81

Real-time Collision Avoidance for Autonomous Air Vehicles

by

Christopher P. Sanders

S.B., Aeronautics and Astronautics
MIT, 1996

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

January 1998

Author ....
Department of Aeronautics and Astronautics
January 16, 1998

Certified by ....
Paul A. DeBitetto
Senior Member of Technical Staff, Charles Stark Draper Laboratory
Thesis Supervisor

Certified by ....
Eric Feron
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by ... .
Jaime Peraire
Associate Professor
Chairman, Department Graduate Committee
Department of Aeronautics and Astronautics

Real-time Collision Avoidance for Autonomous Air Vehicles

by

Christopher P. Sanders

Submitted to the Department of Aeronautics and Astronautics on January 16, 1998, in partial fulfillment of the requirements for the degree of Master of Science in Aeronautics and Astronautics

Abstract

This thesis describes the design and analysis of a collision avoidance system (CAS) for autonomous air vehicles (AAVs). In the future, AAVs will operate in close proximity to one another and cooperate to perform missions. In such environments, a real-time collision avoidance system for the AAVs is needed to ensure the safety of the vehicles and the mission. The CAS development process described in this document represents a balanced approach, concentrating on four key elements: algorithm design, multi-AAV simulation, closed-loop analysis, and actual vehicle flight tests. The testbed for the future flight tests of the CAS is the Draper Small Autonomous Air Vehicle, winner of the 1996 International Aerial Robotics Competition. This balanced design approach yielded more insight into the CAS behavior than if only one or two of the four elements had been used. Furthermore, favoring a simple, functional design over a complex, optimal one allowed the system to be developed quickly and analyzed easily.

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

Thesis Supervisor: Paul DeBitetto
Title: Senior Member of Technical Staff, Charles Stark Draper Laboratory

Acknowledgments

I would first like to thank Paul DeBitetto, who always provides me with valuable guidance and encouragement. I am especially appreciative to him for giving me the opportunity to work on such a challenging and exciting project. I have learned a lot from him and from the DSAAV project. I would also like to thank Professor Eric Feron for always keeping me on track and helping me with the design and analysis of my collision avoidance algorithms.

In addition, I would like the thank the other members of the DSAAV team for their effort and enthusiasm toward the project. Long Phan, Christian Trott, Eric Johnson, Mike Bosse, Chinsan Han, Paul Stukenborg, Bob Faiz, and Rusty Sammon have made the project a success—and a lot of fun, too.

If it were not for Mario Santarelli hiring me to work in the Draper Simulation Laboratory back in 1994, I would have missed out on being a part of both the Simlab and the DSAAV project. I am very grateful to him for giving me these opportunities.

Finally, I would like to thank my family. Their support and guidance have been inspirational to me. I especially want to thank my wife, Marienne Moro Sanders, for her love and support throughout my years at MIT. She has been the source of much patience, understanding, *and* engineering wisdom.

Chris Sanders
January 1998

Table of Contents

List of Figures

List of Tables

Chapter 1

Autonomous Air Vehicles

Over the last couple decades, talk of autonomous vehicles has gradually made its way into discussions on the military and commercial aerospace industries. And while autonomous ground and undersea vehicles have been in service for awhile, the age of the autonomous air vehicle (AAV) is seemingly just dawning.

In reality, the aerospace industry has been moving towards autonomous vehicles for decades. The increased automation in military and commercial aircraft allow flight computers to perform most critical flight operations; and it is conceivable that aircraft pilots will become virtually unnecessary in the not-too-distant future. However, the AAVs we are speaking of are a different class of vehicles with unique missions specifications.

1.1 Why AAVs?

Some of the motivation behind AAVs may be understood by first considering the broader class of *unmanned aerial vehicle*s (UAVs). The UAV class includes AAVs; however, UAVs may also be remotely piloted by a human on the ground (or in another vehicle for that matter). In practice, even most AAV architectures do include a human in the loop, and therefore are not entirely autonomous. However, in most cases this human merely exercises supervisory monitoring and control of the vehicle.

Fully autonomous or not, all UAVs offer some advantages over conventional manned aircraft. All one has to do is to look at the field of human factors engineering to realize how much effort is made to design things for human operation. This is especially the case in the aerospace industry, where the human operator is often at the helm of a highly capable and often dangerous vehicle. Books, reports, university courses, and thousands of engineers have investigated how to make pilots and aircraft crews more comfortable and effective in their respective vehicles. This research often translates into costly displays, control input devices, and life-support systems. In many vehicles, life-support equipment (including the cockpit or crew quarters) constitute much of the weight and size of the vehicle.

Of course, having a human operator in a vehicle also automatically restricts the allowable performance of the vehicle. Fighter aircraft maneuverability is severely limited by the G-levels tolerable by the human body. Furthermore, every time a manned air vehicle takes flight, human lives are being put at risk. This is especially true in combat situations and experimental test flights.

Remotely-piloted vehicles (RPVs) may remedy some of the above-mentioned complications and shortcomings of manned flight. Without a pilot on board, a fighter RPV could perform hi-G maneuvers far surpassing those of manned fighters. Additionally, without a cockpit and life support apparatus, the fighter UAV could be smaller, lighter, and cheaper than its manned counterparts. The loss of an RPV would cost a fraction of that of a manned fighter. Most importantly, the pilot would be unharmed.

Nevertheless, shifting pilot control to the ground introduces a new class of problems. The pilot might be deprived of visual, aural, and motion cues which otherwise would act to enhance his situational awareness. These issues are constantly being investigated and addressed by virtual reality technology and computer-augmented controls. Also, it is no secret that pilots, in general, are opposed to being taken out of the aircraft and put on the ground at the controls of an RPV. However, after wars during which pilots' lives have been lost, there has been noticeably increased support of taking combat pilots out of harm's way.

In some cases, removing humans from the piloting loop altogether has its merits. First of all, humans have finite reaction times. This reaction time includes the time it takes the pilot to be aware of a "situation", as well as the actual physiological reaction time. Under nominal conditions, a human pilot's reaction time is sufficient for safe operation. There are, however, extreme scenarios in which only a computer could react quickly enough to avoid disaster.

At times, human operators practice subjective and emotional decision making. Certainly there are instances in which such traits have proven to be valuable. However, these same traits are quite often to blame for human error in air vehicles. Fatigue, which often plagues overworked or overstressed pilots, is another potential source of human errors. A shift toward supervisory human control or complete autonomy could help eliminate these causes of error.

From a financial standpoint, one of the most expensive aspects of aircraft operation is the number of human operators needed to perform particular missions. With AAVs, any human operators should only be performing supervisory duties. It is reasonable to envision large fleets of AAV operated by as few as one or two people.

1.2 AAV Missions

Autonomous aerial vehicles potentially could perform a number of missions. AAV missions should capitalize on some of the aforementioned advantages AAVs have over manned aircraft. Over the last thirty years, remotely-piloted UAVs have been used almost exclusively in surveillance and reconnaissance missions. Such missions can get away with not having a human on board more than tactical missions, such as air-to-air combat. Predator (Figure 1.1), made by General Atomics, is one of the most battle-proven surveillance UAVs.

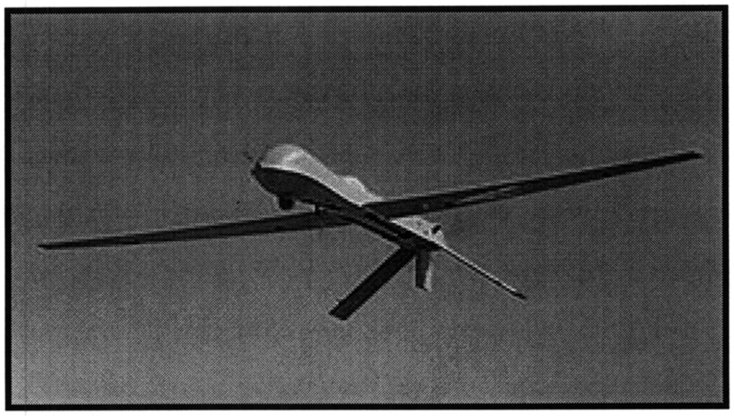


Figure 1.1: *Predator UAV*

Also, AAVs have a further advantage over even RPVs: they can perform missions without concern for the mental or physical condition of a human pilot. In one unique long-duration application, AAVs would form a hovering communications network for long-distance communication. Surveying hazardous radiation or waste areas with AAVs would also keep human operators out of danger.

Without a human cargo, designers have more flexibility with AAV shapes and sizes. The vehicles can be made to have extremely low radar cross-sections, allowing them to operate stealthily. Smaller AAVs could perform surveillance in hard-to-reach areas such as forests and even urban environments. Micro-AAVs, whose sizes may someday shrink down to insect-size, could be deployed in large numbers by soldiers in the field for a look over a ridge. Or a mission could take the micro-AAV inside of buildings, where it would be the proverbial fly on the wall.

Further down the line, AAVs could be used in tactical missions, such as air-to-air and air-to-ground assaults. Since real-time control often plays a part in tactical applications, the communication delays associated with RPVs may make them inappropriate for these missions. Instead, these missions will require advanced control techniques for AAVs, perhaps even "intelligent control" methods which emulate a human operator. While fighter pilots are not quite ready to step out of the aircraft and let a computer fly, as AAV technology advances, these advanced missions will be seriously considered.

AAVs aren't limited to military missions. UAVs are already being used for environmental imaging and data collection. Inevitably AAVs will be used similarly. AAVs even offer something for the entertainment and news industries. Moviemakers will be able to capture shots never before possible with camera-carrying AAVs. News organizations will be able to capture video footage of hard-to-reach or dangerous scenes. And if AAVs prove themselves reliable over time, we may even see passenger-carrying commercial autonomous transports.

1.3 Types of AAVs

Like manned aircraft, AAVs can come in a variety of styles. Everything from autonomous airships to tail-sitters have been attempted. Most AAVs, however, can be described as either a rotorcraft (helicopters and their kin) or a fixed-wing aircraft. Traditionally, RPVs have been predominantly fixed-wing, as hovering a helicopter is a difficult task which relies heavily on vision and motion cues which may be unavailable to the remote pilot.

AAVs, on the other hand, are not restricted by the limitations of remote piloting. In fact, autonomous helicopters have some distinct advantages over fixed-wing AAVs. Helicopters have a unique range of motion, including hover and backward flight. This allows them to perform accurate station-keeping, which is useful in surveillance and reconnaissance. They can more easily maneuver in limited spaces and at low altitudes. Many AAV missions may require them to fly below treetop level or around buildings in cities. Lastly, helicopters and other vertical take-off and landing (VTOL) AAVs do not require an airstrip, and therefore can be operated out of almost anywhere.

1.4 Multi-AAV Operations

As people dream up more and more assignments for AAVs, many of them will involve multiple vehicles working together. At times AAVs with different missions will be required to interact in the same airspace. In other situations, cooperating AAVs will need to safely practice formation flying. For these multi-AAV missions to take place, techniques for safe and intelligent AAV cooperation will be necessary. Those techniques begin which each vehicle's autonomous control system.

1.4.1 Autonomous Control Architectures

The complexity of an AAV's mission may demand that it be able to negotiate multiple—and often conflicting—goals in real-time. The task of designing an autonomous control system which safely accounts for all perceivable scenarios is a formidable one.

Instead of trying to achieve the necessary behavior with a complex, optimal control system, many have suggested using a combination of simple control behaviors. When activated in series or parallel, these simple behaviors may sum to produce much more complicated behavior.

Many biological systems seem to demonstrate this superposition of simple behaviors. In particular, the complex (even chaotic) behavior of swarming flies and mosquitoes has been closely simulated by simple pursuer-evader equations of motion [1]. Similarly, observations and simulations have shown very coordinated and purposeful behavior in ant and bee colonies, where the individuals' behaviors are based on very simple environmental cues [2].

Rodney Brooks formalized this type of vehicle control in what he calls a *subsumption architecture*. The subsumption architecture is a layered control system consisting of *levels of competence*. Each "level of competence is an informal specification of a desired class of behaviors" for the autonomous vehicle. "A higher level of competence implies a more specific desired class of behaviors." Lower levels continuously run with no knowledge of the levels above them. Higher levels, however, can suppress the actions of the lower levels, and inject their own actions instead. When this occurs, it is said that the higher level layers *subsume* the lower levels [3].

A subsumption architecture may be simplified into just two control layers: one concerned with global control and one with local control. This is a common distribution of control in autonomous vehicles. Global control laws are those which utilize either global goals or global information, or both [4]. Global goals describe the overall mission for the vehicle or team of vehicles. Often, global goals are commanded by a centralized controller, which can be either a computer or human operator. Global information is that information which is used by the autonomous vehicle or team to pursue the global goals. This information is obtained through communication among the vehicles and their centralized controller.

Global goals and global information may not be updated very often. A mission schedule may even be uploaded to the vehicles before the mission, with no means for revision during the mission. Unexpected events can occur which disrupt the mission's global goals, in which case it may be necessary to rely on a more reactive, local controller.

Local controllers command reactive behaviors, without considering the global goals. And because they are to react in a timely manner, local control laws may not have time to incorporate global information into their logic. A local controller may be suboptimal, but it might also help the vehicle survive those unexpected disruptions.

Finding the appropriate balance between global and local control is difficult and unique for every autonomous vehicle system. Achieving this balance is often a trade-off between optimality and robustness. At times, global and local goals are in conflict, and we need a way—like subsumption—to negotiate between them.

Conveniently, the multi-AAV cooperation task can also be broken up into global and local control. The global control involves high-level mission planning, while the local control handles the low-level, reactive collision avoidance.

1.4.2 Mission Planning

Mission planning pertains to the strategic routing and scheduling of AAV activities. For example, a mission planner might devise waypoints for the vehicle to follow. These waypoint commands would be taken by the AAV and translated into guidance and control commands.

Robot motion planning has been a hot topic in the robotics, controls, and artificial intelligence communities for years. Parallel efforts have also been made in the air traffic control (ATC) community. Most of the research has focused on numerical optimization methods and so-called intelligent control techniques. Often the line between the two becomes blurred, as they are both based on many of the same principles.

Most of the optimization techniques fall under the classification of numerical programming. In general, numerical programming optimizes a given quantity (e.g. fuel, time, safety) while satisfying certain constraints. The field of operations research (OR) applies these programming techniques to solve a number of real-life problems. While very effective, these methods become computationally intensive as the complexity and number of degrees of freedom in a system increases.

Intelligent controls techniques are an attempt to assign human traits, such as learning and language, to computer control systems. They include fuzzy control, adaptive control, and neural control techniques. Fuzzy control moves away from exact numerical control laws, toward linguistic or fuzzy laws that are used by human operators. For example, a pilot, when asked to describe the landing task, will do so with vague references to “too fast”, “too shallow”, “slight back pressure”, etc. Fuzzy control assigns references like these to numerical control ranges. The fuzzy control law then combines all fuzzy conditions which apply in order to output a single control signal [5].

Adaptive control accounts for a changing environment, enabling the controller to detect such changes and compensate for them. In the complex environment of multi-AAV operations, such flexibility could prove very useful [5].

Lastly, neural control is based on the interconnection of "neural networks" in the human nervous system. A particular combination of sensory stimuli result in an appropriate control action. Often, the relationship between the sensory inputs and motor outputs is a complex one. Therefore, the neural network actually must repeat a task a number of times, each time slightly adjusting the connections until the mapping from input to output is correct [5].

The mission planning task doesn't necessarily have to be performed by a computer either. A human operator could be responsible for planning the mission route, either before the mission, or on-the-fly during the mission. While this may fall short of a fully autonomous mission, the remainder of the guidance, navigation, and control tasks could remain under the computer's control. Such coordination of multiple AAVs would be comparable to the job of an air traffic controller.

1.4.3 Collision Avoidance

Ideally, the higher-level mission planner will handle any potential conflicts between AAVs. In complex multi-AAV operations, however, a mission manager—human or computer—might send errant commands that put two or more AAVs dangerously close to one another. Or a failed communication link with the ground station could leave an AAV virtually blind to others in its vicinity. In these situations, a lower-level, reactive collision avoidance system (CAS) would be necessary to avert an accident.

Like the Traffic alert and Collision Avoidance System (TCAS) in commercial aviation, this CAS would issue commands that would take the involved AAVs out of danger. Unlike TCAS, however, a collision avoidance system for AAVs wouldn't just be issuing warnings to a human pilot; instead, it would basically take over the low-level control of the AAV. Instead of relying on a pilot reacting quickly and taking the correct action, an autonomous CAS can rely on the AAV's guaranteed preprogrammed performance.

As with the mission planning task, there is no obvious technique for collision avoidance. The multi-AAV environment is complex and nonlinear, and many issues have to be considered. The focus of this document is to address some of these issues for a representative autonomous vehicle system. A collision avoidance system for these AAVs is presented and analyzed in the subsequent chapters.

Chapter 2

Draper Small Autonomous Air Vehicle

2.1 Background

The Draper Small Autonomous Air Vehicle (DSAAV) is the testbed upon which our collision avoidance system will be tested. The DSAAV (Figure 2.1) came about in 1995, when an IR&D project was initiated at Draper Laboratory to develop a small autonomous air vehicle. A team of Draper technical staff, Boston University students, and Massachusetts Institute of Technology students was assembled, and they chose the objective of the project: submit an entry in the 1996 International Aerial Robotics Competition.

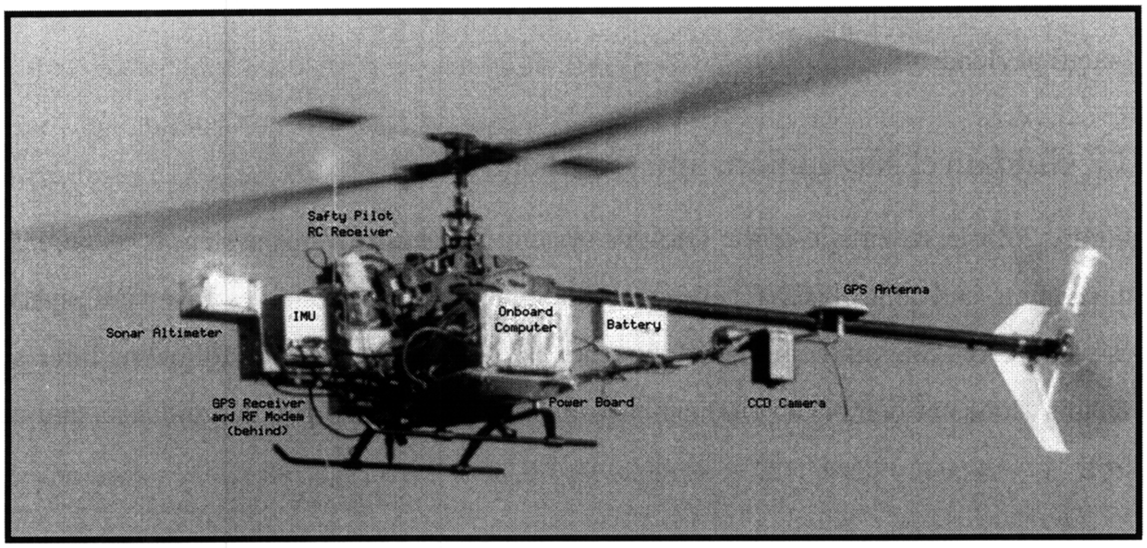


Figure 2.1: *DSAAV in flight*

This annual competition is sponsored by the Association for Unmanned Vehicle Systems, International (AUVSI), and features AAVs designed and built by university and industry teams all over the world. The contest missions change from year to year, but in 1996, it required the vehicles to locate (within one meter accuracy) five barrels scattered throughout a 60 foot by 120 foot field. The AAVs also had to identify each barrel, based on either a radiation label or biohazard label located on each. In addition, one of the drums had a small disk, which was to be retrieved by the AAV and returned to the starting position. The entire mission, including take-off and landing, was to be performed completely autonomously.

The Draper team won the contest, having the only vehicle which successfully performed a fully autonomous mission, from take-off to landing. In fact, in the allotted hour, DSAAV completed seven fully autonomous flights. From these flights, the locations of all five barrels were determined, and two of the barrel labels were successfully identified.

2.2 Vehicle Platform

For a mapping mission in a small area like the contest field, the DSAAV needed to be maneuverable in tight spaces. This requirement naturally pointed to the hover capabilities of helicopters. After assessing payload weight budgets and project cost budgets, TSK's off-the-shelf Black Star remote-control helicopter was selected. Its 2-HP engine and six foot main rotor diameter provided just enough lifting power to hover with an eight to ten pound payload.

2.3 Guidance, Navigation, and Control System

Figure 2.2 is a schematic of the DSAAV system. The heart of this system is its guidance, navigation, and control (GNC) system. Although manned air vehicles have GNC systems as well, AAVs rely on them to take the place of the pilot. Each of the following three subsections looks at one of the elements that make up GNC: navigation, guidance, and control.

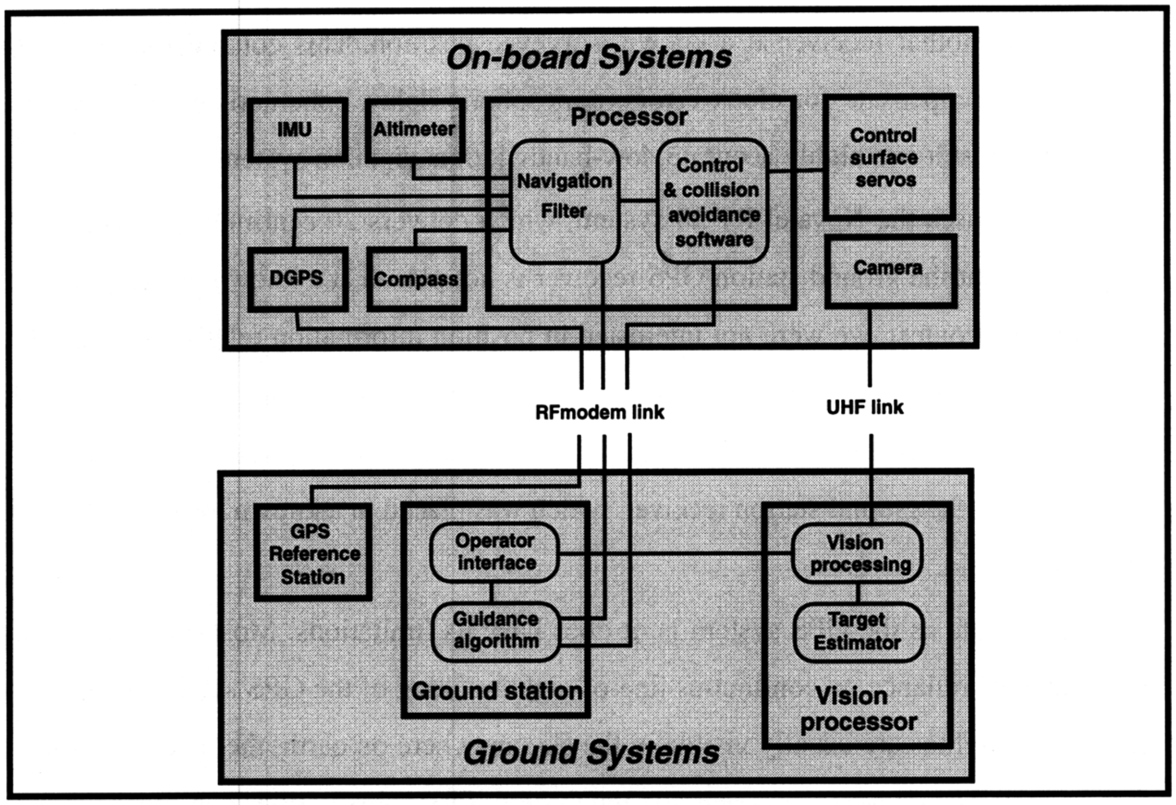


Figure 2.2: *DSAAV system*

2.3.1 Navigation System

Before the DSAAV can autonomously locate barrels in a field, it has to first be able to determine its own position relative to a frame of interest. To do so, it relies on a suite of navigation sensor hardware as well as navigation software which runs on board the vehicle.

Differential Global Positioning System

The global positioning system (GPS) is ubiquitous in today's world. Everyone from aviators to geologists to hikers use this accessible, worldwide navigation system. GPS relies on a constellation of 24 satellites which orbit the earth. A GPS receiver measures its distance to as many satellites as are in view above the horizon. If four or more satellites are in view, the receiver can triangulate its position.

With differential GPS (DGPS), the receiver compares its measured position to the location of another receiver at a known, surveyed position. This comparison allows the receiver to compensate for clock errors, atmospheric delay, ephemeris errors, multipath errors. The result is a highly accurate, low-bandwidth navigation system.

DSAAV uses the Novatel RT-20 system, which delivers 20 centimeter position accuracy. Note that the ground station GPS receiver is not placed at a known geographic location. For the contest, we were not interested in position information relative to an earth or inertial frame. It was important, however, to accurately know the vehicle's position relative to the field. So, the DGPS system provides accurate position information of the vehicle relative to the ground station receiver, which was placed in a known location relative to the contest field.

As accurate as the GPS system is, it does have its limitations. Most its shortcomings stem from its reliance on continuous line-of-sight tracking of the GPS satellites. Although plenty of satellites are usually visible in the sky anywhere on earth, there are a number of factors which can nevertheless degrade the system. First, obstructions can block satellites from view of the GPS antenna. In urban areas, buildings are the obvious culprits. In rural areas, trees are often to blame. In most conventional aerospace applications of GPS navigation, the operation altitudes are above the buildings or the trees. However, as discussed earlier, many of the potential applications for small AAVs involve low-altitude operations, where buildings or trees could interfere with the GPS.

Another GPS problem comes from multipath interference. The ranging signals transmitted from the GPS satellites to the receivers can reflect off of surfaces (such as the ground or buildings) and arrive at the receiver at several distinct times. Each of these arrivals will result in a different computed range to the satellite, essentially confusing the receiver as to what the correct range is. The result is decreased position accuracy.

Inertial Navigation System

Since the helicopter is a unstable dynamic system with relatively fast time constants, the low-bandwidth of the GPS system is insufficient to adequately control the helicopter. Furthermore, with only one GPS antenna on-board, there is no way to sense the rotational motion of the helicopter from GPS. Inertial navigation is a good method of sensing the high-bandwidth and rotational motion of the vehicle.

A typical inertial navigation system (INS) consists of three gyroscopes and three accelerometers. The gyroscopes measure the rotational velocities around three orthogonal axes. The accelerometers measure the vehicle's acceleration along the same three axes. These rates can be fed directly into the navigation and control system, or they can integrated to obtain positions and attitudes which can be used by the navigation and control systems. One of the most important advantages to INS is its self-contained nature. It requires no external communications and works in most any environment.

Like GPS, INS is not without its disadvantages. Angular rates and linear accelerations must be integrated once and twice respectively to obtain attitude and position. Any errors, even small ones, in the rates and accelerations will accumulate as they are integrated. The result is growing errors in attitude and position. Therefore, INS alone is not a reliable sensor for measuring position. However, INS and the accurate, low-bandwidth position measurements of GPS can combine to provide accurate low- and high-bandwidth navigation information. It is the responsibility of the navigation software to sensibly combine the measurements of these two sensors.

The INS system in DSAAV is the Systron Donner Motion Pak, which uses three quartz rate gyros and three quartz flexure accelerometers. Without GPS updates, this inertial measurement unit (IMU) will drift to a 4.5 foot position error in approximately nine seconds.

Sonar Altimeter

Since DSAAV makes autonomous landings, it must very accurately know its position above the ground. Since the GPS/INS navigation provides vertical position relative to an Earth-centered, Earth-fixed reference frame (ECEF), an additional sensor is required to detect the location of the ground below the vehicle. DSAAV uses a sonar altimeter, which consists of a Polaroid sonar ranging module and the necessary electronics to translate the echoes it receives into a range measurement.

Compass

So far, the inaccuracies in the INS position measurements have been accounted for by GPS and the altimeter. But we also have the angular rates, as measured by the INS gyros, which are also integrated to obtain the vehicle's attitude, expressed in the Euler angles, ψ (yaw), θ (pitch), and ϕ (roll). The integration errors for the attitude also could benefit from a correction.

The pitch and roll angles can actually be corrected by the INS accelerometers. The three accelerometers measure, among other vehicle accelerations, the direction of the gravity vector. This direction, relative to the vehicle axes, determines the pitch and roll of the vehicle. That leaves only the heading error uncompensated. The navigation system does compensate for heading drift with a digital compass, which delivers heading measurements with two degree accuracy and a 5 Hz update rate.

Navigation Software

The function of combining the navigation sensor outputs into one single estimate of the vehicle state (which includes linear and angular positions and velocities) is performed in on-board software by the navigation filter. The filter makes this estimate of the state based on all previous sensor measurements; but it is recursive, meaning it does not have to store all the previous measurements in order to compute the present estimate. The trick is to find the optimal recursive filter, which minimizes the root mean square (RMS) error in the state estimate. If the measurement and model noises are gaussian, this turns out to be the Kalman filter. We will discuss the Kalman filter further in Section 4.1.2.

2.3.2 Guidance System

If the navigation system tells the AAV where it is, then the guidance system tells it where it should go. The input to the guidance system are waypoints that the user defines. The waypoints have a position and heading associated with them. Furthermore, at each waypoint, the DSAAV either goes continues on, stops for a certain amount of time, or lands.

The guidance algorithm runs on the ground station (although it is moving on board in the future). To get the DSAAV from waypoint to waypoint, the guidance algorithm outputs position, heading, and velocity commands which are uplinked to the vehicle. These commands are designed to conform with the nominal velocities and accelerations of the DSAAV, and then they are interpreted by the control system, as we will see in Section 2.3.3.

2.3.3 Control System

Given the navigation state estimate and the guidance commands, it is up to the DSAAV control system to get the vehicle to where it is going. It does so by adjusting the helicopter's five control effectors: pitch cyclic, roll cyclic, collective pitch, tail rotor pitch, and throttle. The control algorithm itself treats the DSAAV's dynamics as four decoupled loops: pitch, roll, yaw, and altitude. Just because these loops are decoupled, however, does not mean that all of the effectors are not used in controlling each of the loops.

The control algorithm is a simple PID controller, and it has the ability to adjust the helicopter control surface trim positions in order to reduce steady-state error in the system. Another nice feature of the controller is that it is tunable in mid-flight. The ground station operator can change key control system gains during test flights. And if the vehicle should go out of control, a safety pilot can quickly flip a switch and take control.

2.4 Communication

Two-way communication between the helicopter and the ground station serve a number of purposes. First, the GPS receivers on the helicopter and on the ground station need communication in order to subtract out errors and provide DGPS capability. Secondly, communication provides health monitoring information to the ground operator of the AAV. Lastly, the ground operator can uplink guidance and control commands to the AAV.

For the DGPS communication, the on-board and ground GPS receivers are linked via a spread spectrum radio modem with a 1000 foot range. Due the functionality of the GPS receiver, the telemetry downlink and command uplink also pass through this modem. ASCII-formatted data passed to the GPS receivers by the computers (either on-board or ground station) are transmitted to the other computer via the radio modem and the other GPS receiver.

2.5 Vision System

The DSAAV vision system consisted of three main components: the on-board camera, the UHF downlink, and the off-board image processor. The camera is a small, lightweight, black and white CCD camera, which outputs a standard NTSC video signal. A UHF transmitter with a 500 to 3000 foot range transmits the live video signal down to the ground.

On the ground, SGI Indy processes the video, using standard image processing filters to extract the locations and classifications of the barrels in the contest arena.

2.6 DSAAV Tools

Developing the fully-autonomous DSAAV was a challenging endeavor, but a couple of tools were developed to aid in the development.

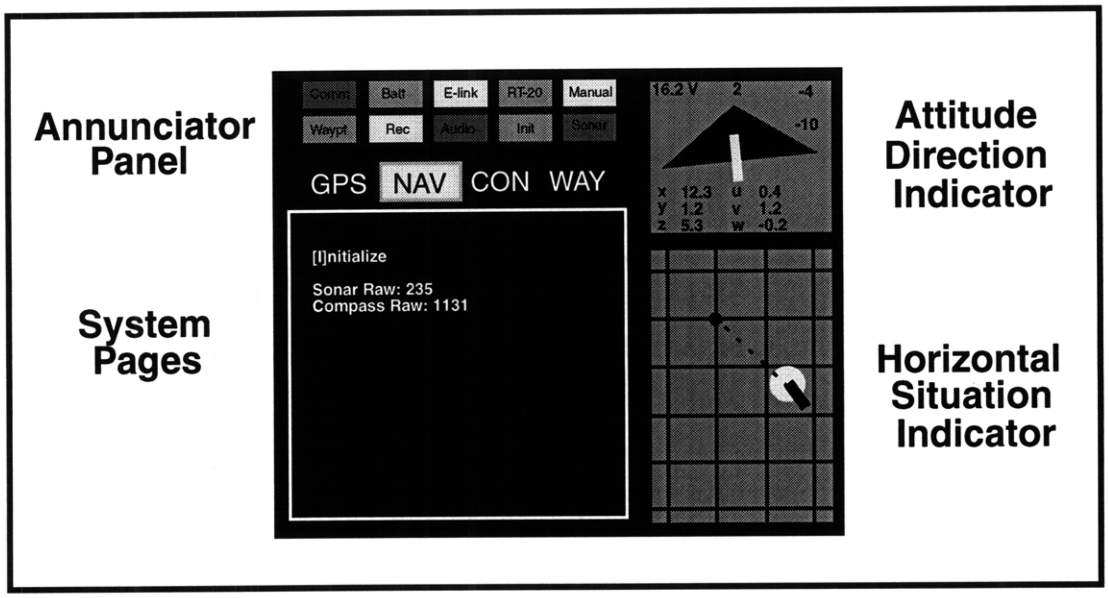
2.6.1 Ground Station

In general, the ground station refers to any of the DSAAV hardware that doesn't actually fly on the vehicle. This includes the reference station GPS antenna and receiver, the ground power supply, the image processing workstation, the radio modem, and the operator interface.

The ground station operator interface consists of a laptop computer running the ground station software. The software generates a graphical user interface (GUI) which allows the ground station operator to monitor the health of the DSAAV and to uplink commands to the vehicle.

The operator may select to view one of four system "pages" at a time. The four pages are dedicated to GPS, the navigation system, the control system, the waypoint guidance system. The GPS page allows the operator to assign the location of the reference station antenna and monitor vital GPS figures of merit, such as number of satellites being tracked and position error standard deviations. The navigation page permits the user to reinitialize the navigation filter, as well as view raw navigation data from the altimeter and compass.

On the control page, control loops can be opened and closed, and their respective closed-loop gains may be adjusted. Finally, on the waypoint guidance page, the waypoints which describe the vehicle's mission may be loaded or entered manually into the guidance system.



```
[I]nitialize

Sonar Raw: 235
Compass Raw: 1131
```

```
x 12.3  u 0.4
y 1.2   v 1.2
z 5.3   w -0.2
```

Figure 2.3: *Ground station user interface*

In addition to the four specific pages, the ground station interface always displays the most vital system information. An annunciator panel provides a brief way to verify that particular systems are functioning properly. Blocks representing the GPS fix, the communications system, sonar, battery voltage, guidance system, navigation filter, control system, and the telemetry recording system are colored red, yellow, or green, depending on their status.

The interface also displays a horizontal situation indicator (HSI) , which shows the horizontal position of the vehicle relative to the contest field. Finally, an attitude direction indicator (ADI), pictorially shows the roll, pitch, and yaw angles of the vehicle. In the same display, those angles, along with the battery voltage and the vehicle position and velocity are numerically represented.

The ground station has proven to be vital in safe, efficient testing of the DSAAV. Most problems which can afflict one of the DSAAV's subsystems can be detected by the ground station operator before it can cause a loss of vehicle. The ground station also allows the DSAAV to be somewhat adaptable when it flies a mission. As mentioned earlier, the waypoints in the guidance system can be changed on the fly by the ground station operator. Additionally, the control system gains can actually be changed and tested in flight, all from the ground station. Most importantly, during a nominal autonomous flight, the ground station operator's hands never touch the ground station, and thus keeping the system truly autonomous.

2.6.2 Simulation

To speed up the development of the DSAAV, and especially its control system, a high-fidelity, real-time simulation of the vehicle was built to run on a Silicon Graphics workstation in Draper's Simulation Laboratory. Not only does the simulation accurately model the vehicle dynamics, it also models the vehicle's navigation and control subsystems. Navigation and control algorithms can be quickly altered to see how they affect the vehicle's performance. Dozens of parameters, from number of satellites being tracked by GPS to individual control gains may be altered on the fly, during a simulation run. In this way, the response of the vehicle to component failure or degradation may be investigated. For convenient analysis, the simulation also renders a 3-D animation of the DSAAV and its surroundings, as seen in Figure 2.4 below. Also, any parameter or states of the vehicle system may be recorded and plotted for analysis.

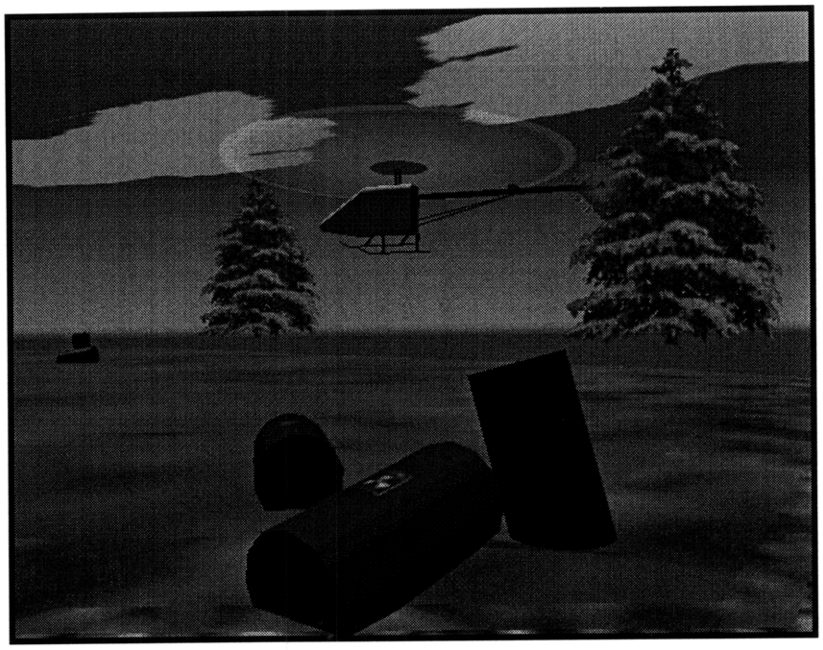


Figure 2.4: *DSAAV simulation display*

The DSAAV simulation was used extensively in the design and analysis of the DSAAV mission manager, control system, and collision avoidance system, which is described in Chapter 4. The ability to quickly change and retest these systems in simulation cut many hours off of the DSAAV development time.

Additionally, the simulation is able to communicate with the ground station, effectively tricking the ground station into believing that it's receiving data from the actual helicopter. Therefore, the ground station was able to be developed and tested with the simulation, eliminating the need for hours of flight testing with the actual vehicle. Lastly, the simulation is able to receive the same servo commands from the safety pilot as the actual vehicle does and inputs them to the vehicle dynamics. With this capability, the simulation can be used to train DSAAV safety pilots.

2.7 DSAAV 2

Following the victory at the aerial robotics competition, the DSAAV development team was tasked to build an improved, follow-on vehicle: DSAAV 2 (see Figure 2.5). The motivation for developing the second vehicle was to demonstrate three things: hardware improvements, cooperation between autonomous vehicles, and a vision navigation system.

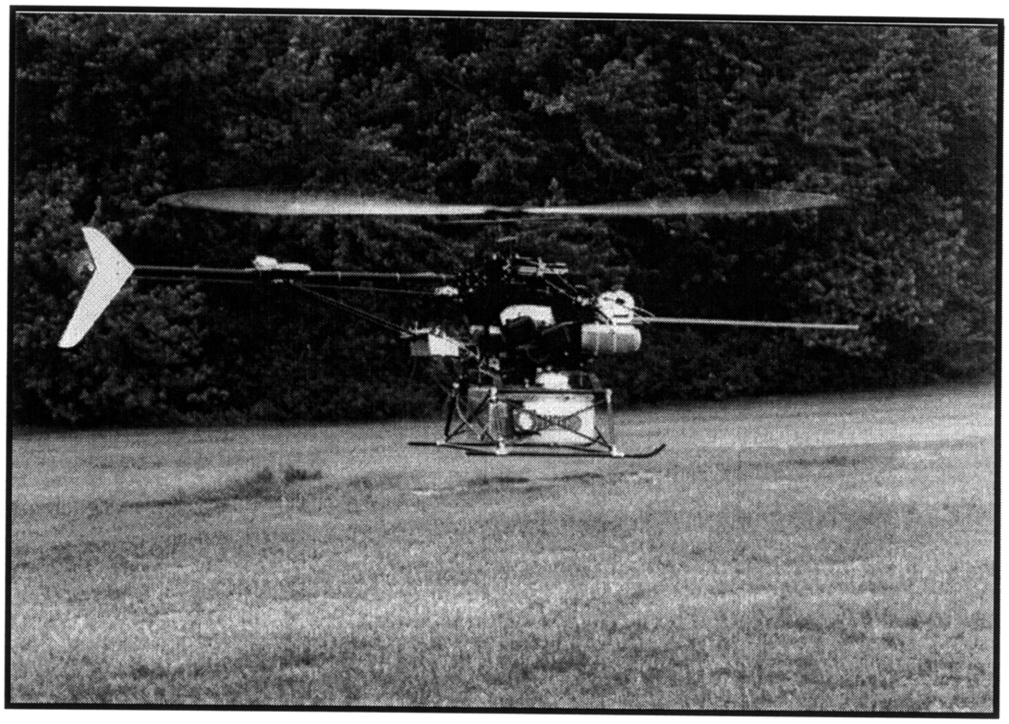


Figure 2.5: *DSAAV 2*

2.7.1 Hardware Changes

Experiences with DSAAV 1 led to a number of hardware design changes for the sequel.

New Helicopter Platform

For DSAAV 2, we opted to replace the TSK helicopter used for DSAAV 1 with the Bergen industrial helicopter platform. The new vehicle, which has reduced vibration and double the payload capacity of the TSK, should be able to accommodate many of the project's future goals.

Sensor Processing Unit

For DSAAV 2, the functions of sampling the inertial measurement unit and sampling the sonar/compass system have been combined and performed by a single PC board, called the Sensor Processing Unit (SPU). The SPU is designed to send IMU/sonar/compass data packets to the computer at a rate of 50 Hz, although that rate has not yet been achieved in practice [6].

Packaging

As seen in Figure 2.1, DSAAV 1 had hardware strewn around its chassis. For DSAAV 2, one vibration-isolated avionics box contains much of the hardware, including the GPS card, IMU, main computer, and the SPU. This design has reduced the amount of wire and connectors between the subsystems, while providing a more marketable look to DSAAV 2. Because the design is not very modular, however, repairs to the components inside the box have been somewhat inconvenient.

2.7.2 Cooperative AAVs

One of the primary objectives in building DSAAV 2 was to demonstrate both vehicles working, cooperating, and avoiding each other in a multi-AAV environment. A demonstration like this has never been attempted and presents many challenges, some of which are discussed in this document. At the time of this writing, the cooperative demo has not yet been performed, but is tentatively expected to happen in the course of the next year. The cooperative DSAAV mission has, however, been extensively analyzed and simulated. Figure 2.6 shows the 3-D output of a simulation run.

Figure 2.6: *Cooperative DSAAV simulation*

2.8 Future

The DSAAV program at Draper continues to make progress and set ambitious goals for its future. The following are some of the potential future directions for the project.

2.8.1 Cooperative DSAAVs

We continue to work toward demonstrating multiple AAVs performing a mission cooperatively. Algorithms, including the ones presented here, have been designed and await their chance to control DSAAV 1 and 2 in their cooperative mission.

2.8.2 Advanced Control Techniques

Currently, the DSAAV control system has very conservative velocity limits (5 feet/sec in forward flight) and a simple controller. Our vehicle platforms, on the other hand, are capable of very maneuverable and high-speed flight. In the future, advanced control algorithms will be designed and tested which allow DSAAV to utilize much more of the vehicles' flight envelopes.

2.8.3 Beyond Line-of-sight Operation

To make DSAAV a more marketable technology, it should someday be able to autonomously fly beyond line-of-sight (LOS) of the operator. To do so will require that the communication links between the vehicle and ground station be robustified and their ranges extended. LOS operation will also require a more capable user interface than the current DSAAV ground station.

Chapter 3

Collision Avoidance System Definition

The first step in developing our autonomous collision avoidance system is defining the multi-AAV system, then determining how this system definition will affect the actual collision avoidance algorithm. A number of factors influence the functional requirements of the CAS; six of them are discussed in this chapter: centralized vs. distributed architecture, algorithm complexity, interface to higher level planners, existing collision avoidance architecture, separation requirements, and vehicle models.

3.1 Centralized vs. Distributed Architecture

One of the most fundamental decisions in defining a collision avoidance system is determining where the processing for the collision avoidance algorithm should occur. This is critical to the processor selection and the design of the algorithm. Generally, a processing architecture may be described as either *centralized* or *distributed.*

In a centralized architecture, all of the processing for each vehicle to avoid collision occurs at one, central location. The appropriate commands are then communicated to the AAVs, which implement the commands at the appropriate times. Such an architecture is analogous to the aviation air traffic control system, where, in general, the paths of many aircraft are coordinated at one location. This one location can be either on the ground, as is the case with ATC, or in the air, like with the AWACS system. One can envision a fleet of AAVs being deployed, with one AAV acting as the mission controller for the others.

In a distributed architecture, each AAV is responsible for computing its own collision avoidance trajectories. This architecture is more along the lines of the *free flight* policy that the FAA is investigating. In free flight, some of the control over the aircraft's flight path is shifted from the ATC centers to the individual aircraft and their pilots. Aircraft will fly without ATC intervention, as long as no other aircraft violate the *Alert Zone* around the aircraft. The Alert Zone is an imaginary cylinder that surrounds the aircraft. If the Alert Zone is violated, ATC will direct the aircraft's' trajectories in order to decrease the risk of the aircraft violating one another's *Protected Airspace Zone*. The Protected Zone is a smaller cylindrical region surrounding the aircraft [7]. The Traffic Alert and Collision Avoidance System (TCAS) is the distributed CAS that is supposed to aid the transition to free flight TCAS and will be discussed further in Section 3.4.

In the free flight concept, it is not obvious when to shift control over to the ATC or back to the individual aircraft. Similarly, deciding between a distributed and a centralized autonomous CAS is not necessarily easy. A number of factors must first be considered; among them are fault-tolerance, communication, autonomy, and processing power.

Fault-tolerance

One concern for a centralized collision avoidance architecture—whether the central controller is on the ground or on one of the AAVs—is that a failure in the CAS could result in a failed mission or loss of vehicle for *all* the AAVs in the system. In a distributed architecture, on the other hand, a failed CAS on an AAV should only endanger that AAV and perhaps any in its immediate area. Still, redundancy and other advances in fault-tolerant systems can keep centralized systems reliable.

Communication

Communication is another consideration in choosing between a centralized and distributed collision avoidance architecture. In a centralized system, the controller must have knowledge of the AAVs' states in order to generate collision avoidance commands. This knowledge could either come from active sensing (i.e. radar) of the vehicles' positions or by receiving state information from a transponder on the individual vehicles.

Likewise, in a distributed architecture, each vehicle could sense others in its proximity via radar, or it could communicate with the other AAVs via data transponders. It appears that neither the centralized nor the distributed architecture requires drastically more communication capability than the other. If there is a slight difference, it would be that communication is simpler in the centralized architecture; each vehicle only has to communicate with the central controller, and not with all the other vehicles. The communication links in a centralized and a distributed architecture are shown in Figure 3.1.

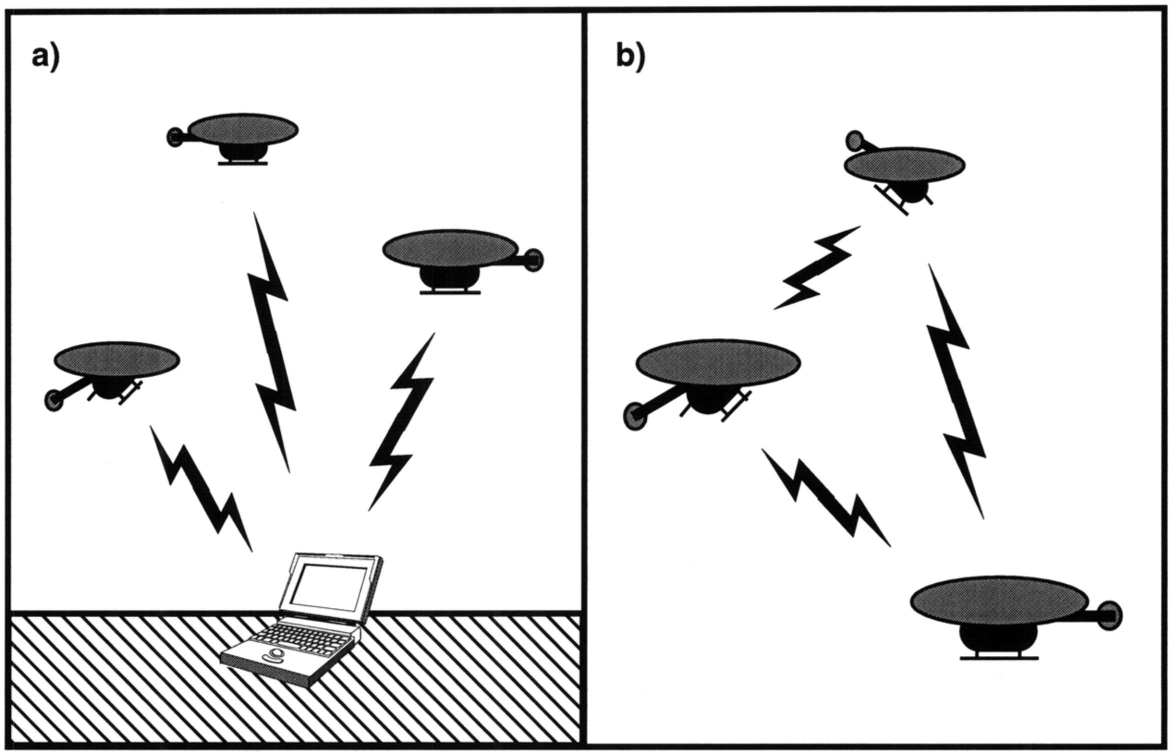


Figure 3.1: *Communication links for a) centralized and b) distributed systems*

Communication time delay is another matter, however. In the centralized system, the controller must wait for each vehicle's state information, compute the recommended maneuvers, then transmit them to each AAV. Therefore, the centralized architecture's time delay will always be greater than or equal to the time delay of the distributed system (by the triangle inequality). If the distance between two vehicles is small relative to their distance from the central controller, this time delay is approximately twice that of the distributed system.

Autonomy

The notion of autonomy, which is independence from a central controller, is somewhat violated by having a centralized collision avoidance architecture. A distributed system, on the other hand, consists of completely self-contained vehicles that only rely on sensing or receiving the other vehicles' states to perform collision avoidance. In reality, however, this distinction might be outweighed by other practical benefits of a centralized system.

Processing Power

Limited processing power on-board the AAVs may force the collision-avoidance system to reside in a central location. The converse case, where the central processor is incapable of computing collision-avoidance solution, is possible but less likely. This decision is closely related to the complexity of the collision-avoidance algorithm, as discussed in the next section.

DSAAV Collision Avoidance Architecture

The DSAAV collision avoidance system is partly distributed and partly centralized. For the sake of increased autonomy and fault tolerance, the collision avoidance algorithm will be processed on board each vehicle. The communication architecture is centralized, however, relying on the ground station as a communication hub. The ground station relays state data from one vehicle to the other, and vice-versa. In the future, we expect this centralized communication architecture to be replaced by a distributed one with inter-vehicle communication. A diagram of the hybrid distributed-centralized architecture is shown in Figure 3.2.

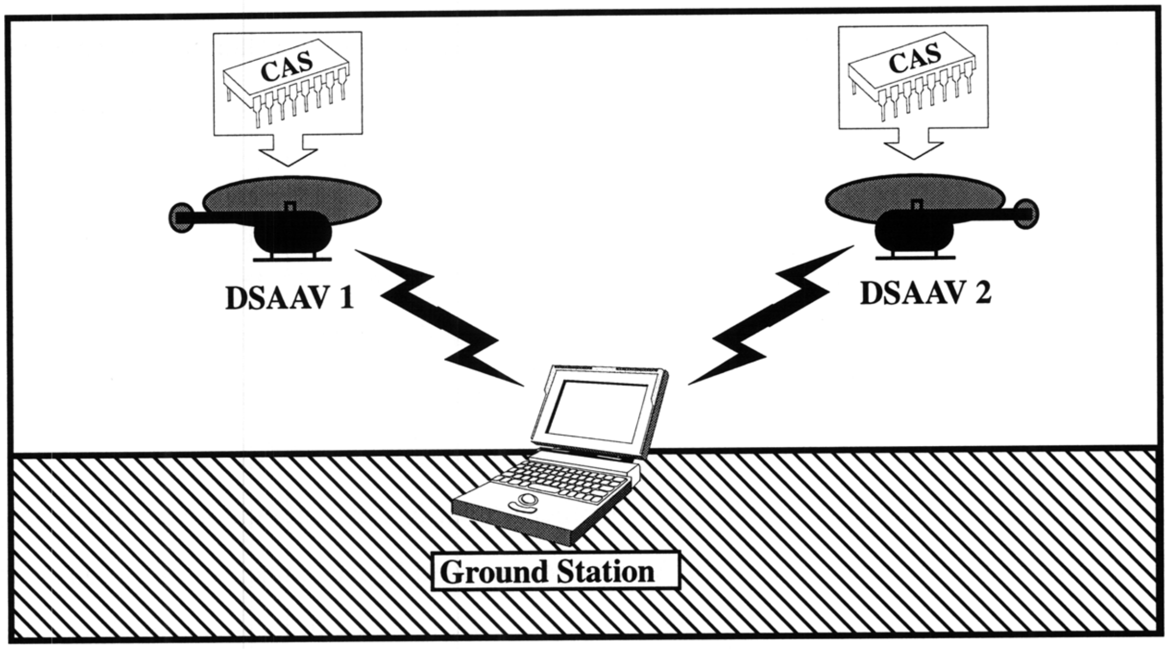


Figure 3.2: *Hybrid architecture*

3.2 Algorithm Complexity

Computational issues in control systems are easily overlooked during design and analysis, but play a critical role in the ultimate implementation of the system. As computer processing speeds and capacities continue to increase, computational requirements may seem less critical. And in all likelihood, today's computational limitations are not as stringent as navigating to the moon on a 64K computer was thirty years ago. However, that does not mean that the edge of the computational envelope is not being stretched today.

One inevitable consequence of increasingly powerful computers is an increase in the complexity of the problems to be solved. Many computational problems that were well out of reach of computers' capabilities in the past are successfully solved today. As more and more of these are solved, more and more difficult problems are introduced to take their place.

The nature of the autonomous collision avoidance problem presents some computational challenges. First, the highly dynamic nature of the multi-AAV system places some bounds on the processing time of the collision avoidance algorithm. The algorithm needs to run fast enough to provide safe commands to the interacting AAVs with their rapidly changing behavior. In a heuristic sense, the speed of the algorithm helps determine the optimality of the avoidance maneuvers. A slowly updating algorithm has to safeguard against drastic changes in the system between updates by not allowing the vehicles to pass very near to one another. A faster algorithm can react more quickly to these unexpected changes and can bring the vehicles closer together.

Perhaps even more important is that the algorithm run in a bounded, known amount of time. In most cases, the vehicles are going to continue their motion, regardless of whether the collision avoidance system has finished computing the appropriate command. This may result in a violation of the separation requirements for the AAVs.

The autonomous CAS problem places other limitations on computation. In the case of DSAAV and other small and micro-AAVs, the size (and hence, capacity) of the on-board computer is severely restricted by the size of the vehicle. In addition, many AAV applications involve fleets of AAVs that are considered almost expendable. In these cases, expensive and powerful computers will be passed up for cheaper, less capable processors.

Ultimately, the complexity of the collision-avoidance algorithm determines the performance of the CAS. The spectrum of algorithm complexity can range from very simple to very complicated. At the simpler end, an algorithm might merely send the vehicles in opposite directions if they get too close to each other. A complex algorithm might calculate the optimal trajectories for all the vehicles, minimizing the combined amount of fuel burnt and guaranteeing the aircraft do not violate their minimum separation criteria.

Global vs. Local Solutions

Tightly coupled with the central vs. distributed question is whether our CAS generates globally or locally optimal solutions. The more global a collision avoidance solution is, the larger number of system variables it considers. A global CAS might try to optimize the trajectories of all the vehicles in a region. As one tries to optimize over an increasingly global system, the complexity and computation time of the problem generally will increase. As they do, it may become less and less feasible to perform these computations on board the AAVs; power and size limitations may make them unsuitable platforms for complex computations. A ground-based processor would likely have fewer such restrictions. Furthermore, as mentioned before, a ground processor is a logical hub in a centralized architecture; it would receive all the information from the vehicles, allowing it to make global decisions.

Collision avoidance solutions that are global in *time* have one additional possible drawback. In nondeterministic systems, which most AAV systems would most definitely be, uncertainty in the state of the system increases the further into the future the algorithm tries to predict. So our AAV CAS could calculate safe vehicle trajectories for the next five minutes, but a disturbance to the system could nullify that solution after only five seconds. If the global CAS could recalculate solutions fast enough (in some bounded time), then it should be able to sustain some (bounded) disturbances. In general, however, there is a trade-off between optimal, global solutions and suboptimal, local solutions that can be computed in real time.

Local collision avoidance, as the name suggests, occurs in a limited region, usually centered around a vehicle of interest. Given this, it seems a waste of time and effort to compute the solution to the local collision-avoidance problem on a platform outside of the region of concern. Performing the computations on one or more of the vehicles in this region is a natural configuration.

Real-time Performance

Algorithm complexity is also driven by how quickly the CAS should react to a dangerous situation. A more complex algorithm will take more processor time to compute. Any algorithm which is to run near real-time will need, at the very least, to consist of a known and bounded number of operations. The performance requirement of the CAS is determined by examining the multi-vehicle system. It is especially important to consider the velocities and other dynamic behavior of the vehicles, how close to one another they operate, and how costly are vehicle collisions. For example, very slow-moving AAVs may not need as reactive a CAS as faster-moving vehicles do.

Number of Vehicles

Multi-AAV environments can contain more than two AAVs. An autonomous CAS can handle this in one of two ways. The first way is for each AAV's CAS to only consider one of the other vehicles at a time. Some selection criterion should be used to determine which other AAV is most critical at the moment. This approach would add little computation to the two-vehicle conflict resolution. Nevertheless, without considering all of the other vehicles, it would be only a local solution.

The second and more global approach is to consider all of the vehicles when computing a collision avoidance maneuver. This approach, however, tends to require a more complex algorithm and create more of a computational burden. However, as computer processors become faster and faster, multi-vehicle collision avoidance will become more and more real-time implementable.

Since the DSAAV is a very agile vehicle, a conflict situation between DSAAV 1 and 2 could arise relatively quickly. Therefore, we decided that we needed our collision avoidance algorithm to run in real time. That means that the algorithm must be simple enough to run—in addition to the other on-board software—on our 486 (586 on DSAAV 2), 50 MHz processor. Since only two DSAAVs currently exist, our algorithm only needs to handle two-vehicle conflicts.

3.3 Interface to Higher Level

In all likelihood, an autonomous collision avoidance system will not be the only controller issuing commands to the AAV. As discussed in Chapter 1, the CAS might act as a safety mechanism which overrides a mission-level planner. If the CAS does act independently of a higher-level planner, it may alter the state of the vehicle such that the mission planner becomes confused or needs to replan from the new state. In such instances, the planner—whether human or computer—would benefit from feedback from the CAS.

In the case where a vehicle is not behaving fully autonomously, there may be a human controller issuing flight commands to the vehicle. These commands can range from high-level, waypoint commands to actual closed-loop, real-time flight control. A feedback mechanism telling the operator that the CAS is active serves several purposes. First, without notification of a collision avoidance maneuver, deviations from the intended path could give the human operator the incorrect idea that the vehicle is malfunctioning. Such a false alarm might result in unnecessary effort from the operator to confirm that the vehicle is behaving properly. This extra work decreases both the efficiency and the autonomy of the system. In a worse-case situation, the false-alarm could result in more drastic action, such as a mission or vehicle abort.

Alerting the operator to a collision avoidance maneuver also gives him/her the chance to replan the vehicles' paths based on their altered trajectories. Generally, a CAS will command the AAV to deviate from its intended flight profile with changes in flight direction or velocity. If these deviations are large enough, the original flight profile may no longer be suitable or optimal for the vehicle. In order to create a modified flight profile, the operator must be alerted to the deviation.

Lastly, feedback is useful when the vehicle is manually telepiloted by the operator and also equipped with an autonomous CAS. Generally, the pilot receives feedback on the state of the vehicle through a visual display. This display may include live video from a camera on the vehicle and/or a computer-generated representation of the vehicle dynamics. If the CAS does not somehow alert the operator when it is active, a confusing discrepancy between control input and display output might be created. When the vehicle is making an avoidance maneuver, the motion of the vehicle as depicted in the teleoperator's display will not correspond with his/her stick movements. This is both disorienting and frustrating to the operator.

Depending on the level of autonomy of the system, the human operator may or may not be actively monitoring the vehicle. This should be considered when designing the operator feedback for the CAS. An operator who is frequently monitoring a visual display for the vehicle or telepiloting the vehicle can probably be notified of a collision avoidance maneuver with a message or graphic on their primary display(s). However, in a system of high autonomy, or when the operator is in charge of monitoring multiple vehicles, it may be necessary to first get his/her attention. Audible signals are an effective way to do this. A simple buzzer, bell, or beep can attract the operator's attention to the display, were a more detailed explanation of the collision avoidance maneuver can be found.

3.4 Existing Collision Avoidance Architectures

As AAVs become more advanced and reliable, they will undoubtedly become more integrated into established air traffic systems. With this in mind, it would be advantageous for the AAV's collision avoidance system to be compatible with any existing collision avoidance architectures. By far the most established and widely used is the Traffic Alert and Collision Avoidance System (TCAS), which has three different versions: TCAS I, TCAS II, and TCAS III. Since 1989, TCAS I has been required on aircraft that can accommodate 10 to 30 passengers. TCAS II is required on aircraft with more than 30 passengers [8].

All of the versions have two interfaces to the pilot, one auditory and one visual. The visual display is a moving-map display centered on the aircraft. It shows the relative positions of nearby aircraft, and uses symbology to indicate how much of a threat of conflict each aircraft poses. When a collision is imminent, the auditory interface notifies the pilot. Alarm sounds are proceeded by a commanded action for the pilot to take to prevent collision.

TCAS I

The most basic of the TCAS versions, TCAS I reports to the pilot the range and bearing of any aircraft within a 4 nautical mile radius. TCAS I also reports the relative altitude of any aircraft equipped with a Mode-C transponder. In addition, TCAS I issues Traffic Advisories (TA) to draw the pilot's attention to a potentially hazardous encounter with another aircraft. When a TA is issued, the pilot will see the symbology of the intruder aircraft change on the TCAS display and will hear an audible "Traffic" alert. It is important to note that TCAS I *only* issues TAs; it does not issue any conflict resolution commands.

The criteria for issuing a TA are based on range to the other aircraft (r), the range rate ($\dot{r}$), the relative altitude between the aircraft (h), and the relative altitude rate ($\dot{h}$). The first test is a range test, which also relies on two threshold parameters, τ_{TA} and DMODTA. TCAS issues a TA if the range to an intruder aircraft is less than distance DMODTA anytime in the next τ_{TA} seconds. If the range to an aircraft is already less than DMODTA, then a TA is issued if the two aircraft are not moving away from each other fast enough. Both DMODTA and τ_{TA} depend on the aircraft altitudes.

For a TA, τ_{TA} can range from 48 seconds above FL300 to 20 seconds at altitudes below 1000 feet above ground level (AGL). DMODTA is 3.0 nmi above FL300 and reaches its minimum of 1.0 nmi between 1000 and 2350 feet AGL.

TCAS uses a similar test in the vertical direction using the parameters ZTHR and τ_V. TCAS issues a TA if the current relative altitude and the relative altitude at the time of closest approach are both less than ZTHR. A TA can also be issued when the relative altitude is greater than ZTHR, as long as the time to coaltitude (τ_V) is small. This happens when the relative altitude at closest approach is less than ZTHR or the relative velocity passes through zero before the point of closest approach [9].

For an autonomous collision avoidance system to be compatible with an existing system like TCAS, there must be both *hardware* and *algorithm* compatibility. The only hardware needed for compatibility with TCAS I (and the U.S. ATC system) is a transponder (preferably a Mode-C transponder). On smaller AAVs, finding the space, power, and payload capacity for the transponder may be an issue. Nonetheless, compatibility with TCAS might be a critical consideration when determining the size and function of an AAV.

TCAS I only issues Traffic Advisories, which require the pilot to make visual contact with the other aircraft before executing any collision avoidance maneuver. Since the pilot—and *not* TCAS—determines this maneuver, there is no easy way to coordinate the collision avoidance maneuvers of the aircraft and AAV. Therefore, the autonomous CAS algorithm only needs to be compatible with TCAS's conflict prediction algorithm.

So, what constitutes algorithm compatibility in TCAS I? First, the autonomous CAS needs to rely only on the information which it receives through the TCAS I system. In addition, the autonomous algorithm should balance safety and efficiency, while minimizing the number of false alarms on the other aircraft's TCAS.

TCAS II

TCAS II has all of the previously-mentioned capabilities that TCAS I does, and also issues collision avoidance commands known as Resolution Advisories (RA). With TCAS II, all RAs are vertical commands, either to climb or descend. The alerting criteria for an RA are the same as those of a TA, with the parameters τ_{TA} and DMODTA being replaced by τ and

DMOD. In other words, if a TA is not resolved and the aircraft continue to close in on each other, the TA can escalate into an RA. The value of τ is generally about fifteen seconds less than τ_{TA}.

Unlike TAs, which require visual contact before a maneuver is initiated, RAs require immediate action. In the event of an RA, TCAS issues an audible resolution command to the pilot. The ten possible RAs are as follows: climb or descend (at 1500 fpm), don't descend or don't climb, and limit descent or climb (to 500,1000, or 2000 fpm). The TCAS logic selects the resolution advisory which minimizes the aircraft's deviation from its current trajectory while still satisfying the minimum separation requirement. The logic also assumes that the resolution maneuver starts 5 seconds after the RA is issued and the vertical acceleration during the maneuver is 0.25 g.

Like with TCAS I, we need to address the compatibility of our autonomous collision avoidance system with TCAS II. The hardware compatibility is the same as with TCAS I, except Mode-C is now required. The autonomous CAS algorithm needs to be compatible with TCAS II in both conflict prediction and resolution. At the very least, this should include the compatibilities mentioned with TCAS I. In addition, we might want to coordinate the resolution directions of the vehicles. If the autonomous collision avoidance system has a choice of maneuvers, it should choose the one which best complements the maneuver chosen by the other aircraft's TCAS.

TCAS III

TCAS III has all the capabilities of TCAS II plus a selection of horizontal resolution maneuvers. Additionally, TCAS III can take advantage of advanced datalink technology between aircraft in order to coordinate the resolution maneuvers. An autonomous collision avoidance system might want to complement TCAS III by also allowing horizontal maneuvers. And like TCAS III, the AAV's CAS could use datalink to negotiate avoidance maneuvers.

3.5 Separation Requirements

Before we can design a collision avoidance algorithm, the separation requirements of the

CAS must first be considered. The separation requirements describe the *protected zone* around the vehicle into which no other vehicle is to enter. The protected zone should reflect how close we never want our AAVs to get to one another. It defines when a collision occurs. In other words, if the protected zone is violated, a collision has occurred, even if the vehicles did not actually come into contact with each other. The two key characteristics of this protected zone are its *size* and its *shape*.

Much of the aircraft and robot collision avoidance literature has looked only at horizontal resolution maneuvers and speed changes, thus reducing the problem to two dimensions [10,11,12,13]. In 2-D, the protected zone is often represented as a circle centered on the vehicle. A circular protected zone makes sense logically and is generally mathematically easy to work with. As we move into three dimensions, the circle is no longer a valid protected zone, and we must look for a new one. In aviation, cylindrical airspaces are often considered. TCAS is designed to enforce a cylindrical protected zone around each aircraft, as shown in Figure 3.3. This is reasonable, since the shape of an aircraft can be approximated with a cylinder.

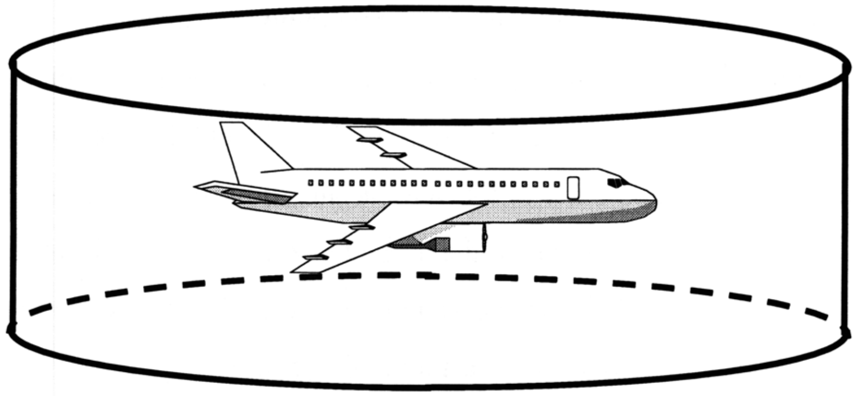


Figure 3.3: *Cylindrical protected zone*

For the protected zone around DSAAV, we could find a cylinder or other shape which best approximates the shape of the vehicle. As we will see in Chapter 5, however, a sphere centered on the vehicle mathematically proves to be a good choice of protected zone. We note that this choice is dependent on our actual collision avoidance algorithm, and other shapes may be more compatible with other algorithms.

The actual size of the protected zone depends on what we consider a "collision" to be. One definition of a collision (the literal one) would be when the two vehicles actually contact one another. In passenger aircraft, passing within one hundred feet of another plane might reflect poorly—at least in the eyes of the passengers—on the airline and the aviation industry. In that case, the protected zone would need to be somewhat larger than the aircraft itself in order to encompass some of those near-miss scenarios.

In the case of the DSAAV, we consider the effect of one vehicle's downwash on the other vehicle. As the AAVs pass over one another, the downwash of the upper vehicle might disturb the lower one. This effect is as of yet unmodeled, so we choose our protected zone so that this downwash interaction is minimal.Currently, our protected zone is a sphere of radius 12 feet that is centered on the DSAAV. Flight tests will show whether the downwash effect is more or less severe than we are estimating it to be.

Note that defining the protected zone does not take into account the dynamics of the vehicles or any uncertainty associated with their motion. It is tempting to compensate for navigation errors, pilot error, or wind gusts by simply expanding the protected zone. We could even try to justify changing the shape of an aircraft's protected zone. Since the majority of an aircraft's motion is generally in the horizontal plane, two planes should be able to pass closer to each other in the vertical direction than in the horizontal. Does that mean we should intentionally flatten the protected zone? No, the above mentioned factors should and will be considered in the actual algorithm design, not in the protected zone specification. This will be discussed further in Chapter 5.

3.6 Dynamic Model

Before designing a CAS, the multi-vehicle dynamic model that the system must satisfy should first be defined. In the case of the DSAAV collision avoidance system, only one-on-one vehicle encounters are considered and solved. Each vehicle considers its motion relative to the other AAV. This generates a relative frame of motion for the two-vehicle system.

First, however, we consider each vehicle's position and velocity relative to a north-east-down frame. The north, east, and down axes of the frame are referred to as the x, y, and z axes.

$$\boldsymbol{x_1} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \qquad \boldsymbol{x_2} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{3.1}$$

$$\boldsymbol{V}_1 = \begin{bmatrix} V_{x_1} \\ V_{y_1} \\ V_{z_1} \end{bmatrix} \qquad \boldsymbol{V}_2 = \begin{bmatrix} V_{x_2} \\ V_{y_2} \\ V_{z_2} \end{bmatrix} \qquad 1 \tag{3.2}$$

Subtracting the state of vehicle two from that of vehicle one gives us the relative state of the vehicle:

$$\boldsymbol{x_r} = \boldsymbol{x_1} - \boldsymbol{x_2} = \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} \tag{3.3}$$

$$\boldsymbol{V_r} = \boldsymbol{V}_1 - \boldsymbol{V}_1 = \begin{bmatrix} V_{x_1} - V_{x_2} \\ V_{y_1} - V_{y_2} \\ V_{z_1} - V_{z_2} \end{bmatrix} = \begin{bmatrix} V_{x_r} \\ V_{y_r} \\ V_{z_r} \end{bmatrix} \tag{3.4}$$

We will only considering avoidance maneuvers in the vertical direction, so only V_z will be time-varying. The horizontal trajectories of the AAVs are assumed to be straight lines, so V_x and V_y are constant. The closed-loop dynamics in the vertical direction are approximated by a first-order system with time constant, τ_v:

$$\frac{dV_x}{dt} = \frac{dV_y}{dt} = 0 \qquad \frac{dV_z}{dt} = \frac{(V_{z_{comm}} - V_z)}{\tau_v} \tag{3.5}$$

So, the two-vehicle system is as follows:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ V_{x_1} \\ y_1 \\ V_{y_1} \\ z_1 \\ V_{z_1} \\ x_1 \\ V_{x_2} \\ y_1 \\ V_{y_2} \\ z_1 \\ V_{z_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_v} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_v} \end{bmatrix} \begin{bmatrix} x_1 \\ V_{x_1} \\ y_1 \\ V_{y_1} \\ z_1 \\ V_{z_1} \\ x_1 \\ V_{x_2} \\ y_1 \\ V_{y_2} \\ z_1 \\ V_{z_2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{V_{z_{1_{comm}}}}{\tau_v} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{V_{z_{2_{comm}}}}{\tau_v} \end{bmatrix} \tag{3.6}$$

The dynamic model of the system should also include any limits on velocity or acceleration. Although the DSAAV has a wide dynamic envelope, its vertical velocities are limited to values between 1 ft/sec (down) and -2 ft/sec (up).

Chapter 4

Collision Avoidance Algorithm Design

Once the requirements of the collision avoidance algorithm have been specified, it is time to actually design the algorithm. Sections 4.1 and 4.2 provide a generic outline for designing a collision avoidance system. Then Section 4.3 describes the design of the DSAAV collision avoidance system

Although the design of a collision avoidance algorithm can be gone about in a number of ways, it is convenient to break up the collision avoidance task into two tasks: *conflict prediction* and *conflict resolution.*

4.1 Conflict Prediction

In conflict prediction, the collision avoidance system tries to determine whether any vehicle's protected zone will be violated by any other vehicles. Depending on how close the CAS thinks the vehicles will pass by one another, it may or may not issue a conflict resolution command. How does it make this decision? The CAS considers three elements in making this decision: *vehicle information*, a *trajectory prediction*, and *conflict thresholds.*

4.1.1 Vehicle Information

Before a collision avoidance system can even begin to detect a conflict, it needs information about the vehicles involved. As discussed in Section 3.1, some of this information may be actively sensed (via radar or other ranging instrument), while other information can be transmitted from the vehicles. In a distributed system, each vehicle's collision avoidance algorithm also will poll its own vehicle's flight control system for information . Some the most useful information might be vehicle *position*, *velocity*, *intent, health,* and *dynamic model.*

The importance of knowing the AAVs' positions is obvious. More important is the relative positions of the vehicles. If we are using a radar-type instrument on board an AAV, relative position is what it will give us. If we are using transmitted position data, the vehicles' absolute positions will have to be subtracted from one another to get their relative positions. In this case, each vehicle needs adequate navigation instruments (such as GPS and/or inertial navigation) to ascertain its own position.

As important as position information is to know where the vehicles are, velocity information is to know where they are going. In fact, in many collision avoidance algorithms, positions and velocity data is the only information that is used for conflict prediction [7,10,11,12,13]. Usually velocity data will be transmitted among the vehicles, although it also can be estimated from successive position updates using filtering techniques discussed in Section 4.1.2.

Intent information describes the future path of the AAVs. Velocity is actually first-order intent information, but we will speak here of other types of intent information. Higher-order state derivatives, such as vehicle acceleration, provide insight into the future trajectories of the vehicles.

Also, intent information can be in the form of knowledge about the future actions of the vehicles. For example, if the AAVs are following a prescribed set of waypoints, knowledge of these waypoints may be used by the CAS. The vehicle velocities and accelerations may indicate a collision is imminent, but the waypoint information might show that one of the vehicles is scheduled to make a drastic change of course that will eliminate the potential conflict. This is illustrated in Figure 4.1.

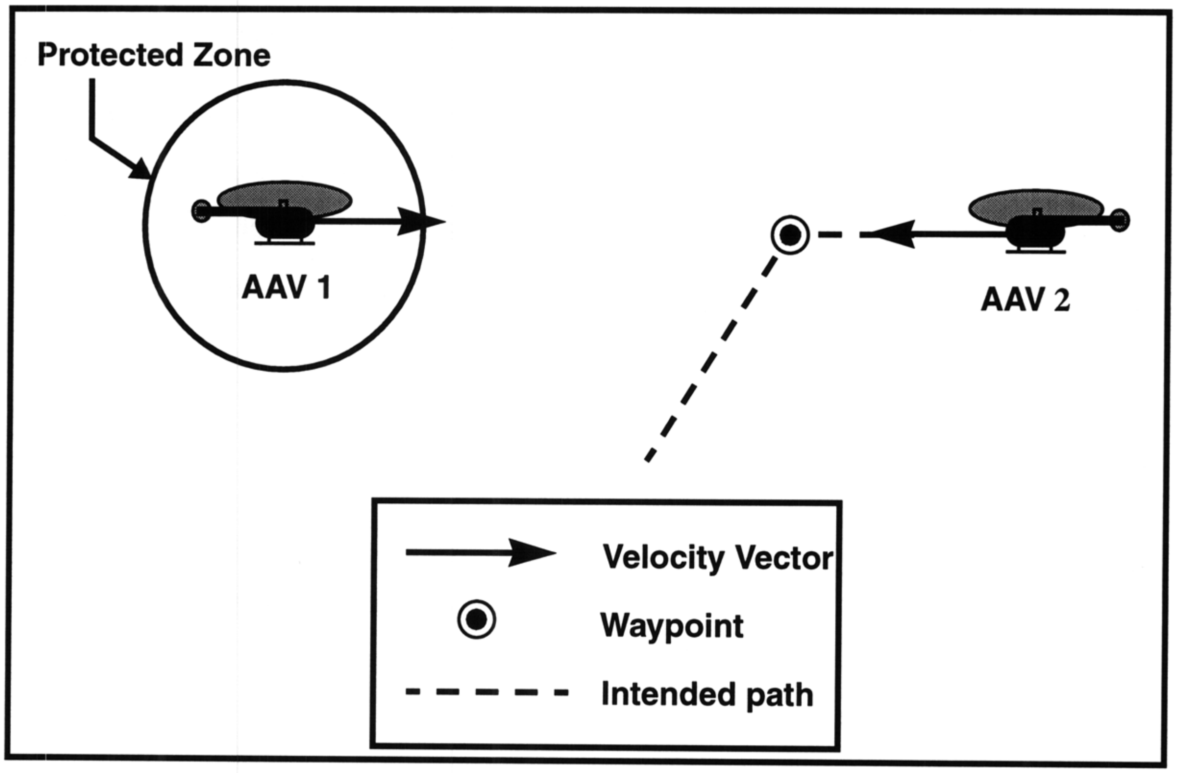


Figure 4.1: *Intent information prevents unnecessary collision avoidance maneuver*

As will be explained later, both the vehicles' health and dynamic models are useful in predicting the AAVs' motions. Vehicle health pertains to whether or not the AAVs are functioning properly. The dynamic models of the vehicles describe the maneuverability and flight envelope of the AAVs. These two elements may not be as fundamental as position, velocity, and intent, but they can assist the collision avoidance algorithm with more accurate conflict prediction.

Ideally, we want our collision avoidance system to have as much information as possible. Realistically, the amount of information available for conflict prediction depends on the available communication bandwidth and the radar/rangefinding capabilities of the AAVs. Even if we have very complete and accurate information about all the AAVs, our algorithm may be too simple to properly utilize this data. In any case, if we are limited to what information our CAS can use, we must prioritize and use the most critical data. In many cases, position and velocity data are considered to be the most essential. Nevertheless, that can vary and must be evaluated for each specific system

4.1.2 Trajectory Prediction

Once the collision avoidance system has information about the vehicles, it takes its next step toward conflict prediction. The CAS must predict the future trajectory of the AAVs, given their state data. This is where the vehicle dynamic models are useful. We assume we know something about how the vehicle moves, and that allows us to predict its motion.

The simplest model assumes the vehicle follows a constant-velocity trajectory:

$$\frac{d}{dt}X(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ v(t) \end{bmatrix} \tag{4.1}$$

So, given the vehicle position and velocity at time t_0, we can calculate its position and velocity and any later time, t. To do so, we must find the state transition matrix, $e^{A(t-t_0)}$, where

$$X(t) = e^{A(t-t_0)}X(t_0) \tag{4.2}$$

Given our state-space model in Equation (4.1), we see that

$$\Phi(s) = (sI - A)^{-1} = \begin{bmatrix} s & -1 \\ 0 & s \end{bmatrix}^{-1} = \begin{bmatrix} s^{-1} & s^{-2} \\ 0 & s^{-1} \end{bmatrix} \tag{4.3}$$

and taking the inverse Laplace transform,

$$e^{A(t-t_0)} = L^{-1}\{\Phi(s)\} = \begin{bmatrix} 1 & (t-t_0) \\ 0 & 1 \end{bmatrix} \tag{4.4}$$

which gives us the future trajectory of the vehicle:

$$X(t) = \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} 1 & (t-t_0) \\ 0 & 1 \end{bmatrix}X(t_0) = \begin{bmatrix} x(t_0) + v(t_0)(t-t_0) \\ v(t_0) \end{bmatrix} \tag{4.5}$$

After extrapolating all of the AAV's trajectories, they can be subtracted to predict the relative motion of the vehicles.

We could do a similar trajectory prediction if given vehicle accelerations. Many maneuvers, such as turns and leveling off at an altitude, can be closely approximated by constant-acceleration dynamics:

$$\frac{d}{dt}X(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ v(t) \\ a(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ v(t) \\ a(t) \end{bmatrix} \tag{4.6}$$

Calculating the state transition matrix and assuming $t_0 = 0$ yields the familiar

$$X(t) = \begin{bmatrix} x(t) \\ v(t) \\ a(t) \end{bmatrix} = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} X(t_0) = \begin{bmatrix} x(t_0) + v(t_0)t + \frac{1}{2}a(t_0)t^2 \\ v(t_0) + a(t_0)t \\ a(t_0) \end{bmatrix} \tag{4.7}$$

which allows us to propagate the vehicle motion forward in time. Once again, subtracting the AAV's trajectories will give us their predicted relative motion.

The above examples use very simple dynamic models. More accurate models that reflect the maneuverability of the AAVs could also be used. Unfortunately, linearized state-space models, which are useful here and will be in the next section, usually require that the vehicle's nonlinear dynamics be linearized about a specific operating state. While the linearization itself is not difficult, storing the linearized dynamics for the vehicle's different flight regimes may become unwieldy. Furthermore, the CAS needs to be able to correctly identify which flight regime the vehicles are currently in.

If the vehicle velocities or accelerations were in fact constant, and we could actually measure them exactly, then the above trajectory predictions would be adequate. But we know this is not the case. First of all, it is highly unlikely that the state measurements from which the trajectory is extrapolated are exact. Looking at Equation (4.7), we see that velocity and acceleration errors will cause the position error to grow linearly and quadratically in time respectively. Also, it seems wasteful to repeatedly update our model with new state measurements while just throwing out the old estimates. Instead, a weighted average of the measurements might produce a more accurate estimate and slow the growth of the errors.

Secondly, the vehicles' velocities and accelerations will not be constant most of the time. And like the measurement errors, such modeling errors will yield position errors that grow with time. Unfortunately, it is impossible to eliminate *all* modeling errors. It would be helpful, however, to have some metric for the accuracy of our trajectory prediction as we extrapolate it forward in time.

Lastly, we face the situation where no state derivative measurements are available, only vehicle position. A single position measurement will tell nothing about the future trajectory of the vehicle. However, if we have a series of position measurements, we should be able to somehow average them to generate an estimate of velocity and acceleration.

Kalman Filter

All of the above limitations can at least partially be overcome with one tool: the Kalman filter. The Kalman filter is an optimal estimator, which is described by Gelb [14] to be

> "a computational algorithm that processes measurements to deduce a minimum error estimate of the state of a system by utilizing: knowledge of system and measurement dynamics, assumed statistics of system noises and measurement errors, and initial condition information."

So each time a new measurement is received, the Kalman filter filters the data and produces an optimal (in a least-squares sense) estimate of the state. In between measurements, it acts as a *predictor* of the state, propagating the linear model forward in time.

The Kalman filter is recursive, which means that all previous measurements influence the state estimate, but none of them need to be stored in memory. Furthermore, the Kalman filter keeps track of the accuracy of the state estimates in a state covariance matrix. Like the state estimates, the covariance gets updated with each new measurement and gets propagated in between measurements. Lastly, the filter can estimate vehicle velocities and accelerations based only on vehicle position measurements [14].

Vehicle health information can be used to update the assumed accuracies of the measurements and the models. These accuracies are important parameters in the Kalman filter. For example, suppose the collision avoidance system learns that one of the AAV's GPS receiver goes from tracking six satellites to only tracking four. The CAS should then increase the noise in position measurements of that vehicle. The end result will be that the CAS Kalman filter will trust the position updates from that vehicle less.

Similarly, the CAS could discover that an AAV has had a mechanical or control system breakdown, and is behaving unpredictably. In that case, the vehicle process noise should be made large to reflect the malfunction. Due to this increased uncertainty, the vehicle's state covariances will grow faster in between measurement updates.

Also, the Kalman filter can act as an all-purpose trajectory predictor in a collision avoidance system. The filter can take in whatever data it can get (radar returns, accurate state data via a transponder, laser rangefinder measurements) about the vehicles, and produce an optimal estimate for their future trajectories. Most AAVs probably already use a Kalman filter to track their own states. These existing filters can just be modified for collision prediction by adding states for the other vehicles.

4.1.3 Conflict Thresholds

Once a conflict prediction algorithm has some idea of where the AAVs are going to be in the future, it faces one last question: will there be a conflict? In other words, should we initiate a conflict resolution maneuver at some point? Given the uncertainty in the vehicles' motion in the future, this is a difficult question to answer with much certainty.

Nonetheless, there are a number of possible approaches to this problem. The first assumes the future trajectories of the vehicles are deterministic and known. The second tries to place lower bounds on how close the vehicles can come to one another. Finally, the last methods seek to establish conflict probabilities for the AAVs.

If we assume that the vehicle trajectories are exactly those found by the trajectory predictor, it is simple to determine if any of the AAVs' protected zones will be violated. This will be done in Section 4.3 for DSAAV's collision avoidance system. As we will see, finding the time and distance of closest approach for constant velocity trajectories requires the solution of a linear equation (a quadratic equation in the constant acceleration case). If the predicted separation distance at closest approach is less than the minimum allowed separation, a conflict is predicted (see Figure 4.2).

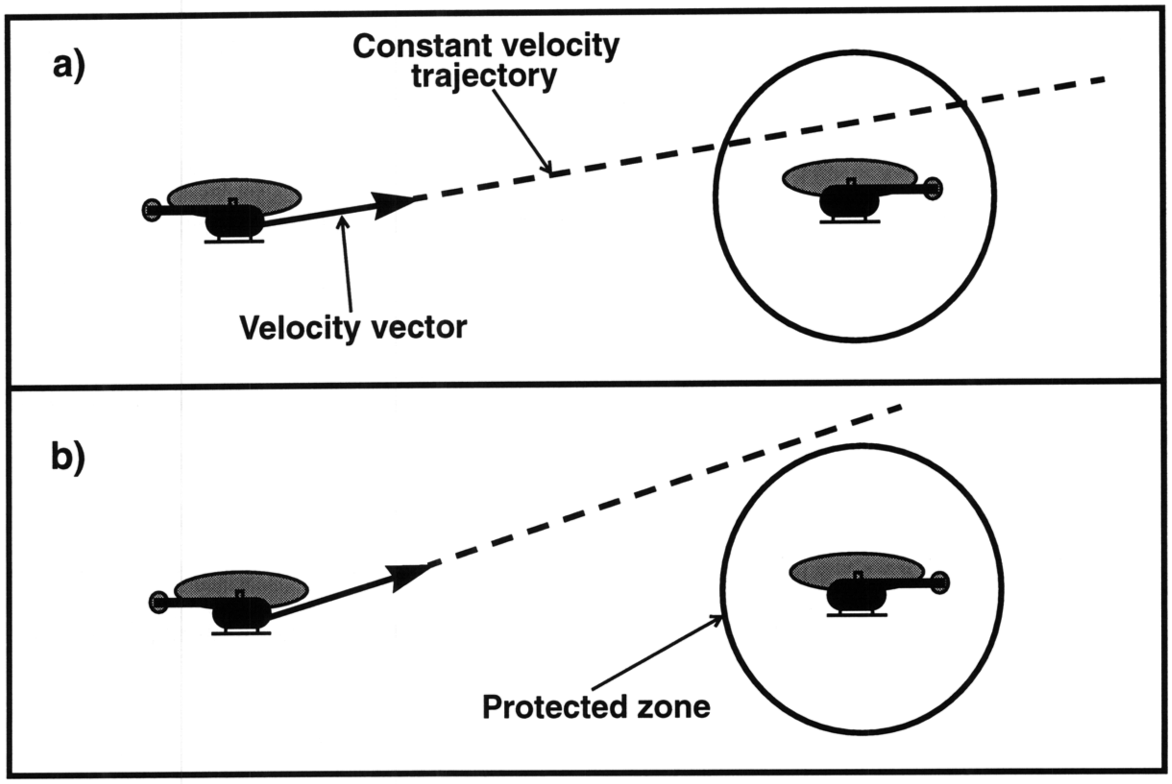


Figure 4.2: *Deterministic conflict detection in the a)conflict and b)no conflict cases*

In either case, as the AAVs stray further and further from these assumed trajectories, the chance of incorrect conflict prediction increases. Depending on which direction they deviate from the assumed trajectory, the result will either be a *missed detection* or a *false alarm*. A missed detection occurs when the algorithm incorrectly decides that no collision will occur. In a false alarm, the algorithm incorrectly believes a collision will occur, perhaps even issuing a resolution maneuver.

In [10], a methodology is introduced that computes a guaranteed lower bound for the closest approach distance of two vehicles, given bounded uncertainties in their motion. For example, one scenario has two vehicles flying on perpendicular trajectories. For a given uncertainty in their along-track accelerations, a lower bound on their miss distance can be computed by checking the feasibility of linear matrix inequalities [10]. Other examples include uncertain velocities, headings, and switch times, as well as a three-dimensional conflict. The algorithm is not prohibitively complex, running in predictable, polynomial time. And while the analysis does not necessarily determine if a conflict *will* occur, it is very useful in that it does determine if a conflict definitely will *not* occur.

Lastly, [15,16] introduce various probabilistic methods for determining the likelihood of a collision. In [15], Monte Carlo simulation is used to generate a look-up table of conflict probabilities. Given the relative locations, velocities, and headings of two vehicles, one can determine the probability that their protected zones will be violated. This method requires extensive Monte Carlo simulation, and each set of Monte Carlo runs is only valid for a to a particular probabilistic model. If any parameters are changed, a new battery of Monte Carlo runs have to be made. This approach does, however, provide a quick and simple means of predicting conflict.

In [16], an analytic solution to conflict probability estimation is presented. First, the position errors of each AAV are assumed to be represented by a gaussian probability density distribution. The root mean square (RMS) error in x- and y-position form the axes of a *position error ellipse* for each vehicle (see Figure 4.3). The two ellipses are then combined into one *combined error ellipse*, which is assigned to one vehicle. This ellipse depicts the RMS errors in the vehicles' relative position, also described by a gaussian distribution. The volume under this distribution is, of course, unity.

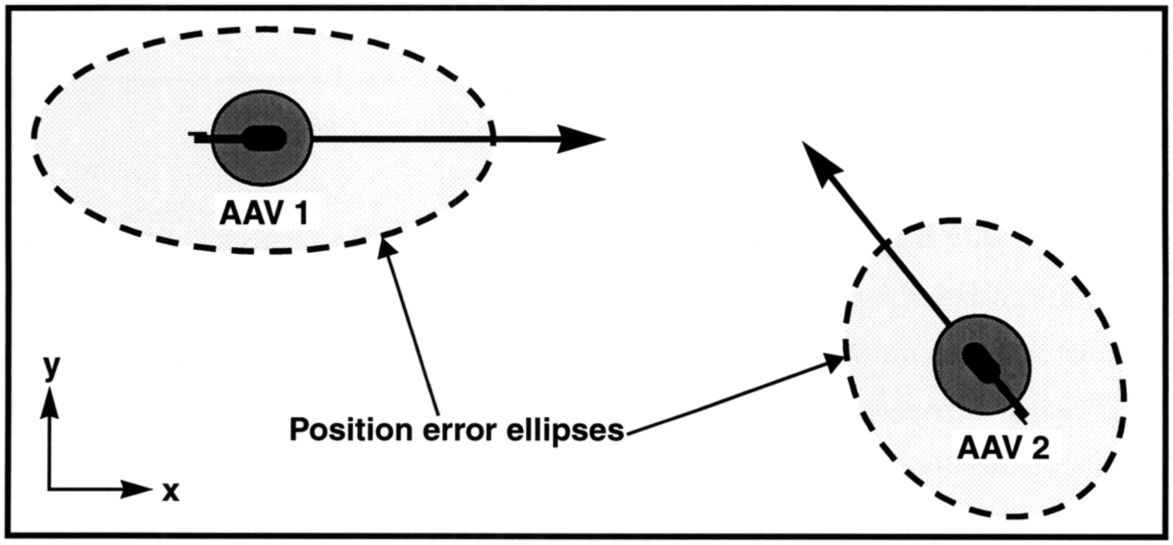


Figure 4.3: *Position error ellipses for the AAVs*

Each vehicle's protected zone consists of a circle; the two protected zones are combined into one, and it is assigned to the other vehicle and moves relative to the combined error ellipse as the vehicles move relative to each other. The region formed by the moving protected zone is the *extended conflict zone*. The conflict probability at any instant is the volume under the probability density distribution that intersects the protected zone. The conflict probability over all time is the volume underneath the gaussian distribution that intersects the extended conflict zone. See Figure 4.4.

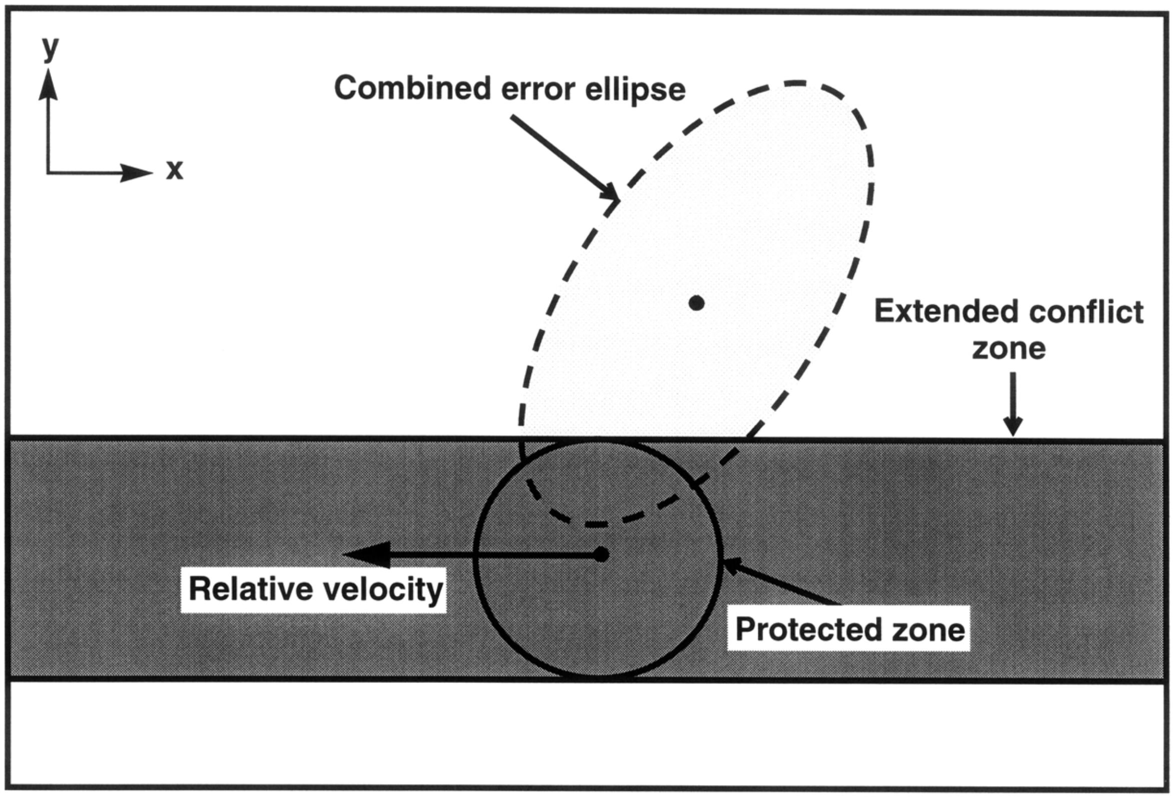


Figure 4.4: *Encounter geometry (adapted from [16])*

To simplify the integration of the probability density, the combined error ellipse is transformed into the unit circle, and the circular protected zone is transformed into an ellipse. After the transformation, the volume under the portion of the probability density surface that intersects the extended conflict zone can be found analytically. This is the total conflict probability. The analysis is validated via Monte Carlo simulation, and is extended into three dimensions.

Note that the above two methods only result in a conflict probability. A decision still needs to be made, however, about whether a collision avoidance maneuver is necessary. Since the position probability distribution is gaussian, the conflict probability is never exactly equal to zero. Therefore, we theoretically cannot completely eliminate the possibility of a collision. We can, however, choose how high the conflict probability should be before we deem it a conflict situation. This cutoff probability is known as the *alerting threshold*.

Determining the location of the alerting threshold consists of finding a balance between safety and efficiency. If we set the value of the threshold too high, we may allow the AAVs too close to one another, increasing the probability of a missed detection. Setting the threshold very low results in a safer system, but probably one that is unnecessarily inefficient. By not letting the vehicles anywhere near each other, a low threshold disrupts the vehicles' intended paths and increases the probability of false alarms. The value of the alerting threshold should reflect the desired level of safety in the conflict prediction. Some approaches to setting thresholds are presented in [9].

4.2 Conflict Resolution

Once a conflict is predicted, a collision avoidance system should command a maneuver that attempts to prevent a collision. There are many potential ways to do this, but key decisions must be made for each design.

4.2.1 Maneuver Direction

Many of the collision avoidance algorithms in the literature seem to choose a maneuver direction with little thought or justification. As we will see, however, this decision can depend on many factors. Collision avoidance maneuvers usually fall into one of four categories: *horizontal maneuvers*, *vertical maneuvers*, *speed changes*, and *combination maneuvers*.

Horizontal Maneuvers

Most algorithms in the collision avoidance literature seem to use horizontal resolution maneuvers. One reason is that they allow three-dimensional environments to be simplified to two dimensions. These planar conflicts are usually simpler to analyze than their 3-D relatives. Furthermore, horizontal conflict resolution techniques for air vehicles can be extended to and borrowed from robots, cars, people, and anything else that operates in a two dimensional environment.

Horizontal maneuvers do have one significant drawback, however. They require vehicles to deviate from their intended trajectories. In the U.S. airspace, for example, the minimum aircraft separation is approximately 15 times greater in the horizontal plane than in the vertical direction. This could lead to inefficient deviations if only horizontal maneuvers are used [16].

Speed Changes

Using along-track speed changes to avoid collision eliminates the need to deviate from the desired trajectory. Furthermore, speed changes are useful in resolving conflicts between vehicles moving with parallel trajectories, where other maneuver directions can keep the vehicles maneuvering back and forth as they try to avoid one another while also maintaining their intended path. Lastly, speed changes also tend to be simpler to analyze and implement.

Speed changes do have their share of limitations, however. First, air vehicles do not usually have a wide range of operating velocities. Therefore, speed changes alone may not be sufficient to resolve some conflict situations. Furthermore, in large aircraft, speed changes tend to take a long time. This, too, reduces their effectiveness in collision avoidance [7]. Lastly, as the relative heading of the two vehicles approaches 180° (head-on conflict), speed changes become less and less effective. In fact when the relative heading is 180°, no speed change—except both vehicles stopping—can prevent an imminent collision.

Vertical Maneuvers

Unlike speed change maneuvers, vertical avoidance maneuvers are equally effective, regardless of the relative headings of the vehicles. In addition, they do not require horizontal trajectory deviations like horizontal maneuvers do. This is especially important when the ground track of the AAV is significant, like in a mapping or surveillance mission.

As we mentioned in the horizontal maneuver section, minimum separation requirements in some airspaces make vertical maneuvers much more efficient than horizontal maneuvers [16]. This is one of the reasons that TCAS uses vertical resolution maneuvers. [16] also mentions that vertical maneuvers might have an advantage in conflicts involving more than two vehicles. In those scenarios, assigning the vehicles to separate altitudes might be the simplest way to prevent conflict. Vertical maneuvers are limited, however, when operating AAVs near the ground, as they are limited in how far they can descend.

Combination Maneuvers

Any maneuver that combines two or more of the above maneuvers can be called a combination maneuver. Combination maneuvers are advantageous in that they can reap the individual benefits from each of the maneuvers. They are generally more complex and hence more difficult to solve and implement

A variation on combination maneuvering is a collision avoidance system that commands horizontal, speed, and vertical maneuvers, but not at the same time. Such a system should apply the resolution trajectory that best suits the specific encounter. For example, depending on the shape of the protected zone and the relative position of the vehicles at closest approach, it would choose a vertical or horizontal maneuver, respectively. For example, in Figure 4.5a, we can see that a vertical maneuver would be more efficient than a horizontal one. But the encounter in Figure 4.5b would require little horizontal deviation to avoid collision.

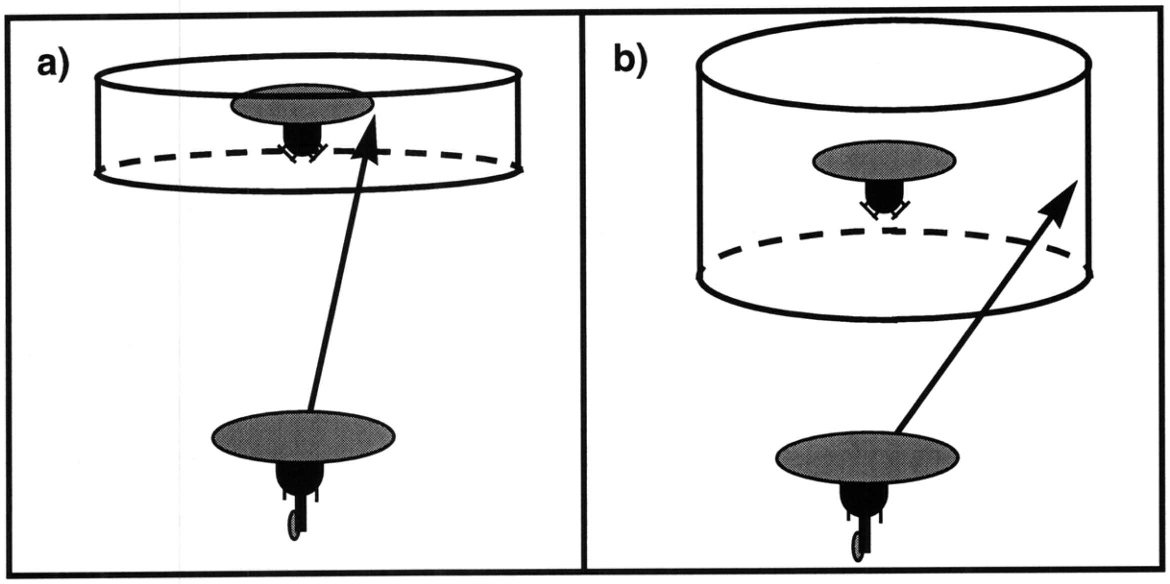


Figure 4.5: *Encounter-specific maneuvers*

4.2.2 Maneuver Commands

The key component of a conflict resolution maneuver is the actual command that is sent from the collision avoidance system. For convenience, we will place these maneuvers in only two groups: *bang-bang* and *continuous* maneuvers.

Bang-Bang Maneuvers

We refer to bang-bang maneuvers as those which generally consist of a one-time avoid command and a one-time return command. The return command brings the AAV back to its original trajectory after the conflict has been eliminated. The avoid and return commands may have one or more preset magnitudes and directions. And generally, the maneuver is open-loop; once it begins, it does not recalculate or readjust unless a new potential conflict has been created. TCAS resolution commands could be considered bang-bang maneuvers; once a command is given, the pilot follows it until the conflict passes.

Because of the open-loop nature of bang-bang maneuvers, it is especially important to start the avoid and return at the right times. Remember that the further in the future the potential conflict is, the less certain we can be that it will actually occur. Hence, maneuvering too early can result in unnecessary maneuvering; sometimes, if let be, the conflicts will resolve themselves naturally. On the other hand, if we wait until too near to the time of the conflict, it may prove to be too late. In [9], a hypothesis testing methodology is presented for determining the best time to initiate the avoidance maneuver.

The timing of the return maneuver is equally critical, and can be looked at using the same techniques found in [9]. It does, however, have some peculiarities that make it different than the avoidance maneuver. The most critical of these is the tendency to start the return too early, thus creating a conflict situation again. This can occur if the return criterion is that the projected return trajectory will not cause a foreseeable conflict. Normally, this will bring the AAVs back to their original path after the conflict passes (see Figure 4.6a). However, if the AAVs are still relatively far away and have just begun an avoidance maneuver, the projected return trajectory also might not cause a conflict. In that case, the trajectory falls *in front of* the protected zone, but fools the system into thinking the conflict is in the past (see Figure 4.6b).

Since the vehicles do not reach their return velocity instantaneously, their return trajectory does not take them back as quickly as the system thinks, and collision once again becomes imminent. The avoidance maneuver is initiated once again. This can repeat itself, producing an unwanted, although not necessarily harmful, on/off switching. This switching is shown for a simulated head-on conflict encounter in Figure 4.7. This phenomenon might be partly eliminated by making the return trajectory more gradual, at least early on the avoidance maneuver.

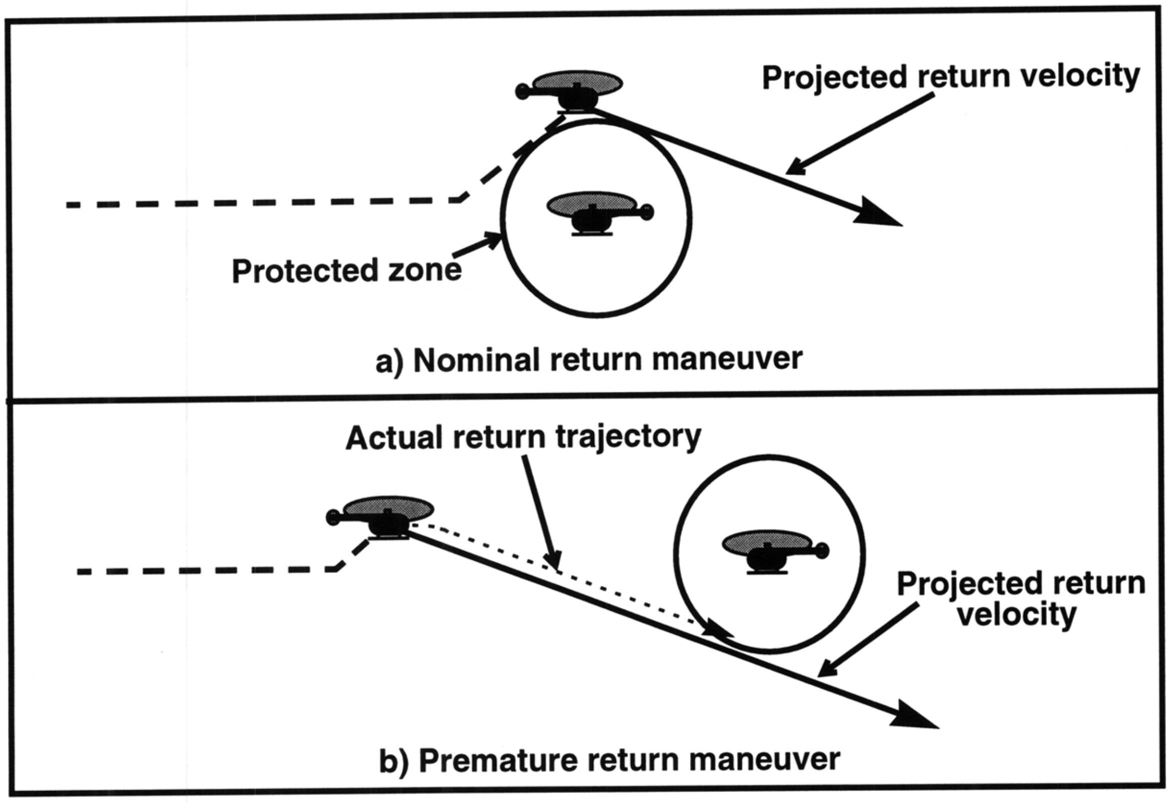

Projected return velocity
Protected zone
a) Nominal return maneuver
Actual return trajectory
Projected return velocity
b) Premature return maneuver

Figure 4.6: *Return maneuvers*

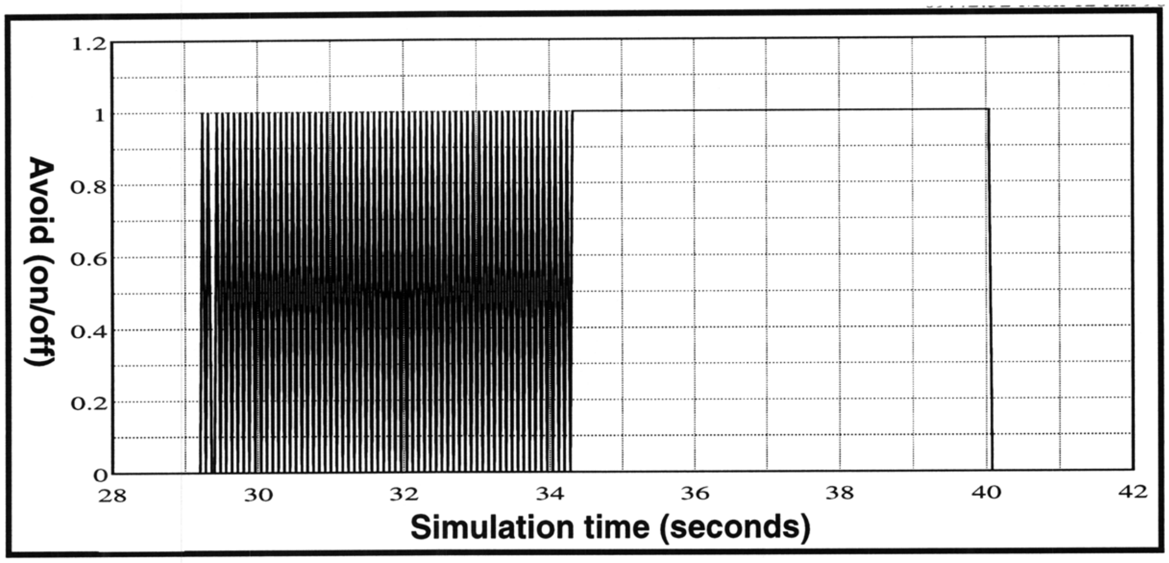


Figure 4.7: *Simulation result: premature return switching*

Continuous Maneuvers

Continuous maneuvers, unlike the bang-bang maneuvers, are closed-loop, reevaluating the avoidance command at each time step. Because these avoidance commands are constantly being recalculated, when they begin and end is not as important as with the bang-bang maneuvers. There are endless possibilities for continuous conflict resolution maneuvers, but most seem to have one thing in common: generating optimized trajectories.

Ideally, continuous collision avoidance controllers would generate globally optimal trajectories in real-time. That generally is not computationally feasible at this point. It is conceivable that such techniques will become feasible in the future, however. Until then, some have used such optimization techniques such as game theory [13] and potential and vortex field motion planning [12] to produce optimal trajectories *off-line*. Their idea is to then approximate the trajectories with line segments and circular arcs. An on-line controller could then follow these approximate paths.

At Seagull Technology, they have solved for optimal collision avoidance maneuvers and generated extensive maneuver charts to be stored and used on-line [7]. Given the relative trajectories of the vehicles, an optimal heading or speed change can be found quickly . The difficulty with this method is generating and storing the maneuver charts. New charts must be made for each specific flight condition, which depends on varying parameters like relative heading and relative velocity. But with the increasing storage capacity of today's computer hardware, more and more charts will be able to fit in the AAV's collision avoidance system.

Finally, collision avoidance controllers can command simpler, locally-optimal maneuvers. One example is the DSAAV collision avoidance controller, described in Section 4.3.

4.2.3 Multi-AAV Maneuvers

Multi-AAV environments can contain more than two AAVs. An autonomous CAS can handle this in one of two ways. The first way is for each AAVs CAS to only consider one of the other vehicles at a time. Some selection criterion should be used to determine which other AAV is most critical at the moment. This criterion will probably be a function of time of closest approach and closest approach distance. One such criterion is suggested and studied in [17].

The problem with only resolving conflicts involving pairs of vehicles is that the avoidance maneuver does not consider the other AAVs; the avoiding AAV may end up in a more severe conflict than the one it just resolved. Nonetheless, [17] shows that a large number of vehicles can interact safely by considering only pairwise conflicts.

The second and more global approach is to consider all of the vehicles when computing a collision avoidance maneuver. In [11], positive semidefinite programming is used to generate resolution trajectories for multiple vehicles. Examples with four to 6 vehicles are solved. The computation time for these solutions is on the order of minutes. Therefore, it probably could not be implemented in real-time with the current processing technology, but probably will someday soon.

There are other, perhaps simpler, avoidance maneuvers for more than two vehicles. [16] suggests distributing all of the vehicles at different altitudes. [13] develops a maneuver protocol similar to the rotaries found on many New England roads (Figure 4.7).

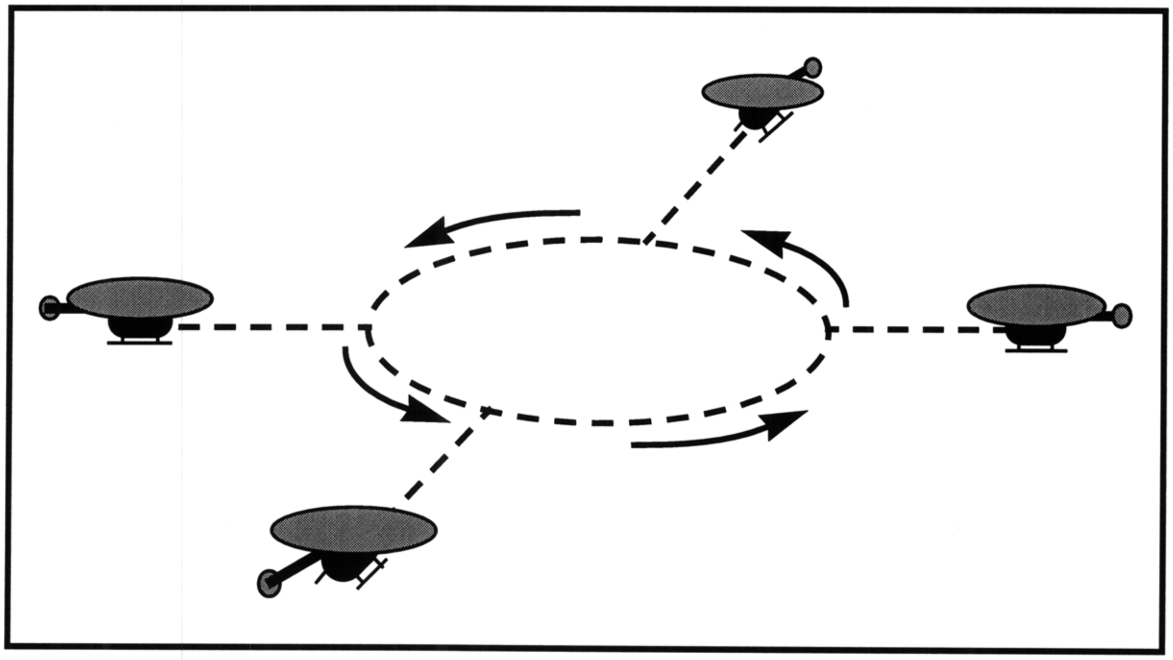


Figure 4.8: *"Rotary in the sky": multi-AAV collision avoidance*

4.3 DSAAV Algorithm Design

Two collision avoidance algorithms—one bang-bang and one continuous—have been designed for DSAAV. First, we will briefly describe the bang-bang controller. Then the continuous controller, which will probably be used in the flight tests, will be described.

4.3.1 DSAAV Bang-Bang Algorithm

The bang-bang controller was designed as a first cut at a simple collision avoidance system for pairs of autonomous helicopters. The algorithm is distributed, and would be run on each vehicle. It was designed and tested using the DSAAV simulation

Conflict Prediction

The CAS uses state data from both vehicles to prediction collision. In the simulation, this data, which includes the position, velocity, and attitude of the intruder vehicle, is easily passed into the collision avoidance algorithm. In actual flights, this data would be transmitted from vehicle to vehicle, either directly or through a communications hub.

The algorithm assumes a constant velocity trajectory extrapolation for predicting conflict. So for the relative position of the vehicles at time, t, is

$$\boldsymbol{x}_r(t) = \boldsymbol{x}_{r_0} + \boldsymbol{V}_r t \tag{4.8}$$

and the vehicle separation, $|\boldsymbol{x}_r|$, is given by

$$|\boldsymbol{x}_r|^2 = \boldsymbol{x}_r^T\boldsymbol{x}_r = \boldsymbol{x}_{r_0}^T\boldsymbol{x}_{r_0} + 2\boldsymbol{x}_{r_0}^T\boldsymbol{V}_r t + \boldsymbol{V}_r^T\boldsymbol{V}_r t^2 \tag{4.9}$$

Now, to find the time of closest approach, t^*, differentiate the vehicle separation with respect to time:

$$\frac{d}{dt}|\boldsymbol{x}_r^2| = 2\boldsymbol{x}_{r_0}^T\boldsymbol{V}_r + 2t\boldsymbol{V}_r^T\boldsymbol{V}_r = 0 \tag{4.10}$$

$$t^* = -\frac{\boldsymbol{x}_{r_0}^T\boldsymbol{V}_r}{\boldsymbol{V}_r^T\boldsymbol{V}_r} \tag{4.11}$$

Substituting this back into Equation (4.9), we find the separation distance at closest approach:

$$(d^*)^2 = x_{r_0}^T x_{r_0} - \frac{(x_{r_0}^T V_r)^2}{V_r^T V_r} \tag{4.12}$$

We do not consider future uncertainty in the vehicle trajectories. The condition for a future conflict is as follows:

$$\begin{aligned} d^* > d &\Rightarrow \text{No collision occurs} \\ d^* < d &\Rightarrow \text{Collision occurs} \end{aligned} \tag{4.13}$$

The value of d used in the DSAAV simulations is twelve feet. The actual value used in flight tests might be larger on smaller as deemed necessary.

Avoidance Maneuver

The DSAAV CAS commands vertical avoidance maneuver to the vehicle. Some of the benefits of vertical maneuvering are discussed in Section 4.2. We will add one other to that list: the DSAAV vehicles respond more quickly to commands in the vertical direction than in the horizontal direction.

In order to minimize the amount of time spent avoiding collision, the bang-bang algorithm will command the vehicles to maneuver at their maximum allowable vertical velocities. Currently, these maximums are two feet per second in the up direction and one foot per second in the down direction. Furthermore, it tries to do so at the last possible moment that conflict can still be avoided with the maneuver.

If a future collision is indeed imminent, the CAS investigates the effect of initiating the avoidance maneuver at that moment. It does not assume an instantaneous velocity change; instead it propagates its current trajectory for another $4\tau_v$ seconds, then inserts an instantaneous velocity change. τ_v is the time constant of a vertical velocity command. After four time constants, the vertical velocity should be at

$$1 - e^{-4} = 0.9817 = 98.17\% \tag{4.14}$$

of its commanded value. From the projected position after four time constants, the vertical velocity is changed to the avoidance velocity, and the time and distance of closest approach are calculated for that new trajectory.

If this distance at closest approach is greater than the minimum allowable separation, then no maneuver is initiated. When the projected distance of closest approach is finally approximately equally to the minimum allowed separation, *d*, the CAS takes over control of the vehicle and the avoidance maneuver is initiated (see Figure 4.9).

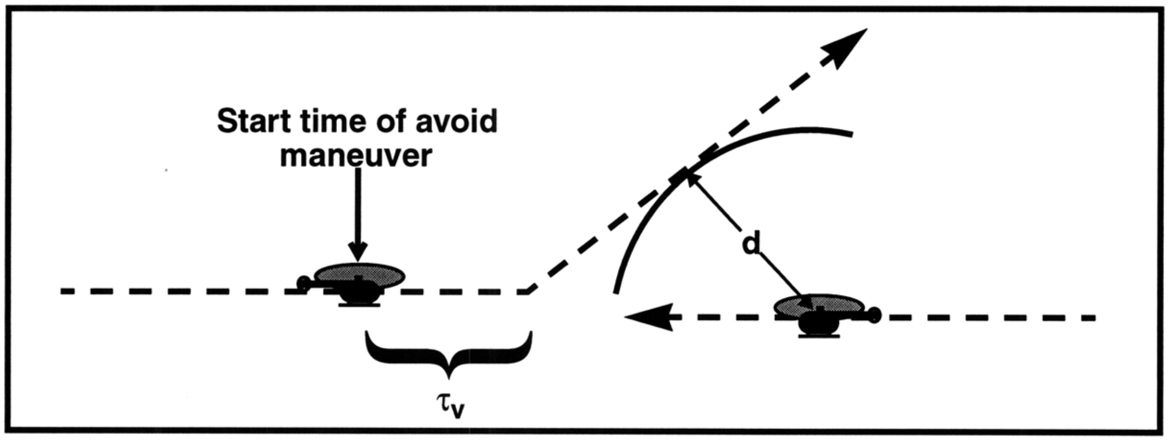


Figure 4.9: *Start of bang-bang avoidance maneuver*

One of the limitations of the algorithm is its need to know whether the other vehicle is running the same collision avoidance logic. This greatly affects the time of maneuver. If both vehicles are cooperating and maneuvering in opposite directions, they can wait longer to maneuver and still achieve the necessary separation. When only one is maneuvering, it must maneuver earlier to achieve the same minimum separation, *d*. This is shown in Figure 4.10.

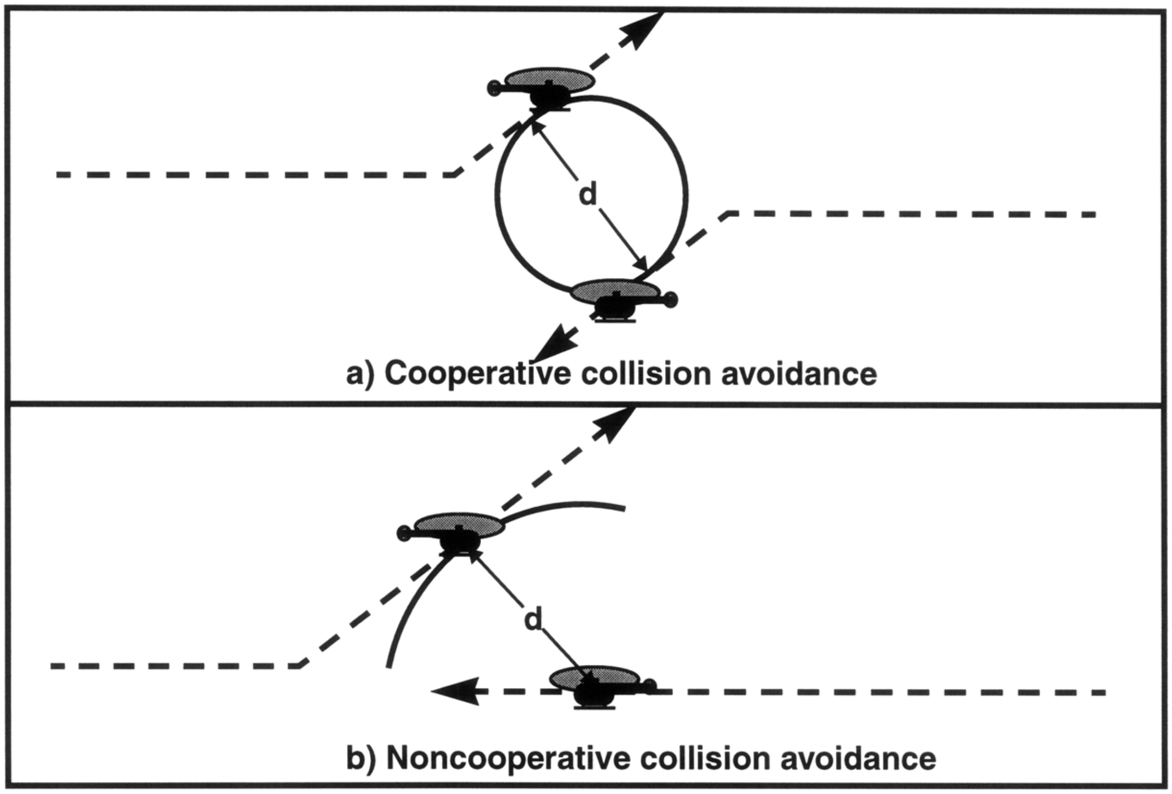

d
a) Cooperative collision avoidance
d
b) Noncooperative collision avoidance

Figure 4.10: *Cooperative and noncooperative avoidance maneuvers*

It is especially important for each vehicle's CAS to know whether the other vehicles are cooperating with them in collision avoidance. If a vehicle maneuvers while incorrectly assuming the other is doing the same, the avoidance will be initiated too late and the separation minimums will be violated. In the converse situation, a vehicle maneuvers to avoid a vehicle that is incorrectly assumed to *not* be maneuvering. In this case, the avoid maneuvers will start early, and will have a added margin of safety. The return maneuvers, as we will see later, can create some difficulty however.

Maneuver Directions

Since the AAVs always have two possible maneuver directions, up and down, the collision avoidance algorithm must set some criterion to choose between the two. So, each vehicle's CAS predicts (constant-velocity extrapolation, again) where its vehicle will be relative to the other at the time of closest approach. If its altitude is higher at that point, then it will choose to maneuver up. If it is lower, than it maneuvers down. This criterion is illustrated in Figure 4.11.

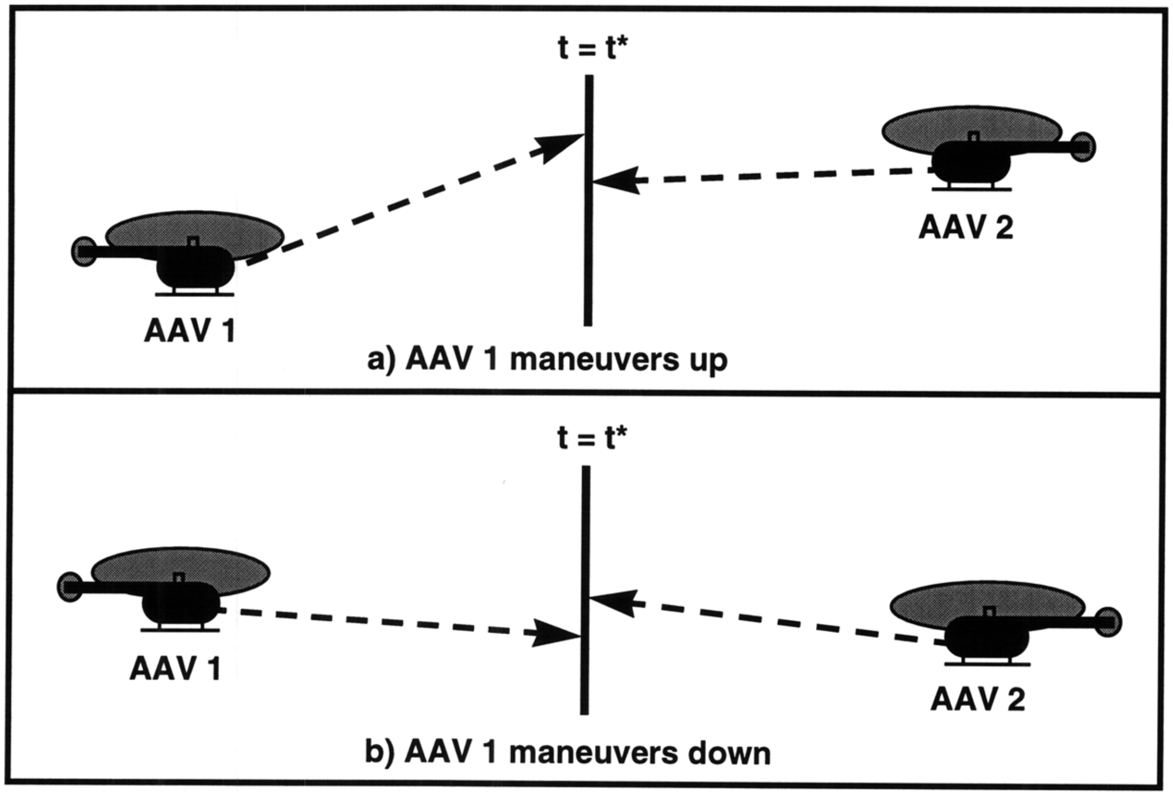


Figure 4.11: *Maneuver direction criterion*

Return Maneuver

The return trajectory of the bang-bang collision avoidance controller begins when the constant-velocity extrapolation of the AAVs' return velocities will not result in a minimum separation violation. Each vehicle finds its own return velocity by polling the guidance system to find the velocity it wants to command to return to the original path. The AAVs assume that the other vehicle's return velocity is its maximum velocity in the return direction (once again, two feet per second up, one foot per second down).

At the beginning of the avoid maneuver, the predicted return trajectories may take the vehicles right into one another. But at some point in time, they will be sufficiently past one another that the predicted trajectories will be safe, as shown in Figure 4.6a. The return maneuver begins at this point; and actually it consists of shutting the CAS off and letting the guidance algorithms bring the AAV back to its original trajectory.

Earlier we alluded to a problem in the return maneuver when the AAVs mistakenly believe that they each are the only one executing an avoidance maneuver. As the vehicles are avoiding one another, they wait until the time that they can safely return to their original trajectory. When this happens, they begin the return maneuver, only to unexpectedly find that the other AAV has also begun a return maneuver. Since they had not planned on this, the vehicles are now lined up for a conflict again, and they avoid once again. The result is that they follow an unnecessary "M"-shaped trajectory. So again, we emphasize the importance of knowing whether the other AAV is cooperating in collision avoidance.

The bang-bang collision avoidance algorithm showed that a simple collision avoidance algorithm could be developed and implemented relatively quickly. It did have its limitations, however. In addition to the ones listed above, it had one other problem which hindered its performance. When deciding to maneuver up or down, each AAV's CAS based the decision on the state of the other vehicle. If this state data was inaccurate, both vehicles could decide to maneuver in the same direction. Sometimes the collision avoidance systems switch on and off and eventually get the vehicles moving in separate, directions. But at other times, the result was a direct collision. If the state data for the other vehicle was transmitted by the other vehicle, then communication time delays could make it inaccurate.

One remedy to this problem would be to implement some means for maneuver direction acknowledgment between the vehicles. Unfortunately, the time delays associated with that might also create new problems or just slow the system down too much. It's these limitations which sparked the design for the continuous DSAAV collision avoidance controller, described next.

4.3.2 DSAAV Continuous Algorithm

The continuous controller was designed to be an improvement over the bang-bang collision avoidance system. Specifically, we wanted to eliminate the shortcomings of the open-loop controller, especially the all-or-nothing behavior which gets it into trouble sometimes. Like the bang-bang controller, the algorithm is distributed and would be run on each vehicle.

Conflict Prediction

Conflict prediction for the continuous collision avoidance algorithm is the same as it is for the bang-bang system. The distance at closest approach, d^*, is calculated and the collision criterion is

$$\begin{aligned} d^* > d &\;\Rightarrow\; \text{No collision occurs} \\ d^* < d &\;\Rightarrow\; \text{Collision occurs} \end{aligned} \quad . \tag{4.15}$$

Avoidance Maneuver

The collision avoidance maneuver of the continuous controller differs quite a bit from the bang-bang version. In the bang-bang controller, the vehicles played a game of AAV chicken by not maneuvering until the last instant, then maneuvering at full velocity. If an error occurred (e.g. the AAVs accidentally maneuver in the wrong direction), the nature of the nature of the algorithm almost ensures that it will not be corrected in time.

The continuous controller, on the other hand, strives to send the vehicles on smooth, gradual trajectories. Any disturbances or errors should generally have time to correct themselves before a conflict occurs. Like the bang-bang algorithm, this one only commands vertical velocities for collision avoidance.

The closed-loop controller on each vehicle aims to have the AAVs right at the allowable miss distance at their point of closest approach. This is equivalent to each AAV aiming its velocity vector tangent to a sphere of radius d centered on the other vehicle. This is visualized in Figure 4.12. Since the algorithm runs continuously in real-time, it is constantly reevaluating and adjusting the commanded velocity to be tangent to this sphere.

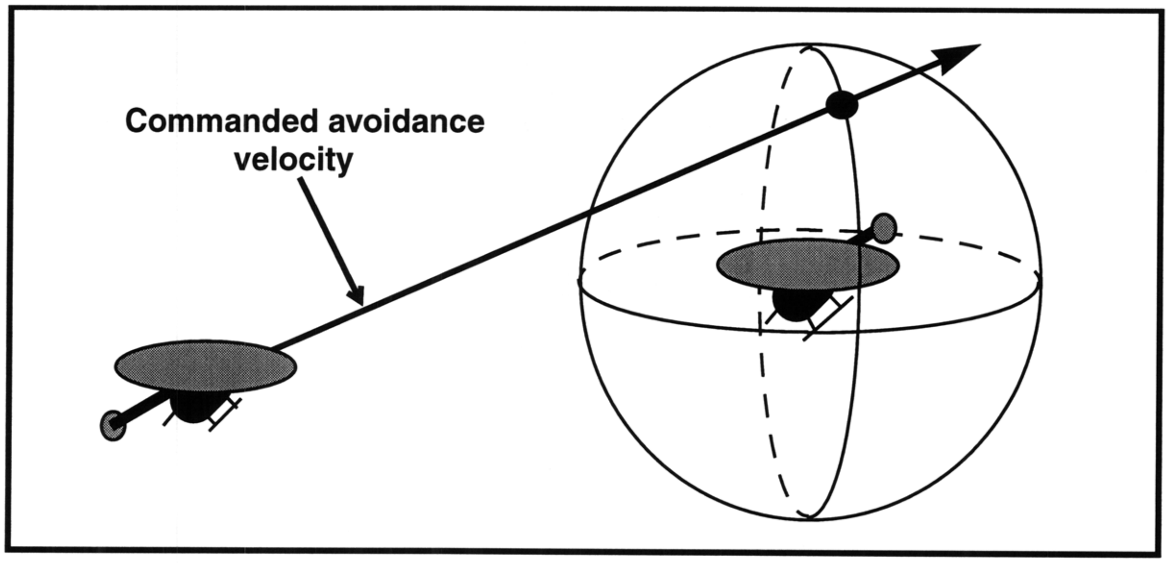


Figure 4.12: *Continuous controller avoidance command*

Equation (4.12) gives us an expression for the vehicle separation at the point of closest approach. Setting the value of separation to the minimum allowable separation, *d*, and then solving for V_{z_r} gives the needed relative vertical velocity to achieve this minimum allowed separation:

$$V_{z_{r_{comm}}} = \frac{-z_r(V_{x_r}x_r + V_{y_r}y_r) \pm \sqrt{\Delta^2[(z_r^2 - \Delta^2)(V_{x_r}^2 + V_y^2) + (V_{x_r}x_r + V_{y_r}y_r)^2]}}{(z_r^2 - \Delta^2)} \tag{4.16}$$

where

$$\Delta^2 = |\boldsymbol{x_r}|^2 - d^2 = x_r^2 + y_r^2 + z_r^2 - d^2 \tag{4.17}$$

$V_{z_{r_{comm}}}$ is the relative velocity needed for the AAVs to pass within *d* of one another at their point of closest approach. The sign of the ± reflects the choice of which vehicle is to pass over top of the other one. This will be discussed in more detail later.

Note that $V_{z_{r_{comm}}}$, is a relative velocity between the vehicles. However, the DSAAV control system requires absolute velocity commands. So, when the collision avoidance algorithms solve for $V_{z_{r_{comm}}}$, that does not necessarily tell them what maneuver they need to execute. $V_{z_{r_{comm}}}$ must be distributed between the two vehicles somehow. We call this *maneuver allocation*.

Suppose one of the AAVs is following its original constant-velocity trajectory, and therefore is not attempting any conflict resolution. In this scenario, it is up to the vehicle that is running collision avoidance (let's call it AAV 1) to make sure the commanded relative velocity, $V_{z_{r_{comm}}}$, is achieved. Remember that

$$V_{z_r} = V_{z_1} - V_{z_2} \tag{4.18}$$

So the collision avoidance system on AAV 1 should be commanding the following vertical velocity:

$$V_{z_{1_{comm}}} = V_{z_{r_{comm}}} + V_{z_2} \tag{4.19}$$

This will result in a velocity vector aimed to be tangent to the sphere of radius *d* centered on AAV 2.

If both vehicles are making evasive avoidance maneuvers, one of two things may occur. One is that both vehicles command a vertical velocity for themselves as if the other AAV is not avoiding conflict. From Equation (4.18), we find that the vertical velocity commands are

$$V_{z_{1_{comm}}} = V_{z_{r_{comm}}} + V_{z_2} \qquad V_{z_{2_{comm}}} = -V_{z_{r_{comm}}} + V_{z_1} \tag{4.20}$$

At first glance, this appears to be collision avoidance overkill. If conflict can be avoided by only one AAV commanding $V_{z_{r_{comm}}}$, surely having both vehicles command it is unnecessary. In fact, it might seem that this could even result in a minimum miss distance that is twice as large as the minimum allowable separation. This does not occur; and having both vehicles try to maintain the full separation has an important advantage.

The collision avoidance controllers are closed-loop, and they will always try to command a relative velocity of $V_{z_{r_{comm}}}$. As long as the controllers are stable and the relative velocities get closer and closer to $V_{z_{r_{comm}}}$, the collision avoidance controllers will have to command less and less of a velocity change (error signal is going to zero). So, theoretically, the vehicles will eventually achieve $V_{z_{r_{comm}}}$ and the collision avoidance systems will no longer need to issue commands. In this situation, it no longer matters that we are commanding twice the relative velocity that we need.

In reality, however, the AAVs will be perturbed from these perfect $V_{z_{r_{comm}}}$ trajectories, and the controllers will try to bring them back. As we will see in Chapter 5, it is in these situations when the extra velocity commanding comes in and affects the stability and response of the system.

The benefit of each AAV commanding the full $V_{z_{r_{comm}}}$ comes when one of the vehicle's CAS is shut off but the other is not aware of it. In this scenario, if the avoiding vehicle commanded anything less than the full $V_{z_{r_{comm}}}$, a collision would occur. We saw a similar result with the bang-bang controller, when an AAV mistakenly believes the other is maneuvering.

In an alternative configuration, the two vehicles could divide $V_{z_{r_{comm}}}$ between themselves. The most logical choice would be for each vehicle to command one-half of $V_{z_{r_{comm}}}$. We will investigate the stability of this approach in Chapter 5. Of course, as we mentioned before this approach relies on the assumption that the other vehicle is cooperating in collision avoidance. If this assumption proves to be wrong, a conflict will undoubtedly occur.

Maneuver Direction

Next we should choose which sign the $\pm$ should take. Because the AAV's CAS is aiming to be tangent to a "sphere of separation" which is centered on the other vehicle, it has two options: maneuver to the top of the sphere or maneuver to the bottom. This is illustrated in Figure 4.13.

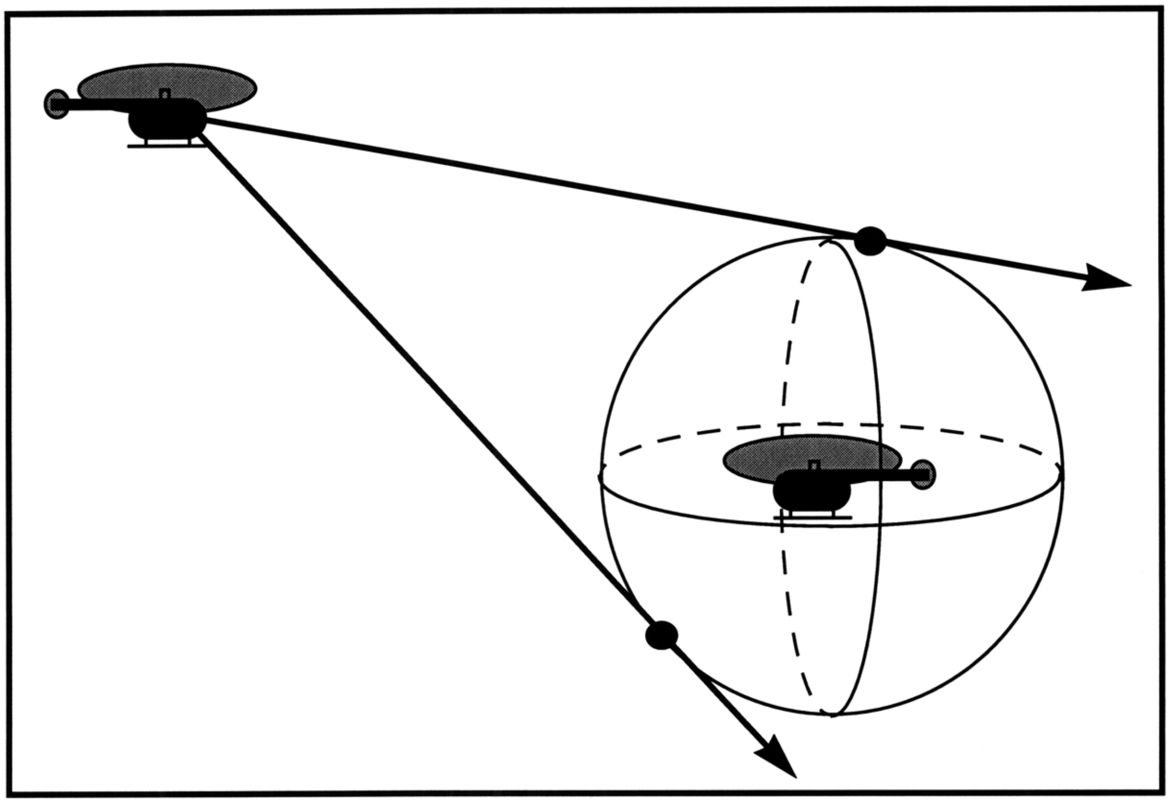


Figure 4.13: *Maneuver options*

We would like to choose the maneuver which minimizes the AAVs' deviations from their current trajectories. This amounts to the same criterion we used in the bang-bang maneuver. So if an AAV predicts that its altitude will be greater than the other AAV's at the point of closest approach, it should maneuver to the top of the "sphere of separation" which surrounds the other vehicle. Conversely, if its altitude is less, it should aim toward the bottom of the sphere.

In other words, if AAV 1 is higher than AAV 2 at the point of closest approach, then $z_r(t^*) < 0$ (remember: the positive z direction is down in the NED frame). In that case, AAV 1 would want to pass over AAV 2, which means that $V_{z_{1_{comm}}}$ should be the lowest it can be. Since V_{z_2} is considered constant, then $V_{z_{r_{comm}}}$ should be the lowest it can. Let's look at $V_{z_{r_{comm}}}$ to see which sign of $\pm$ the that equates to.

First, look at the denominator of the expression for $V_{z_{r_{comm}}}$:

$$z_r^2 - \Delta^2 = d^2 - {x_r}^2 - {y_r}^2 \tag{4.21}$$

The value of this expression is negative when ${x_r}^2 + {y_r}^2 > d^2$. This corresponds to times during which AAV 1 is outside a cylinder that has radius d and is centered on AAV 2. Since the denominator is negative, choosing the $\pm$ to be positive will yield the lowest $V_{z_{r_{comm}}}$.

Something strange, however, happens when AAV 1 is inside the cylinder described by ${x_r}^2 + {y_r}^2 > d^2$. In this situation, the denominator is positive, and choosing the minus sign will produce the lowest $V_{z_{r_{comm}}}$. But, if we look closely, it turns out *we do not* want the lowest $V_{z_{r_{comm}}}$ anymore.

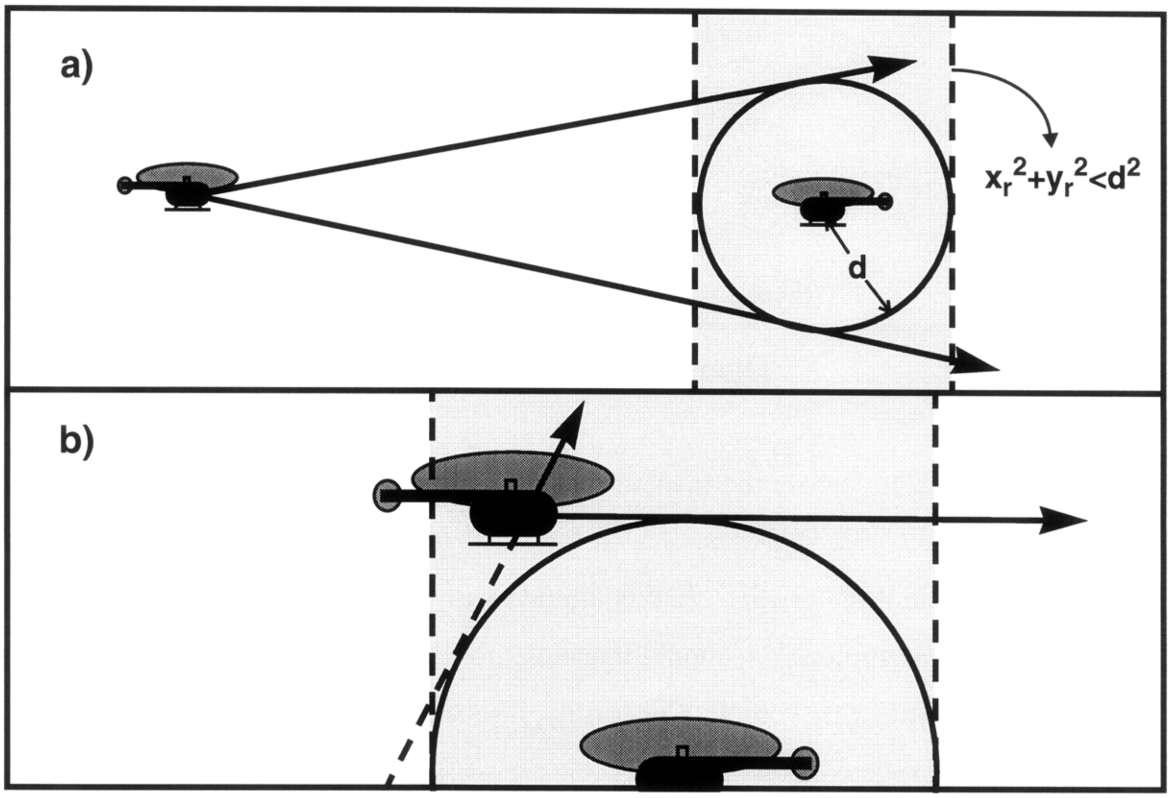


Figure 4.14: *Choosing the maneuver direction*

Looking at Figure 4.14a, it is plain to see the minimum (i.e. most "upward") $V_{z_{r_{comm}}}$ is best for AAV 1 when it is outside of the cylinder. However, as AAV 1 moves into the cylinder, the downward tangent trajectories passes through vertical and starts to point upward (Figure 4.14b). This is a tangent trajectory to the sphere, but it had its point of closest approach in the past ($t < 0$). Furthermore, it actually is a more upward velocity than the proper one. So inside the cylinder, we now want the maximum (most "downward") velocity. Since the denominator is now positive, make the $\pm$ to be positive will once again result in the correct maneuver.

To summarize, the criterion for choosing the sign of $\pm$ is as follows:

$$\begin{aligned} z_r(t^*) > 0 &\Rightarrow + \\ z_r(t^*) < 0 &\Rightarrow - \end{aligned} \tag{4.22}$$

Unlike with the bang-bang system, the two avoidance maneuvers may not necessarily be in opposite directions. This depends on the initial relative positions and velocities of the vehicles. In Figure 4.13, both possible avoidance maneuvers are in the down direction.

It is also possible for $V_{z_{r_{comm}}}$ to have no solution. This happens when the expression under the radical in (4.16) is negative. This can happen in two instances. In the first, the Δ^2 term is negative:

$$\Delta^2 = x^2{}_r + y^2{}_r + z^2{}_r - d^2 < 0 \tag{4.23}$$

When this occurs, the separation of the AAVs is already less the minimum allowable separation. In that case, there is no velocity that would result in a separation of d at closest approach. It is too late for that. The vehicles are already inside the sphere of separation, and there is no constant-velocity trajectory from that initial point that is tangent to the sphere.

The second scenario in which there is no solution for $V_{z_{r_{comm}}}$ is when the rest of the expression under the radical (i.e. not Δ^2) is negative:

$$(z_r^2 - \Delta^2)(V_{x_r}^2 + V_y^2) + (V_{x_r}x_r + V_{y_r}y_r)^2 < 0 \tag{4.24}$$

which can be shown to be equivalent to

$$x^2{}_r + y^2{}_r\Big|_{t = t^*} > d^2 \tag{4.25}$$

In other words, if the *horizontal* separation at the time of closest approach is projected to be greater than d, then there is no $V_{z_{r_{comm}}}$ that will result in a separation of d at t^*. Horizontal velocity commands would be necessary to alter the horizontal trajectories of the

AAVs. The diagram below depicts the AAVs predicted relative positions at t^*.

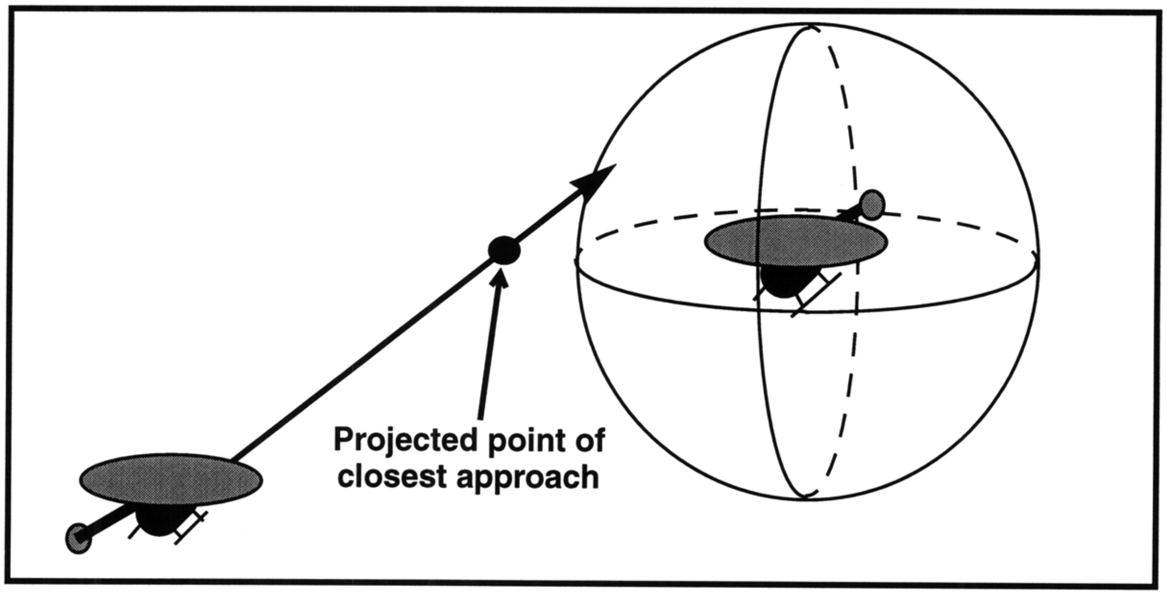


Figure 4.15: *No solution for* $V_{z_{r_{comm}}}$

Return Maneuver

In one regard, the return maneuver is the same as with the bang-bang controller. That is, the CAS shuts off, and the guidance algorithm returns the AAV to its original path. The criteria for switching the CAS off, however, is different.

As soon as a conflict is predicted in the future, the continuous collision avoidance algorithm begins the avoidance maneuver. Now in the bang-bang controller, the CAS shut off when it was predicted that the return maneuver would not cause a conflict. As we discussed, however, that can cause a lot of rapid switching early on in the maneuver. So to prevent that, only let the system switch off after the conflict has passed, we also require there to be no conflict along the avoidance maneuver trajectory either.

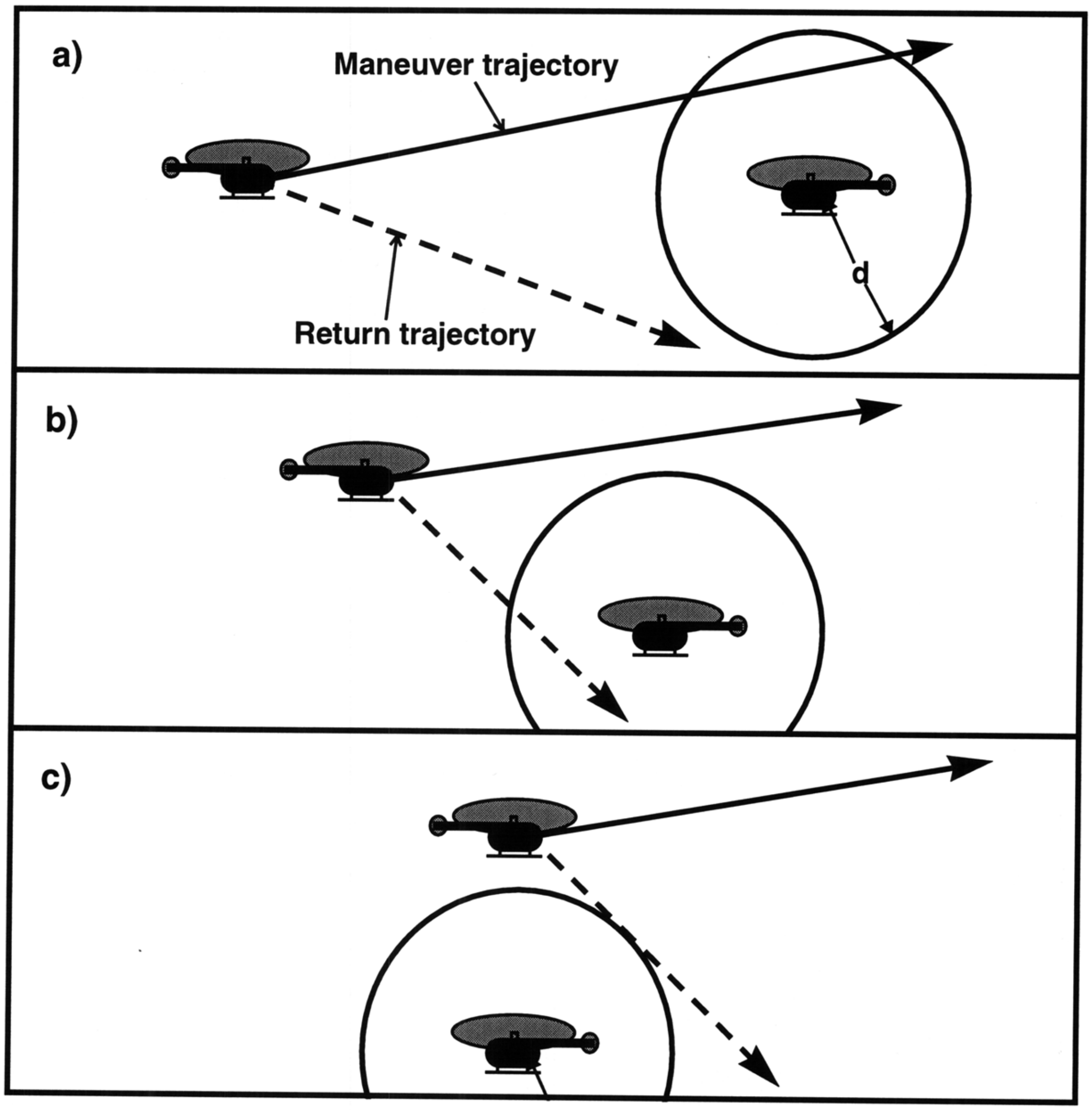


Figure 4.16: *Return maneuver criteria*

This criterion would have been dangerous in the bang-bang controller. Had each vehicle started maneuvering in the same direction by accident, it is likely that the above criterion would have kept them maneuvering in the same direction, eventually causing a conflict.

The continuous controller, though, starts earlier and allows different velocities to be commanded. That helps it generate a smooth, continuous avoidance trajectory. Once the vehicles pass each other, their collision avoidance systems will see that there lies no conflicts along neither the maneuver trajectory nor the return trajectory (Figure 4.16c).

Sometimes the continuous collision avoidance algorithm does create some rapid switching during the maneuver. This occurs early in the maneuver, when the return trajectory does not yet intersect the protected zone. If the initial trajectory violates the protected zone, the controller begins a maneuver. The maneuver may overshoot or get disturbed in such a way that the maneuver trajectory ends up outside of the protected zone. Since the return trajectory does not intersect it either, the controller shuts off. As the AAV tries to return to the original path, its trajectory passes through the sphere of separation, and the system turns on again. This process may be repeated a number of times, as shown in the simulation results below (Figure 4.17).

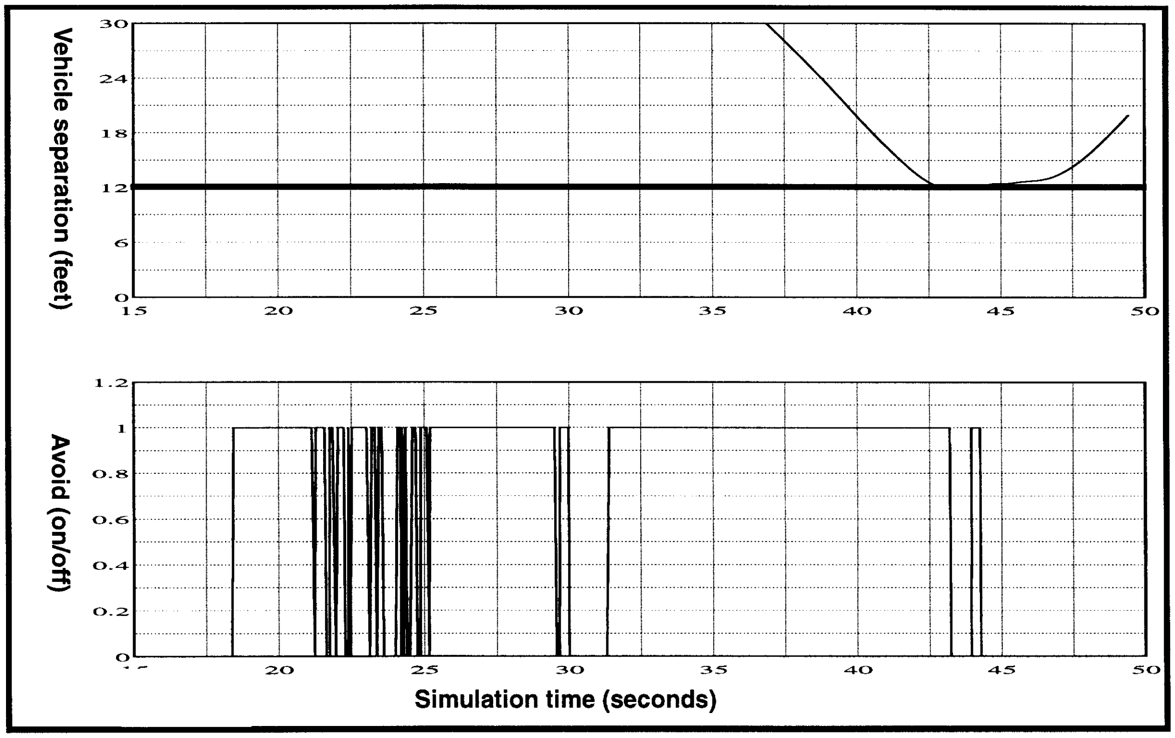


Figure 4.17: *CAS on/off switching*

One common way to remedy such rapid switching like we are seeing is to replace the switch (or *relay*) that is causing the cycling with a *relay with hysteresis*. Whereas a simple relay has one threshold the determines the value of the output, hysteresis considers also the history of the input signal (i.e. what have the values of the input been in the recent past?). Looking at the relay and hysteresis curves in Figure 4.18, we see that the output of the hysteresis curve depends on the direction in which the curve is traversed, as depicted with the arrows. Hysteresis is used in many control applications, including thermostats and spacecraft reaction control systems [18,19].

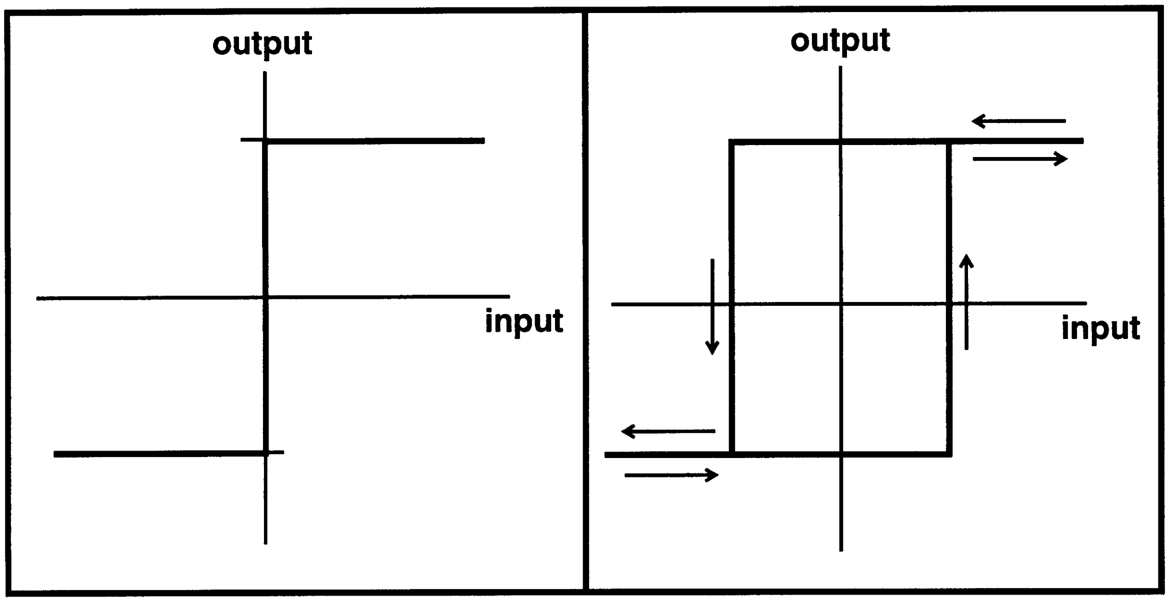


Figure 4.18: *a) Relay and b) hysteresis*

To reduce the on/off switching of the CAS algorithm, we apply a hysteresis loop to our on/off switching criterion. Before, the criterion was to switch on if the projected distance at the point of closest approach, d^*, is lower than the minimum allowed separation, d. If it was not lower, then the collision avoidance would be switched off. This is illustrated in Figure 4.19a.

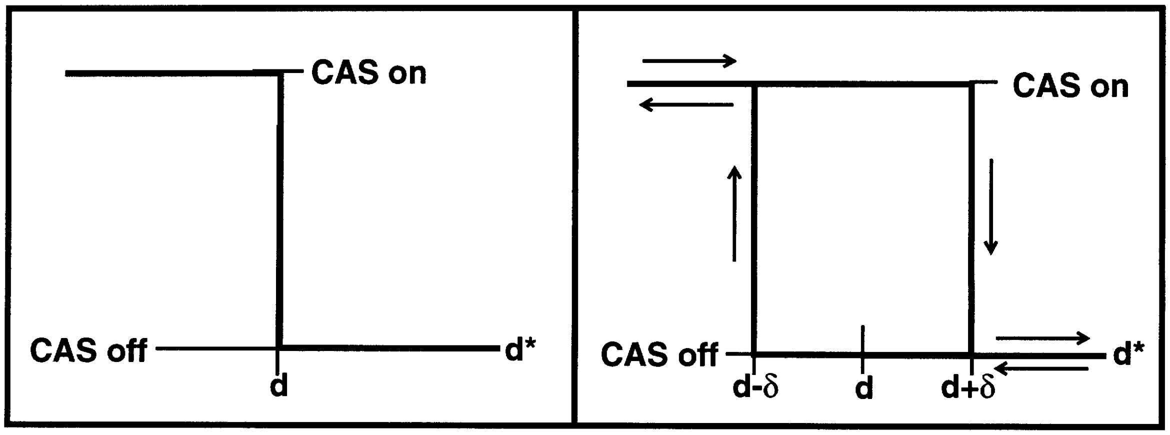


Figure 4.19: *CAS on/off criterion with a) relay and b)hysteresis*

To add hysteresis to the relay, we make whether the system is on or off at $d^* = d$ dependent on the direction in which d^* is moving. Said another way, if the CAS is already off, the value of d^* must drop δ below d before the system switches on. Likewise, if the system is already on, d^* must surpass d by the amount δ before it switches back off. This is shown in Figure 4.19b. The result is less switching at the beginning of the maneuver.

Notice that not all of this switching is eliminated, however. We can try increasing the value of δ, but we may run into the problem that the switching-on criterion requires d^* to be too low, and the system does not switch on in time.

Figure 4.20 shows the system in the same maneuver (DSAAVs moving head-on at one another) as Figure 4.18, with δ = 4 feet. We can see from the simulation plot that adding the hysteresis has reduced the early switching, while not compromising vehicle separation.

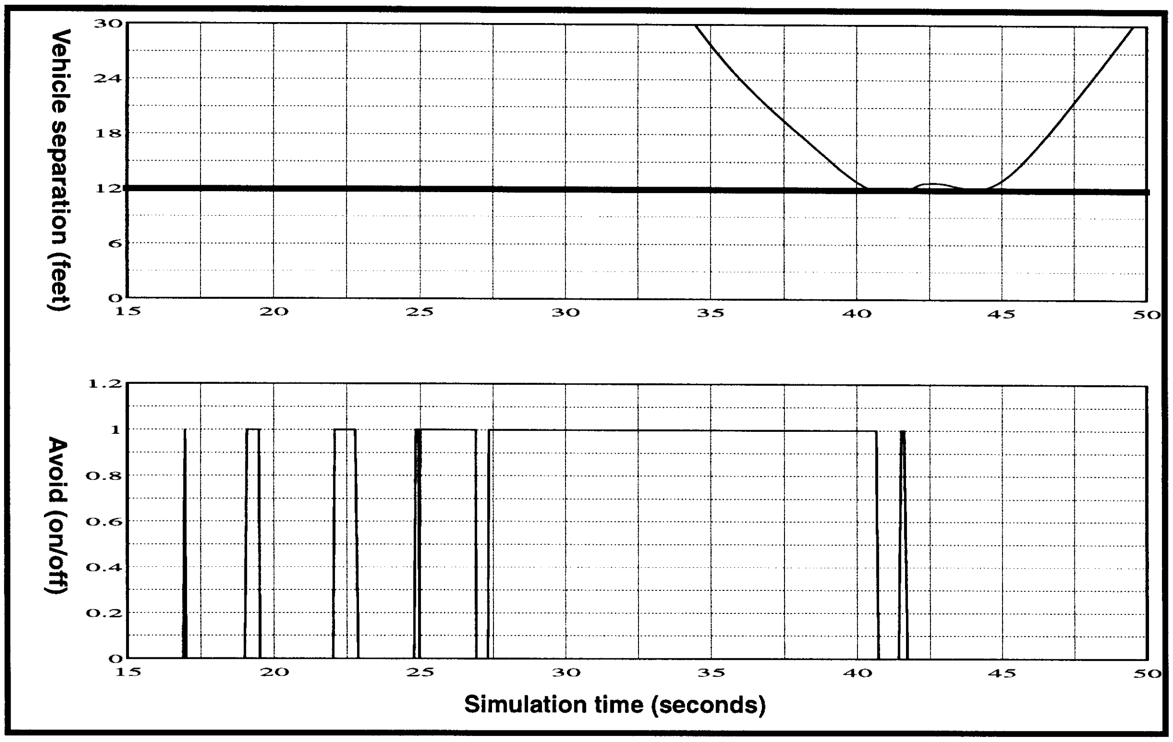


Figure 4.20: *CAS on/off switching with hysteresis; δ = 4 feet*

Time Delay

As we have mentioned before, communication time delay may hinder the performance of the collision avoidance system. This is equivalent to the destabilizing affect of adding time delay to a closed-loop control system, which is what our CAS is. The affect of time delay on the stability of the DSAAV collision avoidance system is discussed in Chapter 5.

In the DSAAV simulation, a communications buffer is established which delays the information being transmitted between DSAAV 1 and 2. The collision avoidance algorithm can, given an estimate of the time delay, attempt to compensate for this time delay. It estimates the current position of the other vehicle by propagating its constant-velocity trajectory for an additional amount of time equal to the estimated time delay. Figures 4.21 through 4.24 show the performance of the algorithm with and without this time delay compensation.

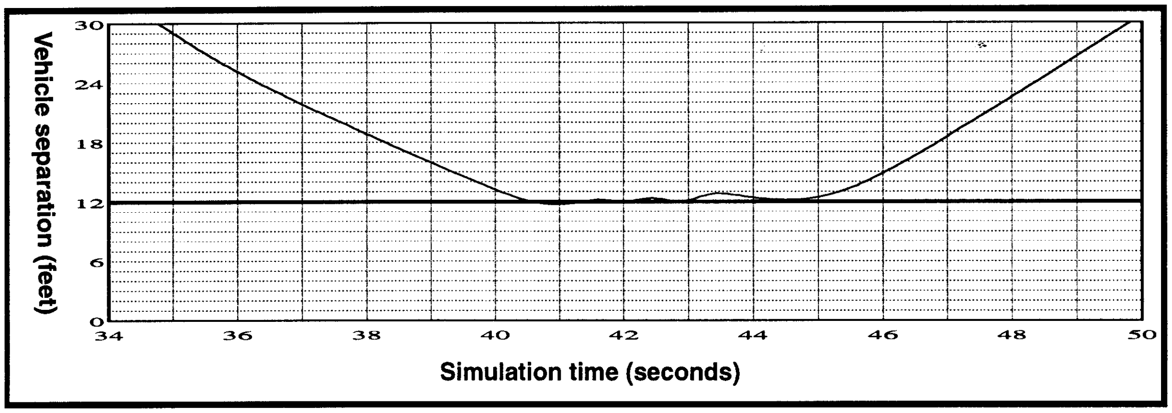


Figure 4.21: *Time delay = 0.0 seconds*

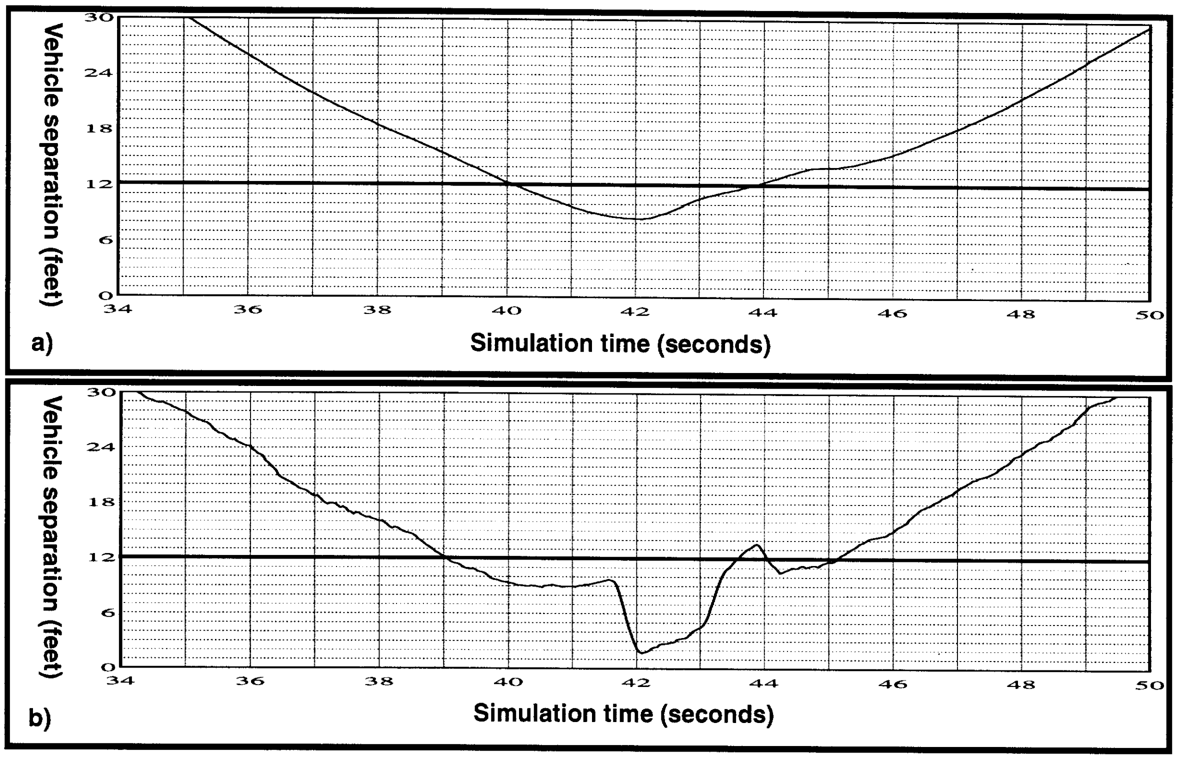


Figure 4.22: *Time delay = 2.0 seconds; a)uncompensated, b)compensated*

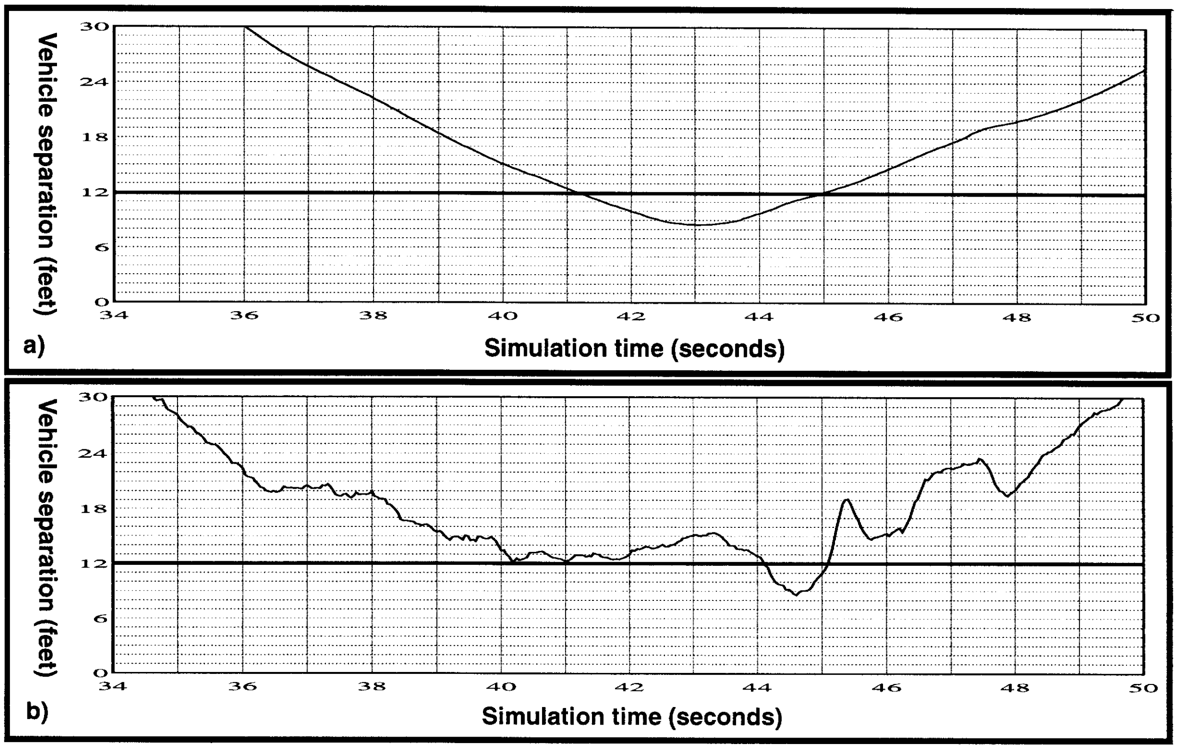


Figure 4.23: *Time delay = 4.0 seconds; a)uncompensated, b)compensated*

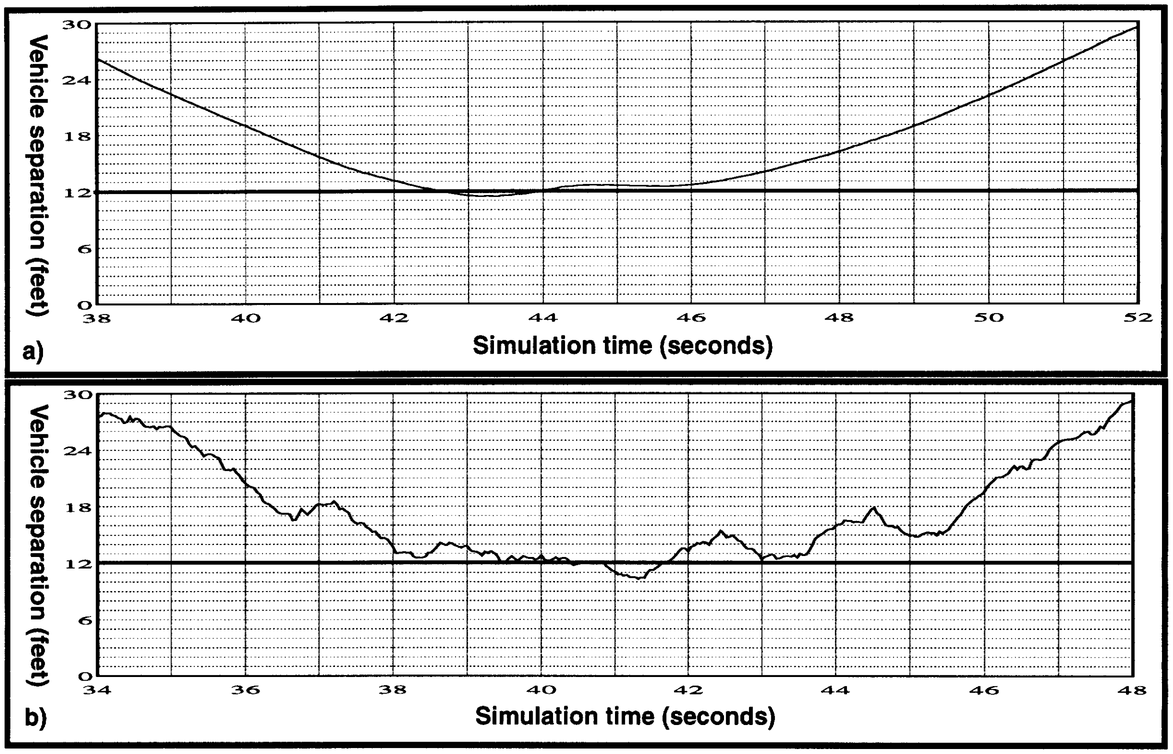


Figure 4.24: *Time delay = 6.0 seconds; a)uncompensated, b)compensated*

As can be seen from the above simulation results, the time delay does worsen the system's performance, but not as much as might be expected. Oddly enough, in this scenario, the CAS performed better with the six second time delay than with the two second delay. This probably will not be the case in general however.

Also notice that the system performance is as bad or worse when time delay compensation is used. As the time delay increases, the intruder velocities received by the DSAAVs statistically will be farther and farther from the actual intruder velocity and thus less accurate for extrapolating the intruder position. Filtering these measurements might improve this problem.

Variable Conflict Threshold

Notice that our collision avoidance system does not account for any uncertainty in the DSAAVs' positions and future trajectories. In the early development of the system, this may not be critical. But before the CAS algorithm is to be test flown on board the vehicles, some consideration would have to be made for the potential inaccuracies and uncertainties in the system. For example, the methods introduced in Section 4.1.3 might be used to set the maneuver thresholds.

Other Limitations

There is one other limitation of the DSAAV continuous collision avoidance system. Since the algorithm is only concerned about *relative* vertical velocities, the absolute vertical velocities of the vehicles do not always keep them near their original altitudes. For example, if both DSAAVs have no vertical velocities, then their avoidance maneuvers should take them in opposite vertical directions. However, if one of their avoidance velocities is perturbed, the other's collision avoidance algorithm will perturb its velocity as well. This can continue until both vehicles are moving in the same direction vertically, moving further away from their original altitudes. This phenomenon can be seen in the simulation plot in Figure 4.25.

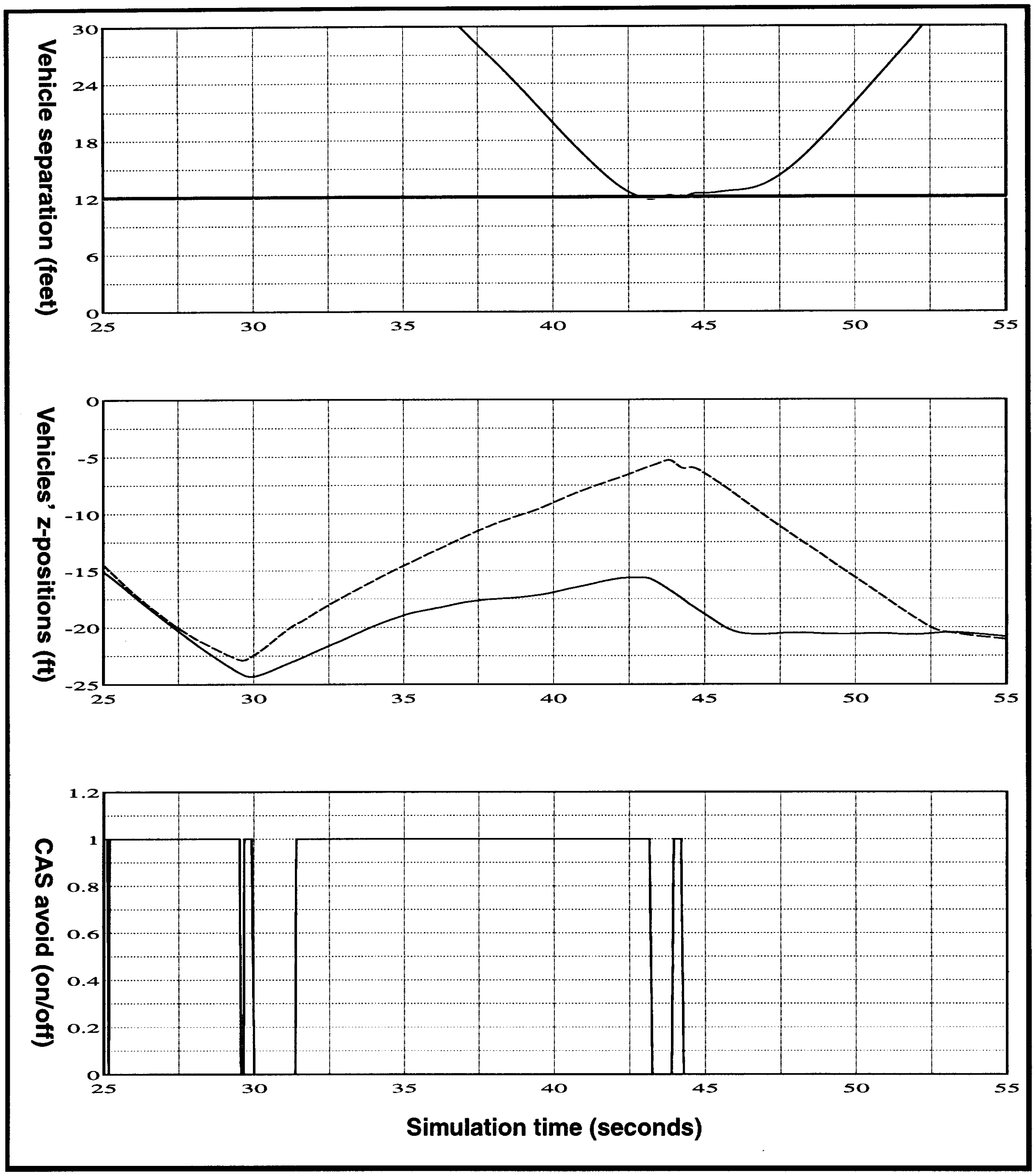


Figure 4.25: *Drifting vehicle altitudes*

The detriment of this behavior depends on how long the vehicles are avoiding one another and the importance of the vehicles' absolute altitude. If the vehicles' collision avoidance systems are only active for a short time, they may not have time to drift too far from their original altitudes. And for low-altitude operations, this drifting altitude phenomenon could drive the DSAAVs into the ground.

One approach to limiting this behavior is to try to reduce the duration of the avoidance maneuver. The best way to do this is by starting the maneuver closer to the time of closest approach, t^*. In Figures 4.26 and 4.27, we see the result of imposing a minimum time to closest approach, t_{min}^*, before the vehicles may avoid one another. The results show that imposing $t_{min}^* = 6$ seconds greatly improves the drifting altitudes while not compromising vehicle separation too much. However, lowering t_{min}^* to 2 seconds creates a violation of the protected zone. Nevertheless, setting a t_{min}^* appears to be an effective method for restricting this unwanted occurrence.

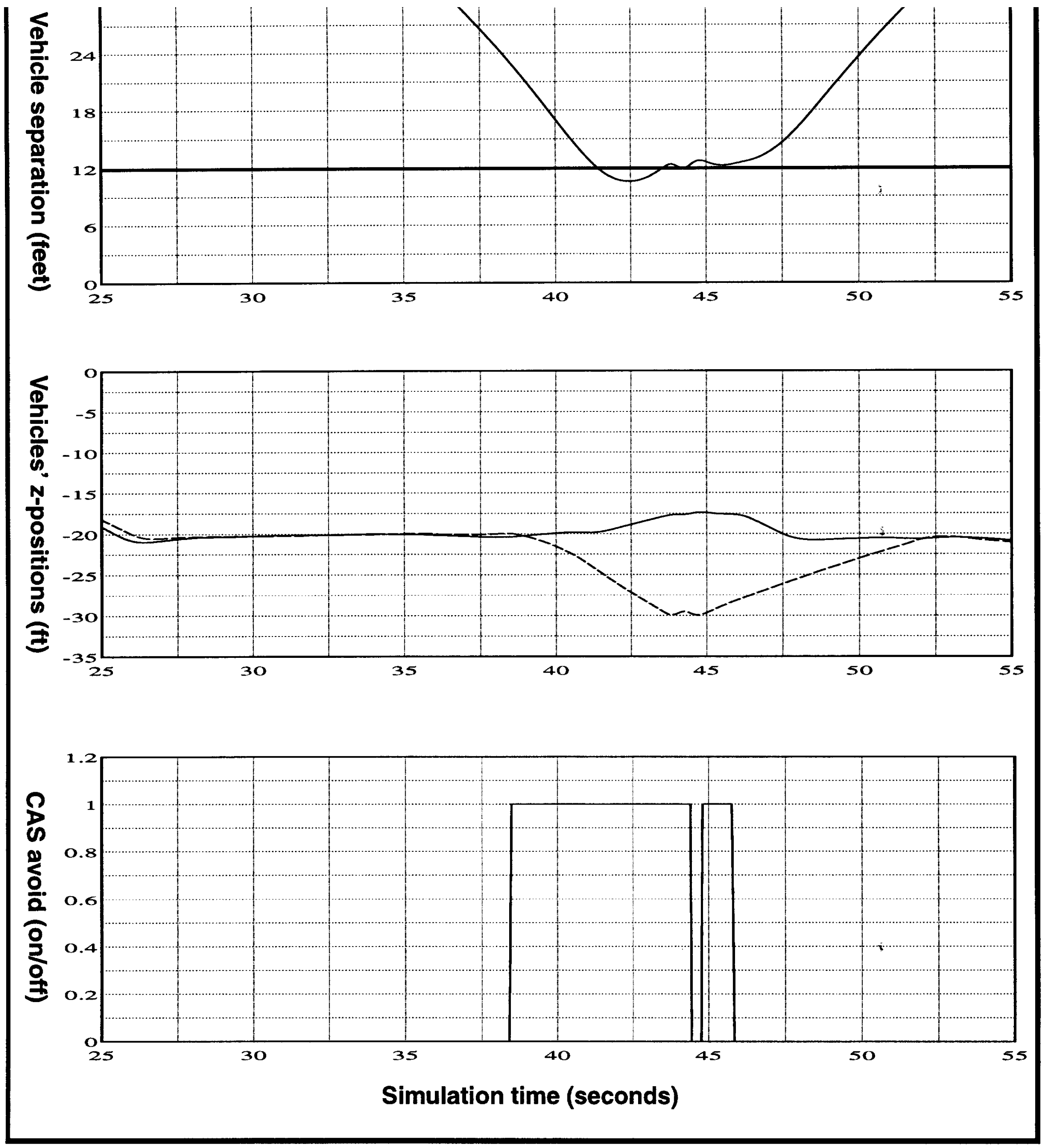


Figure 4.26: *Drifting vehicle altitudes:* $t_{min}^{*} = 6.0$

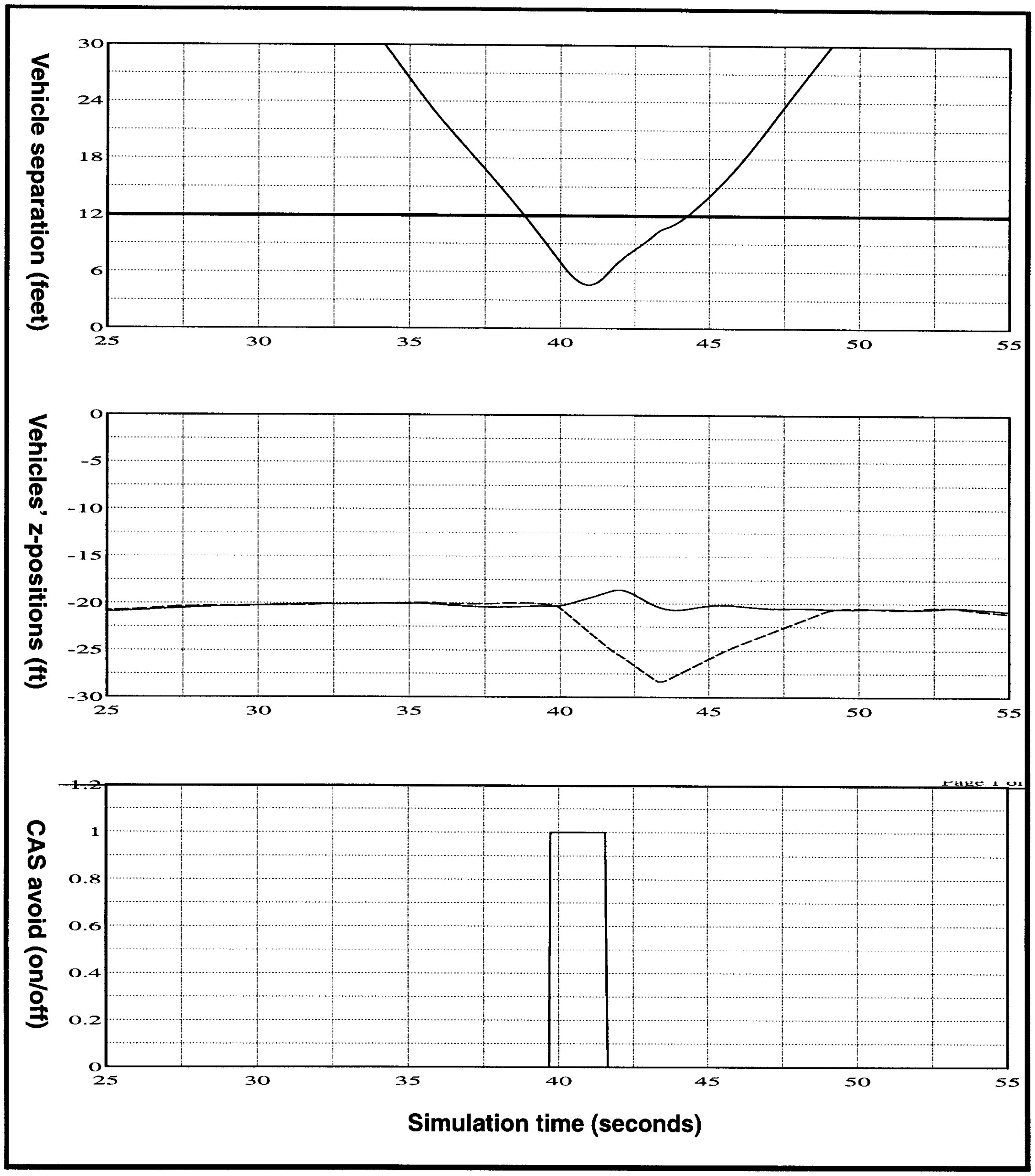


Figure 4.27: *Drifting vehicle altitudes:* $t_{min}^* = 2.0$

4.4 Conclusions

This chapter began with a general outline for designing an autonomous collision avoidance system. Some common approaches to conflict prediction and resolution were introduced, and we discussed some of the issues involved in designing such a system.

The latter half of the chapter described our collision avoidance algorithms and the design processes that led to them. Our algorithm represents a simple but functional CAS, which can run in real-time aboard the DSAAV. The methods used to design the DSAAV CAS can be extended to other collision avoidance systems, and the pitfalls we encountered can serve as lessons to future CAS designers. In Chapter 5, we provide a framework for analyzing a collision avoidance system like the one we have just designed.

Chapter 5

Closed-loop Algorithm Analysis

Now that we have decided on a collision avoidance algorithm that appears to satisfy the system requirements, we should analytically characterize its closed-loop behavior—especially its stability. The dynamics of the system, as we have shown, include the actual vehicle dynamic models and the collision-avoidance feedback controller. The dynamic model is linear; the collision-avoidance controller is nonlinear, and will be analyzed with linearized perturbation analysis. These will be looked at in Section 5.1.

In Section 5.2, we will also analyze the affect of communication time delay in the system. Finally, Section 5.3 briefly discusses some of the unmodeled dynamics—including nonlinearities such as relays and saturation—in the collision avoidance system.

5.1 System Stability and Performance

Once again, the model of the two-vehicle system (from section 3.7) is the following:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ V_{x_1} \\ y_1 \\ V_{y_1} \\ z_1 \\ V_{z_1} \\ x_2 \\ V_{x_2} \\ y_2 \\ V_{y_2} \\ z_2 \\ V_{z_2} \end{bmatrix} = \begin{bmatrix} 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&-\frac{1}{\tau_v}&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&-\frac{1}{\tau_v} \end{bmatrix} \begin{bmatrix} x_1 \\ V_{x_1} \\ y_1 \\ V_{y_1} \\ z_1 \\ V_{z_1} \\ x_2 \\ V_{x_2} \\ y_2 \\ V_{y_2} \\ z_2 \\ V_{z_2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{V_{z_{1_{comm}}}}{\tau_v} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{V_{z_{2_{comm}}}}{\tau_v} \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x}) \tag{5.1}$$

As developed in Section 4.3, the CAS-generated velocity commands are

$$V_{z_{1_{comm}}} = V_{z_{r_{comm}}} + V_{z_2} \qquad V_{z_{2_{comm}}} = -V_{z_{r_{comm}}} + V_{z_1} \tag{5.2}$$

where

$$V_{z_{r_{comm}}} = \frac{-z_r(V_{x_r}x_r + V_{y_r}y_r) \pm \sqrt{\Delta^2[(z_r^2 - \Delta^2)(V_{x_r}^2 + V_y^2) + (V_{x_r}x_r + V_{y_r}y_r)^2]}}{(z_r^2 - \Delta^2)} \tag{5.3}$$

and

$$\Delta^2 = |\boldsymbol{x}_r|^2 - d^2 = x_r^2 + y_r^2 + z_r^2 - d^2 \tag{5.4}$$

Remember, $V_{z_{r_{comm}}}$ is the relative vertical velocity needed for the DSAAVs to pass within distance d of one another at their point of closest approach. Since we want the dynamics to be continuous, we are assuming that the system does not switch on and off and is always trying to maneuver the vehicles within d of each other—even if the avoidance maneuver is not necessary.

The ± determines which DSAAV passes over the other one. In the actual collision avoidance system, a criteria selects whether the ± is plus or minus. As we saw in Section 4.3, our CAS chooses the sign which minimizes the vehicles' deviations from their intended paths. However, to simplify the analysis, we are forcing the ± to be plus. In other words, we are forcing DSAAV 1 to pass over DSAAV 2. So, the vertical velocity commands are those found in Equation (5.2) where

$$V_{z_{r_{comm}}} = \frac{-z_r(V_{x_r}x_r + V_{y_r}y_r) + \sqrt{\Delta^2[(z_r^2 - \Delta^2)(V_{x_r}^2 + V_y^2) + (V_{x_r}x_r + V_{y_r}y_r)^2]}}{(z_r^2 - \Delta^2)} \tag{5.5}$$

5.1.1 Linearized Dynamics

The vehicle dynamics in the vertical direction are nonlinear. So in order to do stability analysis, we apply linearized perturbation analysis techniques [20]. Perturbation analysis assumes that for small deviations about an *equilibrium point*, the dynamics are approximately linear.

Intuitively, the equilibrium points of our CAS should consist of DSAAV states that will result in a miss distance exactly equal to *d*. Along these trajectories, no collision avoidance maneuver is needed to keep the vehicles safely separated. And so V_{z_r} must equal $V_{z_{r_{comm}}}$ in these equilibrium states.

In general, however, equilibrium points are states of the system for which the state derivatives are equal to zero. Looking at the Equation (5.1), we can determine the equilibrium points; they satisfy

$$\left.\frac{d\mathbf{x}}{dt}\right|_{\mathbf{x} = \mathbf{x}_e} = \mathbf{f}(\mathbf{x}_e) = \mathbf{0} \tag{5.6}$$

First of all, we see that for any equilibrium point, the following must be true:

$$V_{x_1} = V_{x_2} = V_{y_1} = V_{y_2} = V_{z_1} = V_{z_2} = 0 \tag{5.7}$$

So, unlike our intuitive definition of equilibrium above, Equation (5.7) prohibits nonzero velocities in any direction. This means that there are equilibrium states for our collision avoidance controller that are not equilibrium states of the overall system. Nevertheless, our "intuitive" equilibrium points are the ones about which we are going to linearize; thus, we will allow the velocities to take nonzero values.

Again, for the collision avoidance equilibrium points, V_{z_r} will equal $V_{z_{r_{comm}}}$. This will result in

$$\begin{aligned}
\frac{d}{dt}(V_{z_1}) &= -\frac{V_{z_1}}{\tau_v} + \frac{V_{z_2}}{\tau_v} + \frac{V_{z_{r_{comm}}}}{\tau_v} = \frac{V_{z_{r_{comm}}}}{\tau_v} - \frac{V_{z_r}}{\tau_v} = 0 \\
\frac{d}{dt}(V_{z_2}) &= -\frac{V_{z_2}}{\tau_v} + \frac{V_{z_1}}{\tau_v} - \frac{V_{z_{r_{comm}}}}{\tau_v} = \frac{V_{z_r}}{\tau_v} - \frac{V_{z_{r_{comm}}}}{\tau_v} = 0
\end{aligned} \tag{5.8}$$

which means the CAS does not need to command a new vertical velocity. We will hold off on discuss specific equilibrium points, but there are certainly a few obvious ones.

Next, we linearize the system around these equilibrium points. Remember that the system is described by the state derivatives, **f**(**x**), from Equation (5.1). To generate the linearized system matrix, differentiate **f**(**x**) by the state, **x**. The result is a first-order, linear approximation to the system around the equilibrium point:

$$\delta\dot{x} = \frac{df}{dx}(x_e)\delta x \tag{5.9}$$

So, for our system

$$J = \frac{df}{dx}(x_e) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{I}{\tau_v} & \frac{L}{\tau_v} & \frac{J}{\tau_v} & \frac{M}{\tau_v} & \frac{K}{\tau_v} & -\frac{1}{\tau_v} & -\frac{I}{\tau_v} & -\frac{L}{\tau_v} & -\frac{J}{\tau_v} & -\frac{M}{\tau_v} & -\frac{K}{\tau_v} & \frac{1}{\tau_v} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{I}{\tau_v} & -\frac{L}{\tau_v} & -\frac{J}{\tau_v} & -\frac{M}{\tau_v} & -\frac{K}{\tau_v} & \frac{1}{\tau_v} & \frac{I}{\tau_v} & \frac{L}{\tau_v} & \frac{J}{\tau_v} & \frac{M}{\tau_v} & \frac{K}{\tau_v} & -\frac{1}{\tau_v} \end{bmatrix} \tag{5.10}$$

where

$$I = \frac{\partial V_{z_{r_{comm}}}}{\partial x_1} = -\frac{\partial V_{z_{r_{comm}}}}{\partial x_2} \tag{5.11}$$

$$L = \frac{\partial V_{z_{r_{comm}}}}{\partial V_{x_1}} = -\frac{\partial V_{z_{r_{comm}}}}{\partial V_{x_2}} \tag{5.12}$$

$$K = \frac{\partial V_{z_{r_{comm}}}}{\partial z_1} = -\frac{\partial V_{z_{r_{comm}}}}{\partial z_2} \tag{5.13}$$

Since the dynamics are the same in the x and y directions (i.e. constant velocity), J and M can be found from (5.11) and (5.12) by simply switching x_r with y_r and V_{x_r} with V_{y_r} in those equations.

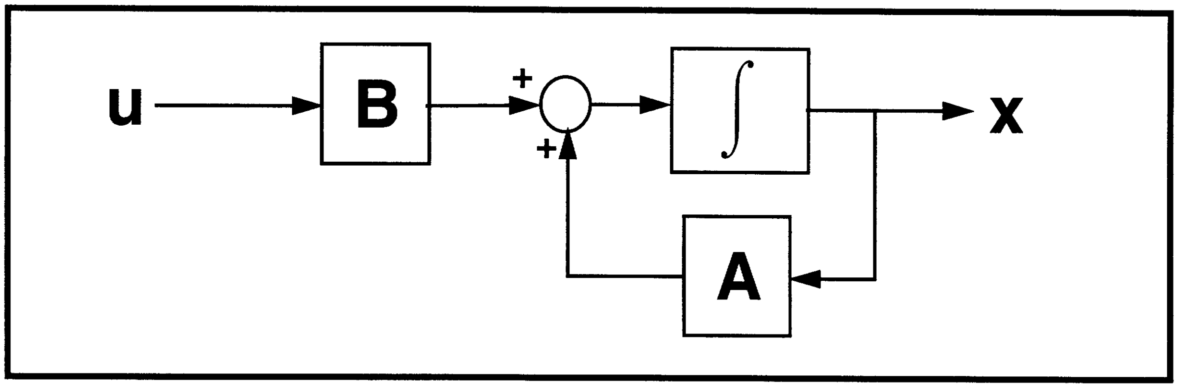


Figure 5.1: *Open-loop dynamics*

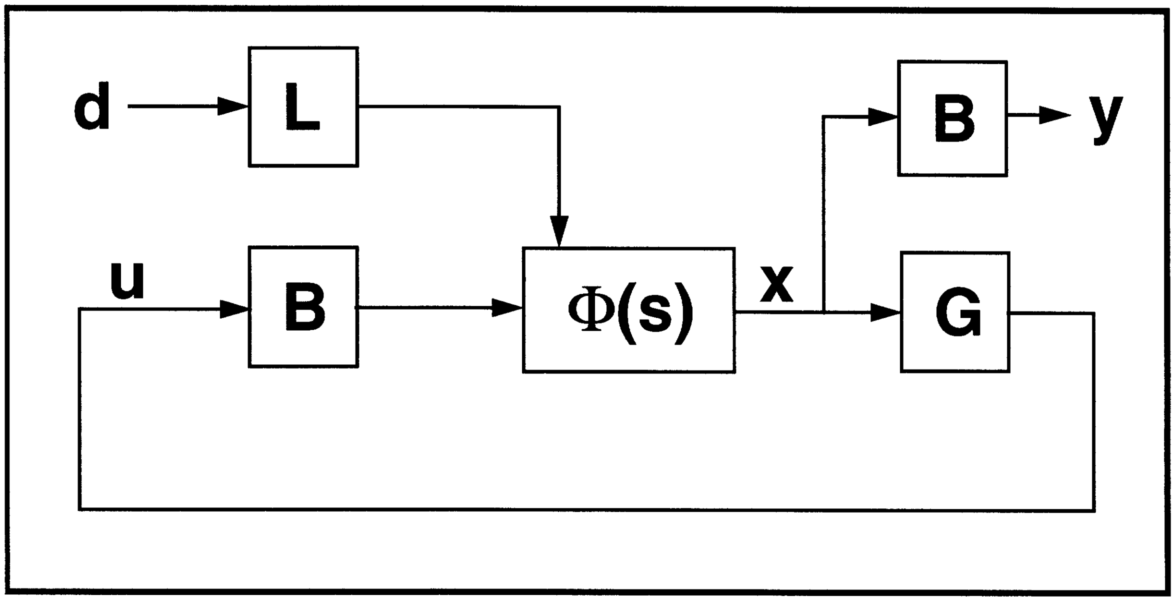


Figure 5.2: *Closed-loop system*

Note that our linearized system, $\boldsymbol{J}$, can be thought of simply as a linear regulator, like the one shown in Figure 5.2, where

$$\begin{aligned} \dot{x} &= A\dot{x} + Bu + Ld \\ y &= Cx \\ u &= -Gx \end{aligned} \tag{5.14}$$

We can see that $\boldsymbol{J}$ is simply the system matrix for the closed-loop system:

$$\begin{aligned} \dot{x} &= (A - BG)x + Ld \\ J &= A - BG \end{aligned} \tag{5.15}$$

From this, we can find the feedback matrix, ***BG***:

$$BG = A - J = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{I}{\tau_v} & -\frac{L}{\tau_v} & -\frac{J}{\tau_v} & -\frac{M}{\tau_v} & -\frac{K}{\tau_v} & 0 & \frac{I}{\tau_v} & \frac{L}{\tau_v} & \frac{J}{\tau_v} & \frac{M}{\tau_v} & \frac{K}{\tau_v} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{I}{\tau_v} & \frac{L}{\tau_v} & \frac{J}{\tau_v} & \frac{M}{\tau_v} & \frac{K}{\tau_v} & 0 & -\frac{I}{\tau_v} & -\frac{L}{\tau_v} & -\frac{J}{\tau_v} & -\frac{M}{\tau_v} & -\frac{K}{\tau_v} & 0 \end{bmatrix} \tag{5.16}$$

So, if

$$u = \begin{bmatrix} V_{z_{1_{comm}}} \\ V_{z_{2_{comm}}} \end{bmatrix} \tag{5.17}$$

and

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \frac{1}{\tau_v} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\tau_v} \end{bmatrix}^{\mathrm{T}} \tag{5.18}$$

then we find that

$$G = \begin{bmatrix} -I & -L & -J & -M & -K & 0 & I & L & J & M & K & 0 \\ I & L & J & M & K & 0 & -I & -L & -J & -M & -K & 0 \end{bmatrix} \tag{5.19}$$

Now that we have a linear system, we can apply traditional linear stability analysis by looking at the eigenvalues and eigenvectors of the system. Due to the large size of our system, we will do this numerically rather than analytically. But before we can proceed with the analysis, remember that we must find some equilibria trajectories for the vehicles.

5.1.2 Equilibria Trajectories

As explained before, an equilibrium state is one that requires no collision avoidance command. In other words, it is when the vehicles' constant-velocity trajectories will bring them within a distance, *d*, of each other at the point of closest approach.

The perhaps the simplest equilibrium trajectory is what we will call the *head-on maneuver.* As the name suggests, it occurs when the two vehicles are moving at one another head-on. We will assume that both DSAAVs are moving along the x-axis. To further simplify the situation, the vehicles in our headon manuever will have zero vertical and lateral velocities ($V_{z_1} = V_{z_2} = V_{y_1} = V_{y_2} = 0$); in other words, their trajectories will be parallel. This is depicted below in Figure 5.3.

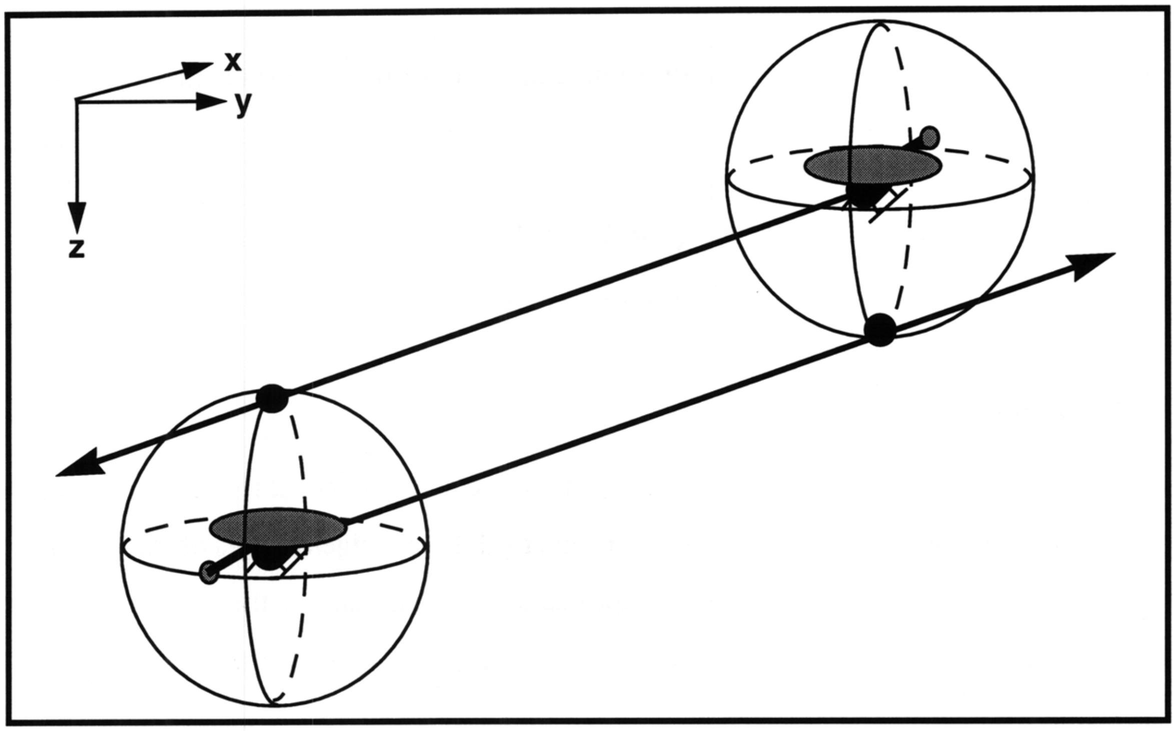


Figure 5.3: *Headon manuever*

In the diagram, the vehicles maintain a constant vertical separation of *d*. The separation need not be completely in the vertical direction, however. When the longitudinal separation ($|x_r|$) is zero—the point of closest approach—the DSAAVs only need to lie on opposite sides of a circle of diameter, *d*, in the y-z plane (see Figure 5.4).

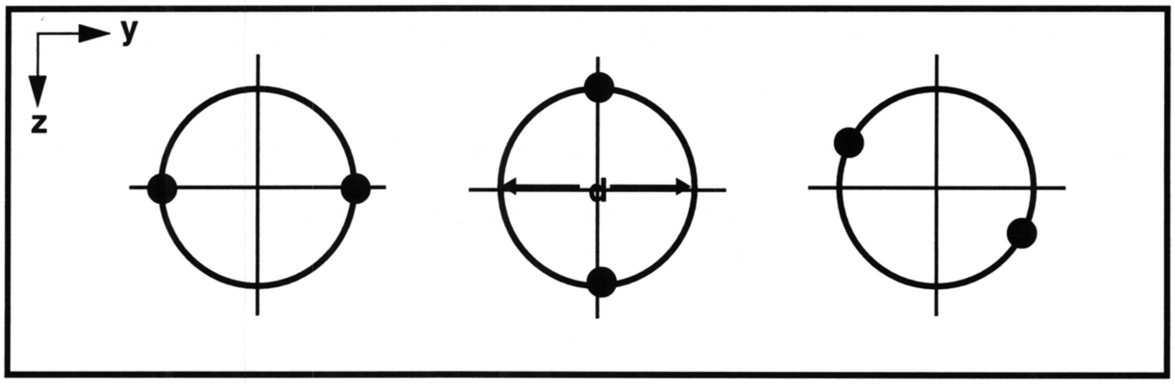


Figure 5.4: *Relative positions at t**

Then, at that point of closest approach ($y_r = y_r{}^*, z_r = z_r{}^*$), the following is true:

$$(z_r^*)^2 + (y_r^*)^2 = d^2 \tag{5.20}$$

And, since $V_{z_r} = V_{y_r} = 0$, the equilibrium condition for the head-on maneuver is as follows:

$$\begin{aligned} V_{y_r} &= V_{z_r} = 0 \\ z_r^2 + y_r^2 &= d^2 \end{aligned} \tag{5.21}$$

5.1.3 Stability Analysis

The Matlab script, *acas.m*, discretizes the state-space and searches it for equilibrium states of the headon maneuver, then computes the eigenvalues and eigenvectors of the matrix **J** at those states. In doing so, it reveals a simplification we can make to the system. That simplification more or less eliminates the need to numerically solve for the eigenvalues of the system.

Decoupled Dynamics

We find that of the system's twelve eigenvalues, two of them are associated with the collision-avoidance dynamics, while the other ten are associated with the constant-velocity dynamics. Those ten eigenvalues are zero, and account for the constant velocities of the vehicles.

Furthermore, the vertical dynamics (z-direction) of the two-vehicle system are completely decoupled from the lateral dynamics (x- and y-directions). Knowing this, we can reduce our twelve-state system to a four-state system, containing only the vertical states.

We now proceed to analytically evaluate the four-state system:

$$\frac{d}{dt}\begin{bmatrix} z_1 \\ V_{z_1} \\ z_2 \\ V_{z_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{\tau_v} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{1}{\tau_v} \end{bmatrix} \begin{bmatrix} z_1 \\ V_{z_1} \\ z_2 \\ V_{z_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{V_{z_{1_{comm}}}}{\tau_v} \\ 0 \\ \frac{V_{z_{2_{comm}}}}{\tau_v} \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x}) \tag{5.22}$$

If we linearize the system, we get the following:

$$\boldsymbol{J} = \frac{d\boldsymbol{f}}{d\boldsymbol{x}}(\boldsymbol{x}_e) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ K & -\frac{1}{\tau_v} & -K & \frac{1}{\tau_v} \\ 0 & 0 & 0 & 1 \\ -K & \frac{1}{\tau_v} & K & -\frac{1}{\tau_v} \end{bmatrix} \tag{5.23}$$

where, as before,

$$\begin{aligned} K &= \frac{\partial V_{z_{r_{comm}}}}{\partial z_1} = -\frac{\partial V_{z_{r_{comm}}}}{\partial z_2} \\ &= -\frac{(V_{x_r}x_r + V_{y_r}y_r)}{z_r^2 - \Delta^2} + z_r\frac{\sqrt{(z_r^2 - \Delta^2)(V_{x_r}^2 + V_{y_r}^2) + (V_{x_r}x_r + V_{y_r}y_r)^2}}{(z_r^2 - \Delta^2)\sqrt{\Delta^2}} \end{aligned} \tag{5.24}$$

Dynamic Modes

First, let's find the eigenvalues and eigenvectors of the linearized closed-loop system. To find the eigenvalues,

$$|\lambda \boldsymbol{I} - \boldsymbol{J}| = 0 \tag{5.25}$$

$$\det\begin{bmatrix} \lambda & -1 & 0 & 0 \\ -K\left(\lambda+\frac{1}{\tau_v}\right) & K & -\frac{1}{\tau_v} \\ 0 & 0 & \lambda & -1 \\ K & -\frac{1}{\tau_v} & -K\left(\lambda+\frac{1}{\tau_v}\right) \end{bmatrix} = 0 \tag{5.26}$$

$$\lambda^2\left(\lambda^2+\frac{2}{\tau_v}\lambda-2K\right) = 0$$

So the eigenvalues of the system are

$$\lambda = 0, 0, -\frac{1}{\tau_v} \pm \sqrt{\frac{1}{\tau_v^2}+2K} \tag{5.27}$$

Next we look at the modes corresponding to the eigenvalue $\lambda = 0$. Calculating the eigenvector, **e**, we get

$$\boldsymbol{Je} = \lambda\boldsymbol{e} \quad \Rightarrow \quad (\lambda\boldsymbol{I}-\boldsymbol{J})\boldsymbol{e} = 0 \quad \Rightarrow \quad \boldsymbol{Je} = 0 \tag{5.28}$$

So, if we assume that

$$V_{z_1} = V_{z_2} = 0 \tag{5.29}$$

then we find

$$\boldsymbol{e} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{5.30}$$

In this mode, if either DSAAV's vertical velocity is perturbed from zero, its vertical position will begin to change linearly with time. In order to maintain proper separation, the other DSAAV's CAS will perturb its vertical velocity by the same amount in order to establish a new equilibrium. The result is that both z_1 and z_2 change linearly with time in the same direction, and both vehicles stray from their initial altitudes.

If the vertical velocities are perturbed from their new equilibrium value, the rate at which z_1 and z_2 are changing will be altered. This drifting altitude phenomenon is discussed in Section 4.3, where it is confirmed with simulation. We again emphasize that the vehicles' relative vertical velocity does not change, and their new trajectories are also equilibria trajectories.

Now, for CAS stability, we need

$$\begin{aligned} \lambda &= -\frac{1}{\tau_v} + \sqrt{\frac{1}{\tau_v^2} + 2K} < 0 \\ \frac{1}{\tau_v} &> \sqrt{\frac{1}{\tau_v^2} + 2K} \quad \Rightarrow \quad \boldsymbol{K < 0} \end{aligned} \tag{5.31}$$

So, K must be less than zero for the closed-loop CAS to be stable.

Like we did with the twelve-state system, let's begin by looking at the simple—yet important—head-on maneuver, where

$$\begin{aligned} z_r &= -d \\ V_{z_r} = V_{y_r} &= y_r = 0 \end{aligned} \tag{5.32}$$

For $K < 0$ we need

$$\begin{aligned} K &= -\frac{V_{x_r} x_r}{d^2 - x_r^2} - d\frac{\sqrt{(d^2 - x_r^2)V_{x_r}^2 + (V_{x_r} x_r)^2}}{(d^2 - x_r^2)\sqrt{x_r^2}} < 0 \\ K &= -\frac{V_{x_r} x_r}{d^2 - x_r^2} - \frac{d\sqrt{V_{x_r}^2 d^2}}{(d^2 - x_r^2)\sqrt{x_r^2}} < 0 \end{aligned} \tag{5.33}$$

First, if $d^2 > x^2$,

$$V_{x_r} x_r > \frac{-d\sqrt{V_{x_r}^2 d^2}}{\sqrt{x_r^2}} \tag{5.34}$$

If V_{x_r} and x_r are the same sign,

$$V_{x_r}x_r > \frac{-V_{x_r}d^2}{x_r} \tag{5.35}$$

Also, a multiplication by V_{x_r} and a division by x_r won't change the direction of the inequality, so the stability condition is

$$x_r^2 > -(d^2) \tag{5.36}$$

This will always be true. But if V_{x_r} and x_r have opposite signs, we discover that

$$x_r^2 < d^2 \tag{5.37}$$

is the necessary condition for stability. That is the same as our beginning assumption. So for $d^2 > x_r^2$, the system is always stable.

When $d^2 < x_r^2$, we find that the system is stable only when V_{x_r} and x_r have opposite signs. Now we see that the system stability depends only the relative vehicle motion in the longitudinal (x) direction. Specifically, as long as the vehicles are moving toward one another along that axis, the system is stable.

So, the CAS is stable in the head-on scenario as long as the DSAAVs are moving toward each other along the x-axis or $x_r^2 < d^2$. Remember, however, that this is a very simple equilibrium trajectory. What happens when y_r, V_{y_r}, and V_{z_r} are not zero?

First of all, if $y_r \neq 0$, then we know that $z_r^2 = d^2 - y_r^2$ (See Equation 5.20). With $V_{y_r} = 0$ and $V_{z_r} = 0$, this is still an equilibrium trajectory; the DSAAVs are still moving toward each other, but one is no longer directly above the other.

Solving $K < 0$ with these values will result in the slightly different condition as above. The system is still stable when the vehicles are moving toward each other. However, instead of the system always being stable for $x_r^2 < d^2$, the range of values of x_r^2 for which the system is stable decreases as the value of y_r^2 increases. So, we have a stable system as long as

$$\frac{V_{x_r}}{x_r} < 0, \quad x_r^2 > z_r^2$$

$$\text{or} \tag{5.38}$$

$$x_r^2 < (d^2 - y_r^2)$$

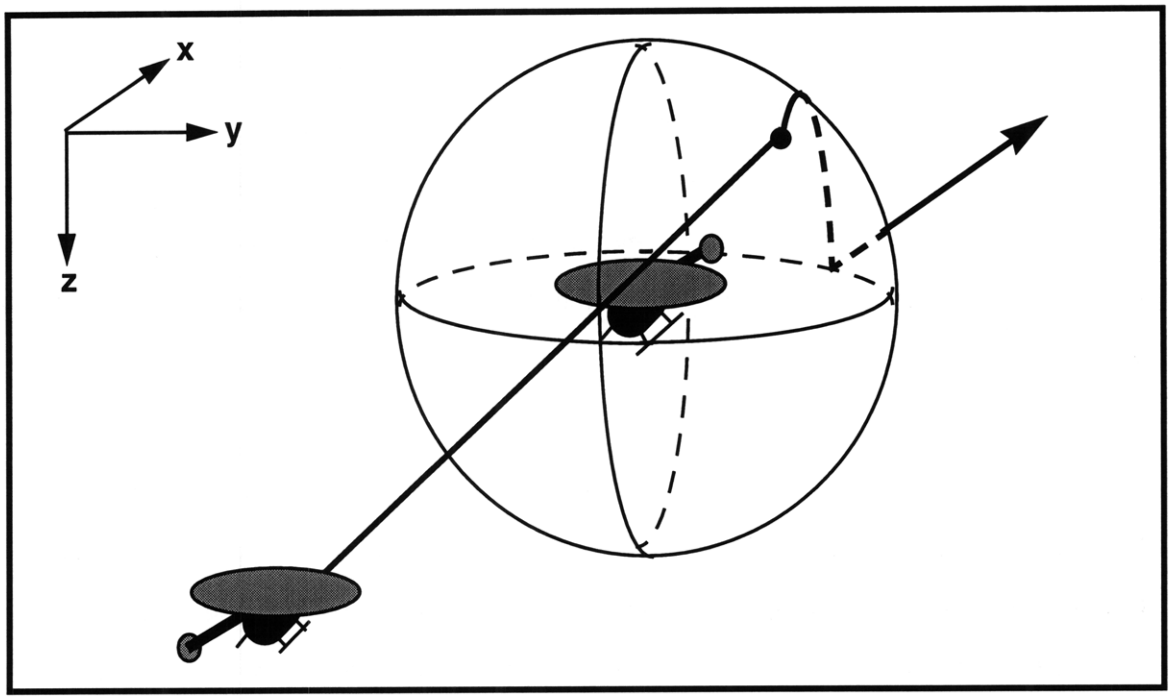


Figure 5.5: *Headon manuever*

In summary, the system is stable as long as there is a vertical velocity which brings each DSAAV tangent to a sphere of radius d centered on the other vehicle. When $V_{x_r}/x_r < 0$, these velocities are constant, and the vehicles' trajectories are both straight lines tangent to the spheres. This is the case in the first segment of the DSAAV's trajectory in Figure 5.5.

While $x_r^2 < (d^2 - y_r^2)$ and $V_{x_r}/x_r < 0$, the collision avoidance velocities will have the DSAAVs continuously move tangent to the spheres—they will trace the surface of the spheres, as seen in the curved segment of the DSAAV's path in Figure 5.5. The greater their lateral separation at crossing, the less of the surface of the spheres they will trace.

As the vehicles continue to move away from one another, they move off the spheres, and after that no vertical velocity can result in a tangential trajectory. That is where the unstable region ($V_{x_r}/x_r > 0, \quad x_r^- > z_r^-$) begins. In Figure 5.5, this unstable region is represented by the straight segment of the vehicle's trajectory coming off of the back of the sphere.

For the $V_{y_r} \neq 0$ case, we come to an observation that is fundamental to the analysis of our system: every $V_{y_r} \neq 0$ case can be transformed into the $V_{y_r} = 0$ case with a simple rotation of the reference frame. If we rotate our coordinate system around the z-axis, eventually all of the vehicles' horizontal motion will be in the x-direction. The transformed system is now just our simple head-on case.

Also notice that $V_{z_r} \neq 0$ does not affect the value of *K*, and therefore does not affect stability. So the generalized stability condition for all encounters is that the horizontal distance (i.e. in the x-y plane) between the two vehicles must be decreasing when $x_r^2 + y_r^2 > d^2$.

5.1.4 Root Locus

Now we will characterize the dependency of the roots of the system on the relative position and velocity of the vehicles. From Equation 5.27, we know that the roots of the system depend only on τ_v and *K*. Table 5.1 summarizes that dependency.

Value of *K*	**Position of poles, λ**
$K > 0$	1 stable, 1 unstable
$K = 0$	$\lambda = 0, -\frac{2}{\tau_v}$
$K = -\frac{1}{2\tau_v^2}$	$\lambda = -\frac{1}{\tau_v}, -\frac{1}{\tau_v}$
$K < -\frac{1}{2\tau_v^2}$	$\lambda = -\frac{1}{\tau_v} + i\omega(K)$

Table 5.1: *CAS root locus*

So the system is stable for all $K < 0$, but becomes more oscillatory as *K* becomes more and more negative. The question now becomes, what is the relationship between *K* and the relative positions and velocities of the vehicles? A plot of K vs. x_r for the head-on case tells us something.

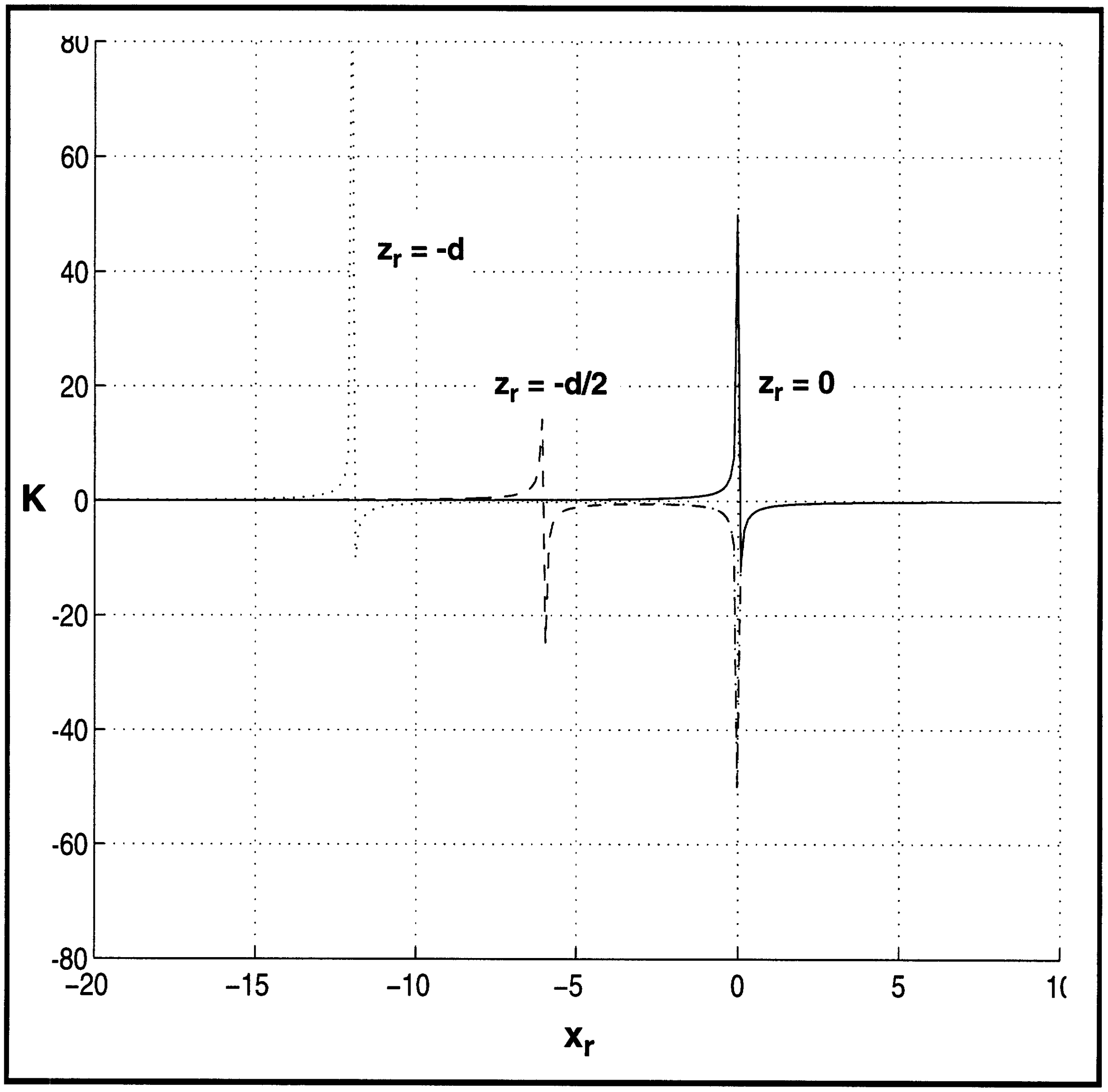


Figure 5.6: *K vs. x_r*

Figure 5.6 is a plot of K vs. x_r for three head-on trajectories in which $V_{x_r} = -1$. In the first, $z_r = -d$, so the two DSAAVs are passing directly over one another. In the second case, $z_r = -d/2$. In the final trajectory, the vehicles are passing side-by-side one another ($z_r = 0$).

We can see that K is small and negative throughout most of the region. Nevertheless, as the DSAAVs approach the $x_r = 0$ point (point of closest approach), K approaches $-\infty$. Thus the system becomes very oscillatory in this region.

The same is true immediately after passing through the $x_r = 0$ point, where the vehicles begin to follow the surface of the sphere. K becomes small and negative again as they follow the sphere, until they run out of sphere to follow ($x_r^2 = d^2 - y_r^2$). As they approach that point, K again approaches $-\infty$, and the system becomes more oscillatory. After passing through that point, K is positive and the system is unstable.

So, as long as the time spent in these highly oscillatory regions is small, they should not present a problem. Fortunately, as Figure 5.6 shows, these regions are narrow. In a true head-on encounter, the vehicles should not be in these regions very long. Unfortunately, when the vehicles are nearly side-by-side, they may be in this region for a longer period of time. Sometime, however, disturbances can knock them out of this region of the state space, or the collision avoidance system shuts off anyway.

Shared-Avoidance CAS

The collision-avoidance system we have just analyzed only uses the current position and velocity information from the other vehicle to generate a collision-avoidance maneuver. Furthermore, it assumes the other vehicle's velocity is constant—that it is not maneuvering to avoid collision. In our stem, however, the other DSAAV is avoiding the collision, and is doing so with the same algorithm. The result is what might appear to be twice as much maneuvering as is necessary. As we have shown, the closed-loop nature continuously corrects for the over-maneuvering and achieves the proper separation between the vehicles. Nonetheless, it might be worthwhile to find out if the stability of the system improves with an algorithm which does not over-maneuver.

To do so, we look at the shared-avoidance system briefly mentioned in Section 4.3, in which half of the relative avoidance velocity is assigned to each vehicle. So our closed-loop system is as follows:

$$\frac{d}{dt}\begin{bmatrix} z_1 \\ V_{z_1} \\ z_2 \\ V_{z_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{\tau_v} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{1}{\tau_v} \end{bmatrix} \begin{bmatrix} z_1 \\ V_{z_1} \\ z_2 \\ V_{z_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{V_{z_{1_{comm}}}}{\tau_v} \\ 0 \\ \frac{V_{z_{2_{comm}}}}{\tau_v} \end{bmatrix} \tag{5.39}$$

where

$$\begin{aligned} V_{z_{1_{comm}}} &= \frac{V_{z_{r_{comm}}}}{2} + V_{z_2} \\ V_{z_{2_{comm}}} &= \frac{-V_{z_{r_{comm}}}}{2} + V_{z_1} \end{aligned} \tag{5.40}$$

So, we end up with

$$\boldsymbol{J} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{K}{2} & -\frac{1}{\tau_v} & -\frac{K}{2} & \frac{1}{\tau_v} \\ 0 & 0 & 0 & 1 \\ -\frac{K}{2} & \frac{1}{\tau_v} & \frac{K}{2} & -\frac{1}{\tau_v} \end{bmatrix} \tag{5.41}$$

which has eigenvalues

$$\lambda = 0, -\frac{1}{\tau_v} \pm \sqrt{\frac{1}{\tau_v^2} + K} \tag{5.42}$$

So we see that for stability, K—the same K as before—again must be negative. But a more detailed root locus (Table 5.2) shows the differences between the original CAS and the shared-avoidance system.

Value of K	**Position of poles, λ**
$K > 0$	1 stable, 1 unstable
$K = 0$	$\lambda = 0, -\frac{2}{\tau_v}$
$K = -\frac{1}{\tau_v^2}$	$\lambda = -\frac{1}{\tau_v}, -\frac{1}{\tau_v}$
$K < -\frac{1}{\tau_v^2}$	$\lambda = -\frac{1}{\tau_v} + i\omega(K)$

Table 5.2: *Shared-avoidance root locus*

Notice that in the shared-avoidance system, the range of values of K for which the system is stable and not oscillatory is twice as large as in the original system. So, it would appear that the shared-avoidance system is a slight improvement on the original system. However, as discussed in Section 4.3, the drawback to this algorithm is that it relies on the assumption that the other vehicle operates under the same collision-avoidance system. Depending on the multi-AAV system, this assumption may or not be a safe one.The original algorithm, however, will succeed even if the other vehicle is not avoiding collision. In fact, we will look next at the performance of our original CAS when one vehicle is noncooperative.

Noncooperative Collision Avoidance

We mentioned in Section 4.3 that the shared-avoidance system fails to meet the system requirement when one vehicle is not cooperating. We also said, on the other hand, that original system succeeds in this noncooperative scenario. Let's now take a look at the stability of the original system in the noncooperative collision-avoidance case.

The closed-loop system becomes

$$\frac{d}{dt}\begin{bmatrix} z_1 \\ V_{z_1} \\ z_2 \\ V_{z_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{\tau_v} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{1}{\tau_v} \end{bmatrix} \begin{bmatrix} z_1 \\ V_{z_1} \\ z_2 \\ V_{z_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{V_{z_{1_{comm}}}}{\tau_v} \\ 0 \\ 0 \end{bmatrix} \tag{5.43}$$

and once again

$$V_{z_{1_{comm}}} = V_{z_{r_{comm}}} + V_{z_2} \tag{5.44}$$

$$\boldsymbol{J} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ K & -\frac{1}{\tau_v} & -K & \frac{1}{\tau_v} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{1}{\tau_v} \end{bmatrix} \tag{5.45}$$

The eigenvalues of the noncooperative system are as follows:

$$\det\begin{bmatrix} \lambda & -1 & 0 & 0 \\ -K & \left(\lambda + \frac{1}{\tau_v}\right) & K & -\frac{1}{\tau_v} \\ 0 & 0 & \lambda & -1 \\ 0 & 0 & 0 & \left(\lambda + \frac{1}{\tau_v}\right) \end{bmatrix} = 0$$

$$\left(\lambda^2 + \frac{\lambda}{\tau_v}\right)\left(\lambda^2 + \frac{\lambda}{\tau_v} - K\right) = 0 \tag{5.46}$$

$$\lambda = 0,\ -\frac{1}{\tau_v},\ -\frac{1}{2\tau_v} \pm \sqrt{\frac{1}{4\tau_v^2} + K}$$

So, once again, $K < 0$, is the condition for stability. As we did with the original system and the shared-avoidance system, we look at the detailed root locus of the noncooperative system (Table 5.3).

Value of K	**Position of poles, λ**
$K > 0$	1 stable, 1 unstable
$K = 0$	$\lambda = 0, -\frac{1}{\tau_v}$
$K = -\frac{1}{4\tau_v^2}$	$\lambda = -\frac{1}{2\tau_v}, -\frac{1}{2\tau_v}$
$K < -\frac{1}{4\tau_v^2}$	$\lambda = -\frac{1}{2\tau_v} + i\omega(K)$

Table 5.3: *Noncooperative CAS root locus*

In the noncooperative system, we find no change in the stability criterion, but we do see a change in the root locus. We see, perhaps surprisingly, that the region of K values which results in a stable, nonoscillating response is half as large as in the original CAS. Furthermore, the oscillations are damped out only half as fast as in the original system. So it seems that any near-instability that was found in the excessive maneuvers of the original system do not improve when one vehicle is noncooperative. We do note, however, that the "drifting altitudes" phenomenon which was seen in the original system does not exist in this system.

5.2 Time Delay

As mentioned in Section 4.3, the communication time delay between the vehicles may limit the performance of the DSAAV collision avoidance system. This section attempts to characterize the affect of the time delay

Before introducing time delay into our two-vehicle model, we must first acknowledge that this delay only affects state information passing between the two vehicles. Each vehicle receives its own state in real-time. With this in mind, it might be worthwhile to separate the system dynamics between the two vehicles. The closed-loop dynamics described by Equation (5.15) can be separated as follows:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = (A - BG)x = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{5.47}$$

This is visualized in the block diagram in Figure 5.7.

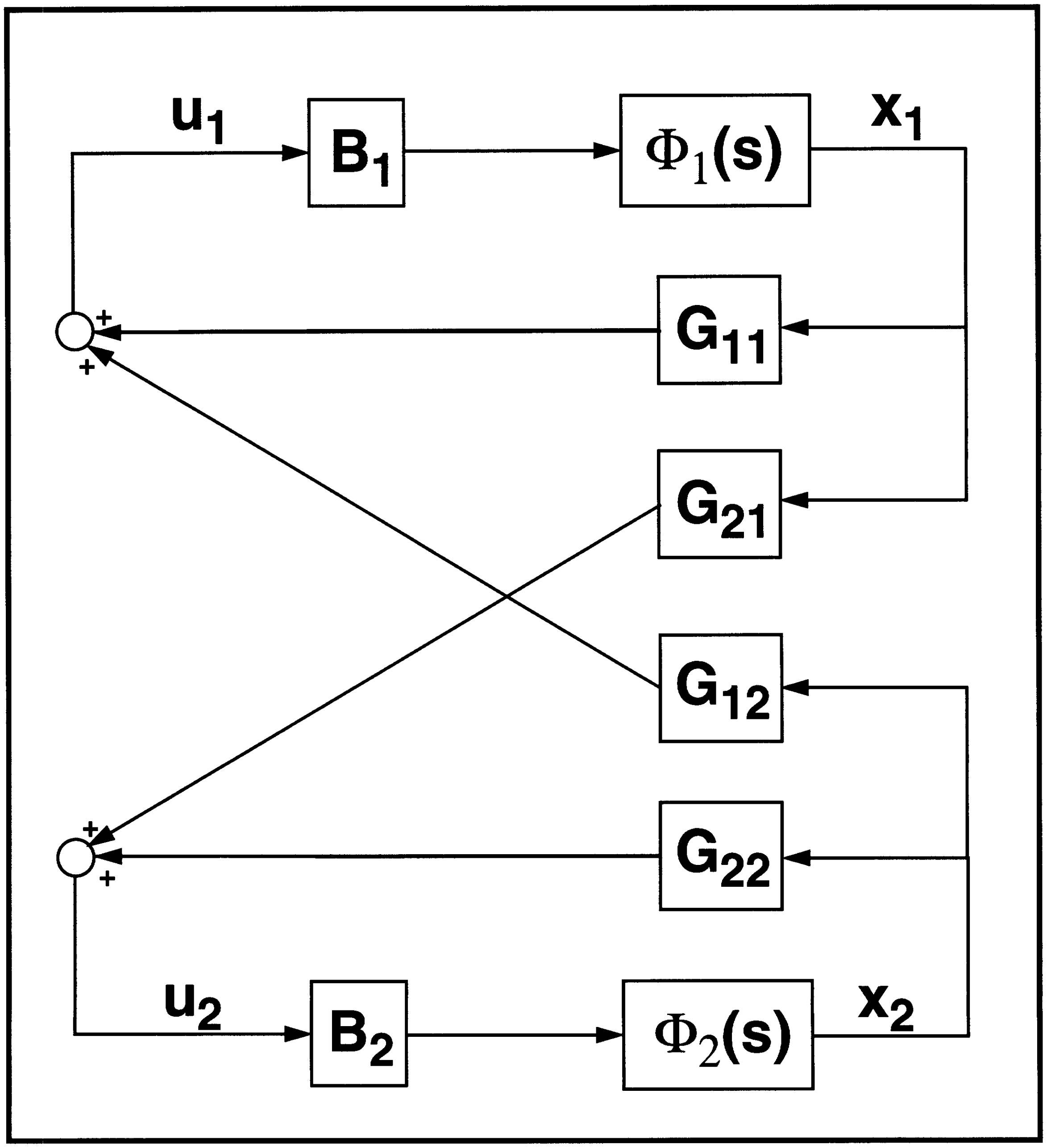


Figure 5.7: *Decoupled system*

Next, we can add time delays to the outputs that are being sent from each DSAAV to the other. Our system now looks like the one in Figure 5.8.

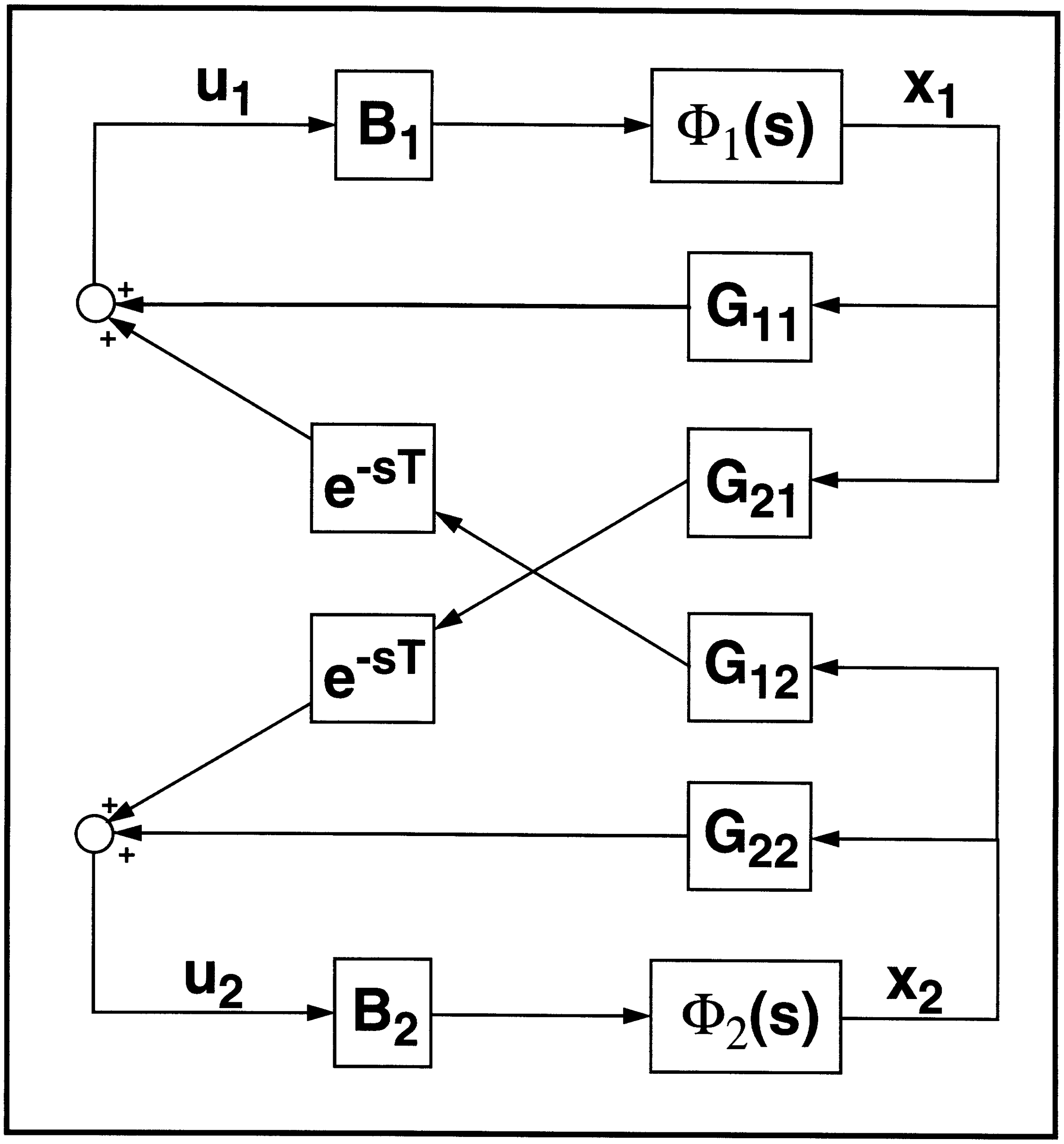


Figure 5.8: *Decoupled system with time delay*

Redrawing the block diagram in Figure 5.8 gives us a clearer visualization of the entire closed-loop system:

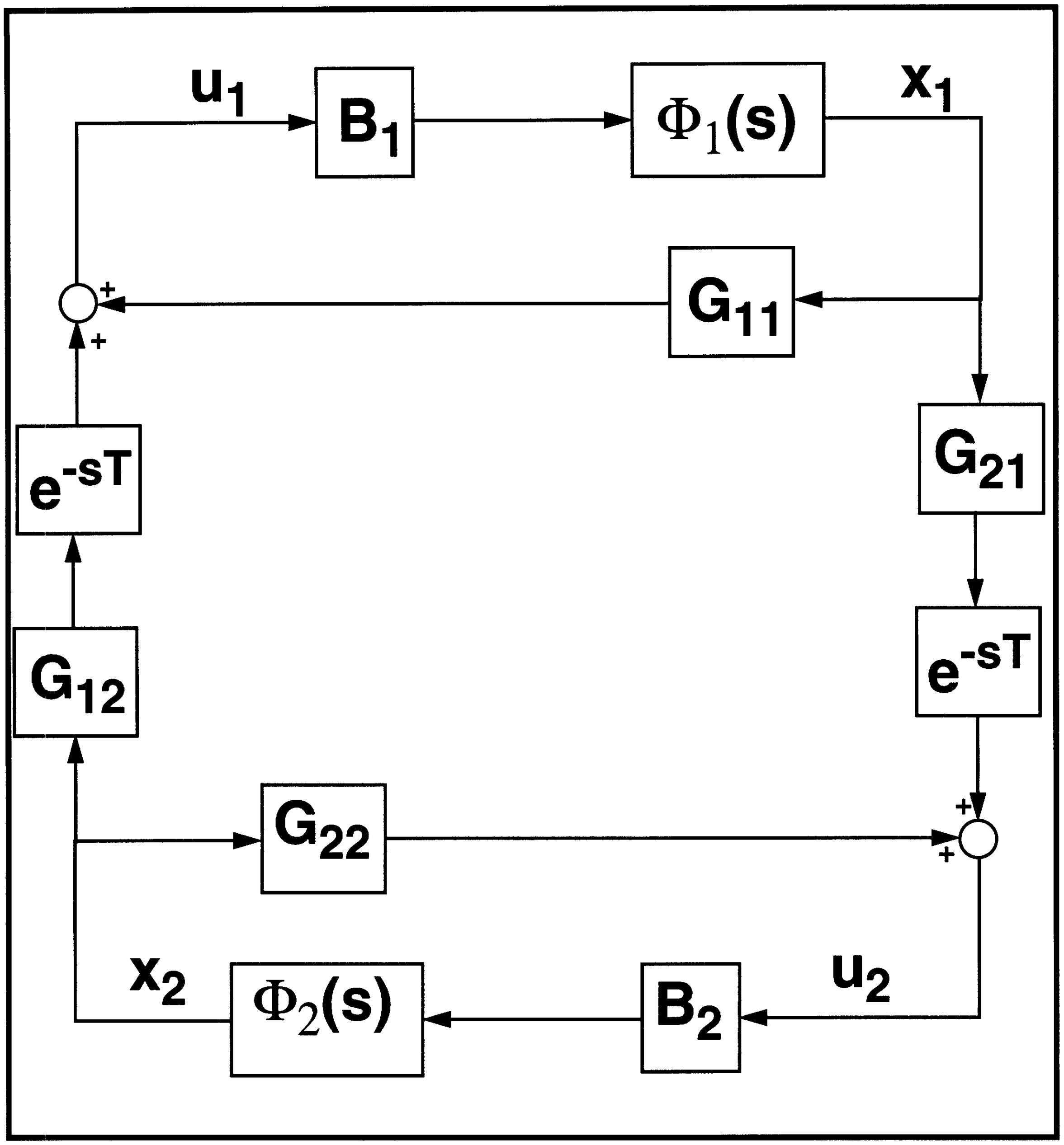


Figure 5.9: *Decoupled, closed-loop system*

To insert a time delay into a state space system, we use a Pade approximation for the time delay of T seconds [21]:

$$e^{-sT} \cong \frac{1 - \frac{T}{2}s}{1 + \frac{T}{2}s} = \frac{y_\Delta(s)}{u_\Delta(s)} = \frac{y_\Delta(s)\xi_\Delta(s)}{\xi_\Delta(s)u_\Delta(s)}$$

$$\frac{\xi_\Delta(s)}{u_\Delta(s)} = \frac{1}{1 + \frac{T}{2}s} \Rightarrow \dot{\xi}_\Delta = -\frac{2}{T}\xi_\Delta + \frac{2}{T}u_\Delta \tag{5.48}$$

$$\frac{y_\Delta(s)}{\xi_\Delta(s)} = 1 - \frac{T}{2}s \Rightarrow y_\Delta = 2\xi_\Delta - u_\Delta$$

From Equation (5.48), we see that the state-space representation of Pade approximation is

$$\begin{aligned} \dot{\xi}_\Delta &= -\frac{2}{T}\xi_\Delta + \frac{2}{T}u_\Delta \\ y_\Delta &= 2\xi_\Delta - u_\Delta \end{aligned} \tag{5.49}$$

Now, the outputs of the vehicle dynamics are the inputs to the time delay, and vice-versa. The result is a closed-loop system which consists of four distinct subsystems: DSAAV 1 dynamics, time delay between DSAAV 1 and 2, DSAAV 2 dynamics, and the time delay between DSAAV 2 and 1.

In the Matlab script, *acas.m*, these four subsystems are linked in a closed-loop manner. The eigenvalues of the resulting system are then computed at several points along a head-on trajectory. The trajectory is repeated for different values of time delay in order to find the maximum time delay, T_{max}, before the system becomes unstable.

First, in Figure 5.10, we plot T_{max} vs. x_r. The relationship is linear, and depends on V_{x_r}. From the plot we find that the value of T_{max} is dependent only on $|V_{x_r}/x_r|$. Specifically, for the system to be stable, the time delay must satisfy the following inequality:

$$T < 2\left|\frac{x_r}{V_{x_r}}\right| \tag{5.50}$$

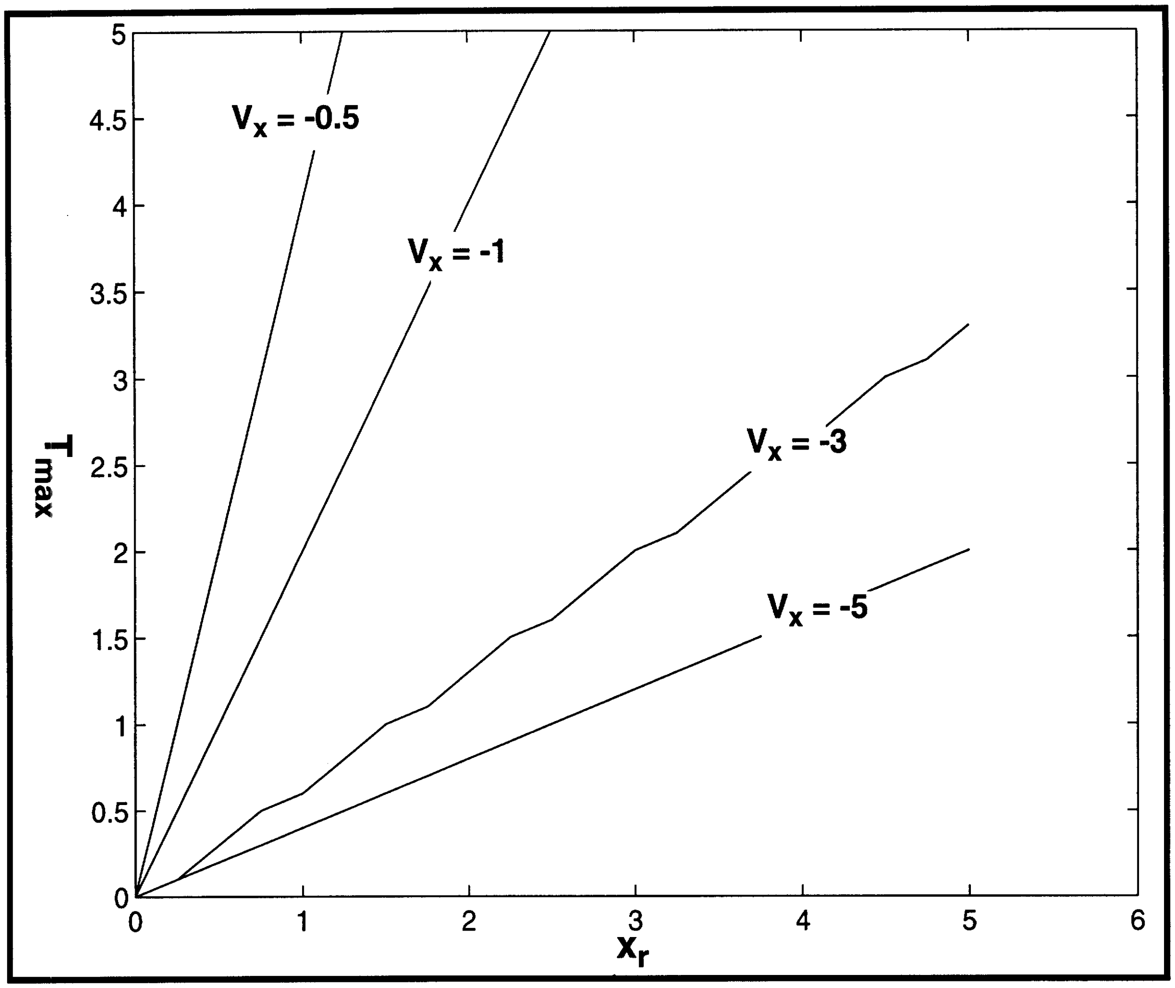


Figure 5.10: T_{max} *vs.* x_r

Now, we point out that for our head-on maneuver, the expression $|x_r/V_{x_r}|$ is actually the time to closest approach, t^*. So our stability criterion becomes

$$T < 2t^* \tag{5.51}$$

So, for a finite *T*, the CAS will in fact become unstable in some region around the point of closest approach.

5.3 Unmodeled Dynamics

Our model of the DSAAV collision avoidance system does not incorporate all of the dynamics of the actual system. And while our model does closely approximate the actual dynamics, it is worth mentioning a couple of the elements it neglects: dynamic coupling between the axes of motion and nonlinearities.

5.3.1 Dynamic Coupling

Our model of the vehicle motion assumes that the dynamics in the x-, y-, and z-directions are completely decoupled. This assumption is tested in the DSAAV simulation, where a more complex and accurate vehicle model is used. The results are shown in Figures 5.11 through 5.13, where the effects of step velocity inputs on the various axes are shown.

We can see that there is indeed some coupling among the axes. This coupling is very small when x- and y-velocities are commanded. When the 2 ft/sec V_z step is commanded, however, there is a substantial resultant V_y (0.2 ft/sec peak). This coupling could very well alter the system response.

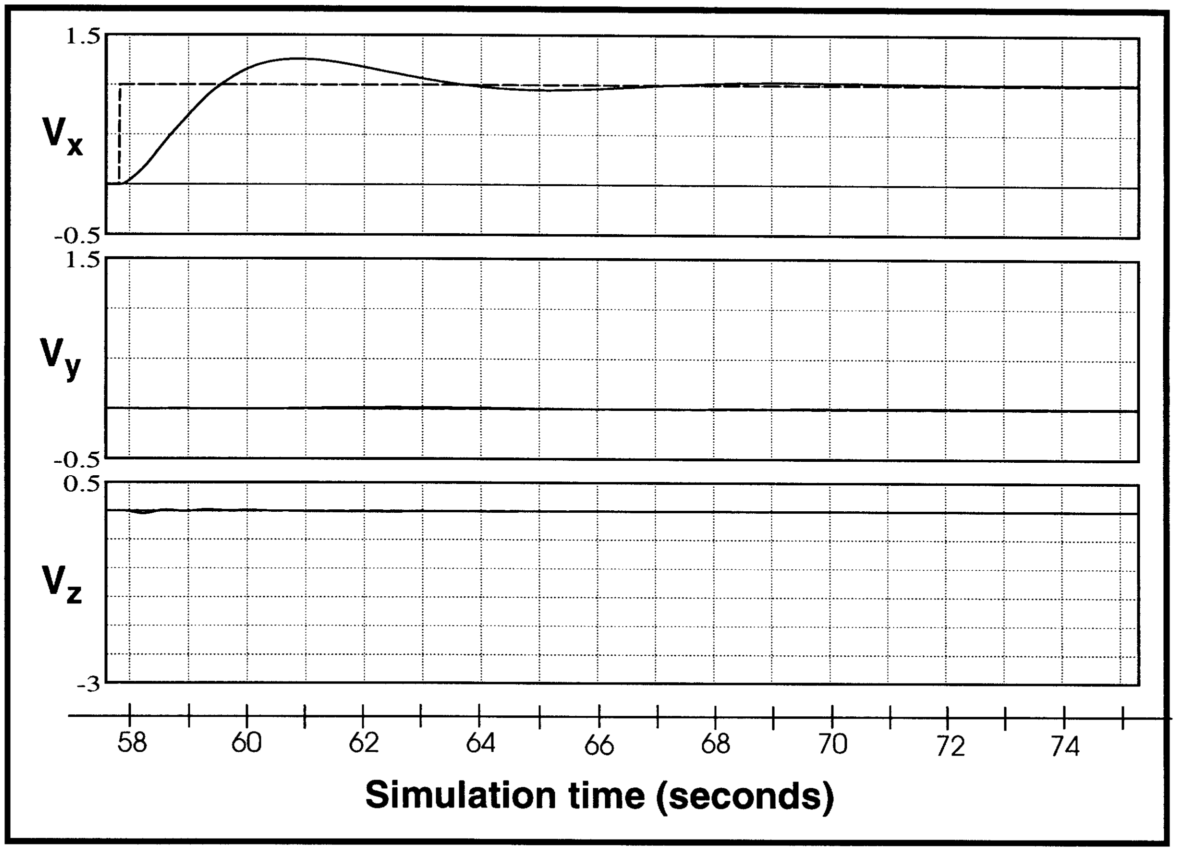


Figure 5.11: *Coupled dynamics in response to step x-velocity command*

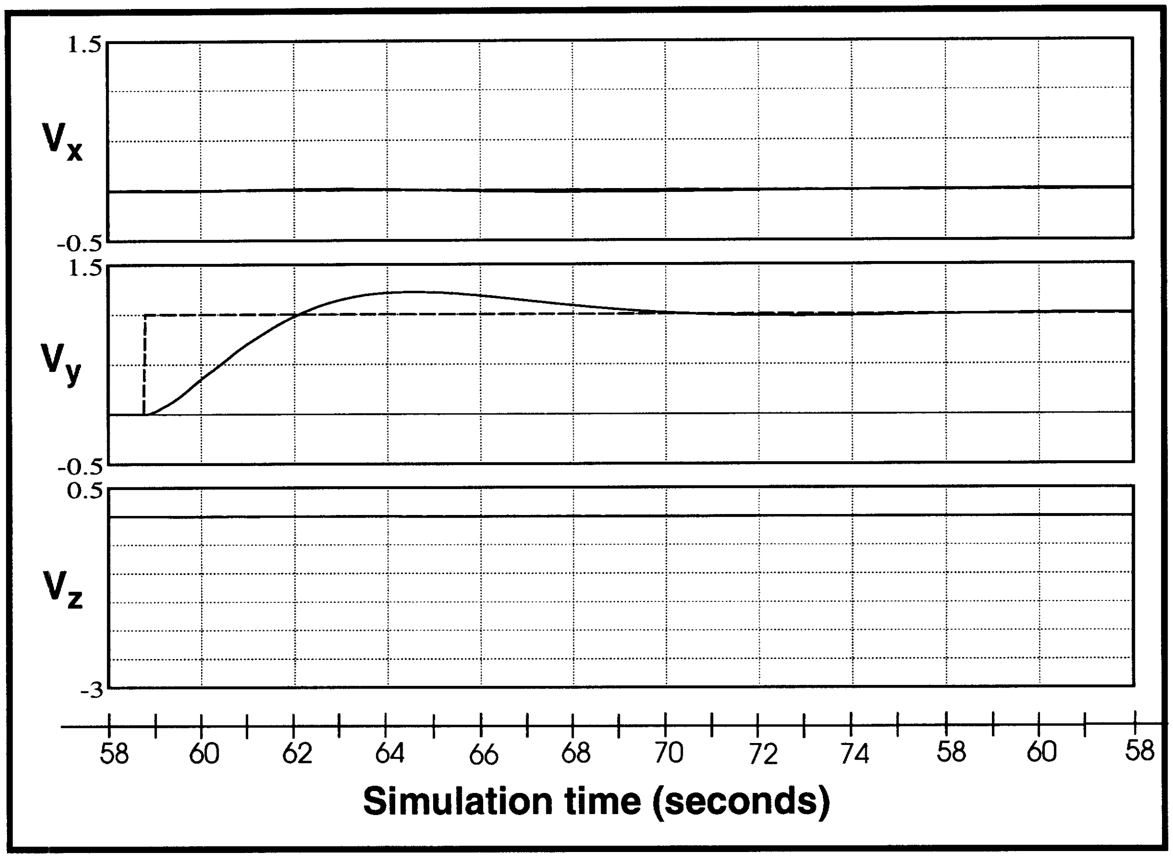


Figure 5.12: *Coupled dynamics in response to step y-velocity command*

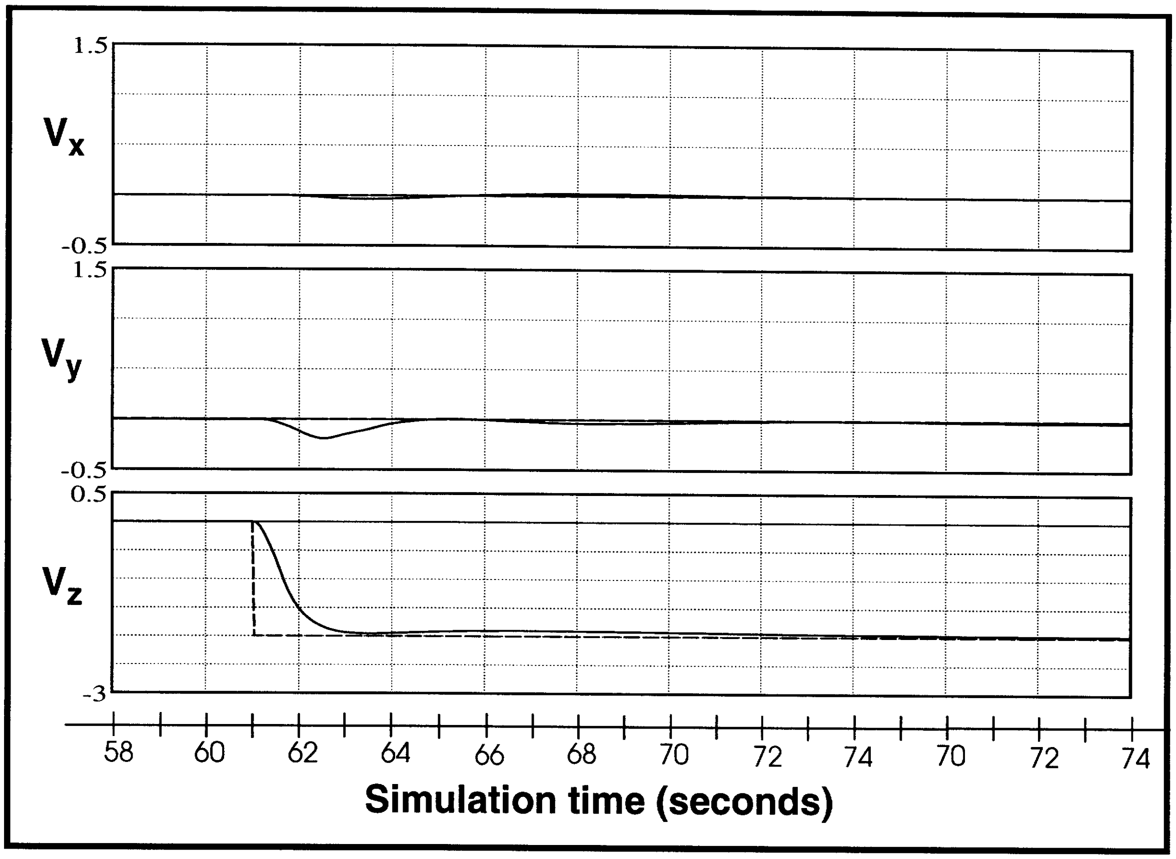


Figure 5.13: *Coupled dynamics in response to step z-velocity command*

5.3.2 Other Nonlinearities

Our CAS model also neglects a number of nonlinearities which are introduced by the collision avoidance algorithm or the vehicle control system. Three of these nonlinearities are collision avoidance relays, which can instantaneously switched between two possible values. The fourth nonlinearity is a saturation.

The first switch determines whether the vehicle is avoiding to the top or bottom of the sphere of separation centered on the other DSAAV. The second switch occurs when the system passes into one of the no-solution situations, as described in Section 4.3.2. Lastly, the collision avoidance system switches completely on and off depending if avoidance maneuvering is needed. The saturation nonlinearity is a limit placed on the vehicle velocities by the control system.

One of the most useful methods for analyzing the above nonlinearities is *describing function analysis*, which is described in [18]. Describing functions approximate the dynamics of the nonlinearity in response to a sinusoidal input function. Using describing functions and traditional frequency domain techniques, the existence of limit cycles and the stability of the system can be determined.

5.4 Conclusions

In this chapter, we have examined the stability and performance of our collision avoidance algorithm. We first took our algorithm and framed it in the context of a feedback control system, allowing us to utilize the many techniques associated those systems. We characterized the stability of the system, and verified that it agreed with our intuition. We found that our two-vehicle CAS is in fact stable for most operating regimes, with a few exceptions,

Next, we attempted to analyze what could be one of the more severe limitations facing real-time, distributed collision avoidance systems: time delay. Most importantly, we again found a way to insert communication time delay in our CAS, while still analyzing it like a traditional control system. Lastly, we mentioned the parts of our system which did not get analyzed here, but should still be considered in the design and analysis of a CAS.

Chapter 6

Conclusions

Autonomous air vehicles will undoubtedly become more and more capable in the coming years. As they do, people will put them to use in greater numbers; everyone from military generals to movie directors are going to want AAVs to go to work for them. Eventually, AAVs are going to find themselves interacting with other AAVs, either cooperatively or in intruder scenarios. The AAVs that safely complete their missions will be those equipped with collision avoidance systems.

6.1 Autonomous Collision Avoidance

6.1.1 CAS Design and Analysis Methodology

First and foremost this document is to be a somewhat comprehensive reference to those investigating multi-AAV operations—especially collision avoidance. Hopefully, by documenting the design and analysis of our autonomous CAS, we have provided a checklist of sorts for future designers of collision avoidance systems. This checklist is by no means complete, but it should at least prevent the omission of some obvious design considerations and allow others to learn from our mistakes.

6.1.2 Strive for Balance in Design and Analysis

The collision avoidance algorithm found in Appendix A bears little resemblance to the one first created with crude sketches nearly a year ago. At first, that simple collision avoidance algorithm probably seemed flawless. And had it stayed on paper, it would have been.

However, part of engineering is transferring ideas from paper sketches and simple models to functioning systems. Since the DSAAV vehicles are not easy to replace in the case of a collision, we have been fortunate to have a high-fidelity vehicle simulation to test our designs on. The DSAAV simulation has pointed out countless scenarios in which the algorithm would have caused very undesirable performance. Yet, for every such scenario found through simulation, we wonder how many others still lurk out there.

That fear leads to more rigorous analysis techniques to find those algorithm flaws that just do not show up on paper or in simulation. For a CAS that has a lot of switching between finite states, it is especially important to understand what states can lead to poor performance. In our bang-bang controller, we occasionally encountered a situation in simulation in which the AAVs maneuvered the same direction and had a direct collision. A finite state analysis of our controller might have revealed that potential hazard.

On the other hand, sometimes the thorough analysis reveals something positive about the system. We faced this in Chapter 5, when we discovered that having both vehicles maneuver as if the other wasn't is more stable than the shared-avoidance case.

So, one key to designing and analyzing a CAS is to maintain a balance between common sense, simulation results, and mathematical analysis.

6.1.3 Choose Functional Over Optimal

Much of the collision avoidance literature is focused on optimal conflict resolution maneuvers, while ignoring practical issues such as computational complexity, time delays, and system disturbances. This may be fine for advancing collision avoidance theory, but it does not always aid in the actual design of a CAS.

When designing a collision avoidance system, it is best to *start very simple and only make the system as complex as it needs to be*. Following this principle, the DSAAV CAS was designed and tested in a relatively short amount of time.

6.1.4 Communication is Important

In general, the more information a CAS has about the other vehicles, the better chance the AAV has of avoiding conflict. Conversely, anything that inhibits the communication among the AAVs will also hinder the performance of the collision avoidance systems. We saw that this was the case with time delay and trajectory uncertainty.

Ideally, AAVs should use advanced communication, such as intent sharing and maneuver coordination, in their collision avoidance systems. Such systems would certainly be more reliable than the basic constant-velocity trajectory extrapolation we are using. Nevertheless, in the absence of these more advanced communication capabilities, its important that the CAS does the best with the information it has.

6.1.5 Safety First

The bottom line with autonomous collision avoidance is that people are not going to trust their vehicles to an unsafe CAS. That means that collision avoidance systems need to have an idea of their expected performance. This is what makes the separation lower bounds in [10] so important. Alternatively, the probabilistic methods found in [9] and [16] can also give a metric of CAS performance.

6.2 Suggestions for Future Research

The investigation of autonomous collision avoidance should not and will not end with this thesis. Plenty of work remains to be done, and some of it will be implemented in actual flight tests.

6.2.1 Nonlinear Analysis

The closed-loop analysis begun in Chapter 5 should be expanded to include the nonlinearities mentioned in Section 5.3.2. These nonlinearities may cause degraded system performance, and should be analyzed and understood. Since the collision avoidance system is a mixture of continuous dynamics and discrete states, it is a *hybrid* system. The interactions between the continuous and the discrete can be complex and difficult to analyze. Nevertheless, understanding them is important in the design of an autonomous CAS.

6.2.2 Flexible Collision Avoidance System

Future collision avoidance algorithms should be flexible in how much and what kind of information they need. Ultimately, vehicle state estimates are used in the collision avoidance algorithm; it should not matter what information was used to generate those estimates.

Using a Kalman filter, measurements could come from a number of sources, and an optimal estimate would be produced. Depending on the accuracy of the measurements, the CAS should adapt its conflict thresholds and avoidance maneuvers to reflect its lack of certainty in the vehicle trajectories. So an AAV receiving only intruder position updates could run the same CAS as one which also receives intent information over a high-bandwidth data link. Because of uncertainty, however, it will not let itself get as close to the intruder.

6.2.3 Three or More AAVs

AAV conflict encounters will not be limited to two vehicles. And while the two-vehicle collision avoidance problem is more fundamental, conflicts involving three or more AAVs should still be investigated.

6.2.4 Flight Test the CAS

Finally, as autonomous CAS designs advance, they need to move away from block diagrams and simulations and move on board actual AAVs. Flight tests of CASs will reveal things that simulation and analytical analysis cannot.

The DSAAV vehicles should be running their collision avoidance systems in flight tests in the coming year. Such a test will be an important demonstration and, as far as we know, the first of its kind.

References

[1] Poggio, M., and Poggio, T. Cooperative physics of fly swarms: an emergent behavior. MIT Artificial Intelligence Laboratory Memo #1512, Cambridge, MA, December 1994.

[2] Gage, Douglas W. Command control for many-robot systems. *Proceedings of AUVS-92*, Huntsville, AL, 1992.

[3] Brooks, Rodney A. A robust layered control system for a mobile robot. MIT Artificial Intelligence Laboratory Memo #864, Cambridge, MA, 1985.

[4] Parker, Lynne E. Local vs. global control laws for cooperative agent teams. MIT Artificial Intelligence Laboratory Memo #1357, Cambridge, MA, March 1992.

[5] White, David A., and Sofge, Donald A., eds. Handbook of intelligent control : neural, fuzzy, and adaptive. Van Nostrand Reinhold, New York, 1992.

[6] Trott, Christian A. Electronics design for an autonomous helicopter. SB/SM Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June 1997.

[7] Krozel, J., Mueller, T., and Hunter, G. Free flight conflict detection and resolution analysis. *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, San Diego, CA, July 1996.

[8] Federal Aviation Administration. U.S. Federal Aviation Regulations, Part 125. Washington, D.C., 1997.

[9] Kuchar, James K. A unified methodology for the evaluation of hazard alerting systems. Ph.D. thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, January 1995.

[10] Shewchun, J. Marc, and Feron, Eric. Linear matrix inequalitites for analysis of free flight conflict problems. Paper submitted to IEEE Conference on Decision and Control, 1997.

[11] Oh, Jae-Hyuk, and Feron, Eric. Primal-dual quadratic programming approach to multiple conflict resolution. Paper submitted to American Control Conference, 1998.

[12] Kosecka, J., Tomlin, C., Pappas, G., and Sastry, S. Generation of conflict resolution maneuvers for air traffic management. Submitted for the International Conference on Robotics and Intelligent Systems, Sept. 1997.

[13] Tomlin, C., Pappas, G., and Sastry, S. Conflict resolution for air traffic management: a case study in multi-agent hybrid systems. Technical report, University of California at Berkeley, 1996.

[14] Gelb, Arthur, ed. *Applied optimal estimation.* MIT Press, Cambridge, MA, 1974.

[15] Yang, L. and Kuchar, J. Prototype conflict alerting logic for free flight. Paper AIAA-97-0220, 35th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January 6-10, 1997.

[16] Paielli, Russel A., and Erzberger, H. Conflict probability estimation for free flight. Paper AIAA-97-0001, 35th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January 6-10, 1997.

[17] Hall, William. Air transportation system model under free flight. Presentation to MIT Operations Research Center, May 1996.

[18] Gelb, Arthur, and Vander Velde, Wallace E. Multiple-input describing functions and nonlinear system design. McGraw-Hill, New York, 1986.

[19] Appleby, Brent D. Reducing shuttle-payload dynamic interaction with notch filters. SM thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, February 1987.

[20] Mohler, Ronald R. *Nonlinear systems, volume 1: dynamics and control.* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[21] Stevens, Brian L., and Lewis, Frank L. *Aircraft control and simulation.* John Wiley and Sons, New York, 1992.

Appendix A

Collision Avoidance Code

Section A.2 contains the C functions that are called from the DSAAV's control software to generate a collision avoidance command. But first, Section A.1 contains an excerpt from the *on_board.spech* file, which defines and initializes the variables and parameters of the CAS.

A.1 Variable Definitions (from *on_board.spech*)

```
%Dir aavhcas_ref {
 |enum heliID id        sw      HELI_A  :helicopter id;
  enum Switch waycomm   sw   ON         :helicopters sharing waypoint info;
  enum Switch on        sw   OFF        :engage collision avoidance system;
  enum Switch coop      sw   ON         :cooperative collision avoidance;
  enum Switch forceAvoid sw  OFF        :force collision avoidance;
  enum Switch delayComp sw   ON         :compensate for time delay;

  float separation        ft            :estimated separation distance;
  float minsep            ft    12      :min. separation distance allowed;
  float minAvoidAlt       ft    -4      :minimum avoidance maneuver alt;
  int    mandir           na            :0=manuever down, 1=maneuver up;
  float manmax            ft/s -2       :maximum vertical maneuver velocity;
  float manmin            ft/s  1       :minimum vertical maneuver velocity;
  float retmax            ft/s -2       :maximum vertical return velocity;
  float retmin            ft/s 1        :minimum vertical return velocity;
  Float delayBuff[DELAYBUFF_SIZE][18]  na
                                        :communication delay buffer;
  float tdComm            sec           :communication time delay;
  int   diffInd           na    0       :number time steps of comm delay;
  float tdEst             sec   .04     :communication time delay estimate;
  float trmax             sec   7       :max. time-to-collision after manvr;
  float tau               sec   .4      :vertical velocity time constant;

  float o_pos[3]          ft            :position of other heli;
  float o_u               ft/s          :body axis velocity of other heli;
  float o_v               ft/s          :body axis velocity of other heli;
  float o_w               ft/s          :body axis velocity of other heli;
  float o_phi             rad           :attitude of other heli;
  float o_theta           rad           :attitude of other heli;
  float o_psi             rad           :attitude of other heli;
  Float o_tBL[3][3]       na            :transform from body to local frame;

 |float xr          ft          :relative x-positions of the AAVs;
 |float yr          ft          :relative y-positions of the AAVs;
 |float zr          ft          :relative z-positions of the AAVs;
 |float Vxr         ft          :relative x-velocities of the AAVs;
|float Vyr         ft            :relative y-velocities of the AAVs;
|float Vzr         ft            :relative z-velocities of the AAVs;
|float Vxrc        ft            :rel. x-velocities after gdc command;
|float Vyrc        ft            :rel. y-velocities after gdc command;
|float Vzrc        ft            :rel. z-velocities after gdc command;
|float D2          ft            :distance - minsep;
|float xrtrue      ft            :relative x-positions of the AAVs;
|float yrtrue      ft            :relative y-positions of the AAVs;
|float zrtrue      ft            :relative z-positions of the AAVs;
|float Vxrtrue     ft            :relative x-velocities of the AAVs;
|float Vyrtrue     ft            :relative y-velocities of the AAVs;
|float Vzrtrue     ft            :relative z-velocities of the AAVs;
|float tstartrue   sec           :time until minimum separation;

|double     zrstar       ft;
|float      sep0         ft          :min. projected separation distance;
|float      tsep0        sec         :time until minimum separation;
|float      sep0c        ft          :min. projected sep. dist. (comm);
|float      tsep0c       sec         :time until min. sep. (commanded);
|float      sepm         ft          :min. sep. distance during maneuver;
|float      tsepm        sec         :time til min. sep. during maneuver;
|enum Switch avoid    sw    OFF      :execute avoidance maneuver;
|float      Vzrcomm1 ft/s            :commanded rel. vertical vel. (up);
|float      Vzrcomm2 ft/s            :commanded rel. vertical vel. (down);
|float      Vzrcomm  ft/s            :commanded rel. vertical vel.;
|float      w_avoid  ft/s            :avoidance maneuver z-axis
                                     :body velocity (+ down);
|float      w_avoid1 ft/s            :vel. command (up);
|float      w_avoid2 ft/s            :vel. command (down);
|float      dstar    ft              :min. projected separation distance;
|float      tstar    sec             :time until minimum separation;
|float      dstarc   ft              :min. projected sep. dist. (comm);
|float      tstarc   sec             :time until min. sep. (commanded);
 float      tstarmin sec             :min. time to closest approach
                                     :before avoiding;
|float      K        na              :collison avoidance variable;
|float      lambda1[2]  na           :eigenvalue of CAS;
|float      lambda2[2]  na           :eigenvalue of CAS;
|enum Switch safeStop sw    OFF  :;
};
```

A.2 Bang-Bang Collision Avoidance Algorithm (from *on_board_hcas.c*)

```
void HCASUpdate1( struct aavnavigation_ref *n,
              struct aavhcas_ref *hcas,
              struct aavon_board_con_ref *c) {
  float xrm,yrm,zrm,z_err,Vx1,Vx2,Vxc,Vy1,Vy2,Vyc,Vz1,Vz2,Vzc,
  wcom1,wcom2,wcom3,manave,tm,sepm,tsepm,xrr,yrr,zrr,tr,sepr,
  sep,xSep,ySep,zSep;
  int coop, mandir;
```

```
GetOtherAAVState( hcas );


/* Convert body velocities (u,v,w) to local frame (Vn,Ve,Vd) */

Vx1 = n->tBL[0][0]*n->u + n->tBL[0][1]*n->v + n->tBL[0][2]*n->w;
Vy1 = n->tBL[1][0]*n->u + n->tBL[1][1]*n->v + n->tBL[1][2]*n->w;
Vz1 = n->tBL[2][0]*n->u + n->tBL[2][1]*n->v + n->tBL[2][2]*n->w;

Vx2 = hcas->o_tBL[0][0]*hcas->o_u + hcas->o_tBL[0][1]*hcas->o_v
    + hcas->o_tBL[0][2]*hcas->o_w;
Vy2 = hcas->o_tBL[1][0]*hcas->o_u + hcas->o_tBL[1][1]*hcas->o_v
    + hcas->o_tBL[1][2]*hcas->o_w;
Vz2 = hcas->o_tBL[2][0]*hcas->o_u + hcas->o_tBL[2][1]*hcas->o_v
    + hcas->o_tBL[2][2]*hcas->o_w;


/* Compute position and velocity relative to other helicopter */

if(!(hcas->delayComp)) {
  hcas->xr = n->pos[0] - hcas->o_pos[0];
  hcas->yr = n->pos[1] - hcas->o_pos[1];
  hcas->zr = n->pos[2] - hcas->o_pos[2];
}
else {
  hcas->xr = n->pos[0] - (hcas->o_pos[0] + Vx2*hcas->tdEst);
  hcas->yr = n->pos[1] - (hcas->o_pos[1] + Vy2*hcas->tdEst);
  hcas->zr = n->pos[2] - (hcas->o_pos[2] + Vz2*hcas->tdEst);
}

hcas->Vxr = Vx1 - Vx2;
hcas->Vyr = Vy1 - Vy2;
hcas->Vzr = Vz1 - Vz2;


/* Compute current estimated separation distance */
hcas->separation = sqrt(hcas->xr*hcas->xr + hcas->yr*hcas->yr
                         + hcas->zr*hcas->zr);


/* Assume other helicopter is non-cooperative if cooperative
   switch has been shut off or if other helicopter is
   below minimum avoidance altitude */

if(!hcas->coop || hcas->o_pos[2] > hcas->minAvoidAlt)
  coop = 0;
else
  coop = 1;


/* Calculate time and distance of closest approach */

if(hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr
                          + hcas->Vzr*hcas->Vzr != 0) {
  hcas->tstar = -(hcas->xr*hcas->Vxr + hcas->yr*hcas->Vyr
                          + hcas->zr*hcas->Vzr)/
                 (hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr
                          + hcas->Vzr*hcas->Vzr);
    hcas->tstar = MAX(0,hcas->tstar);
    hcas->dstar  = ComputeHelSep(hcas->tstar,hcas->xr,hcas->yr,
                          hcas->zr,hcas->Vxr,hcas->Vyr,hcas->Vzr);
  }
  else {
    /* Helicopters not moving-- */
    hcas->dstar = sqrt(hcas->xr*hcas->xr + hcas->yr*hcas->yr
                          + hcas->zr*hcas->zr);
  }

  if(hcas->avoid == OFF) {

    if(hcas->dstar <= hcas->minsep) {

      /* Calculate relative position after rise time
       of vertical velocity command */
      xrm = hcas->Vxr*4*hcas->tau + hcas->xr;
      yrm = hcas->Vyr*4*hcas->tau + hcas->yr;
      zrm = hcas->Vzr*4*hcas->tau + hcas->zr;

      /* Determine direction of avoidance maneuver */

      if(zrm < 0 || (!hcas->coop && n->pos[2] > hcas->minAvoidAlt)){
      /* Maneuver up */
      wcom1 = hcas->manmax;
      if(coop)
        wcom2 = hcas->manmin;
      else
        wcom2 = hcas->o_w;
      mandir = 1;
      }
      else if(zrm > 0) {
      /* Maneuver down */
      wcom1 = hcas->manmin;
      if(coop)
        wcom2 = hcas->manmax;
      else
        wcom2 = hcas->o_w;
      mandir = 0;
      }
      else if(zrm == 0) {
      if(hcas->zr < 0) {
        /* Maneuver up */
        wcom1 = hcas->manmax;
        if(coop)
          wcom2 = hcas->manmin;
        else
          wcom2 = hcas->o_w;
        mandir = 1;
      }
      else if(hcas->zr > 0){
        /* Maneuver down */
        wcom1 = hcas->manmin;
        if(coop)
          wcom2 = hcas->manmax;
        else
          wcom2 = hcas->o_w;
        mandir = 0;
      }
```

```
else if(hcas->zr == 0) {
  if(hcas->id == HELI_A) {
    /* Maneuver up */
    wcom1 = hcas->manmax;
    if(coop)
      wcom2 = hcas->manmin;
    else
      wcom2 = hcas->o_w;
    mandir = 1;
  }
  else if(hcas->id == HELI_B) {
    /* Maneuver down */
    wcom1 = hcas->manmin;
    if(coop)
      wcom2 = hcas->manmax;
    else
      wcom2 = hcas->o_w;
    mandir = 0;
  }
}
}


/* Convert commanded body velocity to local frame */

Vx1 = n->tBL[0][0]*n->u + n->tBL[0][1]*n->v + n->tBL[0][2]*wcom1;
Vy1 = n->tBL[1][0]*n->u + n->tBL[1][1]*n->v + n->tBL[1][2]*wcom1;
Vz1 = n->tBL[2][0]*n->u + n->tBL[2][1]*n->v + n->tBL[2][2]*wcom1;

Vx2 = hcas->o_tBL[0][0]*hcas->o_u + hcas->o_tBL[0][1]*hcas->o_v
+ hcas->o_tBL[0][2]*wcom2;
Vy2 = hcas->o_tBL[1][0]*hcas->o_u + hcas->o_tBL[1][1]*hcas->o_v
+ hcas->o_tBL[1][2]*wcom2;
Vz2 = hcas->o_tBL[2][0]*hcas->o_u + hcas->o_tBL[2][1]*hcas->o_v
+ hcas->o_tBL[2][2]*wcom2;

/* Compute velocity relative to other helicopter */

hcas->Vxr = Vx1 - Vx2;
hcas->Vyr = Vy1 - Vy2;
hcas->Vzr = Vz1 - Vz2;

/* Calculate time and distance of closest
 approach during maneuver */

if(hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr
       + hcas->Vzr*hcas->Vzr != 0) {
tsepm = -(xrm*hcas->Vxr + yrm*hcas->Vyr + zrm*hcas->Vzr)/
         (hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr
         + hcas->Vzr* hcas->Vzr);
tsepm = MAX(0,tsepm);
sepm  = ComputeHelSep(tsepm,xrm,yrm,zrm,hcas->Vxr,
                    hcas->Vyr,hcas->Vzr);
}
else {
/* Helicopters not moving-- */
sepm = sqrt(xrm*xrm + yrm*yrm + zrm*zrm);
}
```

```
      if(sepm <= hcas->minsep) {
      if(mandir == 1) {
        /* Maneuver up */
        hcas->avoid = ON;
        hcas->w_avoid = hcas->manmax;
      }
      else if(mandir == 0 && n->pos[2] < hcas->minAvoidAlt) {
        /* Maneuver down */
        hcas->avoid = ON;
        hcas->w_avoid = hcas->manmin;
      }
      }
      else
      hcas->avoid = OFF;
    }
  }

  else if(hcas->avoid == ON){

    xrr = hcas->Vxr*0*hcas->tau + hcas->xr;
    yrr = hcas->Vyr*0*hcas->tau + hcas->yr;
    zrr = hcas->Vzr*0*hcas->tau + hcas->zr;

    /* Calculate desired return velocity--Guidance system */

    z_err = c->x_cmd[2] - n->pos[2];
    wcom1 = LIMIT(c->z2w*z_err, hcas->retmax, hcas->retmin );

    /* Determine other helicopter's return velocity */

    if(mandir == 0) {
      wcom2 = hcas->retmin;
    }
    else if(mandir == 1) {
      wcom2 = hcas->retmax;
    }
    if(!coop)
      wcom2 = hcas->o_w;


    /* Convert commanded body velocity to local frame */

    Vx1 = n->tBL[0][0]*n->u + n->tBL[0][1]*n->v + n->tBL[0][2]*wcom1;
    Vy1 = n->tBL[1][0]*n->u + n->tBL[1][1]*n->v + n->tBL[1][2]*wcom1;
    Vz1 = n->tBL[2][0]*n->u + n->tBL[2][1]*n->v + n->tBL[2][2]*wcom1;

    Vx2 = hcas->o_tBL[0][0]*hcas->o_u + hcas->o_tBL[0][1]*hcas->o_v
      + hcas->o_tBL[0][2]*wcom2;
    Vy2 = hcas->o_tBL[1][0]*hcas->o_u + hcas->o_tBL[1][1]*hcas->o_v
      + hcas->o_tBL[1][2]*wcom2;
    Vz2 = hcas->o_tBL[2][0]*hcas->o_u + hcas->o_tBL[2][1]*hcas->o_v
      + hcas->o_tBL[2][2]*wcom2;

    /* Compute velocity relative to other helicopter */

    hcas->Vxr = Vx1 - Vx2;
    hcas->Vyr = Vy1 - Vy2;
    hcas->Vzr = Vz1 - Vz2;
    if(hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr
             + hcas->Vzr*hcas->Vzr != 0) {
      tr = - (xrr*hcas->Vxr + yrr*hcas->Vyr + zrr*hcas->Vzr)/
             (hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr
             + hcas->Vzr*hcas->Vzr);
      tr = MAX(0,tr);

      sepr = ComputeHelSep(tr,xrr,yrr,zrr,hcas->Vxr,hcas->Vyr,
             hcas->Vzr);
    }
    else {
      /* Helicopters not moving-- */
      sepr = sqrt(xrr*xrr + yrr*yrr + zrr*zrr);
    }

    if( sepr >= hcas->minsep || tr >= hcas->trmax
      || ((n->pos[2] > hcas->minAvoidAlt) && (wcom1 > 0)) )
      hcas->avoid = OFF;
    }
}
```

A.3 Continuous Collision Avoidance Algorithm (from *on_board_hcas.c*)

```
void HCASUpdate2( struct aavnavigation_ref *n,
                  struct aavhcas_ref *hcas,
                  struct aavon_board_con_ref *c) {
  float Vx1,Vx2,Vxc,Vy1,Vy2,Vyc,Vz1,Vz2,Vzc,z_err,wcom1,qa,qb,qc;
  int coop;


  GetOtherAAVState( hcas );


  /* Convert body velocities (u,v,w) to local frame (Vx,Vy,Vz) */

  Vx1 = n->tBL[0][0]*n->u + n->tBL[0][1]*n->v + n->tBL[0][2]*n->w;
  Vy1 = n->tBL[1][0]*n->u + n->tBL[1][1]*n->v + n->tBL[1][2]*n->w;
  Vz1 = n->tBL[2][0]*n->u + n->tBL[2][1]*n->v + n->tBL[2][2]*n->w;

  Vx2 = hcas->o_tBL[0][0]*hcas->o_u + hcas->o_tBL[0][1]*hcas->o_v
      + hcas->o_tBL[0][2]*hcas->o_w;
  Vy2 = hcas->o_tBL[1][0]*hcas->o_u + hcas->o_tBL[1][1]*hcas->o_v
      + hcas->o_tBL[1][2]*hcas->o_w;
  Vz2 = hcas->o_tBL[2][0]*hcas->o_u + hcas->o_tBL[2][1]*hcas->o_v
      + hcas->o_tBL[2][2]*hcas->o_w;


  /* Calculate intended velocity--Guidance system */

  z_err = c->x_cmd[2] - n->pos[2];
  wcom1 = LIMIT(c->z2w*z_err, hcas->retmax, hcas->retmin );

  /* Convert intended body velocity to local frame */
```

```
Vxc = n->tBL[0][0]*n->u + n->tBL[0][1]*n->v + n->tBL[0][2]*wcom1;
Vyc = n->tBL[1][0]*n->u + n->tBL[1][1]*n->v + n->tBL[1][2]*wcom1;
Vzc = n->tBL[2][0]*n->u + n->tBL[2][1]*n->v + n->tBL[2][2]*wcom1;


/* Compute position and velocity relative to other helicopter */

if( !(hcas->delayComp) ) {
  hcas->xr = n->pos[0] - hcas->o_pos[0];
  hcas->yr = n->pos[1] - hcas->o_pos[1];
  hcas->zr = n->pos[2] - hcas->o_pos[2];
}
else {
  /* Compensate for communication time delay */
  hcas->xr = n->pos[0] - (hcas->o_pos[0] + Vx2*hcas->tdEst);
  hcas->yr = n->pos[1] - (hcas->o_pos[1] + Vy2*hcas->tdEst);
  hcas->zr = n->pos[2] - (hcas->o_pos[2] + Vz2*hcas->tdEst);
}

hcas->Vxr = Vx1 - Vx2;
hcas->Vyr = Vy1 - Vy2;
hcas->Vzr = Vz1 - Vz2;

hcas->Vxrc = Vxc - Vx2;
hcas->Vyrc = Vyc - Vy2;
hcas->Vzrc = Vzc - Vz2;


/* Compute current estimated separation distance */
hcas->separation = sqrt(hcas->xr*hcas->xr + hcas->yr*hcas->yr
                                        + hcas->zr*hcas->zr);


/* Calculate time and distance of closest approach -- Current trajectory*/

if(hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr + hcas->Vzr*hcas->Vzr != 0) {
  hcas->tstar = -(hcas->xr*hcas->Vxr + hcas->yr*hcas->Vyr
           + hcas->zr*hcas->Vzr)/
     (hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr + hcas->Vzr*hcas->Vzr);
  hcas->tstar = MAX(0,hcas->tstar);
  hcas->dstar  = ComputeHelSep(hcas->tstar,hcas->xr,hcas->yr,hcas->zr,
           hcas->Vxr,hcas->Vyr,hcas->Vzr);
}
else {
  /* Helicopters not moving-- */
  hcas->dstar = hcas->separation;
}



/* Calculate time and distance of closest approach -- Guidance trajectory*/

if(hcas->Vxrc*hcas->Vxrc + hcas->Vyrc*hcas->Vyrc
           + hcas->Vzrc*hcas->Vzrc != 0) {
  hcas->tstarc = -(hcas->xr*hcas->Vxrc + hcas->yr*hcas->Vyrc
           + hcas->zr*hcas->Vzrc)/
                (hcas->Vxrc*hcas->Vxrc + hcas->Vyrc*hcas->Vyrc + hcas-
>Vzrc*hcas->Vzrc);
  hcas->tstarc = MAX(0,hcas->tstarc);
```

```
    hcas->dstarc  = ComputeHelSep(hcas->tstarc,hcas->xr,hcas->yr,hcas->zr,
             hcas->Vxrc,hcas->Vyrc,hcas->Vzrc);
  }
  else {
    /* Helicopters not moving-- */
    hcas->dstarc = hcas->separation;
  }


  /* If current trajectory or intended trajectory will cause
     a collision, compute collision-avoidance maneuver */

  if((hcas->dstarc <= hcas->minsep && hcas->tstarc > 0
             && hcas->tstarc < hcas->tstarmin)
             ||(hcas->dstar <= hcas->minsep - hcas->hystWidth
             && hcas->tstar > 0 && hcas->tstar < hcas->tstarmin)
             ||(hcas->dstar <= hcas->minsep + hcas->hystWidth
             && hcas->dstar >= hcas->minsep - hcas->hystWidth
             && hcas->avoid && hcas->tstar > 0
             && hcas->tstar < hcas->tstarmin) ||(hcas->forceAvoid)) {


    /* Relative vertical positions at time of closest approach */

    hcas->zrstar = hcas->Vzr*((hcas->tstar > 0) ? hcas->tstar :
                                             hcas->tstarc) + hcas->zr;


    hcas->D2 = hcas->xr*hcas->xr + hcas->yr*hcas->yr + hcas->zr*hcas->zr
       - hcas->minsep*hcas->minsep;
    qa = hcas->zr*hcas->zr - hcas->D2;
    qb = hcas->Vxr*hcas->xr + hcas->Vyr*hcas->yr;
    qc = hcas->Vxr*hcas->Vxr + hcas->Vyr*hcas->Vyr;


    /* Compute collision avoidance maneuever */

    if( hcas->D2*(qa*qc + qb*qb) > 0 ) {

      hcas->Vzrcomm1 = (-hcas->zr*qb + sqrt(hcas->D2*(qa*qc + qb*qb)))/qa;
      hcas->Vzrcomm2 = (-hcas->zr*qb - sqrt(hcas->D2*(qa*qc + qb*qb)))/qa;

      if(coop) {
             hcas->w_avoid1 = 0.5*hcas->Vzrcomm1 + Vz2;
             hcas->w_avoid2 = 0.5*hcas->Vzrcomm2 + Vz2;
      }
      else {
             hcas->w_avoid1 = hcas->Vzrcomm1 + Vz2;
             hcas->w_avoid2 = hcas->Vzrcomm2 + Vz2;
      }

      if(hcas->zrstar < 0) {
             hcas->K = -qb/qa + (hcas->zr*sqrt(qa*qc + qb*qb))/
                          (qa*sqrt(hcas->D2));
             hcas->Vzrcomm = hcas->Vzrcomm1;
             hcas->w_avoid = hcas->w_avoid1;
             hcas->avoid = ON;
      }
      else if( hcas->zrstar > 0) {
            hcas->K = -qb/qa - (hcas->zr*sqrt(qa*qc + qb*qb))/
                        (qa*sqrt(hcas->D2));
            hcas->Vzrcomm = hcas->Vzrcomm2;
            hcas->w_avoid = hcas->w_avoid2;
            hcas->avoid = ON;
    }
    else {
            hcas->avoid = OFF;
            hcas->w_avoid = 0.0;
    }

    /*if(hcas->id == HELI_A) {
            hcas->Vzrcomm = hcas->w_avoid1 - Vz2;
            hcas->w_avoid = hcas->w_avoid1;
            hcas->avoid = ON;
    }
    else {
            hcas->Vzrcomm = hcas->w_avoid2 - Vz2;
            hcas->w_avoid = hcas->w_avoid2;
            hcas->avoid = ON;
    }*/

    /* Compute eigenvalues of CAS */
    if(hcas->tau != 0) {
            if((1/(hcas->tau*hcas->tau) + 2*hcas->K) >= 0) {
              hcas->lambda1[0] = -1/hcas->tau + sqrt(1/(hcas->tau*hcas->tau)
                        + 2*hcas->K);
              hcas->lambda1[1] = 0;
              hcas->lambda2[0] = -1/hcas->tau - sqrt(1/(hcas->tau*hcas->tau)
                         + 2*hcas->K);
              hcas->lambda2[1] = 0;
            }
            else {
              hcas->lambda1[0] = -1/hcas->tau;
              hcas->lambda1[1] = sqrt(-1/(hcas->tau*hcas->tau) - 2*hcas->K);
              hcas->lambda2[0] = -1/hcas->tau;
              hcas->lambda2[1] = -sqrt(-1/(hcas->tau*hcas->tau) - 2*hcas->K);
            }

    }
  }
  else {
    if(hcas->D2 < 0) {
            /* Helicopters have already violated minimum separation */
            /*             (they're inside the sphere of separation) */

            if(hcas->zrstar < 0) {
              hcas->Vzrcomm = hcas->manmax - Vz2;
              hcas->w_avoid = hcas->manmax;
              hcas->avoid = ON;
            }
            else if(hcas->zrstar > 0) {
              hcas->Vzrcomm = hcas->manmin - Vz2;
              hcas->w_avoid = hcas->manmin;
              hcas->avoid = ON;
            }
            else {
              hcas->avoid = OFF;
              hcas->w_avoid = 0.0;
```

Appendix B

Closed-loop Stability Matlab Script

When it was not possible to use analytical methods to find the eigevalues of the closed-loop system, the Matlab script *acas.m* performed the stability analysis over a discretized state space.

B.1 Matlab Script *acas.m*

```
% acas.m

% Chris Sanders
% Draper Laboratory
% 20 December 1997

clear all
%close all

% Control system time constants
tv = 0.4;

% Minimum allowed separation distance
d = 12;

% All state variables are in relative coordinates
% (AAV #2 relative to AAV #1)
Vx = -2;
Vy = 0;
Vz = 1;

inc = 0.1;

param = 'T';
y = d;

for i = 1:21
x = 0.00005 + (i-1)*.25;

for j = 1:51
if param == 'y'
   y = -d+0.0005 + (j-1)*inc;
elseif param == 'T'
   T = (j-1)*inc;
else
   x = 0.00005 + (j-1)*inc;
end
```

```
z = -sqrt(d^2 - y^2);

if (x^2 == z^2) | (x == 0)
   x = x + 0.0001;
end

tcp = -(Vx*x+Vy*y+Vz*z)/(Vx^2+Vy^2+Vz^2);
d0 = sqrt(x^2+y^2+z^2+tcp^2*(Vx^2+Vy^2+Vz^2)+2*tcp*(Vx*x+Vy*y+Vz*z));
if (abs(d0) - abs(d)) > 0.0001
   d0 = d0
end

Dsq = x^2 + y^2 + z^2 - d^2;
Esq = z^2 - Dsq;
Qsq = Esq*(Vx^2 + Vy^2) + (Vx*x + Vy*y)^2;

% Partial derivatives relative to x-direction
I = (-Vx*z*Esq - 2*x*z*(Vx*x + Vy*y))/Esq^2 + x*Qsq/(Esq*sqrt(Dsq*Qsq)) +
Dsq*(Vx*Vy*y - Vy^2*x)/(Esq*sqrt(Dsq*Qsq)) + 2*x*sqrt(Dsq*Qsq)/Esq^2;
L = (1/Esq)*(-x*z + Dsq*(Vx*Esq + Vx*x^2 + Vy*x*y)/(sqrt(Dsq*Qsq)));

% Partial derivatives relative to y-direction
J = (-Vy*z*Esq - 2*y*z*(Vx*x + Vy*y))/Esq^2 + y*Qsq/(Esq*sqrt(Dsq*Qsq)) +
Dsq*(Vx*Vy*x - Vx^2*y)/(Esq*sqrt(Dsq*Qsq)) + 2*y*sqrt(Dsq*Qsq)/Esq^2;
M = (1/Esq)*(-y*z + Dsq*(Vy*Esq + Vy*y^2 + Vx*x*y)/(sqrt(Dsq*Qsq)));

% Partial derivatives relative to z-direction
K = -(Vx*x + Vy*y)/Esq + z*sqrt(Qsq)/(Esq*sqrt(Dsq));
if K > 0.000001
   K
end
N = -1;

% Assemble Jacobian matrix
Acl = zeros(12);
Acl(1,2) = 1;
Acl(3,4) = 1;
Acl(5,6) = 1;
Acl(7,8) = 1;
Acl(9,10) = 1;
Acl(11,12) = 1;
Acl(6,1) = I/tv;
Acl(6,2) = L/tv;
Acl(6,3) = J/tv;
Acl(6,4) = M/tv;
Acl(6,5) = K/tv;
Acl(6,6) = N/tv;
Acl(6,7) = -I/tv;
Acl(6,8) = -L/tv;
Acl(6,9) = -J/tv;
Acl(6,10) = -M/tv;
Acl(6,11) = -K/tv;
Acl(6,12) = -N/tv;
Acl(12,1) = -I/tv;
Acl(12,2) = -L/tv;
Acl(12,3) = -J/tv;
Acl(12,4) = -M/tv;
Acl(12,5) = -K/tv;
Acl(12,6) = -N/tv;
Acl(12,7) = I/tv;
Acl(12,8) = L/tv;
Acl(12,9) = J/tv;
Acl(12,10) = M/tv;
Acl(12,11) = K/tv;
Acl(12,12) = N/tv;

% Assemble system matrices
A = [0 1 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 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 0 0 0 0 0 0;
      0 0 0 0 0 -1/tv 0 0 0 0 0 0;
      0 0 0 0 0 0 0 1 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 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;
      0 0 0 0 0 0 0 0 0 0 0 -1/tv];
BG = A - Acl;
B = [0 0 0 0 0 1/tv 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 1/tv]';
% Calculate feedback gain matrix
G = B\BG;

% Develop subsystem 1 model

A1 = A(1:6,1:6);
B1 = B(1:6,1);
G11 = G(1,1:6);
G12 = G(1,7:12);
C1 = eye(6);
D1 = zeros(6);

% Add time delay 1
[alpha bravo charlie delta] = pade(T,1);
Ad = alpha*eye(6);
Bd = bravo*eye(6);
Cd = charlie*eye(6);
Dd = delta*eye(6);
[A1c,B1c,C1c,D1c] = series(A1-B1*G11,-B1*G12,C1,D1,Ad,Bd,Cd,Dd);

% Develop subsystem 2 model

A2 = A(7:12,7:12);
B2 = B(7:12,2);
G21 = G(2,1:6);
G22 = G(2,7:12);
C2 = eye(6);
D2 = zeros(6);

% Add time delay 2
[A2c,B2c,C2c,D2c] = series(A2-B2*G22,-B2*G21,C2,D2,Ad,Bd,Cd,Dd);

[At,Bt,Ct,Dt] = feedback(A1c,B1c,C1c,D1c,A2c,B2c,C2c,D2c,1);
[Atest,Btest,Ctest,Dtest] = feedback(A1-B1*G11,-B1*G12,C1,D1,A2-B2*G22,-
B2*G21,C2,D2,1);
Eig(j) = max(real(eig(At)));
```

```
end
Tp = 0:inc/12:22*inc/12;
Ind = find(Eig < 0.000001);
Ts(i) = (max(Ind)-1)*inc;
X(i) = abs(x/Vx);


%plot(Tp,Eig)
%hold on
%axis([0 2 0 1])
%pause

end
plot(X,Ts)
```