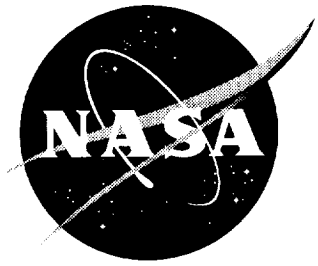
NASA/CR-2001-211022



Aviation Safety Simulation Model

User's Guide

Scott Houser
Logistics Management Institute, McLean, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS2-14361

June 2001

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Contents

FIGURES

Chapter 1
Introduction

The National Aeronautics and Space Administration (NASA) directed LMI to create a model for simulating scenarios involving aviation safety issues. This model provides the second step of a two-step analysis process. In the first step, the analyst determines the probability that a specific piece of equipment will fail. In the second step, the analyst uses simulation to discover what happens when such a failure occurs.

NASA has several heavily detailed, large-scale simulation programs at its disposal. We did not seek to make this model yet another one. The primary benefit of an LMI simulation model is that NASA and LMI analysts can work together to tailor the simulation to investigating new safety technology as the need arises. To maximize this benefit, the model features an open design that ensures that developers can extend it quickly and easily. The simulation is more valuable for its potential to evaluate future issues than for its ability to evaluate current issues. Therefore, ensuring that it has a flexible and extensible design was an even higher priority than using it for a specific application. Nevertheless, having a problem to solve is critical in maintaining proper scope and focus for the model, as well as to show that the model operates properly.

We focused on one relatively simple safety issue: terrain avoidance in the event of an avionics system failure. The terrain avoidance problem applies to current NASA safety research. Therefore, it provides an interesting context for demonstrating the model's usefulness and potential. The model allows the user to

- build a landscape of terrain objects and a flight path through that landscape,
- define a simple description of an aircraft: its speed, track, and location,
- define a set of navigation and surveillance equipment for the aircraft,
- and specify the pilot's ability to react to conditions that arise during flight.

By adjusting these parameters, the user can determine the effect of changes in instrument accuracy or pilot readiness on the outcome of an equipment failure during flight.

The program models the aircraft itself in a rudimentary way. The aircraft is little more than a point in space with a defined track and speed. The user can adjust the pilot's state of readiness and the navigation and surveillance equipment status, but not the aircraft's aerodynamic properties. Although the ability to supply a more

detailed description of the aircraft might be desirable in some situations, it was not necessary for this problem. Therefore, the developers placed their first priority on ensuring that the elements of the aircraft—its pilot, its avionics, and its collision avoidance equipment—provided sufficient flexibility to address several different types of failures and responses.

REPORT ORGANIZATION

The remainder of this report includes the following chapters and appendices:

- *Chapter 2: Description of the Model* describes the model's input and output files and explains how to run it. General users will find this chapter of the most use.
- *Chapter 3: Software Design* discusses the software's object structure and interactions. This chapter is of interest primarily to software engineers and future developers of the model.
- *Chapter 4: Modeling and Algorithms* discusses the mathematical and computational specifics of the model. This chapter is for users, designers, and developers who are interested in understanding the model in more rigorous detail.
- *Appendix: Sample Input and Output Files* provides a set of sample files to which the user can refer when the user is developing a new simulation scenario.

Chapter 2
Model Description

We developed the simulation model in MODSIM, an object-oriented language that includes several built-in simulation features. MODSIM requires a Unix operating system to run. The model answers several types of terrain avoidance questions, such as the following:

- Does the timing of the equipment failure affect the outcome?
- Will the outcome differ if the pilot is less experienced or distracted?
- How will changing the minimal allowable distance between an aircraft and terrain affect the outcome?

The model uses several input files to specify the terrain, the aircraft, and the aircraft's equipment and pilot. A configuration file called "SafetySim.config" tells the model the path and name of each input file. It also provides a path and name that the model uses to write the output. Chapter 3 provides examples of all of these files.

Running a Simulation

To run the simulation, the user must have the SafetySim executable file in his working directory. The configuration file also must be in the working directory. The remainder of this chapter discusses the configuration file and the various input files that the executable requires. Once these files are built, the user need only enter "SafetySim" at the Unix command line prompt to run the simulation.

Configuration File

The configuration file has the format shown in Figure 2-1. Note that the actual file contents are in 12 point Courier New; items that describe the file contents are in Courier New Italic. Additional comments are italicized and inside parentheses. This formatting convention holds throughout the chapter.

Figure 2-1. Configuration File

```
BEGIN HEADER
  Insert a descriptive header here.
END HEADER

BEGIN FILES
  TERRAIN FILE
   Insert terrain file name here.
  PATH FILE
   path file name
  PILOT FILE
   pilot file name here
  AVIONICS FILE
   avionics equipment file name
  AIRCRAFT FILE
   aircraft file name
  SURVEQUIP FILE
   surveillance equipment file name
  DATAREPORTER FILE
   data reporter file name
END FILES
```

The configuration file must have a header block that begins with the BEGIN HEADER label and ends with END HEADER label. Both must be in all caps, and both must reside on their own line of the file. The user can use the header block to write any comments that describe the data in the input files or the purpose of the scenario. The user also might include information such as the creation time and date of the scenario.

Similar to the header block, the files block must begin with a BEGIN FILES label and end with an END FILES label. Both must be on their own line of the file, and both must be in all caps. Likewise, the individual file labels also must be in all caps. The file name corresponding to the label must follow on the next line. If this file is in a different directory than the working directory, the user must specify a full path.

DATA STRUCTURES AND ASSOCIATED INPUT FILES

As the configuration file shows, the model requires seven separate input files. The number of files may be intimidating, but they are designed to minimize the amount of work that a user will need to do. For example, rather than include the same pilot information in a set of several aircraft, the user can place a single pilot description in the pilot file and refer to the pilot's name in every aircraft that will use that information.

When specifying a distance, a coordinate, or a speed, the user must maintain consistent units. For example, it is acceptable to express an aircraft's location in meters or feet. The speed of the aircraft, however, must be expressed in m/s or ft/s, depending on the system used. It is not acceptable to specify positions in feet and speeds in knots, for example. In specifying angular quantities—for example, a heading error—the user should express those values in radians. There is one notable exception: To specify the angle of an intercept's signal trapezoid, the user should express those values in degrees.

In addition to using consistent units within each file, the user also must use consistent units across all files. For example, the user cannot specify aircraft locations in feet and terrain locations in meters.

Pilot

The pilot is the simplest object in the simulation. The purpose of this object is to provide the probability that a pilot would discover a condition or start some task during a given simulation cycle. In other words, the pilot object generates a random reaction time to some stimulus that a real-world pilot might experience. The aircraft object actually performs the task; the pilot merely tells the aircraft when to start.

Because the pilot is a simple object, its file description also is simple (see Figure 2-2).

Figure 2-2. Pilot File

```
PILOTLIST
  number of pilot descriptions
  pilot name
   pilot status
     Pready  Pbusy  Pverybusy
  ... (remaining pilot descriptions)
```

The file starts with a required label, PILOTLIST. The next line contains the number of pilot descriptions in the list. Each pilot description consists of five pieces of information on three lines. The pilot's name appears on the first line of the file. It can be any alphanumeric word (it cannot have spaces). The pilot's status appears on the next line. It can have the following values: 1 for 'ready', 2 for 'busy', and 3 for 'very busy'. The third line contains three probability values associated with each of the three values of the pilot's status. Each probability must be a real number between 0.0 and 1.0.

To understand how the pilot uses this information, consider the example of a pilot description in Figure 2-3.

Figure 2-3. Typical Pilot Description

```
StandardPilot
  2
   .75   .55  .02
```

This pilot's name is "StandardPilot," and the pilot has a status value of 2. Thus, the pilot will begin the simulation in a busy state, and its probability of reacting to a stimulus within a given simulation cycle is 55 percent. With a simulation cycle of one second, there is a 55 percent chance that the pilot will successfully react within the first possible second. The model waits three cycles after providing the pilot with a stimulus before allowing it to respond. Therefore, the best reaction possible is a response that occurs in four seconds. The pilot uses a uniform random number generator to determine success or failure.

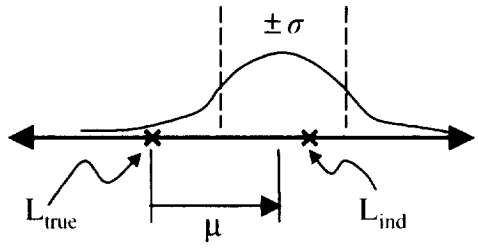
If the pilot fails to react on the first try, it tries again on each subsequent simulation cycle, until it succeeds or times out. The pilot times out after 60 cycles. In other words, if the pilot does not react within 60 cycles, it does not react at all. Therefore, the overall probability of a successful reaction is a function of a geometric random variable with a maximum number of trials equal to 60.

Avionics Equipment and Avionics Package

Whereas the pilot description is the simplest in the model, the avionics equipment description is the most complex. To explain how the avionics object works in the model, we start with a description of how it computes an indicated direction, location, or speed variable.

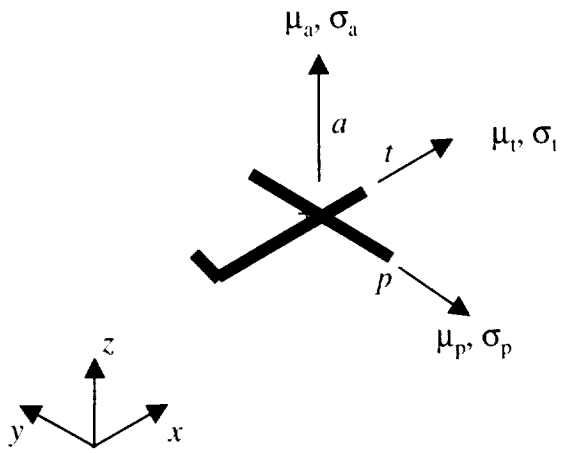
To start this discussion, we first simplify the aircraft to a point location on a single coordinate axis (see Figure 2-4). The model assumes that the error is normally distributed, with a mean μ and a standard deviation σ. The mean corresponds to a steady-state bias error; the standard deviation provides a measure of the dispersion of the randomly chosen error value. Therefore, if the actual location is L_{true}, the indicated location will most often fall within $\pm\sigma$ of $L_{true} + \mu$.

Figure 2-4. Adding Error to a True Location



The avionics object computes an error as discussed above for each of the aircraft's three local coordinate axes: the tangential axis t, the perpendicular (or lateral) axis p, and the altitudinal axis a (see Figure 2-5). Therefore, the user must supply six variables, a mean, and a standard deviation for each axis to specify the location error that a given avionics system will supply.

Figure 2-5. Aircraft Location Error Parameters

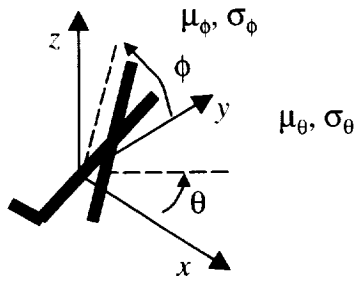


The avionics system measures velocity in addition to location, so the user also must provide a mean and standard deviation for each velocity parameter. Consider that velocity is a function of the aircraft's speed s and its track. Its track is a function of θ, the rotation about the z axis, and ϕ, the angle of rotation about the x axis:

$$v = f(s,\theta,\phi)\,. \qquad \text{[Eq. 2-1]}$$

Therefore, to compute an indicated velocity on the basis of the aircraft's true velocity, the user must define six more values: a mean and a standard deviation associated with s, θ, and ϕ. Figure 2-6 shows the error parameters associated with track error. The speed error also requires a mean and standard deviation, but it is dimensionless and therefore is not shown.

Figure 2-6. Aircraft Track Error Parameters



An avionics equipment description requires three more error values to fully compute the aircraft's location and velocity. These values are map errors. The other types of location errors will lead an aircraft to become uncertain of its own location; a map error will make it uncertain of the location of every terrain feature. In the model, these two uncertainties are essentially equivalent: For example, if every mountain on a map is 500 feet further to the west than the aircraft realizes, the error is equivalent to one in which the aircraft is 500 feet further to the east than it realizes.

Map errors have only a steady-state bias value. They do not have a corresponding standard deviation because the model assumes that they arise from data errors, as opposed to instrument errors. For example, one possible way for a map error to occur is if a synthetic vision system (SVS) database records the location of every terrain item relative to a given reference point and that reference point is in error. Because data errors do not vary from reading to reading, map errors also do not vary. The user specifies a map error in terms of the global reference frame, as opposed to the reference frame of the aircraft. Therefore, the three values for map error correspond to an offset on each of the x, y, and z axes.

In summary, each avionics equipment description involves 15 error values: a mean and standard deviation each for tangential, perpendicular, and altitudinal location; a mean and standard deviation each for heading with respect to rotation about the x axis, heading with respect to rotation about the z axis, and speed; and a value each for the map errors relative to the x, y, and z axes. What if the avionics equipment fails? The equipment description includes two sets of these 15 variables: One set corresponds to a normal operating mode, and one set corresponds to a failed mode. In addition, the user can specify an unlimited number of avionics equipment descriptions in an aircraft's avionics package. In other words, the avionics equipment onboard a given aircraft can have a primary system, a secondary system, a backup secondary system, and so on. When the simulation inserts a failure into an aircraft's primary avionics equipment, the aircraft will respond by asking the pilot if it recognizes the problem. If the pilot responds yes, the aircraft will shift its active avionics equipment from the failed mode of the primary equipment to the normal operating mode of the first backup equipment.

Avionics File

Like the pilot file, the avionics file has a required label, "AVIONICSLIST," that must be in all caps and must reside on its own line at the beginning of the file (see Figure 2-7). Each avionics package begins with its own name and the number of equipment descriptions in the package. Each equipment description also begins with its own name. The 15 error parameters associated with the equipment's normal operating mode follow: first the location parameters, then the velocity parameters, and finally the map error parameters. These values should be separated by white space (no commas). The parameters for the failed mode follow the parameters for the normal mode.

Figure 2-7. Avionics File

AVIONICSLIST

number of avionics packages

name for first avionics package

number of equipment descriptions in package

name of first equipment description

μ_t σ_t μ_p σ_p μ_a σ_a

μ_ϕ σ_ϕ μ_θ σ_θ μ_s σ_s *Nor* } *op. mode*

M_x M_y M_z

μ_t σ_t μ_p σ_p μ_a σ_a

μ_ϕ σ_ϕ μ_θ σ_θ μ_s σ_s *Fa.* } *mode*

M_x M_y M_z

name of primary backup equipment description

...

name of final backup equipment description

...

Initial Status Initial Equipment

... (remaining avionics packages)

The package lists all of its equipment descriptions in proper order. It concludes with two numbers: the initial status and the initial equipment. If the initial status is 1, the avionics package starts the simulation in normal operating mode; if it is 2, it starts the simulation in failed mode. The initial equipment tells the package which equipment is active at the beginning of the simulation. If this value is 1, the active equipment is the primary system. If it is 2, the active equipment is the first backup system, and so on.

Given the complexity of the avionics package, a few examples help. The avionics file in Figure 2-8 describes a typical avionics package. In the normal operating mode, the aircraft location error is normally distributed about the actual location because all of the mean error values are 0.0 for the location parameters. The standard deviation on each location axis is 20.0, which (if we assume units of feet) means that the indicated location is equally accurate along each coordinate axis and typically will fall within $20\sqrt{3}$ feet of the true location. There is no error associated with the aircraft's velocity and no map error; the user is concerned only with location errors in this case. The failed mode is similar to the normal operating mode, except that the location error becomes 12 times larger.

Figure 2-8. Typical Avionics File

```
AVIONICSLIST
   1
    TypicalAvionicsPkg
       1
    PrimarySystem
      0.0 20.0  0.0 20.0  0.0 20.0
      0.0 0.0   0.0 0.0   0.0 0.0
      0.0 0.0 0.0
     0.0 240.0  0.0 240.0  0.0 240.0
      0.0 0.0   0.0 0.0   0.0 0.0
      0.0 0.0 0.0
      1  1
```

The file in Figure 2-9 shows an avionics system with a map error only. This package also includes only one item of equipment. The user specifies no location or velocity measurement errors; all of the parameters related to those errors are 0.0 in the normal operating mode and in the failed mode. The user specifies a map error of 500 feet along the *x* axis, so the aircraft will believe that it is 500 feet further away along the *x* axis from its true location at all times.

Notice that the failed mode and normal operating mode have the same values and therefore are equivalent. This equivalence highlights the following point: The user does not have to create a different failure mode and normal operating mode, then instruct the model to insert an avionics failure in at some time during the simulation. The user also can write the normal operating mode to include the desired failure and fly the aircraft in that mode during the simulation. In that case, the failure mode is ignored.

Figure 2-9. Avionics Package with Map Error

```
AVIONICSLIST
   1
    MapErrAvionics
       1
    PrimarySystem
      0.0 0.0   0.0 0.0   0.0 0.0
      0.0 0.0   0.0 0.0   0.0 0.0
      500.0 0.0 0.0
```

```
0.0 0.0  0.0 0.0  0.0 0.0
 0.0 0.0  0.0 0.0  0.0 0.0
500.0 0.0 0.0
1  1
```

Surveillance Equipment and Surveillance Package

The surveillance equipment is similar to an avionics package, though much less complex. The surveillance equipment determines the indicated distance from the aircraft to the closest terrain object. It also can determine the indicated distance, relative velocity, and relative position with respect to the closest aircraft; this functionality is not necessary for a terrain avoidance problem, however. The model will use this capability when it is extended to address traffic scenarios involving near-miss situations.

The surveillance equipment description assumes that the distance measurement error is normally distributed, with no steady-state bias. Therefore, the user must specify only one value to define that error. The user also must define six velocity error parameters (similar to the avionics equipment). The model ignores those values, however; they are there in anticipation of traffic scenario extensions. The surveillance equipment requires one remaining item: a maximum range. For example, a surveillance radar may have a range of 50 miles, whereas visual indications (after all, the pilot's eyes are surveillance equipment) may have a range of 500 feet or less, depending upon weather situations. If the surveillance equipment computes an indicated distance beyond its range, it will tell the aircraft that it sees nothing.

The surveillance equipment does not have a failed mode; it only has a normal operating mode. A surveillance package has two pieces of equipment: primary equipment and backup equipment. As a result, the surveillance equipment input file structure is somewhat simpler than that of the avionics equipment (see Figure 2-10). As usual, the first line of the file contains a label (in this case, "SURVEQUIPLIST"). The number of surveillance equipment packages immediately follows the label. Each package begins with its own name, followed by the name of the primary equipment.

The primary equipment description starts with σ_d—the standard deviation of the distance error distribution. Then come six values that describe a velocity error distribution; they are identical to those used in the avionics equipment description to specify a velocity error distribution. The last item in the primary equipment description is its maximum range, R. The backup equipment is identical in format to the primary equipment. The package description concludes with an initial status enumeration. If the initial status is 1, the simulation will start with the primary equipment active. If it is 2, it will start with the backup equipment active.

Figure 2-10. Surveillance Equipment File

```
SURVEQUIPLIST
Number of surveillance equipment packages
Surveillance equipment package name
  Primary equipment name
   σd
   μφ σφ μθ σθ μs σs
   R
  Backup equipment name
   σd
   μφ σφ μθ σθ μs σs
   R
  Initial Status
... (remaining surveillance equipment packages)
```

Figure 2-11 shows a typical input file for surveillance equipment. The package name is "TCASRadar"; its primary equipment is called "Operating," and its backup system is called "VisualOnly." Assume that all distances are expressed in feet. The backup system appears to be more accurate than the operating system: The standard deviation of its distance error distribution is only 50 feet, whereas the primary system's standard deviation is 75 feet. The primary equipment has a range of 50,000 feet (approximately 10 miles), however, whereas the backup has a range of only 500 feet. A pilot who waits until he is within 500 feet of a terrain obstruction will have a good idea of the obstruction's location. By the time he sees the obstruction, however, he may be too close to avoid it.

Figure 2-11. Typical Surveillance Equipment Input File

```
SURVEQUIPLIST
1
TCASRadar
  Operating
   75.0
   0.0   0.05236 0.0   0.05236  0.0 15.0
   50000.0
  VisualOnly
```

```
 50.0
 0.0 0.15 0.0 0.15 0.0 100.0
 500.0
1
```

Path

The path data structure provides sufficient information to allow an aircraft to approach from a cornerpost through a set of waypoints until it intercepts the glide slope and localizer and finally lands. The terrain avoidance routines that the model currently runs do not require the aircraft to land, so the approach portion of the path is all a user need worry about. In this section, however, we discuss both the approach and landing portions of the path description, for completeness.

Each path typically consists of one approach and one landing. To create a list of paths, the user creates a list of approaches and a list of landings. The user then creates the paths by selecting an approach and landing from the respective lists.

APPROACH

The approach is primarily a set of path points. Each path point has a designated type, name, and coordinate triple. The approach description itself has a name, followed by the number of points and an ordered listing of each point. The list proceeds from a cornerpost to a handoff point, where the aircraft presumably would intercept the localizer and glide slope.

In the path input file, an approach list has the format shown in Figure 2-12. The list has a label of "APPROACHLIST," followed by the number of approaches in the list. The path points typically start with a cornerpost, follow with a number of waypoints, and end with a handoff point. The type of path point currently has no meaning in the simulation; when it becomes necessary to introduce more complexity, however, the type of point will tell the aircraft to end certain tasks and begin a new task. For example, when the aircraft reaches a handoff point, it will know to fly the remaining portion of its landing by using the runway intercept signals.

Figure 2-12. Approach List

```
APPROACHLIST
Number of approaches
  Name of first approach
  Number of path points
  Cornerpost
   Name of cornerpost
```

```
 X  Y  Z
Waypoint
 Name of waypoint
 X  Y  Z
... (remaining waypoint descriptions)
Handoffpoint
 Name of handoff point
 X  Y  Z
...  (remaining approach descriptions)
```

The sample approach list in Figure 2-13 shows how one might look. This approach list has two approaches: a normal approach and a breakout approach. A breakout approach is not really an approach at all. In the aircraft terrain avoidance problem, the aircraft counters certain abnormal situations by breaking out of its normal approach and flying to a given location and altitude. At that time, it becomes an inactive part of the simulation. This designated location and altitude is called the breakout point, and the path that the aircraft flies to get there is called—for lack of a better term—a breakout approach. In future extensions, the aircraft may hold at the breakout point and attempt to reenter the approach queue. In the current model, it simply becomes an inactive simulation object when it reaches the breakout point.

Figure 2-13. Typical Approach List in Path File

```
APPROACHLIST
2
  ApproachToRunway12
  4
  Cornerpost
   RW12CP
   50000.0  0.0  1000.0
  Waypoint
   RW12WP1
   40000.0  200.0  800.0
  Waypoint
   RW12WP2
   20000.0  500.0  600.0
  Handoffpoint
   RW12WP1
```

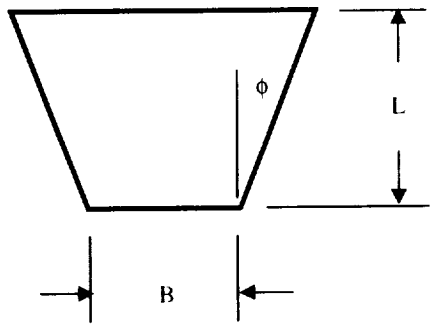
```
 10000.0   800.0   400.0
BreakoutFromRW12Approach
2
Waypoint
 RW12BreakWP1
 10000.0   20000.0   1200.0
Breakoutpoint
 RW12Breakpt
 10000.0   50000.0   2000.0
```

LANDING

The landing description consists of a glide slope intercept, a localizer intercept, and a runway. The glide slope and localizer intercepts have a signal area that is approximated by a trapezoid. When an aircraft is inside an intercept's trapezoid and within the required altitude range, it can pick up the intercept signal.

Three parameters define an intercept's signal trapezoid: the length of the trapezoid along the base, *B*; the length of the trapezoid along its perpendicular, *L*; and the angle ϕ (Figure 2-14). Note that the angle should be expressed in degrees, not radians, for an intercept.

Figure 2-14. Intercept Signal Area



The midpoint of both types of intercepts coincides with the mouth of its corresponding runway. The perpendicular of an intercept is parallel to the runway's orientation. A localizer intercept has a minimum and maximum altitude for its signal; a glide slope does not have explicit altitude limits.

A runway is a three-dimensional (3D) point in space supplemented by a two-dimensional (2D) orientation vector and a length. The altitude of the runway can be set by convention to zero or to the runway's height above sea level.

The landing list's label is "LANDINGLIST" (see Figure 2-15). Each landing has a name. The label for the localizer intercept ("Localizer") immediately follows the landing name. The localizer's signal trapezoid parameters follow. After that comes the label "Altitudes," followed by the localizer's low and high altitude ranges. The glide slope specification includes its own label ("GlideSlope"), followed by its signal trapezoid parameters. The runway description follows the glide slope description. It consists of a label ("Runway"), followed by the 3D coordinate triple of its mouth, the 2D coordinate double of its orientation vector, and its length. The simulation currently does not use the landing description, so we do not include an example.

Figure 2-15. Landing List

```
LANDINGLIST
Number of landings
  Landing name
  Localizer
   B  L  ϕ
   Altitudes
   Low limit  High limit
  GlideSlope
   B  L  ϕ
  Runway
   Runway name
   X Y Z (location)
   X Y   (orientation)
   Length
  ... (remaining landing descriptions)
```

The user builds a path list by referring to the approach list and the landing list that precede the path list in the input file (see Figure 2-16). The path description includes a name for the path, followed by an "Approach" label and the name of an approach from the approach list. A "Landing" label and a landing name chosen from the landing list follow. If the user desires, he can insert a "NoLanding" label in place of the landing label and name;. The model will leave the landing portion of that path empty. Because landing the aircraft is not a part of the current simulation, use of the "NoLanding" feature is wise from a time-saving and error-saving perspective.

Figure 2-16. Path Input File

```
(Approach list)
...
(Landing list)
...
PATHLIST
Number of paths
  Path name
  Approach
   Approach name
  Landing
   Landing name
... (remaining path descriptions)
```

To build a breakout path from a breakout approach, the user must build a path, use the breakout approach for its approach, and specify "NoLanding." The aircraft specification expects a breakout path, rather than a breakout approach, so this step is necessary.

Aircraft

All of the preceding object descriptions are part of an aircraft. The user builds an aircraft by referring to a pilot from the pilot list, a path from the path list, and so on. The user also must specify a location, track, and speed for the aircraft, using a moving point. A moving point is simply a 3D point that has been supplemented with a type, name, heading, and speed. The aircraft also requires two minimum distances: a minimum separation from aircraft and a minimum proximity to terrain.

In addition to the foregoing structures and values, the aircraft has two other items that determine how it will behave during the simulation. The first is an action sequence. An action sequence is a number followed by a keyword that describes the aircraft's overall plan. The number is the amount of simulation cycles from the start of the simulation when the aircraft will begin to act. (By convention, one simulation cycle corresponds to 1 second.) The only keyword implemented to date is "Approach." When the aircraft is supplied with this keyword, it will fly the approach portion of its path from cornerpost to handoff point and stop. Keywords such as "ApproachAndLand," "Land," "LandAndTaxi," and so forth will be implemented as they become necessary to address other issues.

The second item of simulation control information is the event list. An event list has three parts. The first part is a keyword that names the affected equipment ("Avionics," "Pilot," or "SurvEquip"). The second part is an enumeration

(1, 0, –1). A value of 1 means that the aircraft should upgrade the equipment if possible. In an avionics upgrade event, the aircraft switches the active avionics equipment to the next level up: The secondary backup would switch to the primary backup, for example. A pilot upgrade event would raise the pilot's readiness status—for example, from "busy" to "ready." A surveillance equipment upgrade event would change its status from backup mode to operating mode. A value of 0 corresponds to a failed uncovered event for avionics equipment and a downgrade for the pilot and surveillance equipment. An avionics failed uncovered event changes the avionics system's status from normal operating mode to failed mode. If the pilot successfully detects this event, the aircraft will automatically downgrade the avionics to its next available backup system and restore the status to normal operating mode. A value of –1 corresponds to a downgrade for all three objects. The third item in an event is the simulation cycle in which the event occurs.

The aircraft input file starts with a moving point list (see Figure 2-17). Each moving point has a type, a name, and the following data: a 3D coordinate triple (x, y, z) that specifies the point's initial location, a track vector T that specifies the point's initial heading, and the speed of the point, s. The aircraft has no aerodynamic properties. As far as the simulation is concerned, it literally is a 3D point moving through space. Although this simplification reduces the realism of the simulation, it makes the equations of motion easy to formulate. The simplification also allowed the programmer to place a higher priority on interactions between the aircraft and its various components.

Figure 2-17. Moving Point List

```
MOVINGPOINTLIST
Number of moving points
   Type of moving point
     Name of moving point
     x  y  z
      Tx  Ty  Tz
      s
... (remaining moving point descriptions)
```

The moving point list belongs at the top of the aircraft file. The aircraft list follows (see Figure 2-18). Each aircraft is an aggregate of components defined in the various other input files, supplemented by its separation distances, action sequence, and simulation events. Unlike the other items, the aircraft refers to the moving point by *type*, not by name.

Figure 2-18. Aircraft File

```
(Moving point list)
...
AIRCRAFTLIST
  Number of aircraft
   Moving point type
     Aircraft name
     Pilot
        Pilot name
     Avionics
       Avionics package name
     Path
         Path name
     BreakoutPath
         Breakout path name
     SurvEquip
         Surveillance equipment name
     MinimumDistances
       Aircraft separation   Terrain separation
      ActionSequence
       Start Time  Approach
     Events
        Number of events
     Equipment Keyword Upgrade/Downgrade/Fail Time
```

A typical aircraft input file might look like Figure 2-19. The aircraft name is "Aircraft1." It gets its initial speed, track, and position from the Boeing737 moving point. The moving point will start the simulation in a slight descent, at a little more than 150 ft/s. It uses a pilot named "StdPilot," an avionics package named "GPSBasedAvionics," and surveillance equipment named "GPSBasedSurvEquip." It must maintain a minimum distance of 750 feet to other aircraft and 1,000 feet to terrain. If it comes within 1,000 feet of terrain—as measured by its surveillance equipment—it will break out of its normal approach route, "JuneauApproachToRunway26," and fly the "JuneauBreakout" path.

Figure 2-19. Typical Aircraft File

```
MOVINGOBJECTLIST
  3
   Boeing737
     MP1
    50000.0   30000.0   5000.0
     .9998   0.0   -.0199
     150.2
AIRCRAFTLIST
  1
   Boeing737
     Aircraft1
    Pilot
       StdPilot
    Avionics
     GPSBasedAvionics
    Path
       JuneauToRunway26
    BreakoutPath
       JuneauBreakout
    SurvEquip
       GPSBasedSurvEquip
    MinimumDistances
      750 1000
     ActionSequence
      1500.0   Approach
    Events
     5
     Avionics -1 2000
     Pilot -1 2000
     Avionics 0 2500
     Pilot 1 3000
     SurvEquip -1 3500
```

The aircraft's action sequence indicates that it will begin its approach 1,500 seconds (or 25 minutes) after the start of the simulation. On the way, five events should occur: The avionics system will need to go to its backup system, and the pilot will become busy at 2,000 seconds. At 2,500 seconds, the backup avionics system will fail. As a result, the aircraft will fly with the failed system or, if the pilot detects the problem, switch to the secondary backup system (if one exists). After 3,000 seconds, the pilot will cease to be busy and return to ready status. After 3,500 seconds, the surveillance equipment will shift from the normal operating system to the backup system.

Although the navigation and surveillance systems both depend on the global positioning system (GPS)—at least, if their names are indicative of their configuration—the simulation treats them as independent, distinct systems. Therefore, it is perfectly acceptable for the avionics system to fail catastrophically while the surveillance equipment remains fully operational. In designing an event sequence, the user should consider possible interrelationships between the aircraft's components.

Terrain

The user builds a terrain map for the simulation by creating a list of terrain objects. Each object is a simple rectangular solid. To build more complex terrain, the user can create several terrain objects stacked on top of each other, with various orientations. For most scenarios, this level of fidelity should be acceptable. If a scenario requires more realistic topography, the terrain object is based on a set of geometry data structures that will allow it to take on more complexity.

The first step in defining a terrain object is to define its eight vertices in a point list. Each point in the list has a unique ID number and a 3D coordinate triple (see Figure 2-20).

Figure 2-20. Terrain Point List

```
POINTLIST
number of points in list
id1  x1  y1  z1
id2  x2  y2  z2
...
idn  xn  yn  zn
```

The terrain list follows the point list in the terrain input file. Each terrain object has the format shown in Figure 2-21. The "Lower" and "Upper" keywords are required labels that specify the lower and upper face of the rectangular solid,

respectively. The "id_{Li}" and "id_{Ui}" refer to the ID numbers of points in the point list. Several rules must be obeyed in specifying vertices for the terrain object:

- Each vertex in the lower face must have the same z coordinate value; the same holds true for each vertex in the upper face.
- The vertices in each face must be listed in counterclockwise order, as viewed from the top.
- The first vertex in the upper face must be directly above the first vertex in the lower face.

Although these restrictions are somewhat onerous to the user, they make it easier for the terrain object to use advanced geometry routines from the more generic 3D solid data structure on which the object is based. It also allows the user more freedom in orienting the object.

Figure 2-21. Terrain Description

```
Terrain object name

Lower
  id_L1 id_L2  id_L3  id_L4
Upper
  id_U1 id_U2  id_U3  id_U4
```

The sample terrain input file in Figure 2-22 shows how the point list and the terrain list work together. The first four points in the list make up the terrain object's lower face; the second four points make up the upper face. The user can verify that each of the four points on the lower and upper faces have the same elevation, that points 1–4 and points 1–5 are listed in counterclockwise order as viewed from above, and that point 5 is directly below point 1.

Figure 2-22. Typical Terrain Input File

```
POINT LIST
1 39470.1 100469.3 0.0
2 52156.2 113155.2 0.0
3 45813.2 119498.5 0.0
4 33127.0 106812.4 0.0
5 39470.1 100469.3 3000.0
6 52156.2 113155.2 3000.0
7 45813.2 119498.5 3000.0
```

```
8 33127.0 106812.4 3000.0


TERRAINLIST
1
Obstruction1
  Lower
  1 2 3 4
  Upper
  5 6 7 8
```

Data Reporter

The data reporter is a simple class that collects relevant output during a simulation and records it in an output file, with a name that is based on the data reporter input file. For example, if the user names the data reporter input file "sim.data," the simulation will dump its output to the file "sim.data.out." The data reporter input file contains only two items of information: the minimum terrain distance for the simulation and the full printout flag. Because the structure of the input file is relatively obvious, we skip directly to an example (Figure 2-23).

Figure 2-23. Typical Data Reporter Input File

```
MINIMUM TERRAIN DISTANCE
  500
FULL PRINTOUT (Y/N)
  Y
```

The minimum terrain distance is separate from the minimum terrain distance given to each aircraft. If the aircraft comes closer to a terrain object than its own minimum distance, it will take action to correct the situation by breaking out of its approach. If the aircraft gets closer than the data reporter's minimum terrain distance, it is assumed that the aircraft is in imminent risk of a crash.

If the user selects "Y" for the full printout flag, the data reporter will record the following data for every aircraft during every simulation cycle:

- The aircraft name
- The simulation time
- The indicated position
- The true position

- The true separation distance from the closest terrain object.

In every case, the data reporter also will record various simulation events of note. For example, it would record when an aircraft reaches a waypoint, when another aircraft breaks out of an approach, and when a third aircraft crashes. Each message will have the simulation time and the aircraft name.

Chapter 3
Software Design

The simulation's narrow focus on aircraft terrain avoidance results in few complex interactions between its actors. The aircraft will complete their approach unless they get too close to terrain. If they get too close, they will dodge the terrain and fly a safe missed approach path to a single breakout point. Therefore, there is little need to describe the model dynamically (using state diagrams). This design description focuses on the simulation's object hierarchy, using Rumbaugh's Object Modeling Technique.

The simulation world object controls all of the other objects in the model and launches the simulation. It also gives the data reporting class the opportunity to interrogate each object of interest and provide output for important events that occur in the simulation. These events might include a missed approach, a pilot's failure to discover an equipment failure, or a controlled flight into terrain.

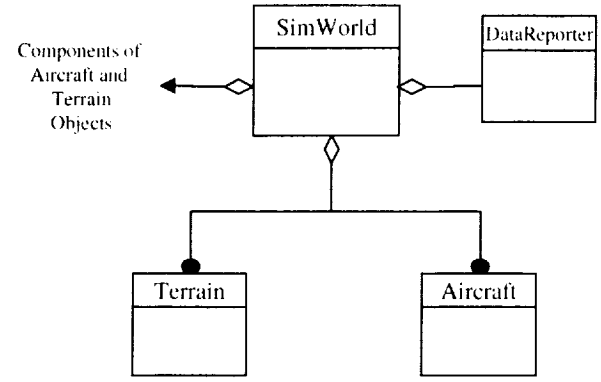
The terrain and aircraft are the primary objects in the simulation world; all other objects are components of one or the other of these objects. The remainder of this chapter discusses the organization of the simulation world as a set of lists of simulation components and describes the structure of the terrain and aircraft objects in detail.

THE SIMULATION WORLD

The simulation world loads all of its objects from a set of input files, launches the simulation, and provides a central vantage point for detecting noteworthy events. It owns a terrain objects list, an aircraft list, and a data reporter object (see Figure 3-1). It also owns lists of components that are used to build each aircraft and terrain object.

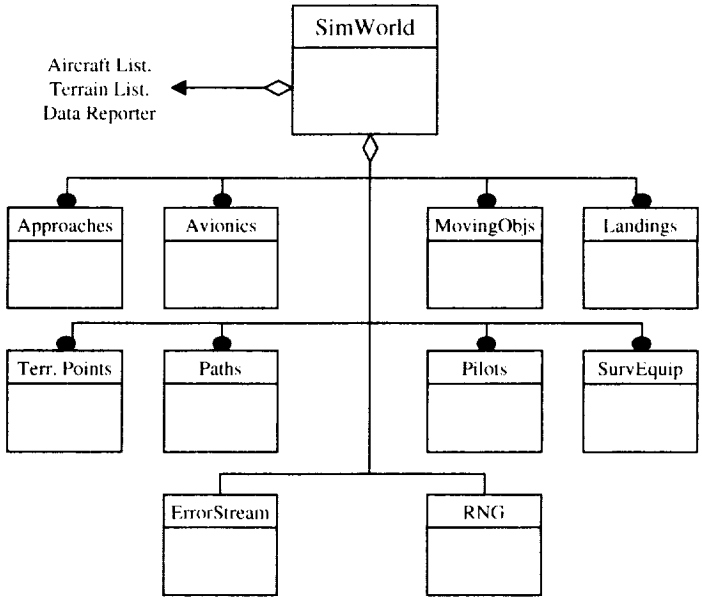
The simulation world feeds each object the information it needs to perform its set of actions. For example, the simulation world provides each aircraft with a reference to its closest terrain object. The aircraft interrogates that terrain object to determine its indicated distance. The simulation world also tracks the true distances from each aircraft to each terrain object. It feeds that information to the data reporter, which decides whether an aircraft has strayed close enough to a terrain object to warrant noting it in the output file. The simulation world also accepts messages from the aircraft—for example, the aircraft notes when it breaks out of its approach path—and relays those messages to the data reporter for processing. This central control reduces the interactions between the classes in the model. The reduced interactions enhance the model's maintainability and extensibility.

Figure 3-1. Simulation World and Major Components



The simulation world loads the items required to build the terrain and aircraft objects separately (see Figure 3-2). For example, each terrain object refers to a set of points in a master list to create its boundaries. This design allows several terrain objects to use the same point as a vertex. Similarly, several aircraft can refer to the same avionics object, for example. Although building terrain and aircraft objects from external lists adds to the programming complexity of the data loading procedure somewhat, it simplifies the input file structure and eliminates redundant effort for the user.

Figure 3-2. Other Simulation World Component Lists



All of the component lists in the simulation world structure hold aircraft components, except for the terrain point list. The structure also includes two objects in addition to the component lists: An error stream object provides troubleshooting data to an output file when the program discovers an input file error, and a random number generator (RNG) provides random numbers to all of the objects that

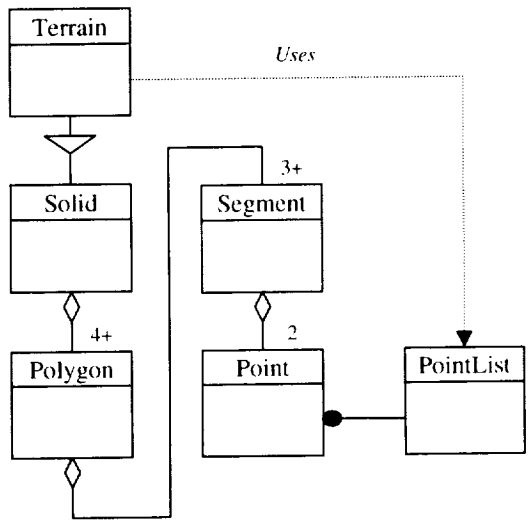
use random variables. For example, each avionics object refers to the RNG when it provides an indicated distance between the aircraft and its next waypoint.

TERRAIN OBJECT AND GEOMETRY MODELING

A consistent and robust geometry framework is critical for creating an aircraft safety simulation. The safety model borrows geometry algorithms and objects from the Flight Segment Cost Model, a module of the Aviation Systems Analysis Capability that discusses the geometry-based algorithms used in this model. The terrain object uses all of the geometry classes, so it provides a convenient context for discussing their class structure.

The terrain object is a specialization of the more generic solid class (see Figure 3-3). The solid is an aggregate of its boundary elements: four or more polygon objects. Similarly, the polygon is an aggregate of three or more segments, and the segment is a pair of points. To facilitate easier input/output routines, we created a point list object as well. The terrain object chooses its eight vertices from a point list; then it uses those vertices to construct its six rectangular faces.

Figure 3-3. Terrain Object and Geometry Classes

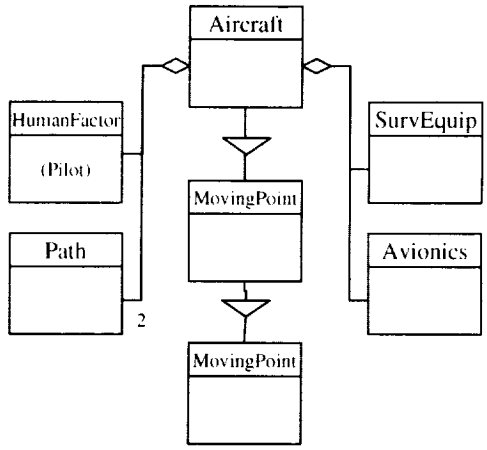


The terrain object uses the solid's geometry routines to compute distances to aircraft and to detect when an aircraft has entered its boundaries. When an aircraft lies on or inside the boundary of a terrain object, we know that it has crashed. Most of the physical objects in the model are child objects of one or more geometry objects or are composed of one or more geometry objects. For example, an aircraft is a child of the point object because the model represents aircraft as 3D locations with headings and speeds. To determine the distance between an aircraft and a terrain object, the aircraft feeds its location to the terrain object, which checks its distance relative to each of its six faces.

AIRCRAFT OBJECT

The aircraft object is a child of the moving point object—which, in turn, is a child of the point object (see Figure 3-4). The aircraft has one human factor object, two paths, a surveillance equipment object, and an avionics object. The human factor object contains routines that allow the aircraft to determine whether the pilot will detect an error condition, such as a gauge failure. The aircraft's two flight paths include a normal approach and a breakout path. The breakout path provides a safe route to take in case of a major equipment failure or unacceptably close proximity to terrain or other aircraft. The surveillance equipment object and avionics object provide two functions that are distinct from a programming perspective: surveillance (tracking other objects and their proximity to the aircraft) and navigation (tracking the proper path). From a physical perspective, one or more pieces of equipment may share in these responsibilities.

Figure 3-4. Aircraft Object



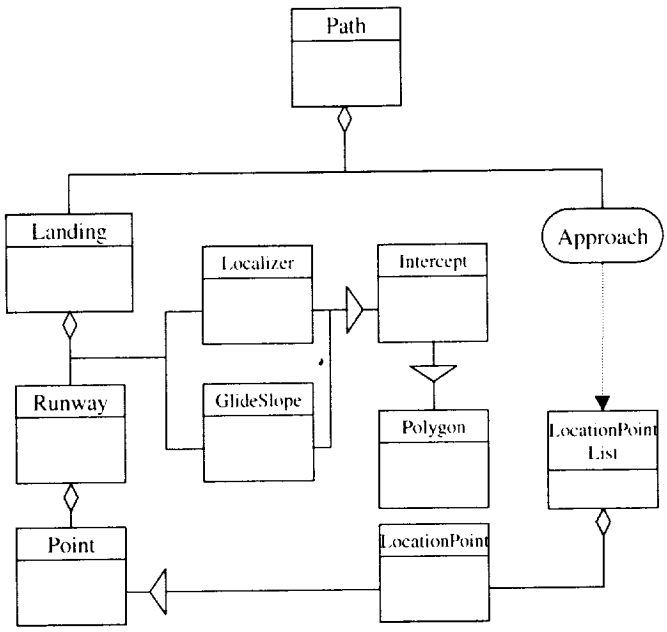
The human factor object is a relatively simple structure. The path object and the navigation/surveillance equipment are considerably more complex. The following two sections describe them further.

PATH

In narrowing the scope of the model to address terrain avoidance issues first, we left many of the design features from the original design unimplemented. For example, we have not implemented controllers because they are not necessary for a simple terrain avoidance scenario. The path, however, has several implemented features that the current model does not use. These features will allow the model to handle other issues, such as runway traffic and landing blunders. We may develop these scenarios relatively soon, so it is instructive to describe them in this report.

The flight path currently includes separate approach and landing objects (see Figure 3-5). The simulation loads a set of approaches and landings separately. It is possible to mix and match them when building a path, provided that the handoff between them occurs at the same location.

Figure 3-5. Path Object



Approach

The approach is the only portion of the path that the simulation currently uses. The approach is an ordered list of location points, which are basic 3D points that are enhanced to include a name and type. For example, the user can specify whether a particular point is a cornerpost, waypoint, or handoff point. A breakout path is a special type of "approach" that ends at a cornerpost or other designated point, rather than a handoff point. The types of path points currently have little meaning; they will become significant when the model expands to include controllers and their control domains.

Landing

The landing portion of a path object has three components: a localizer, a glide slope intercept, and a runway. The glide slope and localizer are children of the more general intercept class. This class is a polygon object—specifically, a trapezoid—that specifies a region of space in which aircraft can detect the intercept signal. Once the aircraft concludes its approach, it will rely on the intercept signals, rather than its own avionics, to determine its position for the remainder of its flight.

Runway

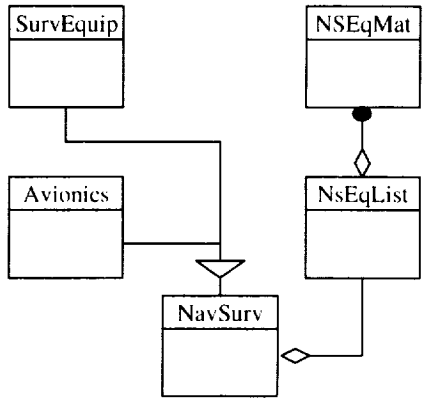
Each localizer and glide slope pair is associated with exactly one runway. The runway has a point that describes its entrance, a length, and a 2D orientation.

NAVIGATION/SURVEILLANCE OBJECTS

The aircraft's avionics object performs its navigation functions, and the surveillance equipment object performs its surveillance function. In reality, one type of equipment may handle some or all of these two functions. For example, a GPS may tell the aircraft where it is and how far away the nearest mountain is. The software design separates those functions into two objects with no direct interaction.

Both objects are children of a generic navigation/surveillance parent object (NavSurv in Figure 3-6). The parent object contains a list of Navigation/Surveillance Equipment Matrices (NSEqMats), which are objects that hold a specific set of error parameters attributed to the accuracy of a specific piece of equipment. The avionics object can have an unlimited number of these items in its list; the first equipment matrix describes the primary navigation system, the second describes the secondary system, and so forth. Each matrix has a set of values associated with nominal operating accuracy and failure accuracy. Typically, when the item fails, the pilot will have the opportunity to detect the failure and switch to a backup system.

Figure 3-6. Navigation and Surveillance Objects



The surveillance equipment object only holds one equipment matrix in its list. The nominal mode for the surveillance equipment corresponds to a working surveillance system, be it a GPS system or collision avoidance radar. The failed mode corresponds to visual indications only.

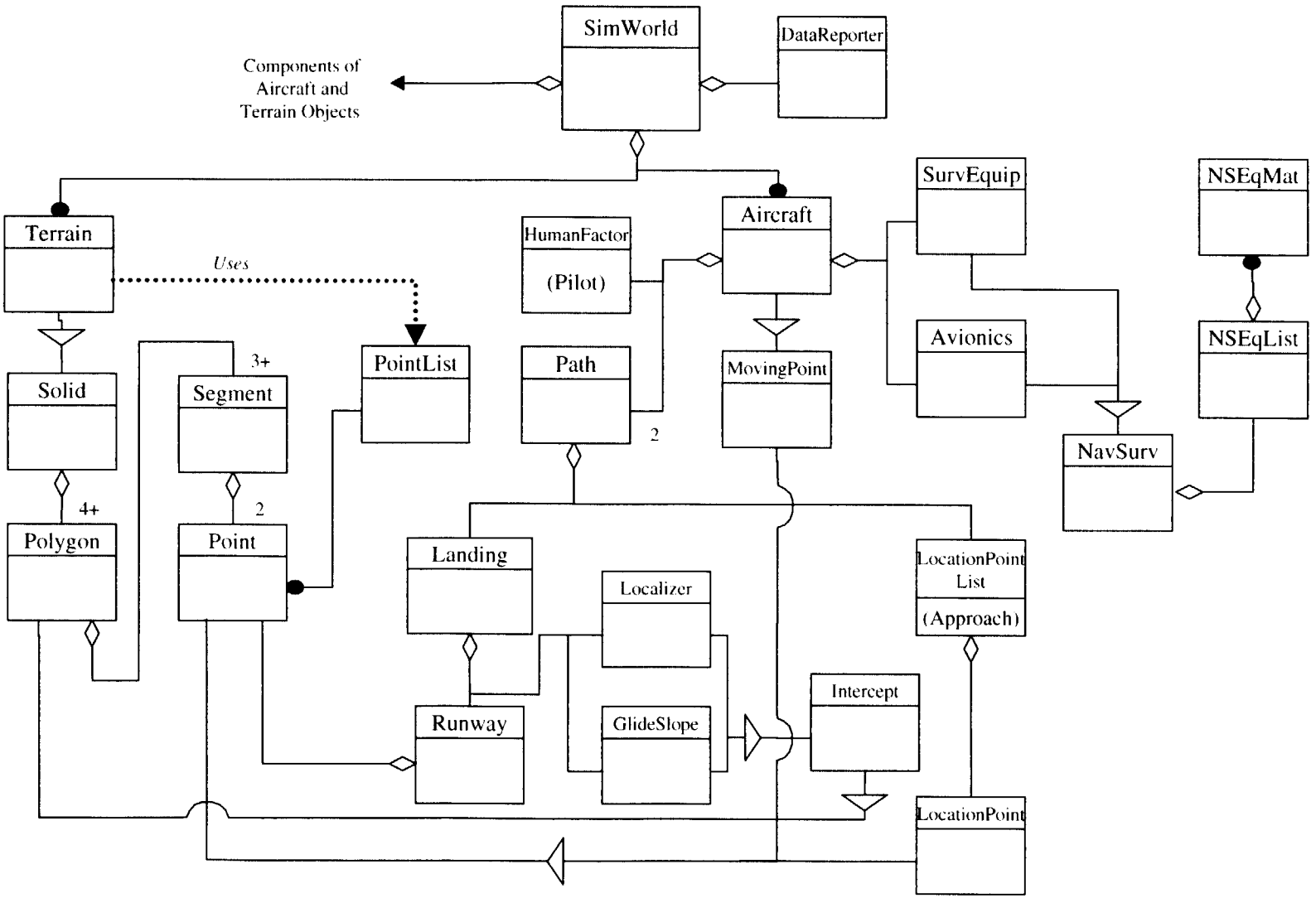
FUTURE EXTENSIONS

In designing and building an aviation simulation model in a reasonable amount of time, the risk of overdesign far outweighs the risk of designing a system that is too simple to be useful. Simply put, adapting a simply designed, narrowly focused model to new problems is easier than designing a monolith that addresses every conceivable aviation safety issue. The need for simplicity led to the goal of a simple, one-page design, as shown in Figure 3-7. To create a useful framework for further work, however, simplicity cannot be the only goal; the design also must be extensible.

To maximize the design's extensibility, we placed special emphasis on a hierarchical approach to object interactions. For example, the object that models a pilot has no direct knowledge of the aircraft object or any other object. It simply provides, in accordance with user-specified probabilities, a go/no-go check that an aircraft uses before acting. The avionics object, path objects, and navigation/surveillance objects are similarly limited; for example, the surveillance equipment object does not interact directly with a terrain object to determine the aircraft's indicated distance from it. Instead, the aircraft interacts with the terrain object to determine the true distance, and the surveillance object provides an error that is consistent with its equipment accuracy. This approach makes the object interactions easier to build and maintain, which simplifies the task of extending the model.

The most obvious extension to this model will be to add a landing algorithm for the aircraft. The model's current collection of objects is sufficient for implementing this algorithm. To simulate traffic scenarios, it will be necessary to develop a controller class. A capability to model traffic scenarios will be a valuable extension to this model, so we have already considered the controller object's composition and interactions in conceptual design activities.

Figure 3-7. Overall Software Design



Chapter 4
Modeling and Algorithms

In this chapter, we discuss the modeling methods we used to simulate aircraft flight, compute important geometric parameters, and model human reaction to a stimulus. These methods constitute the bulk of the mathematical groundwork required to model a terrain avoidance scenario. Because the safety simulation model is working code, the reader may wish to learn about possible ways to extend and enhance the methods described. Therefore, we describe algorithm modifications that may prove useful in future work, when applicable.

FLYING AN APPROACH

The first step in flying an approach is to place the aircraft at the cornerpost, as indicated by its avionics equipment. Once there, the aircraft turns until it is pointing at the next waypoint on the path and then flies in a straight-line motion until it reaches that point. It continues this pattern until it reaches the handoff point.

Turning to a Waypoint

The aircraft must first determine which direction to turn. For example, if the aircraft is turning toward a waypoint that is 45 degrees to starboard, it can turn 45 degrees to starboard or 315 degrees to port. Clearly, turning to starboard is preferable. The aircraft determines which way to turn by calculating the direction vector l toward the goal waypoint and taking the cross-product of the track vector h and l. The z axis component of $h \times l$ tells the aircraft which way to turn: If its sign is negative, the aircraft should turn to starboard; otherwise, it should turn to port.

The next step for the aircraft is to determine its radius and center of turn. The model currently assumes a 15 degree bank angle for all normal turns. Equation 4-1 computes the magnitude of a turn radius r, based on a given bank angle ϕ_b:

$$r = v_{xy} / g \tan \phi_b , \qquad \text{[Eq. 4-1]}$$

where v_{xy} is the magnitude of the component of the velocity in the $z = 0$ plane and g is the acceleration due to gravity. To compute the center of turn, the model must first compute the direction of the radius of turn. The radius direction is h_{xy}, the component of the track vector in the $z = 0$ plane, rotated ±90 degrees, depending on the direction of turn:

$$r_x = (h_{xy})_y = h_y / \|h_{xy}\|$$
$$r_y = -(h_{xy})_x = -h_x / \|h_{xy}\| \qquad \text{[Eq. 4-2]}$$

if turning to starboard and

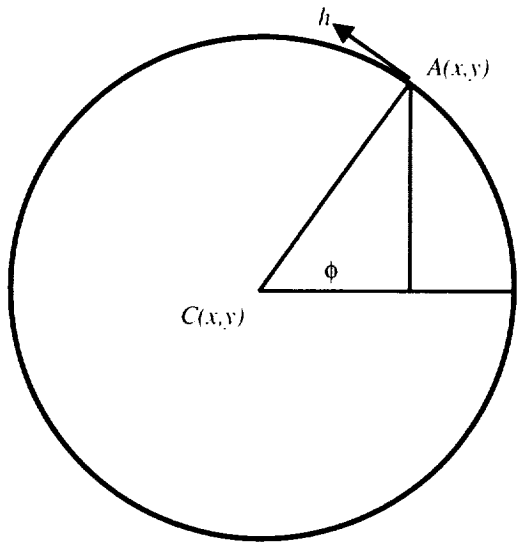
$$r_x = -(h_{xy})_y = -h_y / \|h_{xy}\|$$
$$r_y = (h_{xy})_x = h_x / \|h_{xy}\| \qquad \text{[Eq. 4-3]}$$

if turning to port.

The turn radius r and turn center $C(xy)$ create the arc that the aircraft flies while turning (see Figure 4-1). To determine the aircraft's location $A(x,y)$ on that arc, the model computes the angle θ swept between the x axis and the current direction of the turn radius. Equation 4-4 computes θ:

$$\theta = \tan^{-1} \frac{A_y - C_y}{A_x - C_x} \qquad \text{[Eq. 4-4]}$$

Figure 4-1. Aircraft's Location on Turn Arc



With r and q, we now have the aircraft's location expressed in cylindrical coordinates. Given that the angular velocity and linear velocity are related such that

$$\dot{\theta} = \frac{v_{xy}}{r}, \qquad \text{[Eq. 4-5]}$$

we know that after turning for one simulation time step t the aircraft's new angular location would be

$$\theta^t = \theta^{t-1} + t\dot{\theta} \quad . \qquad \text{[Eq. 4-6]}$$

(The t superscript denotes the current time step; the $t - 1$ subscript denotes the previous time step.) The aircraft's new location after a simulation time step would be

$$\begin{aligned} A_x^t &= C_x + r\cos\theta \\ A_y^t &= C_y + r\sin\theta \\ A_z^t &= A_z^{t-1} + v_z t \end{aligned} \qquad \text{[Eq. 4-7]}$$

After each time step, the model takes the cross-product of the track vector and the vector defined by the direction from the aircraft to the waypoint. When the z component of the cross-product changes sign, the aircraft has completed the turn. The model assigns the aircraft a track vector that will take the aircraft to the waypoint along a linear path.

Flight on a Linear Path

Flying a linear path is much simpler computationally than turning. The equations of motion for linear flight are

$$\begin{aligned} A_x^t &= A_x^{t-1} + v_x t \\ A_y^t &= A_y^{t-1} + v_y t \\ A_z^t &= A_z^{t-1} + v_z t \end{aligned} \qquad \text{[Eq. 4-8]}$$

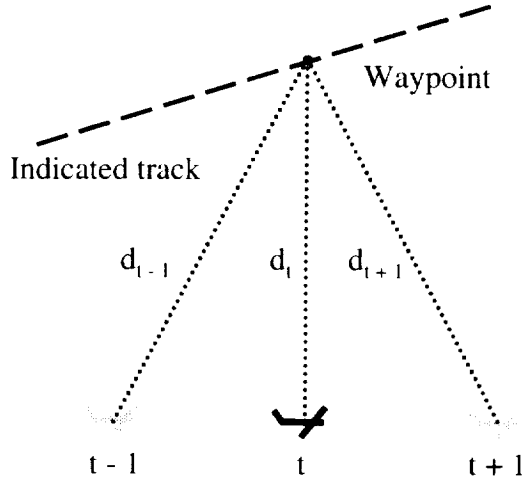
Determining when the aircraft has arrived is more difficult: Because of errors in the avionics package, the aircraft could be quite a distance from the actual waypoint when it believes that it has reached the waypoint. The simulation decides when the aircraft has reached the location by

- Computing the aircraft's expected location at the next time step
- Computing the distance between the waypoint and the current position
- Computing the distance between the waypoint and the position at the next time step
- Comparing the two distances.

When the distance between the expected position and the waypoint is greater than the distance between the current position and the waypoint, the model assumes

that the aircraft has reached its goal (see Figure 4-2). At that time, it will either turn toward the next waypoint or end the approach, depending on whether it has reached the end of the path.

Figure 4-2: Aircraft at Minimum Distance from Waypoint

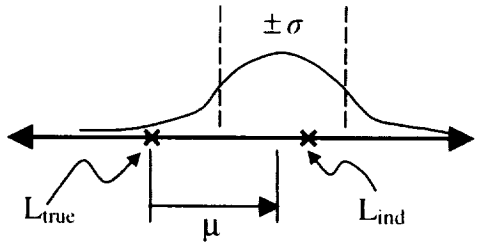


True track and location

NAVIGATION EQUIPMENT: CALCULATING INDICATED VERSUS ACTUAL VALUES

Before we consider how to incorporate instrument errors into indicated location, track, and speed values, we review how the navigation object models those errors. Recall (see Chapter 2) that the navigation object represents an error for a given value as a normally distributed offset from the true value (Figure 4-3). The mean of the normal distribution corresponds to a steady-state bias error; the standard deviation defines the variance of the distribution.

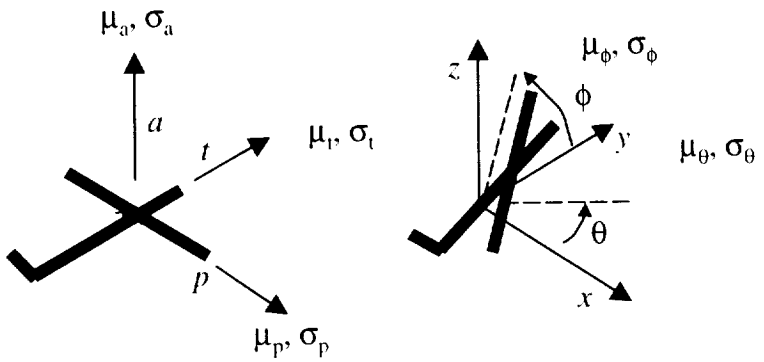
Figure 4-3: Error as Normally Distributed Offset of a Value



The navigation and surveillance data structures apply these normally distributed errors to six of the aircraft's location and velocity parameters: the x, y, and z coordinates; the track's angular rotation with respect to the x axis, ϕ; the track's angular rotation with respect to the z axis, θ (see Figure 4-4), and the speed. The

navigation object's location errors are relative to the aircraft's local coordinate frame. This local frame's altitudinal axis, *a*, aligns with the global *z* axis. The aircraft's tangential axis, *t*, aligns with the track of the aircraft, as projected onto the $z = 0$ plane. A perpendicular axis, *p*, lies at a right angle from the tangential axis in the $z = 0$ plane.

Figure 4-4: Aircraft Position, Velocity, and Error Parameters (Speed Not Shown)



Indicated Location

The navigation equipment introduces location errors into the simulation in two situations:

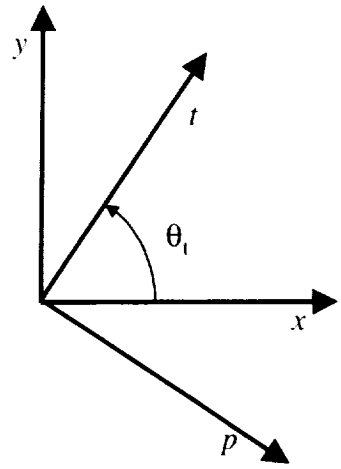
- When it adds an error vector to the true location to produce an indicated location.
- When it sets an aircraft's true location, based on the expected instrument error and the desired initial indicated value.

Both situations clearly require the same set of equations to relate the indicated location coordinates to the true location coordinates:

$$\begin{aligned} x_{ind} &= x_{true} + e_x \\ y_{ind} &= y_{true} + e_y \\ z_{ind} &= z_{true} + e_z \end{aligned} \qquad \text{[Eq. 4-9]}$$

This set of equations expresses the errors in terms of the global coordinate frame, however; the navigation object stores its error parameters in the aircraft's local coordinate frame. To use Equation 4-9, the model must transform the error values from the local frame to the global frame. The *a* axis of the local frame aligns with the global *z* axis. The *t* axis of the local frame aligns with the projection of the track vector *h* onto the $z = 0$ plane. We define the *p* axis by rotating the *t* axis clockwise 90 degrees in the $z = 0$ plane. As viewed from above the $z = 0$ plane, the local and global coordinate axes appear as shown in Figure 4-5.

Figure 4-5: Transforming from Aircraft to Global Coordinates



The local coordinate frame transforms to global coordinate frame according to the equations

$$t = \cos\theta_t \hat{i} + \sin\theta_t \hat{j}$$
$$p = \sin\theta_t \hat{i} - \cos\theta_t \hat{j} \qquad \text{[Eq. 4-10]}$$

Because the t axis aligns with the track vector, we can derive expressions for the cosine and sine terms in Equation 4-10 in terms of h_{xy}, the projection of the track vector onto the $z = 0$ plane:

$$\cos\theta_t = \frac{(h_{xy})_x}{\|h_{xy}\|}\hat{i} = c$$
$$\sin\theta_t = \frac{(h_{xy})_y}{\|h_{xy}\|}\hat{j} = s \qquad \text{[Eq. 4-11]}$$

Recall that the errors in the altitudinal, tangential, and perpendicular directions are random numbers that the simulation generates via a normal distribution:

$$e_t = N(\mu_t, \sigma_t)$$
$$e_p = N(\mu_p, \sigma_p) \qquad \text{[Eq. 4-12]}$$
$$e_a = N(\mu_a, \sigma_a)$$

The errors in terms of the global coordinate system therefore are

$$e_x = ce_t + se_p$$
$$e_y = se_t - ce_p \qquad \text{[Eq. 4-13]}$$
$$e_z = e_a$$

Indicated Velocity

The navigation object stores the velocity error parameters in spherical coordinates: (s, θ, ϕ). Therefore, to compute an indicated velocity, we must convert the track vector h from Cartesian to spherical coordinates. The true speed, s, is equivalent to the radius r of the spherical coordinate triple (r, θ, ϕ). Therefore, computing an indicated speed is trivial. One must simply generate an error value, using a normally distributed RNG, and add it to the true speed. To compute an indicated track, one must perform the following conversion on the true track vector:

$$h(x, y, z) = h_x\hat{i} + h_y\hat{j} + h_z\hat{k}$$
$$h(\theta, \phi) : \theta = \tan^{-1}(h_x / h_y), \phi = \sin^{-1}(h_z) \qquad \text{[Eq. 4-14]}$$

Once converted, the model computes the normally distributed random numbers e_θ and e_ϕ and adds each to the appropriate value to generate an indicated $h(\theta, \phi)$. It then computes the track vector back to Cartesian coordinates, using the inverse of Equation 4-14.

TERRAIN/AIRCRAFT DISTANCE CALCULATION

Because the model represents the aircraft as a point in 3D space and the terrain object as a rectangular solid, computing the distance between them is simple. When the terrain object receives its eight vertices, it receives them in a known order (see Chapter 2). It uses this known order to create the six boundary faces of the terrain object. When it creates those faces, it computes a unit normal vector n for each face. With this vector, it is possible to represent the plane of a face as a halfspace:

$$0 = n_x x + n_y y + n_z z - (n_x x_0 + n_y y_0 + n_z z_0), \qquad \text{[Eq. 4-15]}$$

where (x_0, y_0, z_0) are the coordinates of one of the face's vertices. The halfspace has a useful property: It is possible to get the distance d from the halfspace to any point (X, Y, Z) by plugging that point into the halfspace equation:

$$d = n_x X + n_y Y + n_z Z - (n_x x_0 + n_y y_0 + n_z z_0). \qquad \text{[Eq. 4-16]}$$

Furthermore, d is directional: If it is negative, we know that it is on the inside of the halfspace; if it is positive, we know that it is on the outside of the halfspace.

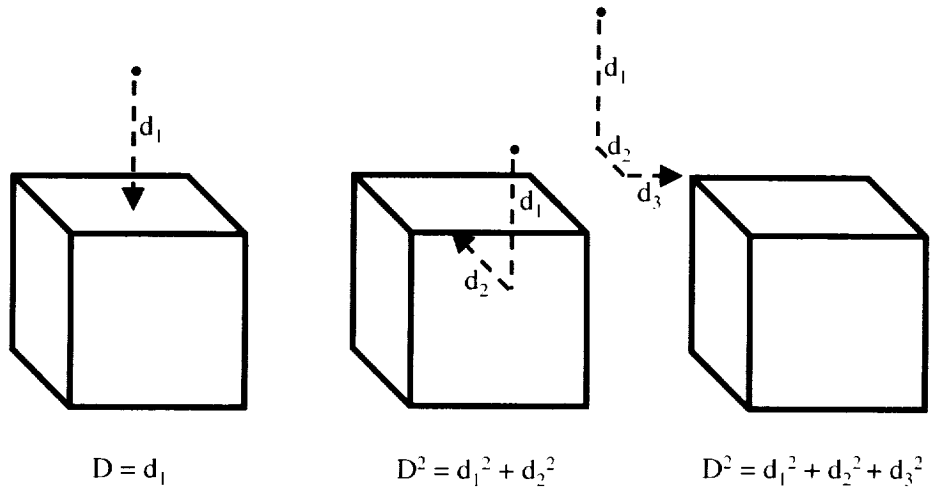
We can use the directional property of d to determine the distance from a point to a terrain object (or any rectangular solid). The algorithm is as follows:

1. Plug the point into the halfspace equation of each of the solid's six faces. If the point is inside the solid (in the case of an aircraft, that means it has

crashed), all six of the d_is will be less than or equal to zero. In that case, the model defines the distance between the solid and the point to be zero.

2. If the point lies outside the solid, it will lie outside one to three of the solid's faces. Figure 4-6 shows these three possibilities.

Figure 4-6. Distances from a Point to a Rectangular Solid



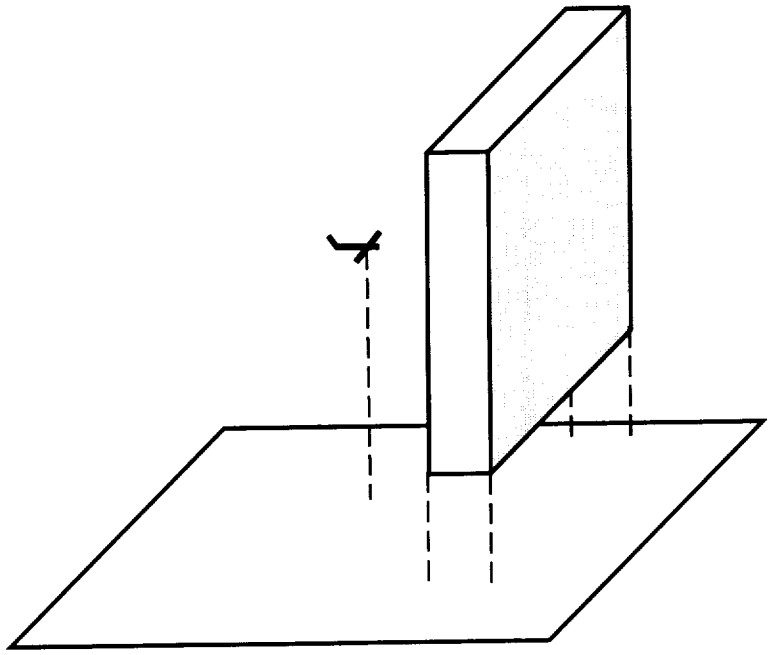
3. Square all non-zero d_is. Sum them and take the square root of the sum to get the distance from the point to the solid.

This algorithm relies on the fact that all faces of a solid that share a vertex are mutually perpendicular. A more general algorithm, using halfspaces, also is possible, but these methods are more complex. Nonconvex solids with a large number of faces can add considerable complexity—potentially, enough to greatly reduce the execution time of the model. If a new scenario requires highly complex terrain geometries, the developers should consider integrating a commercially available geometry kernel into the model.

CHOOSING TURN DIRECTION TO AVOID TERRAIN

When a pilot breaks through a haze and finds that the aircraft is too close to a terrain object, the aircraft must climb to avoid it. The pilot usually will turn the aircraft to avoid the object as well. Deciding which direction an aircraft should turn to avoid an object is easy for a pilot—but for a computer simulation. The method involves projecting the object point (the aircraft), its line of sight (track vector), and the obstructing solid's vertices (terrain object) onto the $z = 0$ plane (see Figure 4-7).

Figure 4-7. Projecting Terrain and Aircraft onto $z = 0$

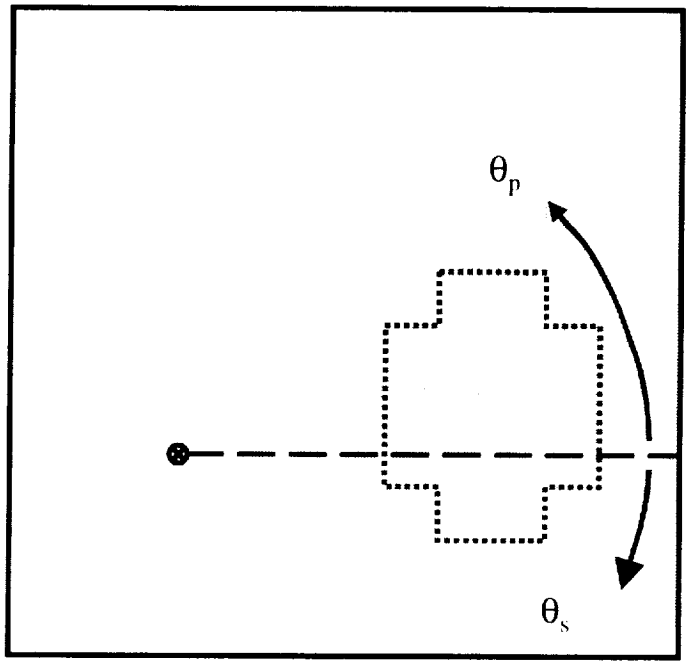


Using the projected points as a reference, getting the angle swept from the line of sight to the starboard-most and port-most vertex of the terrain object is straightforward (see Figure 4-8). To determine the angle swept between the line of sight l and the vector v that extends from the aircraft to any vertex of a terrain object, the model takes the cross-product of the vector and the line of sight. The angle swept from the line of sight to the vector comes from the following equation:

$$\frac{l \times v}{\|l\|\|v\|} = \sin\theta \qquad \text{[Eq. 4-17]}$$

The side with the smallest of the two angles is the direction that the aircraft should turn to avoid the object. As Figure 4-8 implies, this method does not require the terrain object to be a rectangular solid. In fact, the terrain object can be a general polyhedron.

Figure 4-8. Determining Smallest Turn Angle to Avoid Terrain



MODELING PILOT REACTION TIME

Two possibilities exist for computing pilot reaction times: The model can test to see that the pilot successfully reacts and then generate a reaction time on the basis of some random distribution, or it can test the pilot periodically during the simulation to see whether it succeeds. The reaction time then becomes the amount of time required to generate a successful test. Both methods are relatively straightforward to implement. The latter method is a little more intuitive, and it offers more flexibility to adjust for the difficulty of a task than the former. We therefore chose the latter method.

The reaction time T_{rxn} is a function of a geometric random variable; the number of tests required to successfully react, n; and the time elapsed per test, T_{test}:

$$T_{rxn} = \begin{cases} nT_{test} + T_{\min} & for\ n \le n_{\max} \\ \infty & otherwise \end{cases} \qquad \text{[Eq. 4-18]}$$

The reaction time is defined only if n is less than some maximum number of tests n_{max}. Otherwise, the simulation presumes that the pilot does not respond to the stimulus. The pilot data structure generates the reaction time according to the following algorithm:

1. Wait the minimum reaction time T_{min}.
2. Set $n = 0$.

3. Until a successful test occurs or $n = n_{max}$:

   a. Wait one test duration time T_{test}.

   b. Generate a uniform number $u = U(0, 1)$

   c. Compare u to the pilot's probability of a successful test p: if $p \geq u$, the test is successful. Otherwise, increment n by 1 and repeat step 3.

Only one currently implemented event forces a pilot to react: an uncovered failure of the avionics (navigation) equipment. Therefore, the model uses a constant T_{min} of 3 seconds, a constant T_{test} of 1 second, and an n_{max} of 60 cycles, corresponding to a 60 second window in which the pilot must react. It would help the user to know what value of p corresponds to a given probability that the pilot ultimately will fail to react to the event. The pilot ultimately fails to react when n, the outcome of a geometric random variable that we call X, equals or exceeds n_{max}. Equation 4-19 allows the user to calculate the probability of that outcome:

$$P\{X \geq n_{\max}\} = (1-p)^{n_{\max}-1} \qquad \text{[Eq. 4-19]}$$

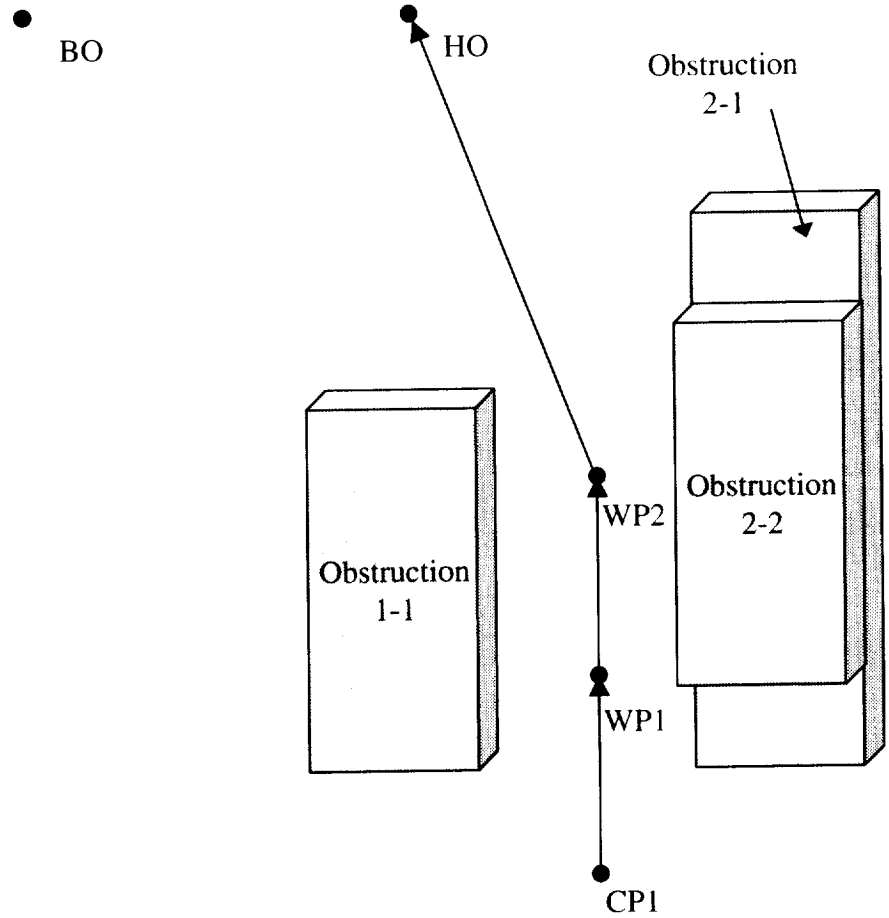
To ensure that a pilot fails to react at least 50 percent of the time, the user must select a $p < .012$.

This algorithm for computing pilot response time extends to other human elements that developers may add in future enhancements. For example, this algorithm can be used to determine the time required for a controller to notice that an aircraft has strayed off course. In that scenario, it may be necessary to lengthen T_{test} from 1 second to 5 seconds, because the controller's radar screen may take that long to complete a sweep. For other types of stimuli, it may be necessary to adjust T_{min}. In the event of more complex stimuli, the developer also can raise the number of successful tests that the human must perform. For example, if a human must diagnose an error condition by looking at four separate gauges, it is possible to adjust the algorithm to wait until four successful tests occur before declaring a successful reaction.

AppendixA
A Simple Example

This appendix shows a simple example involving two aircraft flying an approach of four waypoints through a valley (Figure A-1). The aircraft proceed from the cornerpost ("CP1") to the handoff point ("HO"), provided that they do not encounter terrain. If they come too close to any of the obstructions, they will break out of the normal approach and fly the breakout path, which involves turning from wherever they happen to be and heading to breakout point "BO." We made the scenario geometry as simple as possible to aid the reader in understanding the data presented in the input and output files. With more complex geometry, it becomes difficult to picture what the simulation does simply by reading the numeric output.

Figure A-1. Terrain and Path for Example



The scenario flies two different aircraft. One has fully accurate avionics and operational surveillance equipment. The other has avionics that are fully accurate, with the exception of a significant map error with respect to the *x* axis. This aircraft also has severely limited surveillance equipment. The map error acts to push the right-hand obstruction into the path of the second aircraft, and the limited surveillance equipment prevents the aircraft from detecting the problem quickly. The simulation output shows that this aircraft takes an emergency turn/climb to port and flies to the breakout point after clearing the terrain. We do not provide a full printout of the output file; that would be tedious for the reader. Instead, we provide excerpts from the file and comment on the data.

The main purpose of this example is to provide a working set of input files that readers can use as a template for creating their own scenarios. Creating a valid set of input files is not easy; we emphasized algorithm and object development over user-friendliness. When we do provide comments about the input files, they appear at the end of the file listing that they discuss. Comments are interspersed with the output file excerpts. The file listings are in 12 point Courier New; additional comments are in 12 point Times New Roman.

CONFIGURATION FILE (SAFETYSIM.CONFIG)

```
BEGIN HEADER
  Safety Simulation sample configuration file, 17
January 2001.
END HEADER


BEGIN FILES
  TERRAIN FILE
   sample.terrain
  PATH FILE
   sample.path
  PILOT FILE
   sample.pilot
  AVIONICS FILE
   sample.avionics
  AIRCRAFT FILE
   sample.aircraft
  SURVEQUIP FILE
   sample.radar
```

```
  DATAREPORTER FILE
   sample.data
END FILES
```

This configuration file shows a consistent naming convention. In creating a scenario, naming all the files in the scenario similarly is a good idea, to avoid confusing them with files for other scenarios.

TERRAIN FILE (SAMPLE.TERRAIN)

```
POINTLIST

 24

 1 6000.0 1000.0 0.0
 2 9000.0 1000.0 0.0
 3 9000.0 13000.0 0.0
 4 6000.0 13000.0 0.0
 5 6000.0 1000.0 1000.0
 6 9000.0 1000.0 1000.0
 7 9000.0 13000.0 1000.0
 8 6000.0 13000.0 1000.0

 9 15000.0 1000.0 0.0
 10 18000.0 1000.0 0.0
 11 18000.0 18000.0 0.0
 12 15000.0 18000.0 0.0
 13 15000.0 1000.0 1000.0
 14 18000.0 1000.0 1000.0
 15 18000.0 18000.0 1000.0
 16 15000.0 18000.0 1000.0

 17 15000.0 7500.0 1000.0
 18 18000.0 7500.0 1000.0
```

```
 19 18000.0 16500.0 1000.0
 20 15000.0 16500.0 1000.0
 21 15000.0 7500.0 2000.0
 22 18000.0 7500.0 2000.0
 23 18000.0 16500.0 2000.0
 24 15000.0 16500.0 2000.0


TERRAINLIST
3

Obstruction1-1
  Lower
  1 2 3 4
  Upper
  5 6 7 8

Obstruction2-1
  Lower
  9 10 11 12
  Upper
  13 14 15 16

Obstruction2-2
  Lower
  17 18 19 20
  Upper
  21 22 23 24
```

This file demonstrates the use of two terrain objects stacked atop each other to create a more detailed topography. Obstructions 2-1 and 2-2 combine to create the terrain on the right side in Figure A-1.

PATH FILE (SAMPLE.PATH)

```
APPROACHLIST
2
  SampleApproach
  4
  Cornerpost
   CP1
   12000.0   0.0   2500.0
  Waypoint
   WP1
   12000.0   5000.0   2000.0
  Waypoint
   WP2
   12000.0   10000.0   1500.0
  Handoffpoint
   HO
   8000.0   20000.0   500.0

  SampleBreakoutApproach
  1
  BreakoutPoint
   B0
   0.0   20000.0   2500.0

LANDINGLIST
0

PATHLIST
2
  SamplePath
   Approach
     SampleApproach
```

```
 NoLanding
SampleBreakoutPath
 Approach
  SampleBreakoutApproach
 NoLanding
```

PILOT FILE (SAMPLE.PILOT)

```
PILOTLIST
  1
  StdPilot
   1
   .95  .75  .25
```

This pilot will react fairly quickly regardless of the level of readiness. Even when the pilot is very busy, the probability of pilot reaction to a stimulus is .25. To create a 50 percent chance that the pilot will fail to react to a stimulus completely, the probability would have to be approximately .012.

AVIONICS FILE (SAMPLE.AVIONICS)

```
AVIONICSLIST
2

  PerfectAvionics
1
  Primary
    0.0  0.0   0.0  0.0   0.0  0.0
  0.0  0.0   0.0  0.0   0.0  0.0
  0.0  0.0  0.0
    0.0  0.0   0.0  0.0   0.0  0.0
  0.0  0.0   0.0  0.0   0.0  0.0
  0.0  0.0  0.0
1   1

  MapErrorAvionics
```

```
1
Primary
 0.0 0.0  0.0 0.0  0.0 0.0
0.0 0.0  0.0 0.0  0.0 0.0
-3100.0 0.0 0.0
 0.0 0.0  0.0 0.0  0.0 0.0
0.0 0.0  0.0 0.0  0.0 0.0
-3100.0 0.0 0.0
1  1
```

In this scenario, we decided to use two avionics packages. The other option would be to create one package: The normal operating mode would get the perfect error values, and the failed mode would get the map error.

The map error specified will cause the aircraft with this set of avionics to measure its position 3,100 feet to the left of where it actually is. Thus, when the aircraft thinks it is centered on the path, it will be flying directly toward the obstruction on the right side of Figure A-1.

AIRCRAFT FILE (SAMPLE.AIRCRAFT)

```
MOVINGOBJECTLIST
  1
  StdPlane
    nm1
      0.0 0.0 0.0
    0.0 1.0 0.0
    100

AIRCRAFTLIST
  2
    StdPlane
     StandardAC
    Pilot
     StdPilot
    Avionics
```

```
    PerfectAvionics
  Path
   SamplePath
  BreakoutPath
   SampleBreakoutPath
  SurvEquip
   PerfectRadar
  MinimumDistances
    750 750
ActionSequence
  0.0  Approach
  Events
  0

  StdPlane
   MapErrorAC
  Pilot
   StdPilot
  Avionics
    MapErrorAvionics
  Path
   SamplePath
  BreakoutPath
   SampleBreakoutPath
  SurvEquip
   FailedRadar
  MinimumDistances
    750 750
ActionSequence
  400.0  Approach
  Events
  0
```

Both aircraft use the same pilot and fly the same path. The second aircraft uses flawed navigation and surveillance equipment. The second aircraft starts 400 simulation cycles (400 seconds) after the first, which ensures that the output data are uncluttered. To run a parametric study, the user can define several aircraft that are similar in all respects except one and see how that variation affects the outcome.

SURVEILLANCE EQUIPMENT FILE (SAMPLE.RADAR)

```
SURVEQUIPLIST
2
PerfectRadar
  Operating
   0.0
   0.0 0.0  0.0 0.0  0.0 0.0
   50000.0
  Backup
   85.0
   0.0 0.15 0.0 0.15 0.0 100.0
   850.0
  1
FailedRadar
  Operating
   0.0
   0.0 0.0  0.0 0.0  0.0 0.0
   50000.0
  Backup
   85.0
   0.0 0.15 0.0 0.15 0.0 100.0
   850.0
  2
```

These radars are identical except for their status: "Perfect Radar" has a status of 1 (operating) and "Failed Radar" has a status of 2 (failed). It would be just as easy to assign both aircraft the same surveillance equipment and create an event in the second aircraft that causes its radar to fail before it begins its approach.

DATA REPORTER FILE (SAMPLE.DATA)

```
MINIMUM TERRAIN DISTANCE
  500.0
FULL PRINTOUT (Y/N)
  Y
```

OUTPUT FILE EXCERPTS (SAMPLE.DATA.OUT)

The simulation output is primitive, but consistent. Every line of data includes the relevant aircraft's name, the simulation time, and data, which can be locations in three dimensions or messages concerning simulation events.

The simulation starts with the first aircraft, "StandardAC," making a turn toward the first waypoint. Because StandardAC's track already aligns with the waypoint, the turn is complete in one simulation cycle. The data specify the aircraft, then the simulation time (1), followed by its indicated x, y, and z coordinates and its true x, y, and z coordinates.

```
StandardAC 1, IND: 12000, 0, 2500, TRUE: 12000, 0,
2500
StandardAC 1 completed turn.
```

After the aircraft completes its turn, it flies to the first waypoint:

```
StandardAC 2, IND: 12000, 99.5037, 2490.05, TRUE:
12000, 99.5037, 2490.05
StandardAC 3, IND: 12000, 199.007, 2480.1, TRUE:
12000, 199.007, 2480.1
StandardAC 4, IND: 12000, 298.511, 2470.15, TRUE:
12000, 298.511, 2470.15
StandardAC 100, IND: 12000, 9751.36, 1524.86, TRUE:
12000, 9751.36, 1524.86
StandardAC 101, IND: 12000, 9850.87, 1514.91, TRUE:
12000, 9850.87, 1514.91
StandardAC 102, IND: 12000, 9950.37, 1504.96, TRUE:
12000, 9950.37, 1504.96
StandardAC 102 has arrived at goal wp.
```

Notice that even though the avionics system has perfect accuracy, the aircraft does not directly coincide with the first waypoint. The aircraft travels in discrete steps, and the waypoint falls between two of those steps.

The aircraft continues to fly the route without incident, concluding at simulation time 212.

```
StandardAC 212, IND: 7990.21, 20024, 497.593, TRUE:
7990.21, 20024, 497.593
StandardAC 212 has arrived at goal wp.
```

At simulation time 401, the second aircraft, "MapErrorAC," begins its flight. Notice that its true location is 3,100 feet from its indicated location. This error puts the left edge of Obstruction 2-2 in its path.

```
MapErrorAC 401, IND: 12000, 0, 2500, TRUE: 15100, 0,
2500
MapErrorAC 401 completed turn.
MapErrorAC 402, IND: 12000, 99.5037, 2490.05, TRUE:
15100, 99.5037, 2490.05
MapErrorAC 403, IND: 12000, 199.007, 2480.1, TRUE:
15100, 199.007, 2480.1
```

As time passes, the aircraft approaches the obstruction. The aircraft has a minimum operational separation distance of 750 feet, according to its input file specification (see aircraft file). The simulation notes that the aircraft has violated its minimum operational distande at simulation time 469. Three more cycles are required for the pilot to react and attempt an emergency turn/climb. By then, the aircraft is less than 500 feet from the obstruction. The data reporter lists 500 feet as the minimum safe distance for an aircraft, so the simulation notes that a crash is imminent. The emergency maneuver is successful, however, and the aircraft increases its distance beyond 500 feet once again at simulation time 482. Two seconds later, the aircraft completes the turn/climb; one time step after that, it increases its distance beyond 750 feet.

```
MapErrorAC 467, IND: 12000, 6567.25, 1843.28, TRUE:
15100, 6567.25, 1843.28
MapErrorAC 468, IND: 12000, 6666.75, 1833.33, TRUE:
15100, 6666.75, 1833.33
```

MapErrorAC 469, IND: 12000, 6766.25, 1823.37, TRUE: 15100, 6766.25, 1823.37

MapErrorAC, 469 below min. terr. dist.

MapErrorAC 470, IND: 12000, 6865.76, 1813.42, TRUE: 15100, 6865.76, 1813.42

MapErrorAC 471, IND: 12000, 6965.26, 1803.47, TRUE: 15100, 6965.26, 1803.47

MapErrorAC 472, IND: 12000, 7064.76, 1793.52, TRUE: 15100, 7064.76, 1793.52

MapErrorAC, 472 crash imminent.

MapErrorAC 473 Attempting emergency turn/climb

MapErrorAC 474, IND: 11991, 7163.73, 1853.52, TRUE: 15091, 7163.73, 1853.52

MapErrorAC 475, IND: 11971, 7241.1, 1913.52, TRUE: 15071, 7241.1, 1913.52

MapErrorAC 476, IND: 11939.9, 7314.75, 1973.52, TRUE: 15039.9, 7314.75, 1973.52

MapErrorAC 477, IND: 11898.5, 7383.12, 2033.52, TRUE: 14998.5, 7383.12, 2033.52

MapErrorAC 478, IND: 11847.6, 7444.77, 2093.52, TRUE: 14947.6, 7444.77, 2093.52

MapErrorAC 479, IND: 11788.4, 7498.4, 2153.52, TRUE: 14888.4, 7498.4, 2153.52

MapErrorAC 480, IND: 11722, 7542.88, 2213.52, TRUE: 14822, 7542.88, 2213.52

MapErrorAC 481, IND: 11649.8, 7577.26, 2273.52, TRUE: 14749.8, 7577.26, 2273.52

MapErrorAC 482, IND: 11573.4, 7600.82, 2333.52, TRUE: 14673.4, 7600.82, 2333.52

MapErrorAC, 482 crash no longer imminent.

MapErrorAC 483, IND: 11494.4, 7613.06, 2393.52, TRUE: 14594.4, 7613.06, 2393.52

MapErrorAC 484, IND: 11414.5, 7613.73, 2453.52, TRUE: 14514.5, 7613.73, 2453.52

MapErrorAC 484, completed turnclimb.

MapErrorAC, 485 clear of terrain.

MapErrorAC 485, IND: 11334.6, 7612.76, 2513.52, TRUE: 14434.6, 7612.76, 2513.52

```
MapErrorAC 486, IND: 11235.2, 7623.27, 2513.44, TRUE:
14335.2, 7623.27, 2513.44

MapErrorAC 487, IND: 11138, 7646.62, 2513.36, TRUE:
14238, 7646.62, 2513.36

MapErrorAC 488, IND: 11044.7, 7682.41, 2513.28, TRUE:
14144.7, 7682.41, 2513.28
```

When the aircraft completes its turn/climb, it begins to fly its breakout path. The model assumes that once an aircraft breaks out, it will keep well clear of any further encounters with terrain. Therefore, the model does not monitor the aircraft as it flies the breakout path, and it is not necessary to include this portion of the flight. The addition of traffic to this scenario will change that assumption, however. In anticipation of that eventuality, the model continues to fly the aircraft until it reaches the breakout point.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2001	Contractor Report

4. TITLE AND SUBTITLE
Aviation Safety Simulation Model
User's Guide

5. FUNDING NUMBERS
C NAS2-14361
WU 706-87-21-01

6. AUTHOR(S)
Scott Houser

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Logistics Management Institute
2000 Corporate Ridge
McLean, Virginia 22102-7805

8. PERFORMING ORGANIZATION REPORT NUMBER
LMI-NS010S1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-2199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/CR-2001-211022

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Robert Yackovetsky.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 03 Distribution: Nonstandard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The Aviation Safety Simulation Model is a software tool that enables users to configure a terrain, a flight path, and an aircraft and simulate the aircraft's flight along the path. The simulation monitors the aircraft's proximity to terrain obstructions, and reports when the aircraft violates accepted minimum distances from an obstruction. This model design facilitates future enhancements to address other flight safety issues, particularly air and runway traffic scenarios. This report shows the user how to build a simulation scenario and run it. It also explains the model's output.

14. SUBJECT TERMS
Aviation safety; Simulation; Object-oriented software design

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL