieved$_F$ (WAYPOINT LAND MODE) = TRUE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: TAKEOFF MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **instate** RUNUP MODE	T
Waypoint Achieved$_F$ (RUNUP MODE) = TRUE	T

Send "Mode Change Request" **to** Vehicle
Message Contents: WAYPOINT LAND MODE

Set Mode Confirmation Status **from** ACKNOWLEDGED **to** UNACKNOWLEDGED

Estimated Flight Mode **instate** TAKEOFF MODE	T
Waypoint Achieved$_F$ (TAKEOFF MODE) = TRUE	T