Proximity Operations of a Miniature Inspector Satellite Using Emulated Computer Vision

by

Christine Marie Edwards

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

June 2008

Author .....
Department of Aeronautics and Astronautics
June 6, 2008

Certified by .....
David W. Miller
Professor
Thesis Supervisor

Accepted by .....
Prof. David L. Darmofal
Associate Department Head
Chairman, Committee on Graduate Students

Proximity Operations of a Miniature Inspector Satellite Using Emulated Computer Vision

by

Christine Marie Edwards

Submitted to the Department of Aeronautics and Astronautics
on May 27, 2008, in partial fulfillment of the requirements for the degree of
Master of Science in Aerospace Engineering

Abstract

If a micrometeoroid, a piece of space junk, launch debris, or a major system failure impacts the Crew Exploration Vehicle (CEV), it can cause life-threatening damage. Past International Space Station (ISS) and Space Shuttle repair missions have shown that inspection of a damaged system is crucial for planning the EVA to repair it. To assist the CEV team with inspection and contingency planning, an inspector satellite can be an essential tool. This thesis presents the idea of using a miniature satellite to inspect the CEV for damage while flying in formation. In this research, the satellite test bed SPHERES has been used to develop, demonstrate, and flight-test an inspector-satellite operations design and controller. The design utilizes an autonomous control algorithm that combines Linear Quadratic Regulator (LQR) and Artificial Potential Field (APF) control. This controller is designed to navigate through waypoints, follow the contours of an inspected spacecraft's surface, avoid obstacles, operate in real-time onboard the inspector satellite, work with or without a-priori knowledge of the inspected spacecraft's geometry, interface with a computer-vision system, and handle the loss of computer-vision information. Since the SPHERES camera system is currently under development, computer vision data was emulated using the current SPHERES global metrology. Results from simulations and successful flight tests aboard the ISS are discussed.

Thesis Supervisor: David W. Miller
Title: Professor

Acknowledgments

This work is dedicated to my parents, Bill and Debbie Edwards, and to my fiancé, Andrzej Stewart. Thank you for the love, support, and inspiration that you have given to me. The spirit with which you live your life inspires those around you. Also, thank you to my sister Lindsey and brothers William and Andrew. I have been blessed to have such a good family.

I would like to give thanks to Professor Dave Miller for his suggestions and questions that have helped me to improve as an engineer and to develop this work into a meaningful contribution to engineering research.

Thank you to Maj. Shawn McCamish, Dr. Marcello Romano, Dr. Riccardo Bevilacqua, and Dr. Simon Nolet for the collaboration to flight-test the NPS controller on SPHERES. Also, I would like to give thanks to the SPHERES team, including Dr. Alvar Saenz-Otero, Christophe Mandy, Swati Mohan, Amer Fejzic, Jacob Katz, Matthew Jeffrey, Brent Tweddle, Caley Burke, Georges Aoude, and John Merke.

Thank you to the astronauts who have performed SPHERES test sessions on the ISS, and thank you to the NASA and DoD teams who make these test sessions possible. I would like to give a special thanks to astronaut Dan Tani, who performed the flight tests discussed in this thesis.

Additionally, thank you to my friends and extended family for their encouragement and support over the years.

Lastly, thanks to God for creating this wonderful universe and giving us the drive to explore and discover.

Contents

List of Figures

1 Introduction

1.1 Motivation

If a micrometeoroid, a piece of space junk, or launch debris impacted a manned spacecraft, it could cause life-threatening damage. Precious consumables could start spewing into space. Or, hot atmospheric gases could stream into a hole during reentry, potentially causing the spacecraft to disintegrate, killing the crew. If damage is suspected, then inspection is needed.

Currently, the inspection methods for manned spacecraft are limited. A spacecraft arm can search for tile and RCC-panel damage over parts of space shuttle. Also, astronauts in the International Space Station (ISS) can photograph the space shuttle belly before it comes back to Earth. If damage is found, an EVA can perform close-up inspection and repair. However, an arm has limited view, the ISS will not always be the destination, and an EVA is dangerous and consumes time and resources. Additionally, the new Crew Exploration Vehicle (CEV) will have extensive missions before coming home, increasing the chances of being significantly damaged by micrometeoroids. NASA has developed a means of detecting potential micrometeoroid impact for the CEV. If an acoustic sensor detects a potential impact and its location cannot be observed from the vehicle, an EVA or another kind of inspection must be performed to determine the extent of the damage. We are entering a new age of space exploration, voyaging to the Moon, Mars, and beyond. Robust inspection methods must be developed and carried along on the journey.

To inspect the CEV for damage, a micro satellite could detach itself from a docking station on the CEV, fly in formation along the vehicle's outer wall to the location of concern, then transmit motion video for real-time analysis or photographs for later study. After the satellite achieved its task, it could return to its station, autonomously dock, and refuel. All the commodities the satellite would need are onboard the command module. The docking port would provide power, compressed gas for propulsion, and data transfer. This inspection concept has received preliminary interest from the Lockheed Martin CEV team.

Studying the damage of spacecraft using a small orbiting satellite would have been very useful in the past and could be invaluable in the future. The extent of the damage to the Apollo

XIII capsule could have been assessed earlier in the mission. Also, the damage to the Columbia would have been detected. This inspection could be performed before committing to three major legs of the upcoming moon missions. The satellite could inspect the Earth Departure Stage (EDS) prior to leaving Earth orbit, the Lunar Surface Ascent Module (LSAM) prior to committing to lunar descent, and the CEV prior to reentering Earth's atmosphere.

A satellite test bed called Synchronized Position Hold Engage and Reorient Experimental Satellites (SPHERES) can be used to develop and test controls algorithms for satellite inspection. Developed by the MIT Space System Laboratory (SSL), SPHERES has been used to test formation flight and autonomous docking on a 2-D laboratory platform, in NASA's Reduced Gravity Airplane, and in the ISS [1-3]. There are currently three SPHERES in the ISS. This testing in the ISS offers four major benefits for using SPHERES. These satellites are man-rated, they have already been flown in close proximity to another man-rated spacecraft (ISS), and the astronauts are comfortable using them. Additionally, the ISS is a risk-tolerant environment for testing the 3-D maneuvers of a new control algorithm. A problem with a developing algorithm can be discovered without loss of the satellite. Thus, SPHERES can be used for testing inspection-control algorithms and potentially as an inspector satellite for the CEV. Figure 1-1 illustrates SPHERE satellites taking images of the CEV surface.

The goal of the research in this thesis is to design, implement, and flight-test a controller for inspection operations using the SPHERES laboratory. This controller is designed so that it will expand the inspection operations modes that have been developed in the past by other research groups.

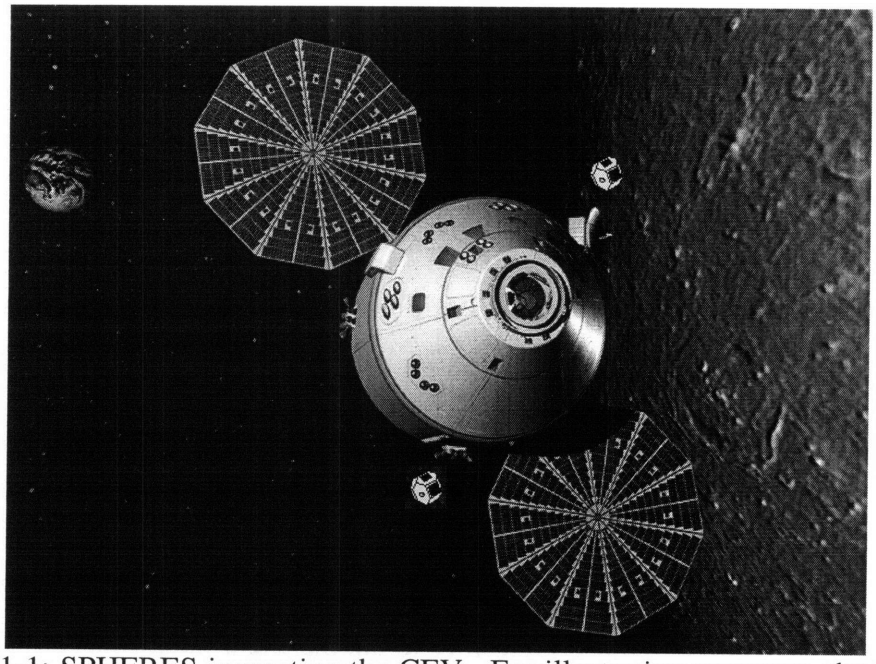


Figure 1-1: SPHERES inspecting the CEV. For illustration purposes, the SPHERES are not shown to scale.

1.2 Background

In the last decade, research groups have developed several satellite-inspection operations. After the Columbia accident, the Orbiter Boom Sensor System (OBSS) was designed to inspect the space shuttle for launch damage. Looking at the operations and control-system design for this boom-inspection system is useful for determining which aspects of the operations design have worked well for NASA. Additionally, a couple of teams have developed operations designs for inspector satellites. These teams are the NASA Autonomous Extravehicular Robotic Camera (AerCam) groups and Drs. Roger and McInnes of the University of Glasgow. AerCam Sprint is a 14-inch-diameter satellite that was flight-tested on the Space Shuttle in 1997 [9]. Later, the system was redesigned with a 7.5-inch diameter, and renamed the Mini Aercam [10]. The operations designs of the OBSS, AerCam, and the Roger & McInnes Free-Flyer will now be discussed.

For the inspection of the space shuttle, the OBSS was developed. The OBSS system consists of a 50-foot boom that attaches to the end of the space shuttle's arm. This boom extends the length of the arm so that it can reach the underside of the space shuttle. Also, camera and laser sensors are located at the tip of the boom to inspect the space shuttle's surfaces. These sensors are located on a pan-and-tilt mechanism that can adjust their viewpoints. The operations mode

for inspecting the space shuttle's under-surface is named the Auto Control Mode [11]. During inspection operations in this mode, the arm is commanded to maneuver its tip through a series of user-defined waypoints. The arm automatically adjusts its configuration so that the tip with the inspection sensors moves along a straight line from one waypoint to another. When a series of these waypoints are combined and commanded in sequence, they are called a survey trajectory [12]. Figure 1-2 shows the beginning and ending locations of the OBSS for one survey trajectory [12]. Note how the waypoints for this survey trajectory are placed at equal distances away from the space-shuttle surface. When the space shuttle is undocked from the ISS, the OBSS is commanded to fly along waypoints that are at a height of 5 feet from the space-shuttle surface. When the space shuttle is docked to the ISS, the waypoints are 2 feet from its surface [11]. To perform a complete inspection operation, the arm maneuvers through several of these survey trajectories until its sensors have imaged the entire leading edge of the wing. This inspection operation has been successfully implemented into the space-shuttle missions [13]. However, this operation will not be useful for the CEV, because the current CEV design does not have an arm.

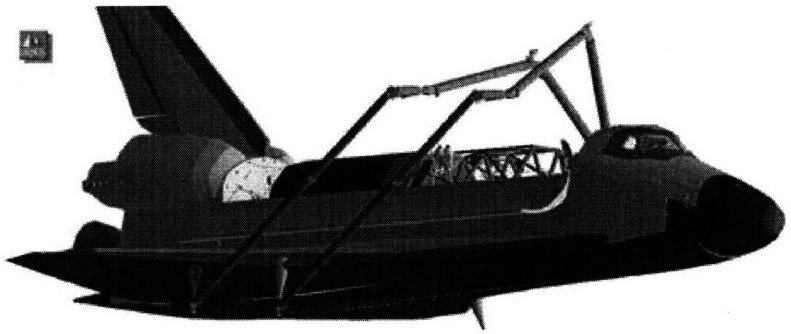


Figure 1-2: Beginning and ending waypoints of the OBSS during one survey trajectory [12].

Inspection operations for an inspector satellite are different from those of an inspection boom, because a satellite free-flies in formation with the inspected vehicle, while the boom is attached to the vehicle. The AerCam teams have developed and tested operations modes for inspecting the ISS. One mode was simulated by the Mini Aercam team in Reference [10]. In this mode, the satellite flies in straight paths through waypoints. Figure 1-3 shows this mode, in which the AerCam flies through a series of waypoints in front of the Space Shuttle's wing. To navigate through these waypoints, the system is designed to use a relative GPS system when in

low earth orbit (LEO), and the AutoTRACK Computer Vision System if the satellite is not in LEO [10].

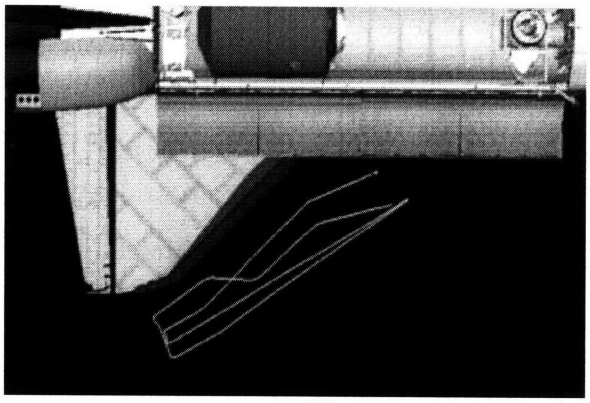


Figure 1-3: Simulated path of the AerCam using waypoint algorithm [10]

Also, a semi-autonomous mode for the AerCam was developed by Choset et. all [14]. In this mode, the user defines a goal point, and the AerCam autonomously navigates to that point. The AerCam utilizes a generalized-Voronoi-graph (GVG) algorithm that creates a roadmap based on a known model of the ISS, and it chooses the path that will take the inspector to the goal point [14]. This original GVG path curves around the ISS. The curved GVG path is then simplified into a series of straight-line trajectories, defined by waypoints along the GVG path. An additional algorithm adjusts the waypoints to ensure that the straight-line trajectories stay above a minimum distance from the ISS surfaces. This algorithm was implemented into a simulation in Reference [14]. Figure 1-4 shows a trajectory that has been defined by this AerCam path planner. This is a viable path-planning method for an operation that transfers the inspector to a point of concern. However, the trajectories that it produces do not provide views that smoothly follow the surface of the inspected spacecraft, like those currently provided by the OBSS in space shuttle operations.

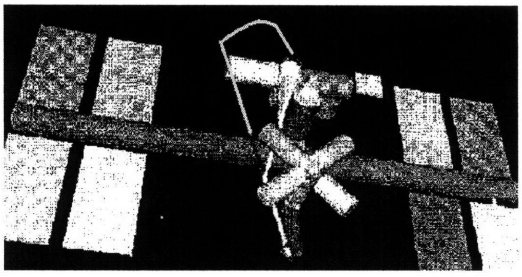


Figure 1-4: Simulated path of the AerCam using GVG algorithm [14]

Other inspector-satellite operation modes have been designed by Dr. Roger and Dr. McInnes. These modes were also developed for inspection of the ISS, and were designed to utilize relative GPS. The first mode takes the satellite from its docking port to a point close to the location of concern. This mode is called the Transfer to Observation Point. For this mode, the inspector follows a trajectory along an Ellipse of Safety [9]. This ellipse is designed so that if the inspector system fails during the maneuver, the inspector cannot drift into the ISS. According to Reference [9], the transfer is designed to take the satellite at least 200 meters away from the ISS so that the satellite can receive accurate GPS information without interference from the ISS. Figure 1-5 shows this operation mode in which the satellite flies out 200 meters, and then returns to a position near the location of concern via the Ellipse of Safety.

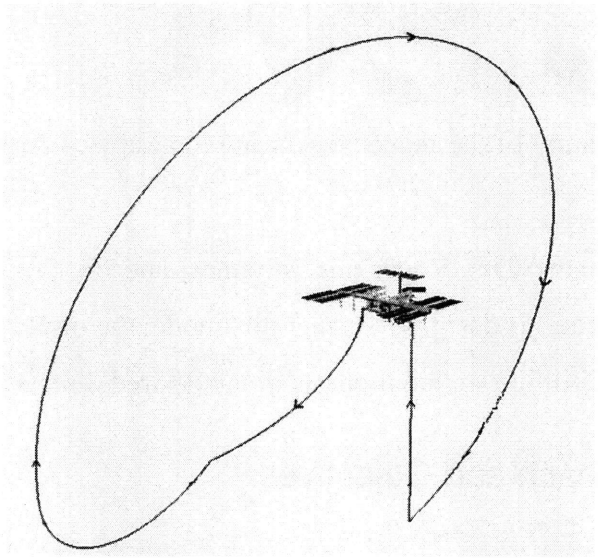


Figure 1-5: Simulation of the inspector trajectory around the ISS during the Transfer-to-Observation-Point mode [9].

For the second operation mode, Roger and McInnes developed a path planner that utilizes Laplace potential functions. This second mode is called Laplace Artificial Potential Guidance. The Laplace potential fields bend a path around obstacles. In this case, a predefined model of the ISS is used to create the Laplace potentials. Using the Laplace potentials, the inspector satellite plans a path around the complex ISS shapes to the location of concern. The velocities are kept low to avoid damage if the inspector were to malfunction. Figure 1-6 shows a simulated path using the Laplace potential fields. The satellite only fires its thrusters periodically during the maneuver to keep fuel usage low, which causes the scalloped shape of the trajectory. One usual problem with potential-field obstacle avoidance is that the satellite can get stuck in a local

minimum. However, a benefit for using the Laplace potential fields is that they contain no local minima. Unfortunately, they are very computationally intensive, and are thus difficult to implement in real-time satellite operations. According to Reference [9], if this path-planner is implemented on a satellite system, the paths will likely be calculated on ground or ISS computers, and then uploaded to the inspector.

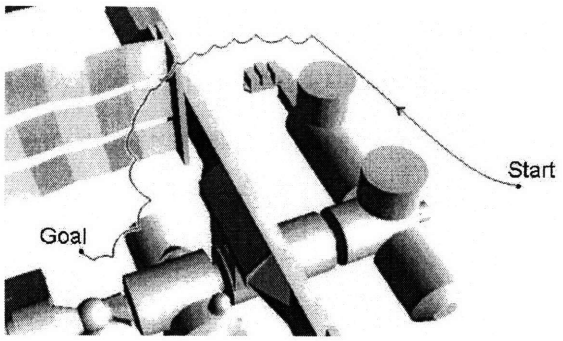


Figure 1-6: Simulation of the maneuver during the Laplace Artificial Potential Guidance mode [9]

Lastly, the AerCam teams, Drs. Roger and McInnes, and the OBSS all included a station-keeping mode in their operations designs. During this mode, the inspector holds its position, and the operator can adjust its attitude so that it points in the desired direction for inspection.

1.3 Research Approach and Objectives

As an initial implementation of inspection operations and testing using SPHERES, a controls algorithm that is useful for CEV inspection has been developed, implemented, and flight-tested on a SPHERE satellite. This algorithm is designed to add another useful inspection operations mode to those that have been developed in previous research. For the current inspection operations of the Space Shuttle, the OBSS has the ability to smoothly follow the surface of the space shuttle at a constant height. This useful aspect of the OBSS operations is not present in the operations modes for the inspector satellites that have been developed thus far. Thus, an objective for this research is to develop a satellite controller that provides inspection operations that are similar to those successfully performed on the Space Shuttle using the OBSS. A satellite controller is developed that provides smooth-wall-following and autonomous navigation through waypoints.

For the application of inspecting the CEV, the CEV has a surface with greater curvature than the underside of the Space Shuttle. Also, it has solar panels and antennas that protrude from its surface. With obstacle avoidance, the inspector would then be able to navigate around protrusions that are between waypoints. Thus, another objective for the controller is to provide obstacle avoidance.

Additionally, the SPHERE satellites and other potential inspector systems have limited computational ability. The inspection controller that is developed in this research must be computationally inexpensive enough to be usable on SPHERES.

Also, the SPHERES team is currently developing a computer-vision system for its satellites. The controller discussed in this research will be designed to interface with this computer-vision system. Because of this parallel development, and because relative GPS sensors would not be useful in moon missions, the primary sensor system for this inspection controller will be computer-vision. The computer-vision information that is used by the control algorithm will be emulated, because the actual system is still under development.

Finally, to design the control algorithm so that it can be applied to inspection operations of spacecraft other than the CEV. The controller can be designed so that it can utilize, but does not require, an a-priori model of the inspected spacecraft. Instead, the controller can adapt its knowledge of the inspected-spacecraft's shape as it flies.

As a summary, the objectives for this research are:

1. To develop a satellite controller that provides the following operations aspects that have been successfully used in Space Shuttle inspection:
   a. Autonomous navigation through waypoints
   b. Smooth wall following
2. To develop this controller so that it avoids obstacles
3. To develop the controller to interface with a computer-vision system
4. To design the controller so that it can use, but does not require, a-priori knowledge of the inspected-spacecraft's shape, and can update its model of the inspected spacecraft as it flies.
5. To design the controller such that it is computationally inexpensive enough to be implemented on SPHERES and can operate in real-time
6. To implement the controller on SPHERES hardware

7. To flight-test the controller on the ISS using the SPHERES laboratory
8. To demonstrate SPHERES as a laboratory for testing new inspection algorithms
9. To demonstrate SPHERES as a micro-satellite tool for inspecting the surface of NASA's CEV.

In the following sections, a satellite controller design will be discussed that satisfies these objectives. The basis of the controller is a path planner that can utilize information from a computer vision system to navigate around obstacles. A computer-vision system can identify feature points along the surface of the inspected spacecraft. Then, placing exclusion zones around these feature points, can create a navigation mesh, as shown in Figure 1-7. These exclusion zones will be referred to as NavZones, and the inspected spacecraft will be called the target. This mesh can produce a smooth virtual surface that is located a pre-determined distance above the target spacecraft, along which the spacecraft can maneuver.

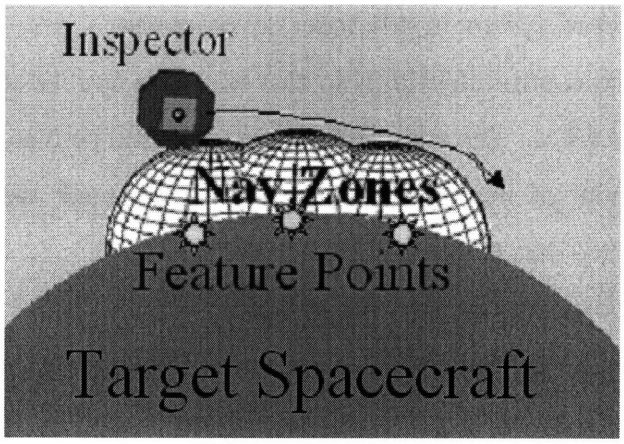


Figure 1-7: Illustration of inspector maneuvering around mesh of NavZones.

A path planner can then be utilized to maneuver around this navigation mesh. One path planner that satisfies the research objectives above is a combination of a Linear Quadratic Regulator (LQR) and Artificial Potential Functions (APF) developed by Dr. McCamish [4-7]. The LQR controller plans a maneuver so that it minimizes an energy cost function, thus providing fuel-efficiency for the controller. And the APF provides robust obstacle avoidance. When the LQR and APF are combined together, they provide smooth trajectories around obstacles. Additionally, the LQR and APF algorithms can be implemented in a computationally inexpensive manner. Therefore, they can be implemented on the SPHERE satellites. In this

research, Dr. McCamish's controller was redesigned for inspection operations. This redesigned controller will be referred to as the NavZ.

1.4 Outline

This thesis discusses the development of the NavZ controller, its implementation on the SPHERE satellites, and flight-test results. The thesis structure has the following organization:

- Chapter 2 briefly describes the SPHERES testbed, the inspection control architecture, and several inspection operations modes that can utilize the NavZ.
- Chapter 3 discusses the development of the NavZ controller, beginning with Dr. McCamish's LQR and APF controller that was implemented on SPHERES as a baseline, then the redesign for the application of inspection maneuvers. The chapter includes ISS flight-test results for the baseline and NavZ controllers.
- Chapter 4 discusses the continued improvements to the NavZ controller after its first flight test. These improvements include a pointing algorithm and the implementation of waypoints.
- Chapter 5 explains how the computer-vision system was emulated, explores the accuracy of computer-vision information, and shows simulation results of how the NavZ can adapt when a feature point is not detected.
- Lastly, Chapter 6 discusses the design for two final ISS tests of the NavZ algorithm, and shows simulation results of maneuvers around a full-scale CEV.

2 SPHERES Inspector Satellite

Before giving the details of the development of the NavZ controller, the SPHERES testbed will first be discussed. Also, this chapter gives an overview of the controls and operations frameworks for an inspector satellite using the NavZ.

2.1 SPHERES Testbed

The MIT Space Systems Laboratory (SSL) has developed a miniature-satellite testbed to provide researchers with an experimental laboratory for testing formation flight, rendezvous, docking, and autonomy algorithms. The SPHERES facility [1-3] consists of three 20-centimeter-diameter miniature satellites. One of these satellites is shown in Figure 2-1. Each SPHERE satellite can autonomously control its relative positions and orientations in a six degree-of-freedom (6-DOF) environment. The testbed is primarily designed to operate inside the ISS, but it can also operate onboard NASA's Reduced Gravity Aircraft and in 2-D environments on flat floors. The flat floors that have been utilized for SPHERES testing include one at the NASA Marshall Space Flight Center Flight Robotics Laboratory and the SSL laboratory air table [16-19]. Figure 2-2 shows the SPHERES test volume inside the ISS, with the coordinates that are used for ISS testing. By operating inside the ISS, SPHERES tests algorithms in a microgravity environment while providing direct recovery of the satellite when real or simulated GN&C failures occur. Therefore, SPHERES provides a fault tolerant environment to test advanced algorithms and operations.

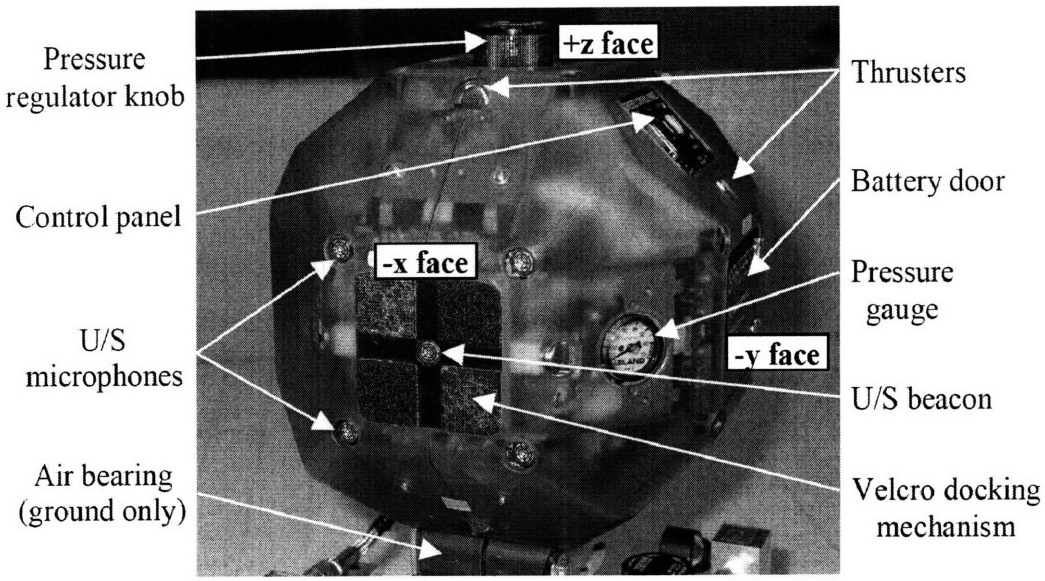


Figure 2-1. A SPHERE satellite and its visible components. [15]

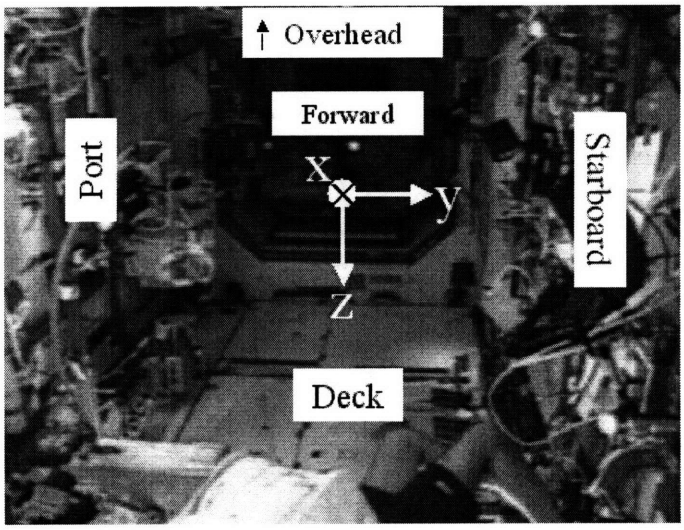


Figure 2-2. SPHERES ISS test volume and coordinates.

Each SPHERE satellite is identical, and contains most of the subsystems of a standard satellite. It has self-contained power, propulsion, computer, communication, and navigation systems.

- **Power:** The power system consists of AA batteries.
- **Propulsion:** SPHERES move in the testing environment by emitting small jets of CO_2 gas using pulse-width modulated ON-OFF micro-valve thrusters.
- **Computer:** The computer has a real-time satellite command structure. A control cycle is a section of time in the command structure in which the controller code is run by the

satellite computer. The control cycle typically runs at a frequency of one hertz, but this is adjustable.

- **Communication:** The satellites use two low-power wireless (RF) links to communicate with each other and a laptop computer.
- **Navigation:** For navigation, the SPHERES use ultrasound. In the close distances that are seen inside the ISS module, ultrasound is useful for triangulating satellite position. Ultrasonic signals are emitted from beacon transmitters that are mounted at known locations around the ISS test volume. Figure 2-3 illustrates the locations of the beacons. Microphone receivers on the satellite detect these signals.

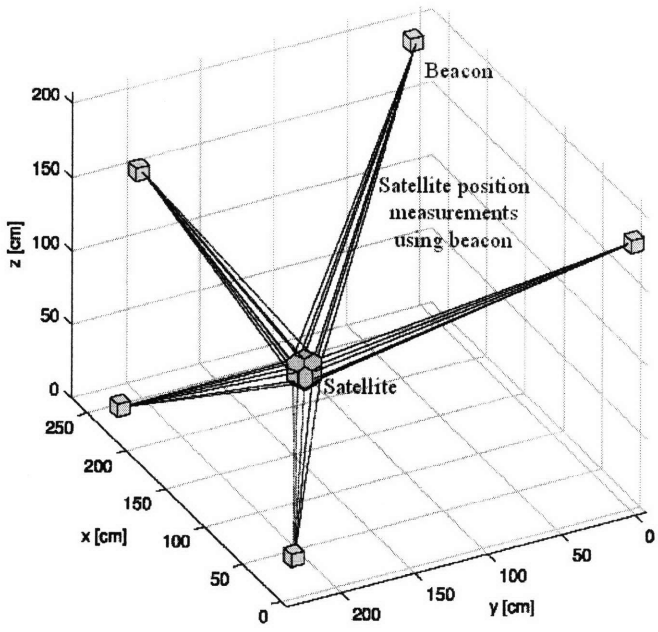


Figure 2-3: Beacon locations and ranges measured by a SPHERE [28].

With five beacons mounted on the walls and 24 microphones on each satellite, there is a potential of 120 measurements per satellite during a control cycle. This data is converted to range data using the known speed of sound in the ISS. Ultrasound measurements combined with data from three gyroscopes are processed using the navigation software module to compute a 6-DOF state solution. The resulting precision of the estimates is two millimeters in position and approximately two degrees in attitude in most of the testing volume [15]. When using a control cycle frequency of one hertz, the satellites can maintain a position uncertainty of ±2 cm inside of the testing volume, even when neglecting the orbital dynamics.

The onboard guidance, navigation, and control (GN&C) software is written in C. It consists of a series of modules, each accomplishing a specific task (estimation, control, thruster management, maneuver sequencing) [19]. Modularity in the software facilitates integration of algorithms developed by scientists outside MIT. The SPHERES team has developed a Guest Scientist Program [20]. A Guest Scientist can work with the SPHERES team to implement a test. In this program, the SPHERES team worked with Dr. McCamish to flight-test his LQR and APF controller [8]. Additionally, modularity in the software allows flexibility in designing an experiment through the selection of different combinations of modules. Inheritance between experiments and reuse of previously validated software modules increases the robustness of the code and permits engineers to incrementally mature GN&C technologies [1].

In the future, the SPHERES team has two potential contributions for inspector satellite operations. First, the current SPHERES testbed can be utilized by NASA and CEV engineers to develop and flight-test inspection algorithms. Second, with its man-rated and flight-qualified systems, the SPHERE satellites could be adapted for use as inspector satellites. To operate outside the ISS, several subsystems must be modified and qualified for the space environment. The ultrasound navigation system must be replaced with other navigation sensors. Since the ultrasound system is designed to imitate relative GPS data, one potential replacement would be relative GPS for operations in LEO. Otherwise, the ultrasound navigation information can be replaced with the computer-vision system that is currently under development. With these developments, the SPHERE satellites could be used for CEV inspection.

2.2 SPHERES Inspection System and Operations Design

In this section, a high-level design for the inspector satellite system and operations will be discussed. This framework was kept in mind during the development of the NavZ controller. Figure 2-4 shows a diagram of a control system for an inspector satellite. The pilot of the inspector sends waypoint, position, and pointing commands to the satellite through a Human Interactive Computer Interface (HICI). Those commands are sent to the inspector via an RF link. If waypoints are commanded, the inspector utilizes a path planner to determine the maneuvers that are needed to fly between waypoints. A supervisory control computer determines the thruster firings needed to maneuver along the path, while controlling the other satellite systems. Navigation sensors provide data for determining inspector position and orientation. Also, a

global map of the shape of the CEV and the location of the inspector is uploaded or created at the beginning of the operation, and then updated during the operation. The status of the inspector satellite and operations are displayed for the pilot on the HICI.

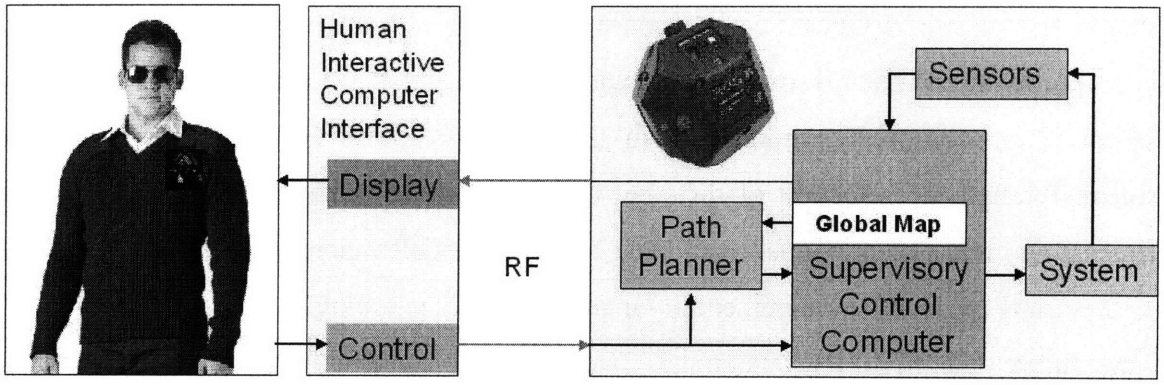


Figure 2-4: Block diagram of the inspection control system.

During an inspection operation, the pilot will utilize several inspector-satellite modes. Three possible inspector operations modes are illustrated in Figure 2-5. First, in the Semi-Autonomous Waypoint Mode, the satellite navigates around the CEV surface through a series of pre-determined waypoints, automatically scanning for damage. The camera system points almost directly at the surface, with its view slightly tilted in the direction of the path to identify feature points. Then, in the Semi-Manual Waypoint Mode, the pilot specifies a maneuver height and goal location. The inspector automatically plans a path to the goal. The camera system can be directed to point at a specific location on the CEV surface. Lastly, in the Manual Mode, the pilot directly controls 2-DOF of position or attitude while the autonomy holds the other 4-DOF steady. Automation provides collision avoidance, but the pilot can override it if necessary. The NavZ controller discussed in the next chapter can be used for either of these waypoint modes.

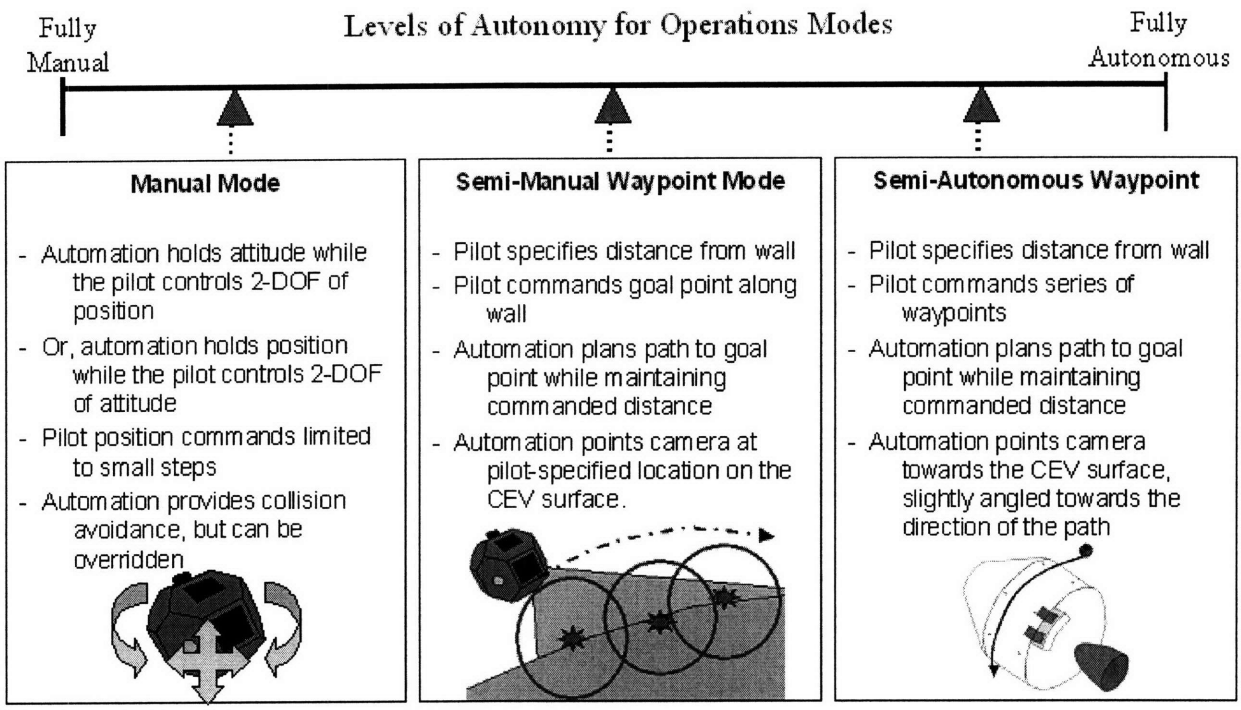


Figure 2-5: Three operations modes for an inspector satellite.

2.3 Summary

In this chapter, a brief description of the SPHERES testbed and satellite systems was given. Then, an inspector-control-system architecture was discussed. Lastly, several operations modes were introduced. The NavZ controller was designed within this general design architecture, and can be used in waypoint modes. The next chapter will discuss the development and flight-test results of the NavZ.

3 Inspection Path Planning and Maneuvering

The controller designed for SPHERES inspection maneuvers includes a path-planning controller, which fits into the simplified controller diagram as shown in Figure 3-1. This path-planning controller takes the control input from the satellite pilot. Then, it autonomously determines the forces that will maneuver the satellite along a desirable path. Next, the control computer regulates the thruster firings to apply those forces to the satellite.

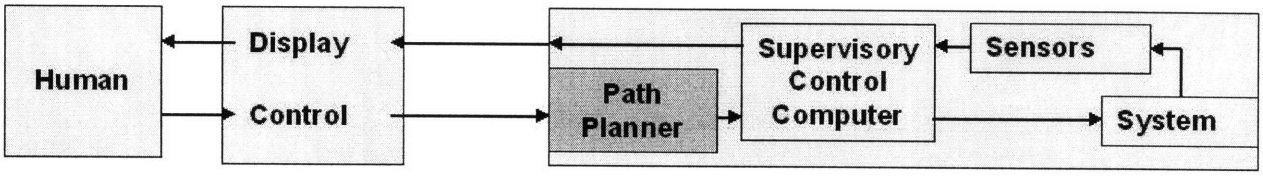


Figure 3-1. Overview of inspector satellite control system, with path planner highlighted

For the operations design introduced in Section 2.2, a path-planning controller has been developed that has the desired smooth-wall-following behavior discussed in Section 1.3. The baseline design for the SPHERES inspection controller is the Multiple Spacecraft Close-Proximity Control Algorithm (MSCPC) developed by Dr. McCamish at the Naval Postgraduate School (NPS) [4-6]. Section 3.1.1 gives an overview Dr. McCamish's original algorithm.

Through a joint-effort between the SPHERES team and NPS, their algorithm was implemented on the SPHERE satellites [8]. The contributions for this thesis include collaboration with Dr. McCamish and Dr. Nolet to implement the MSCPC on the SPHERES testbed, translate the simulation into flight code, perform verification tests, and analyze flight results. Sections 3.1.1-3.1.4 describe the implementation of MSCPC on SPHERES, verification of the code, and ISS flight-test results.

After the MSCPC was successfully implemented, it was redesigned for the application of inspection operations. Section 3.2 discusses the modifications that evolve the MSCPC into an inspector-satellite controller. Also, ISS flight-test results of this new algorithm are discussed in Section 3.2.3.

3.1 Baseline Controller Design Using LQR and APF

Originally, the MSCPC planner was designed to provide the control needed for multiple satellites to approach from a distance, maneuver around each other without colliding, and then dock together. Each satellite is treated as an obstacle surrounded by a spherical exclusion zone. The other satellites avoid maneuvering into that zone. To create these exclusion zones, the MSCPC algorithm combines two types of controllers: the Linear Quadratic Regulator (LQR) and Artificial Potential Functions (APF). It incorporates the efficiency of LQR with the robust obstacle avoidance provided by APF. References [4], [5], and [6] give detailed explanations about how this algorithm works. Section 3.1.1 gives an overview of the algorithm developed by Dr. McCamish. Also, it discusses how Dr. McCamish and the SPHERES team adapted MSCPC to work on SPHERES.

3.1.1 Algorithm of Baseline Controller Design on SPHERES

The MSCPC algorithm combines LQR and APF controllers to plan fuel-efficient paths around obstacles. In every control cycle, the desired forces are determined by LQR and modified by APF. The LQR finds the desired forces that will push the satellite towards the goal point. Then the APF controller decreases the LQR forces that would have pushed the satellite towards spherical obstacle zones. At the end of the control cycle, the satellite fires its thrusters to apply those forces.

Another way to view how the LQR and APF controller works is to look at the path. In each time step, the LQR plans a straight path from the starting point to goal point, with gradual acceleration and deceleration to minimize fuel usage. Meanwhile, each obstacle has a sphere of influence. If the satellite enters any the sphere-of-influence of an obstacle, then APF shifts that path so that it curves around the obstacle. This concept is illustrated in Figure 3-2.

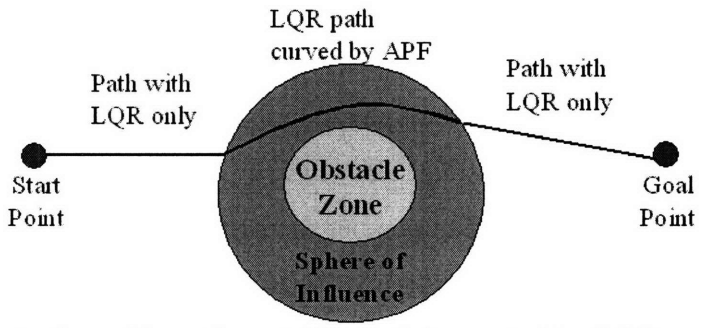


Figure 3-2. Illustration of how the satellite path is created by LQR and shifted by APF.

If multiple obstacle zones are placed near each other, then they can combine to form a mesh, as shown in Figure 3-3. Then, the satellite sees APF contributions from nearby obstacles. If these zones are placed along the surface of a spacecraft, they can define the shape of that surface. Thus, multiple interconnected obstacle zones can potentially define complex geometries. This idea is the basis for the NavZ algorithm, which is the main contribution of this thesis.

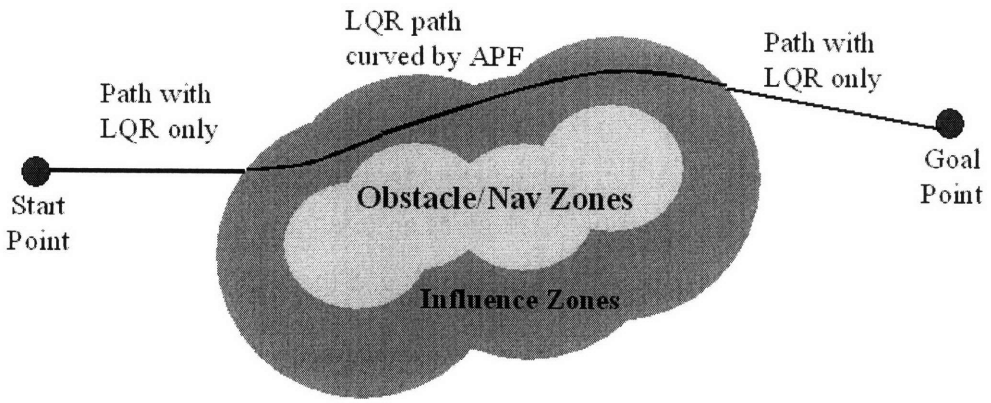


Figure 3-3. Illustration of a path that is shifted by APF from a mesh of obstacles.

In order to implement the MSCPC controller on SPHERES, a state-space model has been created. The state space model used by Dr. McCamish is based on the Hill-Clohessy Wiltshire linearized equations of relative motion. This state space model is shown in Equation 3.1 [1].

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3\omega^2 & 0 & 0 & 0 & 2\omega & 0 \\ 0 & 0 & 0 & -2\omega & 0 & 0 \\ 0 & 0 & -\omega^2 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \tag{3.1}$$

The position coordinates of the satellite x, y, and z are in the Hill's frame, whose origin is in a circular orbit around the Earth. The variable ω is the orbital velocity of the target point. The 6x6 matrix in Equation 3.1 will be referred to as A, and the 6x3 matrix is B.

Because SPHERES operates in a small test volume that is approximately a cubic meter, the affects of the orbital velocity on this path planner are negligible. Thus the A and B matrices can be simplified to

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

Using the block-partitioned form, A and B look like the following matrices, where I is a 3x3 identity matrix.

$$A = \begin{pmatrix} 0 & I \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 \\ I \end{pmatrix} \tag{3.3}$$

The A and B matrices in this state space model are inputs into the LQR function. With the simplifications described above, the inputs to LQR can be simplified to Equations 3.4. This simplification is useful, because it eases the development of an LQR solver for SPHERES. Please see Appendix C for more details.

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.4}$$

LQR is a type of optimal control algorithm that minimizes an energy quadratic cost function. For this application, the LQR uses the following cost function

$$J = \frac{1}{2}\int_0^T \left(\bar{x}^T Q \bar{x} + u^T R u\right) dt \tag{3.5}$$

in which x is the state error vector, u is the acceleration vector, and J is the cost variable that is minimized. In Equation 3.5, Q is the state penalty matrix. It penalizes the error in the position and velocity of the satellite with respect to the desired final state. Equivalently, R is the control effort penalty matrix. It scales the cost of the energy needed to accelerate or decelerate.

To solve this cost function, the assumption of an infinite time horizon is applied. Thus, the steady-state Ricatti equation can be used to solve Equation 3.5. Even though this assumption gives this problem a continuous infinite time, the penalty matrices are altered every control cycle to discretize the function and recover the finite-stopping nature of Equation 3.5. The values in Q

and R are altered every control cycle using Equations 3.6. This approach was developed by Dr. McCamish [5], but changed for SPHERES using the same block-partitioned simplification shown in Equation 3.4.

$$Q = \begin{pmatrix} \frac{\alpha_{Q1}}{r_{goal}^2} & 0 \\ 0 & \frac{\alpha_{Q2}}{v_{max}^2} \end{pmatrix} \qquad R = \begin{pmatrix} \frac{\beta_R}{a_{max}^2} \\ 0 \end{pmatrix} \tag{3.6}$$

In this equation, r_{goal} is the range to the target point, v_{max} is the maximum maneuver velocity set by the satellite operator, and a_{max} is the maximum satellite acceleration. The variables α_{Q1}, α_{Q2}, and β_R are gains that are fine-tuned to give the desired performance based on simulation results. For SPHERES,

$$\alpha_{Q1} = \alpha_{Q2} = \beta_R = r_{goal} \tag{3.7}$$

These Q and R matrices are inputs into the LQR function along with the A and B matrices. The LQR solver then finds the u that minimizes J in Equation 3.5. This u is the desired acceleration a_{LQR}, shown in Equation 3.8.

$$\min(J) = \frac{1}{2}\int_0^T \left(\bar{x}^T Q \bar{x} + u^{*T} R u^*\right) dt \qquad a_{LQR} = u^* \tag{3.8}$$

After this a_{LQR} vector is found, it is modified by APF so that it curves around obstacles. This modification is applied by adding the vector a_{APF} to a_{LQR}. Whenever the satellite enters the sphere of influence of an obstacle, that obstacle adds an a_{APF} vector. The APF developed by Dr. McCamish applies Equation 3.9 for a_{APF}.

$$\vec{a}_{APF} = k_v\left(\frac{\vec{v}_o}{\Delta t} + \vec{a}_o\right) \tag{3.9}$$

The vector v_o is the component of the satellite's velocity towards the center of the obstacle, and the vector a_o is the component of a_{LQR} towards the obstacle. If the satellite's velocity is away from the obstacle, then $v_o = 0$. Likewise, if a_{LQR} is away from the obstacle, then $a_o = 0$. The k_v is a potential-field shaping function. Thus, the first term in Equation 3.9 decreases the velocity towards an obstacle, and the second term decreases the acceleration towards an obstacle.

The k_v shaping function was developed by Dr. McCamish to provide a smooth Gaussian potential function [5].

$$k_V = \frac{e^{-r_o^2/(2\sigma^2)} - e^{-D_o^2/(2\sigma^2)}}{e^{-L_o^2/(2\sigma^2)} - e^{-D_o^2/(2\sigma^2)}} \tag{3.10}$$

$$\sigma = \frac{D_o}{3} \tag{3.11}$$

In these equations, r_o is the distance between the satellite and the obstacle and D_o is the diameter of the obstacle's sphere of influence. The L_o parameter is the minimum distance that the center of the satellite can be from the obstacle center such that the satellite does not enter the obstacle zone. The equation for L_o is

$$L_O = R_S + \text{uncert} + R_{obs} \tag{3.12}$$

in which R_s is the maximum radius of the satellite, *'uncert'* is the uncertainty of the satellite's position, and R_{obs} is the radius of the obstacle zone. In this application for SPHERES, R_s is 0.153 meters, *'uncert'* is 0.002 meters, and R_{obs} is 0.316 meters.

If the satellite is outside the sphere of influence ($r_o > D_o$), then $a_{APF} = 0$. Otherwise, a_{APF} is defined by Equation 3.9. For D_o, Dr. McCamish uses Equations 3.13 and 3.14.

$$D_O = d_{o1} L_O + d_{o2}\,\text{stopdist} \tag{3.13}$$

$$\text{stopdist} = \frac{v_o^2}{4 a_{max}} \tag{3.14}$$

The variable *'stopdist'* is the stopping distance of the satellite if it uses its maximum acceleration. Also, d_{o1} and d_{o2} are positive stopping distance constants. In this SPHERES application, these constants were set such that

$$D_O = 4 L_O + 3 \frac{v_o^2}{4 a_{max}} \tag{3.15}$$

Plugging these equations into Equation 3.9 gives a_{APF}, which is found for every obstacle whose influence zone covers the satellite. These APF accelerations are subtracted from the a_{LQR} to find the desired satellite acceleration in Equation 3.16.

$$\vec{a}_{desired} = \vec{a}_{LQR} - \sum_{obs=0}^{n} \vec{a}_{APF}^{obs} \tag{3.16}$$

The desired forces for the SPHERES satellite are Equation 3.17

$$\vec{F}_{desired} = M_s \vec{a}_{desired} \tag{3.17}$$

in which M_s is the mass of the satellite. For this application, M_s was set to 4.3 kg, which is the SPHERE satellite mass with about half fuel capacity. In the MSCPC development, this force vector was then rotated to inertial coordinates. For SPHERES, it is rotated to the laboratory coordinates. Then, the SPHERES mixers are utilized to determine the thruster firings needed to apply those forces.

With the parameters and modifications discussed in this section, the MSCPC algorithm was implemented in the SPHERES simulation before it was tested on hardware. The next section discusses the simulation set up and results.

3.1.2 Simulation of Baseline Controller Design

The experiment that was designed to test the implementation of the MSCPC algorithm on SPHERES includes three satellites. One is a chaser, another a target, and the third an obstacle. The chaser was set a meter away from the target, with the obstacle halfway between them and slightly offset. During the simulation, the chaser would need to maneuver around the obstacle to approach and dock to the target. The MSCPC algorithm was implemented in the SPHERES MATLAB simulation introduced in Section 2.1. The position and velocity of the simulated satellite are shown in Figure 3-4.

These satellite states were animated in Satellite Tool Kit (STK) using the method described in Reference [7]. Figure 3-5 shows screenshots of the STK animation. In Figure 3-5a, the satellite on the left is the chaser, middle is the obstacle, and right is the target. The cone on the chaser shows the docking face, and a faint sphere around the obstacle satellite shows its obstacle-zone. Commanded forces are illustrated with white plumes. The chaser satellite successfully navigates around the obstacle in Figure 3-5b, approaches the target in Figure 3-5c, and docks with the target in Figure 3-5d.

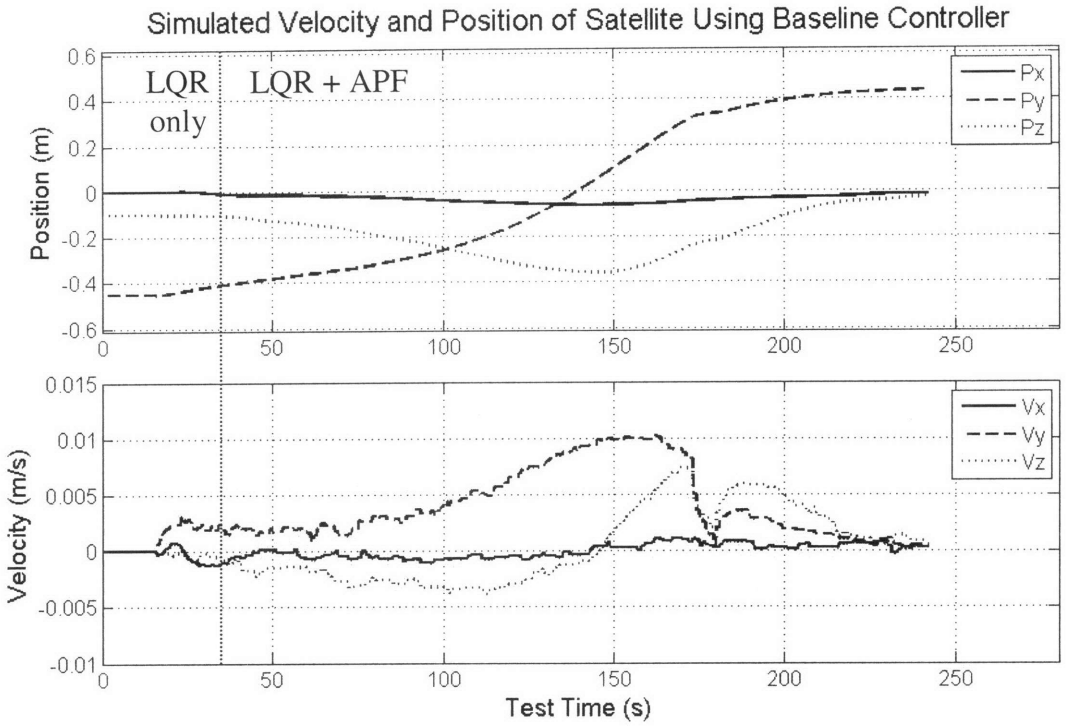
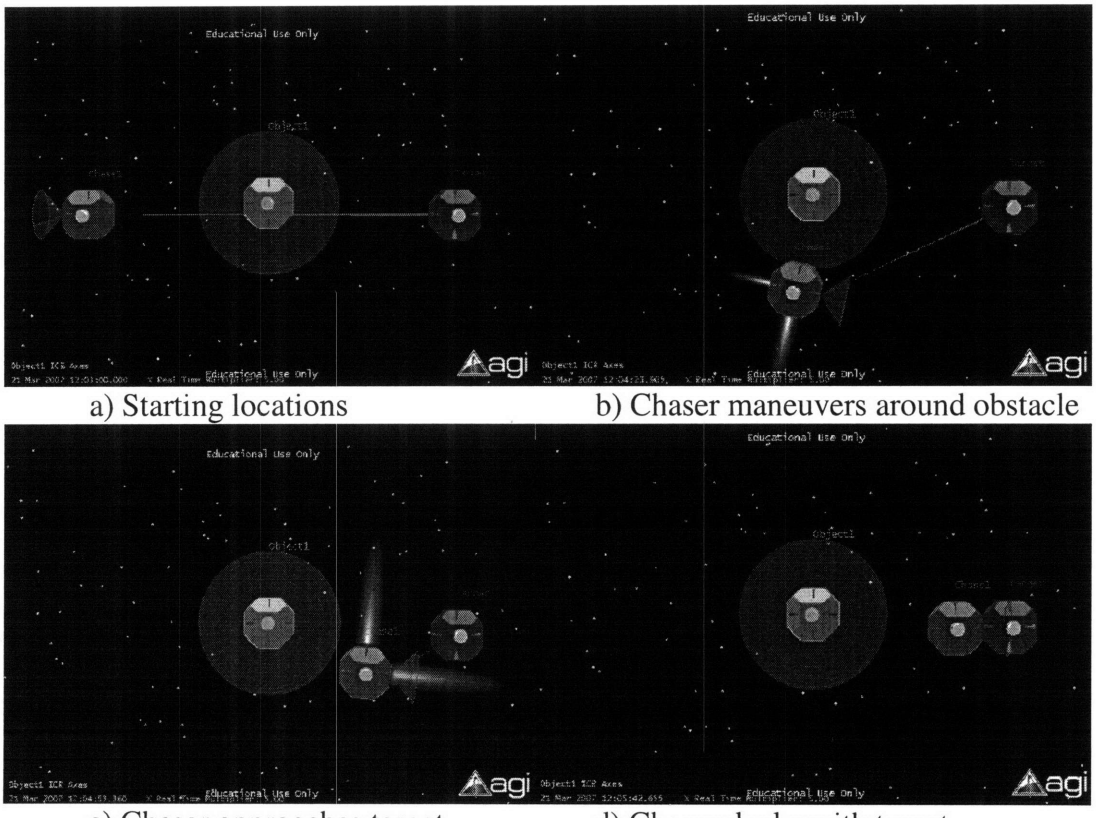


Figure 3-4. Simulated position and velocity of satellite.



a) Starting locations b) Chaser maneuvers around obstacle

c) Chaser approaches target d) Chaser docks with target

Figure 3-5. Results of the MSCPC algorithm implemented in the SPHERES MATLAB simulation [8].

3.1.3 Ground and In-Flight Verification of Baseline Controller Design

There were several steps that were performed to verify that the MSCPC algorithm was successfully implemented on the SPHERES facility. As discussed in the previous section, the algorithm was implemented in the SPHERES simulation, and the behavior of the satellites was checked to ensure that they were maneuvering correctly. Then, the controller code from the MATLAB simulation was translated into the *C* programming language using Microsoft Visual *C++* to run on the satellite [21]. The control code for the SPHERES satellite contains a library that includes the matrix manipulation functions that are utilized in the NPS controller. Thus, the only additional function that needed to be developed in *C-code* was an LQR solver. This LQR solver needed to be efficient enough to run onboard the SPHERE satellites. Using a combination of algorithms in Reference [22], a simple LQR solver, adapted to a double integrator system, was derived for controlling the position of the satellites in all three axes. Please see Appendix C for the LQR *C-code* and more details about its development.

The first step to verify that the NPS controller was successfully implemented in *C-code* was to compile the *C-code* translation into a MEX-file accessible by the SPHERES MATLAB simulator [23]. The resulting trajectories were compared to those of Dr. McCamish's original MATLAB simulation. They were found to be practically identical to the original ones, demonstrating the accuracy of the *C-code* translation of the NPS controller.

The second step was to test the *C-code* translation through hardware-in-the-loop testing. When the NPS controller ran on the SPHERES computer, it required ~10 ms (±2 ms) per iteration. Since the algorithm would typically only be called at a frequency of 1 Hz, this computation delay was judged to be acceptable for the real-time command structure of the satellite.

The third step was to run the test on the flat table in the SPHERES laboratory. Because the NPS control algorithm uses very small thrust levels (thruster opening times on the order of 20 ms), the ground satellite had difficulty overcoming the stiction, friction, and slope irregularities of the table. However, this ground test was helpful to visually verify that the satellite was thrusting in the correct directions during different moments in the maneuver.

Lastly, because the ground tests could not clearly demonstrate that the thrust-levels were correct, the telemetry was analyzed after this controller was run briefly on a SPHERE satellite in the ISS during December 2007. To confirm that the flight controller calculated the correct

outputs, the controller inputs from the telemetry were run through the simulation. The resulting simulation thrust-levels and flight-telemetry thrust-levels are plotted in Figure 3-6. These plotted forces are in the X-direction, using the coordinates of the ISS test volume shown in Figure 2-2. During this initial test of the controller, the chaser and target satellites were deployed too close to each other. When the test started, they were already touching each other. As a result, the commanded forces in Figure 3-6 are extraordinarily small and oscillatory. Thus, this flight-test was not useful for demonstrating the full maneuver, but was useful for validating the controller. Both flight-telemetry and simulation-results matched very closely, indicating that the onboard controller behaves very similarly to the MATLAB controller. From the results of these four verification steps, it was concluded that the controller was successfully implemented on the SPHERES hardware.

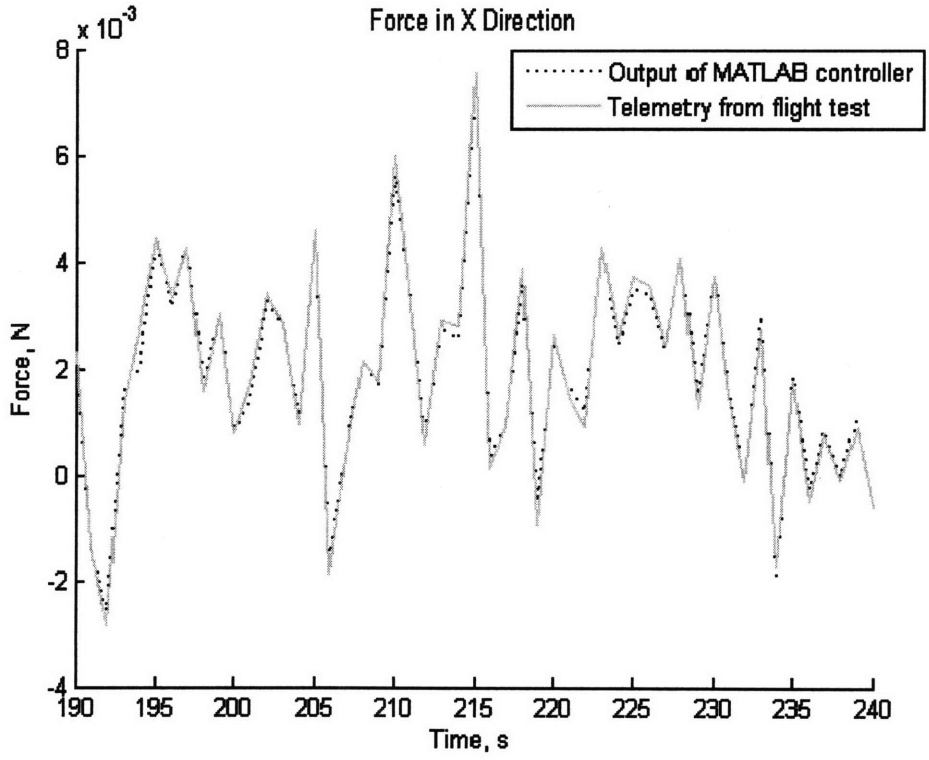
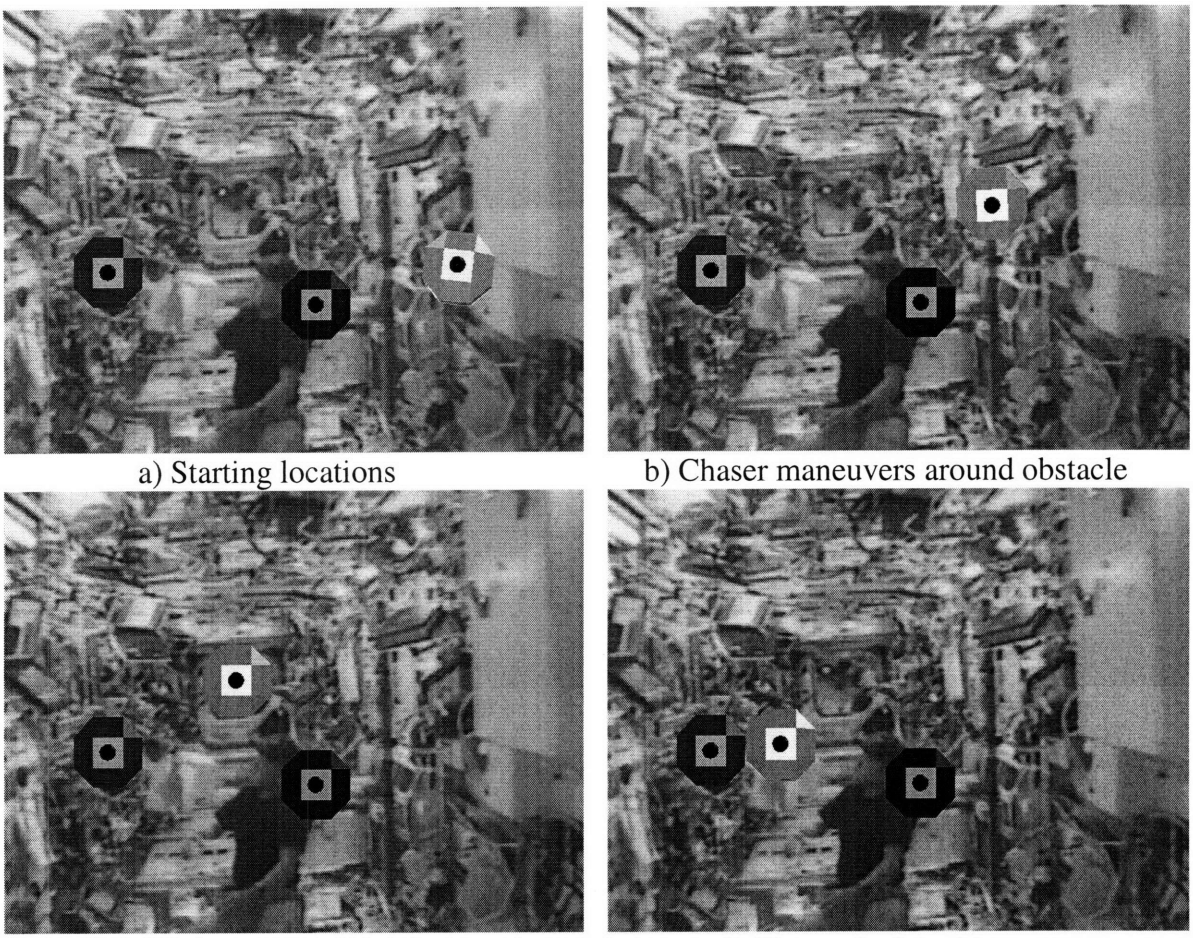


Figure 3-6. Comparison of the Computed Thrust Levels in the X-direction between Simulation and Flight

3.1.4 Flight Results and Evaluation of Baseline Controller Design

On December 2, 2007, December 29, 2007 and January 27, 2008, the NPS test was executed a total of six times. It was part of the SPHERES Test Sessions 10, 10a, and 11, all performed by astronaut Dan Tani onboard the International Space Station. During the four tests in Test Session 10 and the one in Test Session 10a, three SPHERES satellites were used (a chaser, a target and an obstacle). The satellites did not successfully complete their maneuvers mainly because of confusion with the deployment instructions sent to the crew. Thus, although valuable information was collected through the telemetry, the experiments performed during these two test sessions did not demonstrate the capabilities of the NPS Multiple Spacecraft Close-Proximity Control Algorithm.

For Test Session 11, the on-orbit supply of SPHERES batteries was low, with a new supply scheduled to launch in February 2008. To conserve batteries, the test was modified to use a single satellite, with the location of the target and the obstacle pre-determined and hard-coded in the software, to simulate a virtual target and a virtual obstacle. Figure 3-7 shows a few frames taken from an animation of the telemetry collected during the experiment. At the beginning of the test shown in Figure 3-7a, the chaser is on the right, the virtual target on the left, and the virtual obstacle in the center. Although not physically present during the experiment, the virtual target and the virtual obstacle were drawn to better illustrate the trajectory followed by the chaser. The chaser successfully maneuvered above the virtual obstacle, as shown in Figure 3-7b. Then, it approached the virtual target following a trajectory that would have led to docking, given a real target at that location. The approach is shown in Figure 3-7c and docking in Figure 3-7d.



a) Starting locations b) Chaser maneuvers around obstacle

c) Chaser approaches target d) Chaser docks with target

Figure 3-7. On-orbit Experimental Results: Animation of the telemetry collected during the single-satellite docking experiment in the ISS.

The telemetry provided the position and velocity of the chaser, which are plotted in Figure 3-8. The origin of the coordinate system is located in the center of the test volume, and also corresponds to the geometric center of the obstacle for this experiment. During the first 47 seconds, the onboard navigation system converged to a solution, and the satellite moved to a proper initial position given the constraints of the test volume. The MSCPC algorithm controlled the chaser from 47 seconds to the end of the experiment. Throughout the experiment, the trajectory was very smooth. The apparent noise on the velocity estimates was caused by noise in the ultrasonic range measurements. Docking would have occurred at 230 seconds. At that time, the velocities along all three axes were estimated to be less than 1 mm/s. This experiment was very successful, as it would likely have resulted in docking if an actual satellite were physically present at the location of the virtual target.

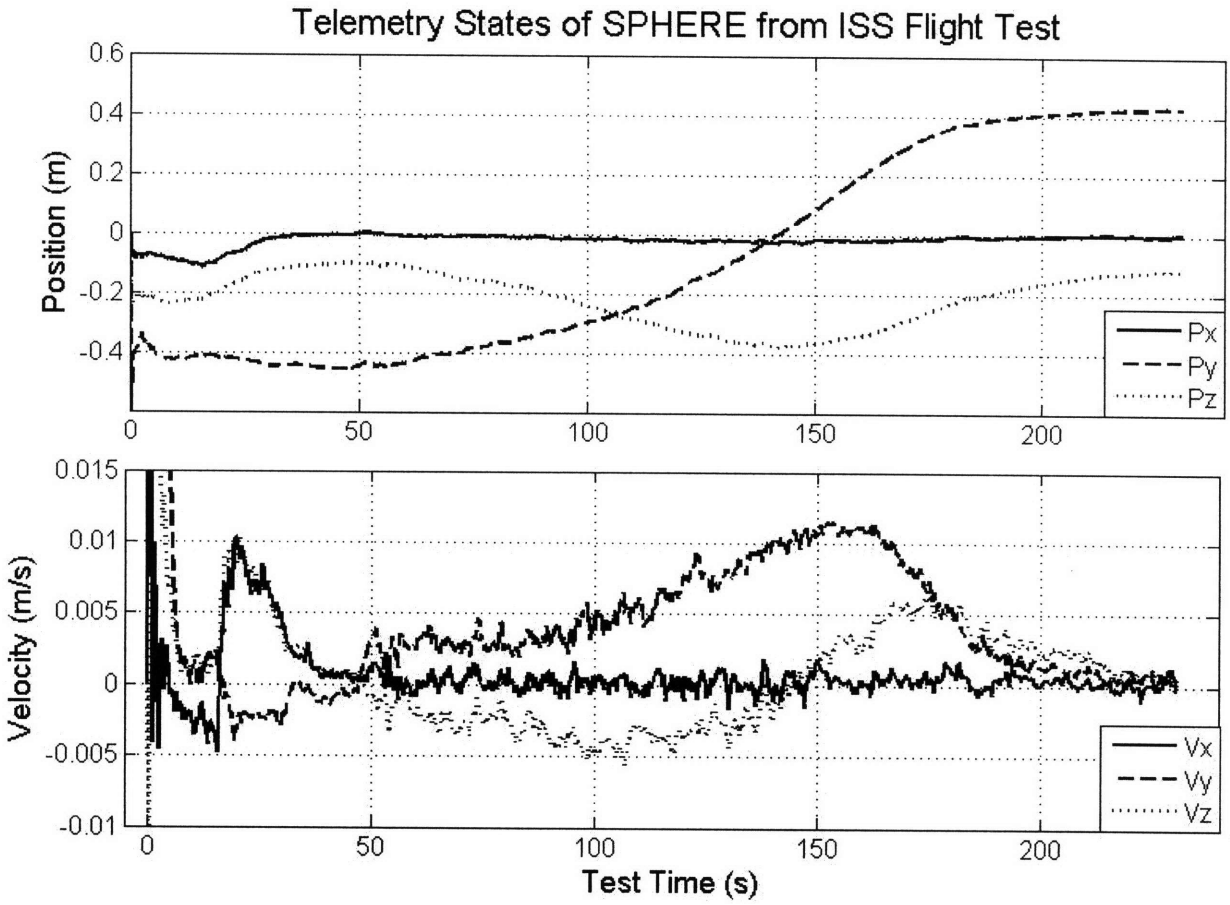


Figure 3-8. On-orbit Experimental Results: Position and velocity of the chaser during the single-satellite docking test in the ISS.

Now that the real-time, hardware-in-the-loop implementation and performance of the MSCPC algorithm has been validated for obstacle avoidance and docking, the next section will extend this work to create an algorithm that can interface with a computer-vision system and be used to navigate around complex obstacle shapes for inspection operations.

3.2 Inspection Controller Design using Mesh of Navigation Zones (NavZ)

For the NavZ inspection controller, the MSCPC algorithm has been modified so that it works with one miniature satellite navigating around a larger spacecraft body. As introduced in Section 1.3 and shown in Figure 3-9, exclusion zones can be placed around the feature points detected by a computer-vision system, thus forming a mesh of NavZones. Also, the exclusion zones can be ellipsoidal instead of spherical so that they closely follow the contours of the spacecraft body. This section describes developments that extend the MSCPC algorithm for navigation around obstacle meshes. Also, it discusses how the APF functions are changed to create ellipsoidal NavZones instead of spherical ones. Lastly, this section discusses flight-test results of the new algorithm aboard the ISS.

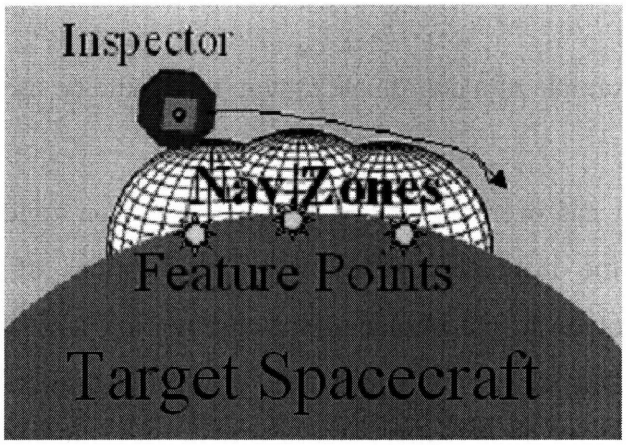


Figure 3-9: Illustration of inspector maneuvering around mesh of NavZones.

3.2.1 Algorithm for Inspection Controller using NavZone Mesh

The first change that was made to the MSCPC was the navigation coordinate frame. For the MSCPC, the body coordinates of the target spacecraft were used for navigation. For inspection operations, the coordinates of the inspected-spacecraft body will be used. This coordinate system is useful as the frame of reference, because the smaller inspector navigates around this larger body. For inspecting the CEV, the x_{CEV} coordinate could be in the direction of the inspector's docking port, and the z_{CEV} coordinate could be along the length of the vehicle. Figure 3-10 illustrates how these coordinates could be oriented on the CEV. The origin is in the center

of the CEV and aligned with the docking station. In the case of SPHERES testing on the ISS, these navigation coordinates were set to the ISS test-volume coordinates shown in Figure 2-2.

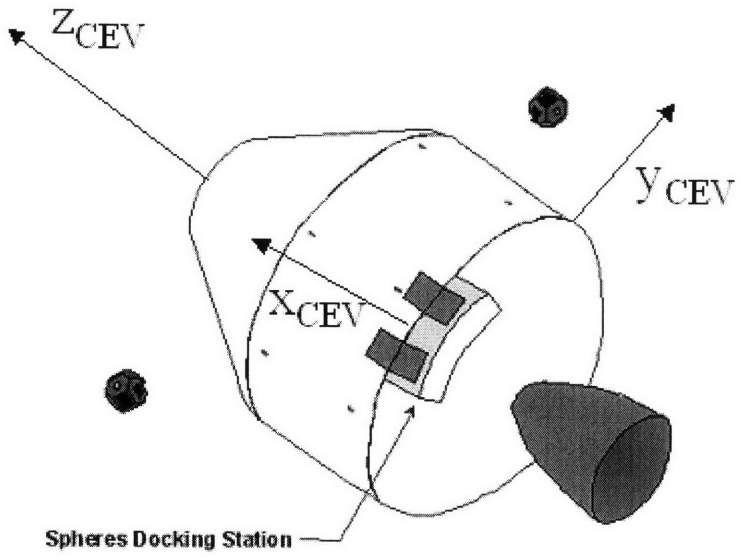


Figure 3-10: Example of a coordinate system that could be used by an inspector satellite for the CEV

Also, the algorithm has been adapted so that the exclusion zones surround feature points along the surface of the inspected spacecraft. The locations of these feature points are determined each control cycle by emulating the information that would be received from a camera system. For the initial implementation of this controller, the feature points have predetermined locations. With an actual camera system, the computer vision would determine the satellite's distance from feature points. In this emulated case, the satellite determines its distance from these feature points by extracting the information from the SPHERES global metrology system. This initial emulated computer vision and further improvements to the emulation are discussed in Chapter 5.

When the NavZones are placed at feature points, they create a mesh that outlines the shape of the target spacecraft. With LQR and APF controllers, an inspector maneuvers in paths that outline the mesh of NavZones. Thus, the inspector navigates at a specified distance above the target's surface, determined by the width of these NavZones. Because of the method described in Section 3.1.1, that gradually combines the APF with the LQR as the SPHERE approaches a NavZone, the path does not dip between NavZones, but smoothly follows the outline. Each NavZone that has the inspector inside its influence zone contributes an APF to Equation 3.16,

rewritten below. These APF contributions of the nearby NavZones are summed for every control cycle, thus creating the effect of a mesh of NavZones. Figure 3-11 illustrates this concept. The spherical NavZones are placed to outline a complex shape, and the snapshots of the SPHERE show that it smoothly maneuvers along that surface.

$$\vec{a}_{desired} = \vec{a}_{LQR} - \sum_{obs=0}^{n} \vec{a}_{APF}^{obs} \tag{3.16}$$

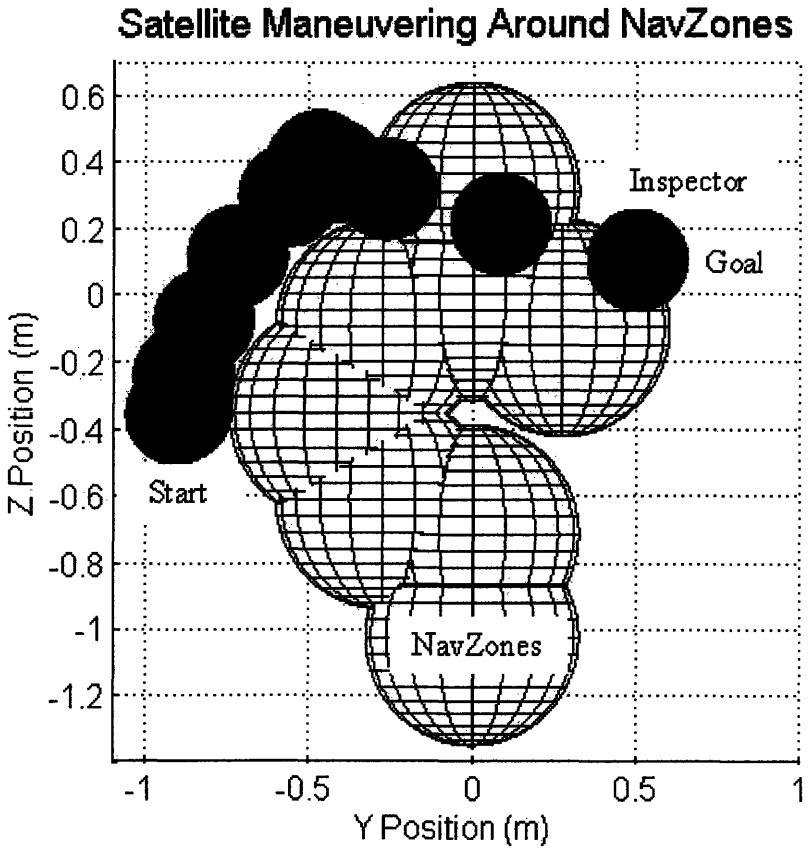


Figure 3-11 Snapshots of a satellite maneuvering around navigation zones in simulation

In the case of complex protrusions from the spacecraft body, such as solar arrays or antennas, other potential functions could be used to outline the surface in greater detail. The spherical potentials used for the MSCPC are beneficial, because they are not computationally expensive, as described in Section 3.1.1. The calculations are fairly simple compared to other potential functions. For each control cycle, the main information that is needed is the distance from each obstacle that is in view. To provide greater detail to the surface mesh of NavZones, yet maintain much of the computational simplicity of the original MSCPC algorithm, an ellipsoidal potential function has been implemented. When the model of the inspected spacecraft is already known,

ellipsoids could be placed strategically to give greater definition to the shape. Or, the ellipsoid zones could be stretched so that they provide more overlap than the spheres, and thus a smoother surface model, as shown in Figure 3-12. The ellipsoid NavZones on the right of the figure have the same separation distance as the spheroid NavZones on the left, but they provide a smoother surface. Note how the dips between the ellipsoids are smaller than those between the spheroids.

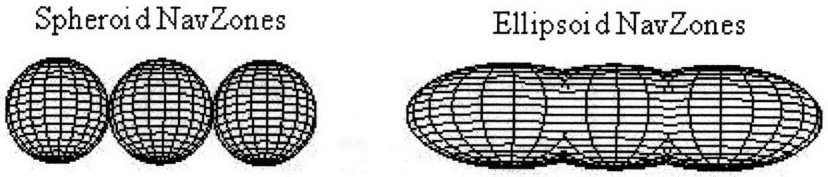


Figure 3-12: Spheroid and ellipsoid NavZones

Each ellipsoid has its own coordinate frame with an origin at the center of the ellipse, labeled x_{obs}, y_{obs}, and z_{obs}. An ellipsoid is then defined by the orientation of its coordinate frame and its three radii: *a*, *b*, and *c*. These radii are illustrated in Figure 3-13. The location of the spacecraft along the edge of the ellipsoid is defined using the angles λ and β, as shown in Figure 3-13.

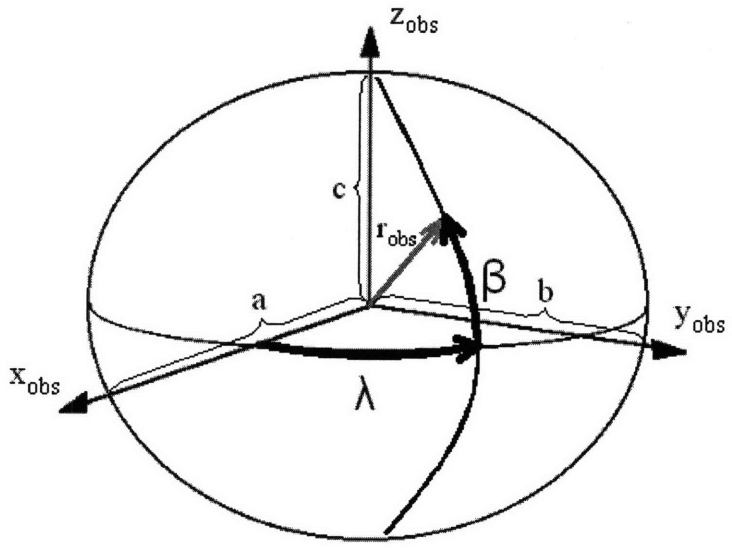


Figure 3-13. Ellipsoid coordinates

From the view of the inspector satellite, changing the potential functions from spherical to elliptical causes the width of a NavZone to change as the inspector flies around it. So for each time step, the width of the NavZone in the direction of the inspector satellite must be computed to determine how close the inspector is to its surface. First, the position vector from the origin of

the ellipse to the inspector satellite is found in the ellipse's frame of reference, r_{obs}. The width of the NavZone along r_{obs} is determined by finding λ and β in Equations 3.18 and 3.19, and then finding the magnitude of Equation 3.20.

$$\lambda = \tan^{-1}\left(\frac{a}{b}\frac{r_{y\,obs}}{r_{x\,obs}}\right) \tag{3.18}$$

$$\beta = \tan^{-1}\left(\frac{b}{c}\frac{r_{z\,obs}}{r_{y\,obs}}\sin\lambda\right) \tag{3.19}$$

$$Obs = \begin{pmatrix} a\cos\beta\cos\lambda \\ b\cos\beta\sin\lambda \\ c\sin\beta \end{pmatrix} \tag{3.20}$$

The width of the NavZone along r_{obs} is the magnitude of Equation 3.20, $|Obs|$. This magnitude is then included in Equation 3.21, which is used in the controller instead of the original Equation 3.12.

$$L_o = R_s + uncert + |Obs| \tag{3.21}$$

This revised L_o is then used in Equation 3.15 to determine D_o. This change to D_o affects the APF acceleration adjustment a_{APF}, via Equation 3.9, causing the path to wrap around the ellipsoidal shape. Thus, whenever the inspector enters the influence zone around an ellipsoid, the APF uses the method in Section 3.1.1 to calculate a_{APF} with the NavZone radius defined by the ellipsoid. The path is shifted around that ellipsoid using a_{APF}.

These NavZone meshes and ellipsoidal potential functions were incorporated into the algorithm and simulated in the SPHERES MATLAB simulator. The simulation results will be discussed in the next section.

3.2.2 Simulation of Controller Design using Mesh of Navigation Zones

In the first implementation of this NavZ controller, only spherical NavZones were used. The target-spacecraft body coordinate system is set to the ISS test-volume coordinate system in Figure 2-2. Also, the NavZones are placed at [0 0 0.1], [0 -0.275 -0.103], [0 -0.416 -0.36], and [0 0.275 -0.103] meters using ISS test-volume coordinates. The radius of each NavZone is set to

0.153 meters. Additionally, v_{max} from Equation 3.6 is set to 0.02 m/s. Thus, the simulation is designed to be a slow-moving first implementation.

When the inspector begins near the edge of a NavZone's sphere-of-influence at [0 -0.8 -0.1] meters, its path follows that shown in Figure 3-14. Each inspector-satellite image is a snapshot of the location of the SPHERE at evenly spaced intervals of 40 seconds during the simulation. The inspector appears to smoothly maneuver around the spheres-of-influence of the NavZones.

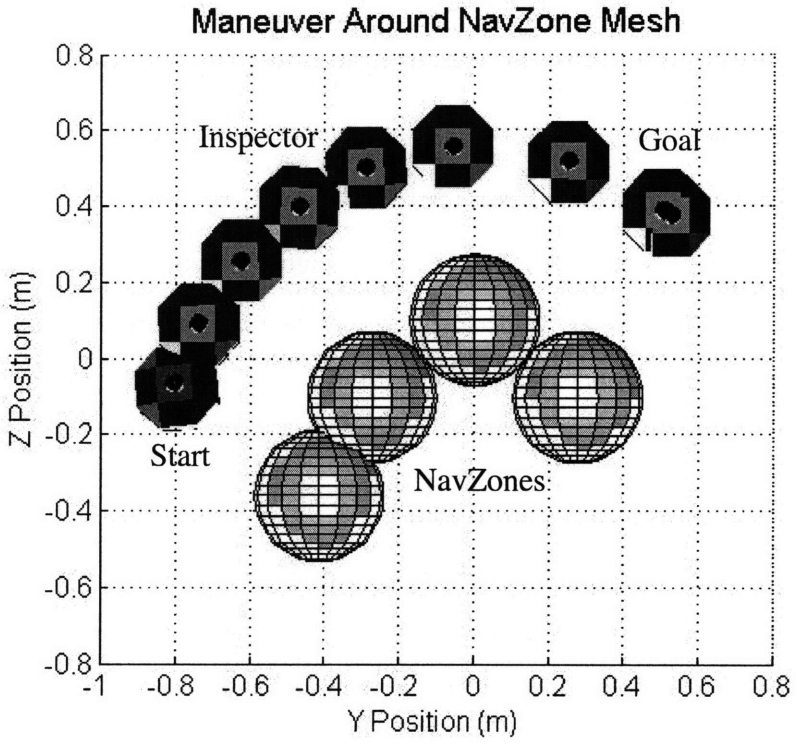


Figure 3-14. Snapshots of satellite maneuvering navigation zones

The simulated telemetry is shown in Figure 3-15. With the maximum velocity set as 0.02 m/s, the total time needed for the maneuver is around 350 seconds. Also, the change in velocity is gradual, as is expected when using LQR.

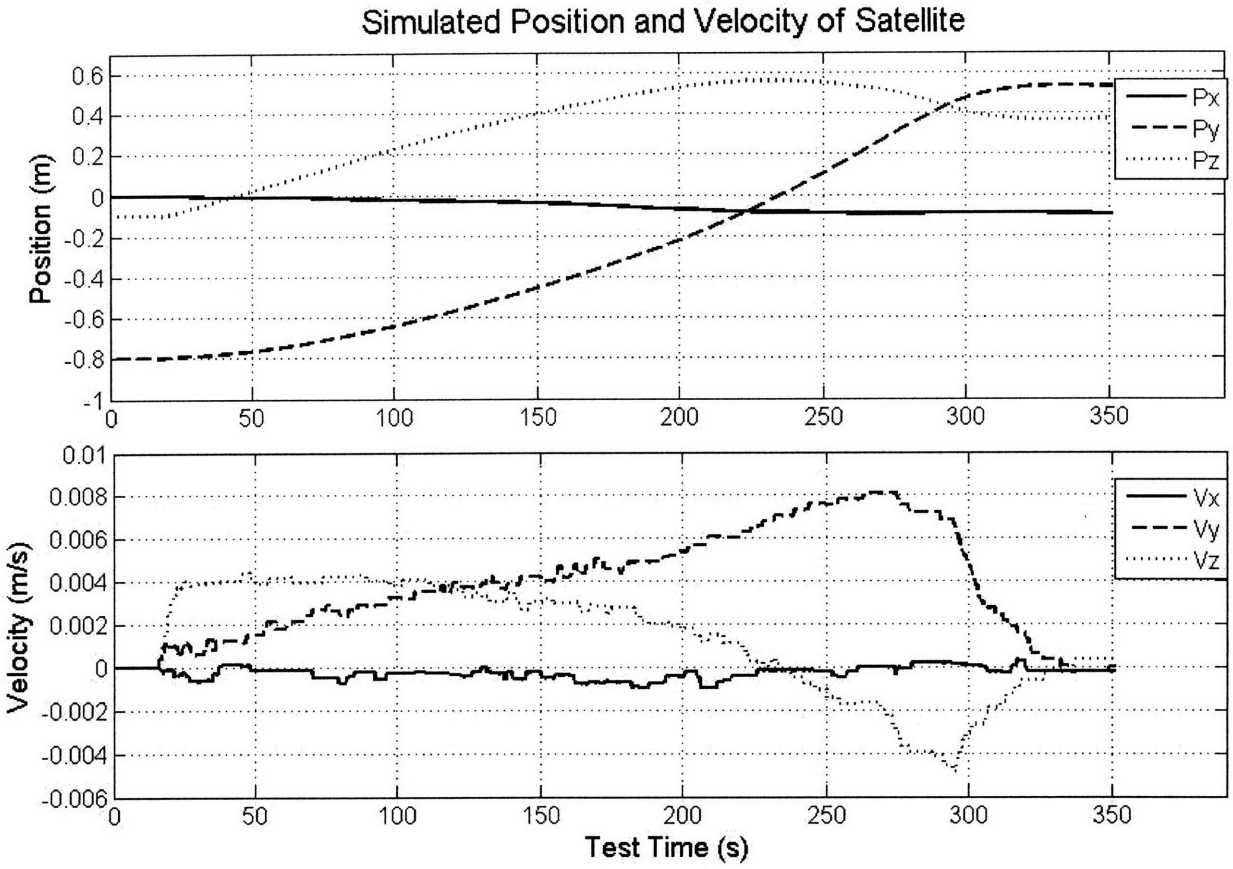


Figure 3-15. Simulated telemetry of the SPHERE maneuvering around NavZones

While the desired behavior is for the satellite to maneuver closely around the NavZones, in this implementation the satellite maneuvers at about twice the width of the NavZones. This behavior is cause by the influence zones around the NavZones, illustrated in Figure 3-16. According to Equation 3.15, the width of the influence zone is more than twice that of the NavZone. A conclusion from this simulation is that the width of the influence zones could potentially be altered to adjust how close the inspector flies to the NavZones. In Section 4.1, Equation 3.15, which defines the width of the influence zones, is altered so that the satellite maneuvers closely along the surfaces of the NavZones.

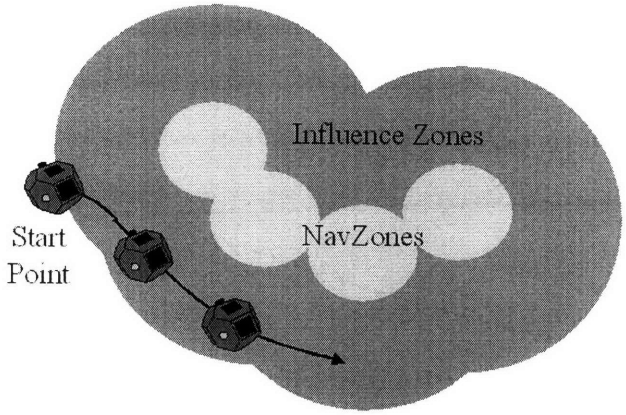


Figure 3-16: Size of influence zones affects how close the inspector flies to the NavZones.

For the first implementation of the ellipsoid potential function, one ellipsoidal NavZone is placed at the center of the ISS test volume with its axes aligned with those of the test volume. The radii of the ellipsoid are set to a = 0.6 meters, b = 0.6 meters, and c = 0.3 meters. The changes to the influence zone introduced above and detailed in Section 4.1 were implemented for this simulation. Thus, the satellite closely followed the edge of the NavZone. Figure 3-17 shows the results of this simulation and gives a three-dimensional view of the simulation trajectory. These ellipsoid potential functions were fully integrated into the rest of the control code after the first ISS test, and this integration is discussed in Chapter 6.

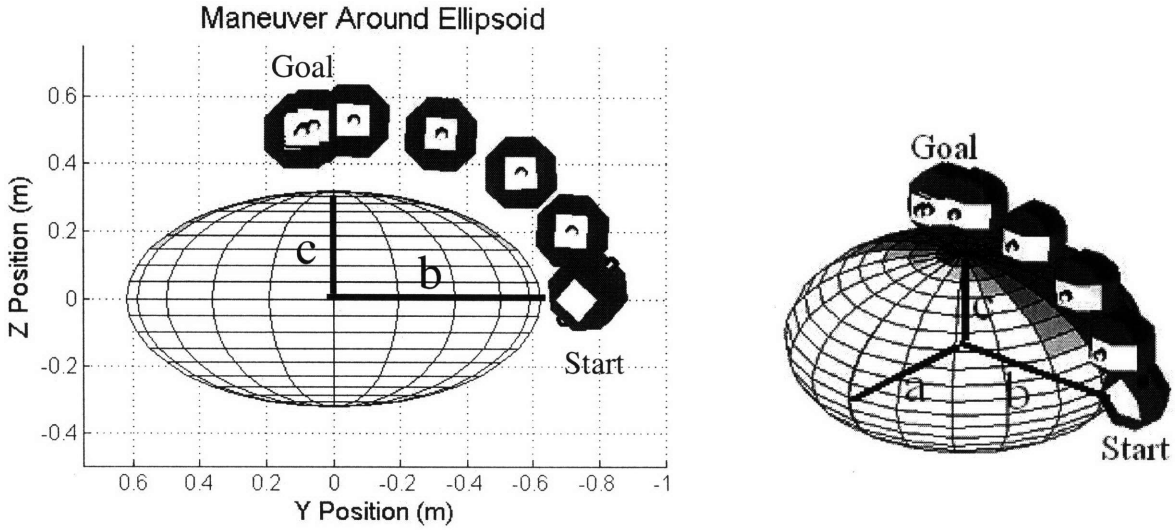
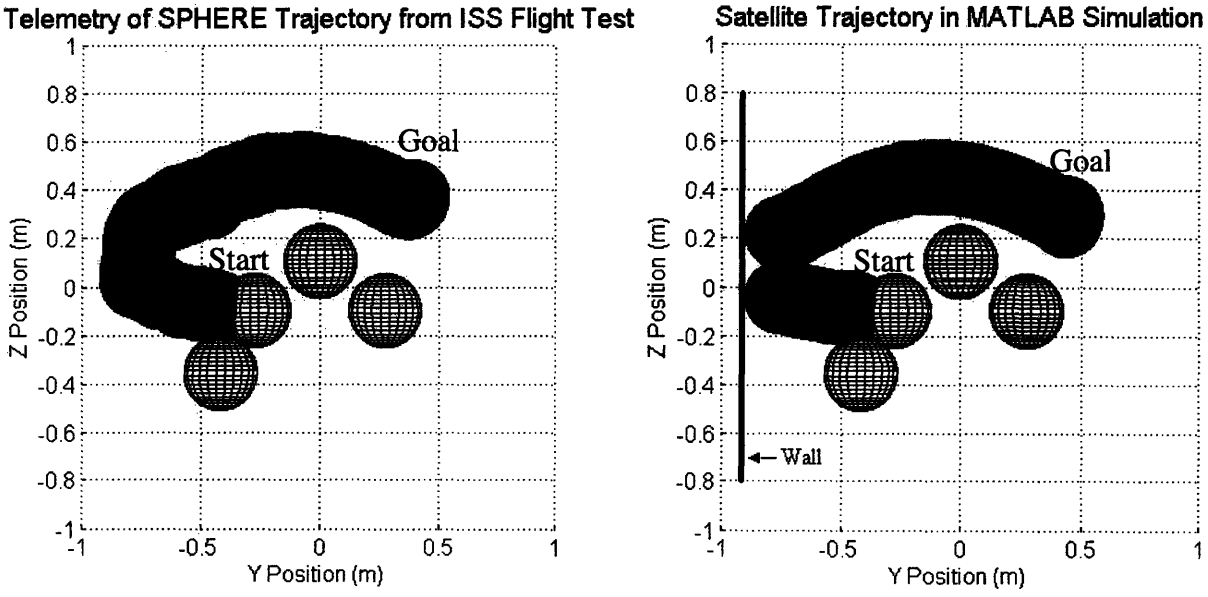


Figure 3-17. Plot and 3-D visualization of a satellite maneuver around an ellipsoid using the SPHERES MATLAB simulation.

3.2.3 Flight Results and Evaluation of NavZ Controller

The maneuver simulated in Section 3.2.2 was tested aboard the ISS during SPHERES Test Session 11 on January 27, 2008. The results are shown in Figure 3-18a. These results look very different than the simulated results shown in Figure 3-14, because the satellite had a different starting point. During the ISS test, the satellite started at [0 -0.45 -0.1] meters, which was partially inside a NavZone. Because the repulsion force grows exponentially towards the NavZone center, the satellite pushed out of the navigation zone with large forces. This caused the satellite to move quickly towards the wall. Once it touched the wall, it recalculated its LQR and APF forces, and began moving towards the goal point. The start and goal points from the test results were then put into the simulation, and the trajectory is plotted in Figure 3-18b. The gap in between trajectories is due to the satellite touching and moving along the wall of the ISS. Since the wall is not included in the MATLAB simulation, the simulation trajectory was plotted to the point that it reached the wall. Then, another simulation trajectory was started at the point that the satellite left the wall in the flight test. This shows that when the satellite begins inside the navigation zone, the simulation results closely match the ISS Test results if the wall stops the satellite.



a) Flight-test data b) Simulation

Figure 3-18: On-orbit experimental results and corresponding simulation.

The cause of this change in starting point was poor configuration control management. During development of this controller, the satellite starting position was set at [0 -0.8 -0.1] meters. Then during Test Session 10, the SPHERES team discovered that a white box had been installed that encroached into the test volume. Now the negative y limit of the test volume was tighter. The satellite could not go farther than y_{pos} = -0.45 meters in the test volume. Because of this change, the starting point in the flight code was changed. However, the corresponding change to the simulation starting point was forgotten. Thus, the configuration of the simulation was older than that of the flight code, and the test was not executed as it was originally simulated. Two important lessons: Configuration Control and Test as You Fly. The configuration control management processes were improved for the continued development of the NavZ controller after the first flight test.

3.3 Summary

A unique path planner was designed for inspection operations. Its strength lies in its ability to control the satellite in an optimal energy maneuver that smoothly follows the surface curvature of another spacecraft. The baseline algorithm was the MSCPC developed by Dr. McCamish at the NPS. This controller was modified to run on the SPHERE satellites and adapted to control maneuvers around meshes of NavZones at feature points on a target spacecraft's surface. These feature points are analogous to feature points that could be detected by a computer-vision system. Furthermore, the spherical-based APF was changed to an ellipsoidal-based APF. These ellipsoid potential functions allow for a better-defined model of the inspected spacecraft, while still maintaining much of the simplicity of the original MSCPC algorithm.

Both the MSCPC and the NavZ algorithms were flight tested with the SPHERES hardware aboard the ISS. The algorithms successfully performed like the simulations. Also, several potential improvements for NavZ were discovered. These continued developments are discussed in Chapter 4.

4 Inspection Operations Development After First Flight Test

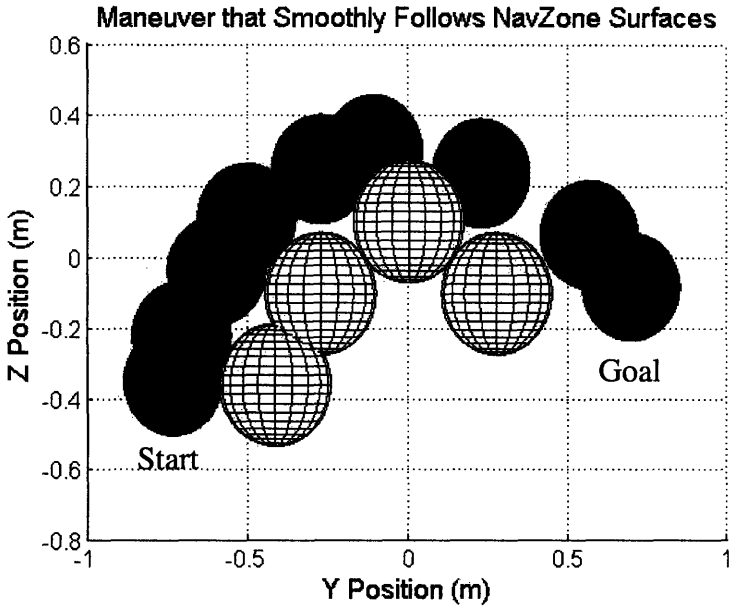
After the first test on the ISS, the NavZ algorithm was improved through several other developments. The ability for a pilot to change the height of the maneuvers above the target's surface was implemented and simulated. Also, a pointing algorithm was developed that is unique for this NavZ path planning. Each section below describes the algorithm and simulation of a NavZ development. Also, fully integrated simulations that combine all of these improvements are discussed in Chapter 6.

4.1 Changing Influence Zone

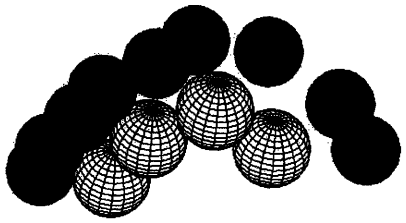
As discussed in Section 3.2.3, the influence zone width needed to be changed so that the satellite would follow the NavZones closely. To make this change, the Equation 3.15 for the width of the influence zone was adjusted to Equation 4.1. This change makes the influence zone width approximately 50% smaller than before.

$$D_0 = 2\left(L_0 + \frac{v_0^2}{4a_{max}}\right) \tag{4.1}$$

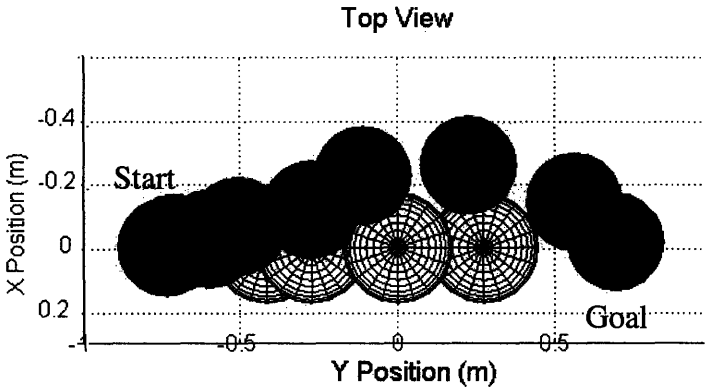
Because of this change, the satellite now maneuvers closely along the edge of the navigation zones. This behavior is evident in the simulation results shown in Figure 4-1. Figure 4-1b shows snap-shots of the satellite position as dark-colored disks, and the NavZones are shown as wire-frame white spheres. Figure 4-1b shows a three dimensional view of the maneuver. In these graphs, the satellite maneuvers up and around the back of the mesh, while closely following the NavZones surfaces. Note how the trajectory closely follows the edge of the zones.



a) Front View



b) 3-D View



c) Top view

Figure 4-1: Simulation results with new influence-zone equation.

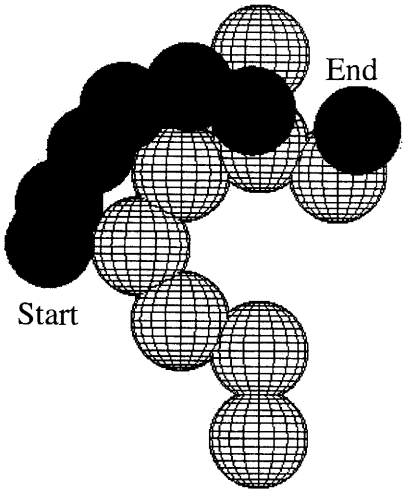
This is an improvement to the controller simulated and tested in Chapter 3. For an inspection operation, the pilot of the inspector can command it to maneuver at a specified distance above the target's surface. This specified distance sets the size of the NavZones (L_o). For the controller in Section 3, the satellite moves outward to about twice L_o away from the target's surface, then completes the maneuver needed to get to the goal point. Now, with the new Equation 4.1, the satellite's trajectory is consistently at L_o above the target's surface. Thus, the satellite maneuvers at the specified distance above the target's surface.

4.2 Controlling Height Above the Target's Surface

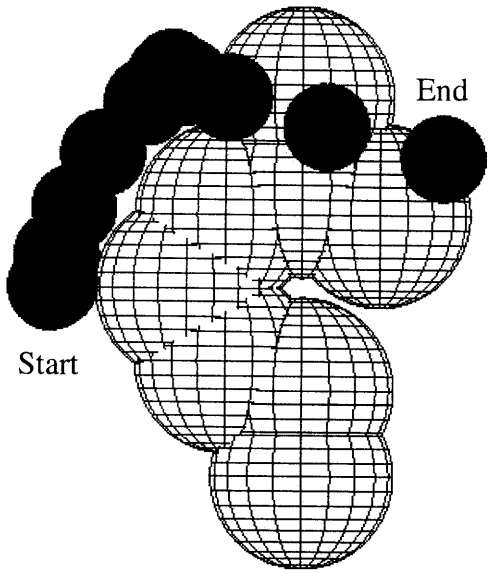
With the development described in Section 4.1, the inspector pilot can now control the height above the target's surface at which the inspector maneuvers. Changing the variable that defines the size of the NavZones, L_o, changes the height of the maneuver. When the pilot inputs a new desired height, the variable L_o is reset to this value. This is a useful aspect for the NavZ controller. An inspector pilot may need to have it maneuver closer to the target surface in order to see particular features. Or, the inspector pilot may need to fly farther away for quicker maneuvers or broader situational views.

Simulations were performed implementing commands for maneuvering different heights above the target surface. In the operations modes discussed in Section 2.2, when a pilot specifies a new height, the satellite first flies to that height. Then, the NavZones are sized to that height, and the maneuver begins. To imitate this operations mode, in these simulations the start point is set at the surface of a NavZone. Thus, that start point shifts if the NavZone increases or decreases in size.

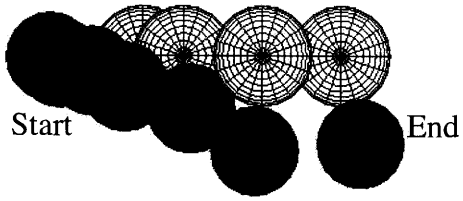
Graphs from two of these simulations are shown in Figure 4-2. Both of the start points are set so that they are located on the edge of far-left NavZone. Also, both have the same goal point. Figure 4-2a shows a simulation with the height L_o set to 0.153 meters. Figure 4-2b shows a simulation in which the height L_o was changed to 0.3 meters, and the start point was offset by this new height to remain on the NavZone edge. These results show that a pilot can change the value of L_o in order to inspect closer or farther away from the vehicle.



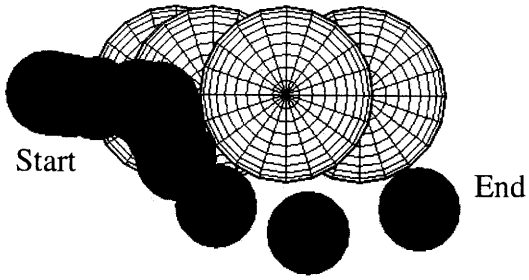
a) Front view of low maneuver



b) Front view of high maneuver



c) Top view of low maneuver



d) Top view of high maneuver

Figure 4-2. Inspector satellite maneuvers at two different heights above the vehicle hull.

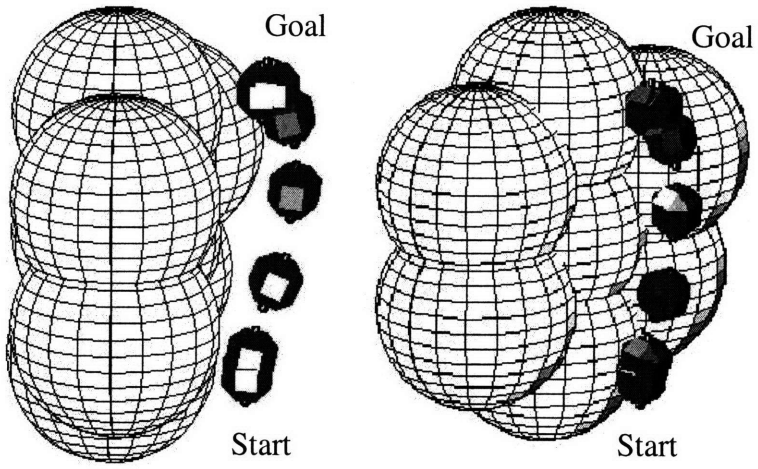
4.3 Controlling Maneuver Speed

In addition to controlling the height of the maneuver, an inspector pilot will want to control the speed of the maneuver. When there are other important CEV operations as well, an inspection operation will have a specified time schedule. In this case, a maneuver's time could be limited, and the speed adjusted accordingly.

The variable v_{max} can be utilized to control the time it takes to perform a maneuver. This variable adjusts the speed of the whole maneuver by affecting the LQR accelerations through Equation 3.6. If v_{max} is increased, the accelerations calculated by LQR increase, thus increases the maneuver velocities. Because of the design of the combined LQR and APF controller, the APF also adjusts to handle these increased velocities. Looking at Equation 3.9, if v_{max} is

increased, then the velocity towards an obstacle v_o increases, and the corresponding APF accelerations increase. Thus with an increase in v_{max}, the values of a_{LQR} and a_{APF} adjust and balance to create a quicker maneuver that still successfully avoids the navigation zones.

Several simulations were performed with different values for v_{max}. The maneuver that was used for these simulations is graphed in Figure 4-3. The following figures show the results of changing v_{max}. The maneuver in Figure 4-4 implemented a v_{max} of 0.05 m/s, and shows that the maneuver finished in 100 seconds. For Figure 4-5, v_{max} was 0.5 m/s, and the maneuver finished in 80 seconds. Thus, changing the variable v_{max} successfully changes the maneuver speed.



a) Front View b) 3-D View

Figure 4-3: Maneuver performed for the simulations testing changes in v_{max}.

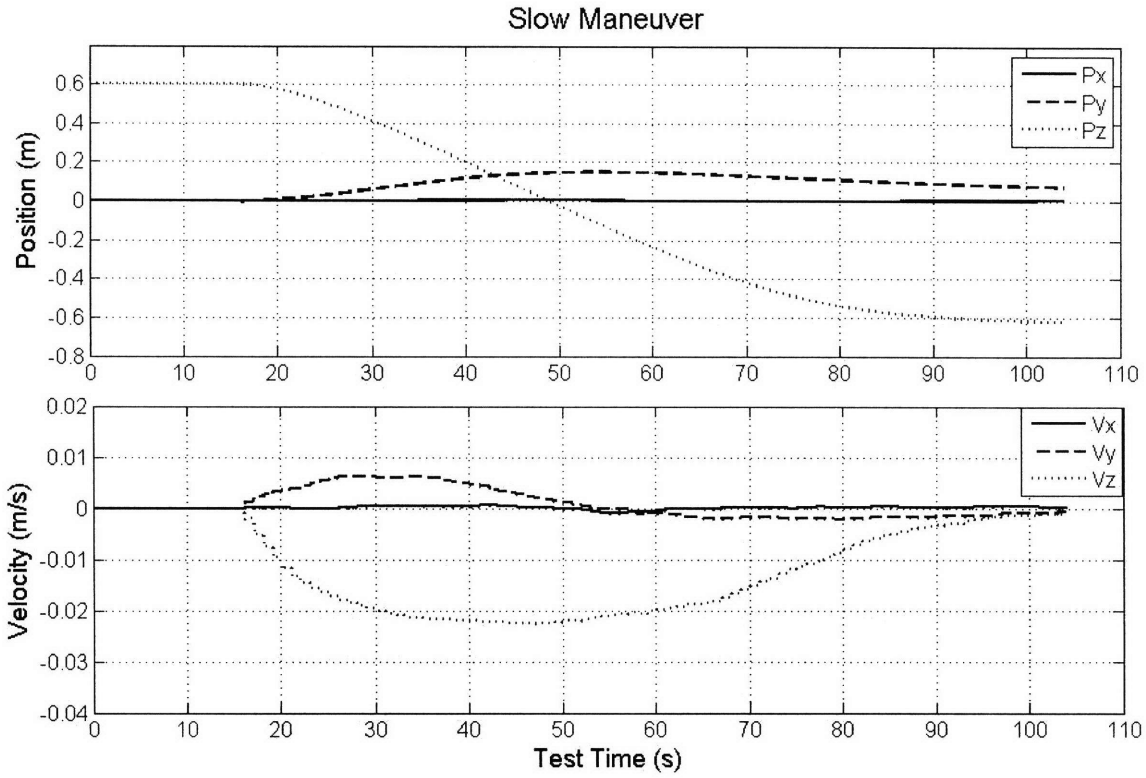


Figure 4-4: Maneuver performed with a value of 0.05 for v_{max}.

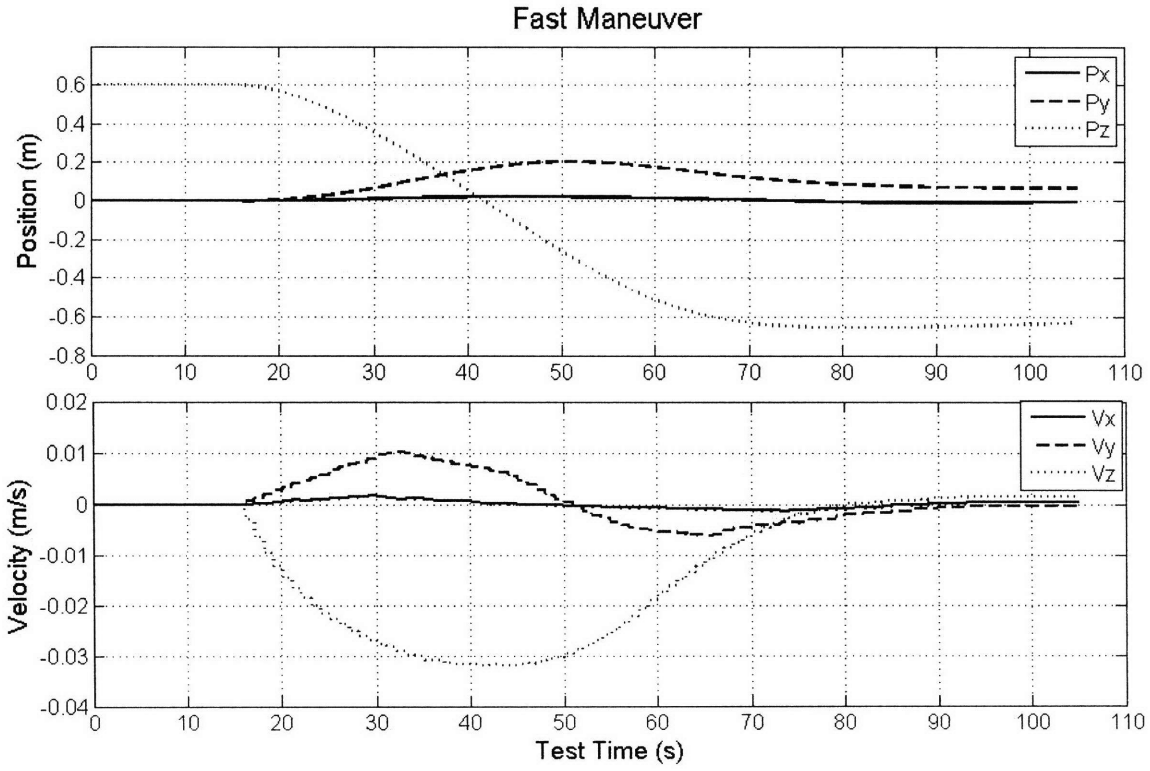


Figure 4-5. Maneuver performed with a value of 0.5 for v_{max}.

4.4 Pointing

For most inspection operations, an imaging system needs to point at the surface of the target. One of the useful aspects of the NavZ algorithm is that the inspector's velocity and path are mostly parallel to the surface of the target. This aspect is useful when it comes to the automated pointing control that is introduced in Section 2.2. Knowledge of the location of the target surface is limited by knowing only the location of feature points. Because these feature points are discontinuous, this could make a smooth automated pointing algorithm difficult to implement. However, if one face of the inspector is controlled to always point along the parallel path, then an adjoining face to that one can be controlled to always point towards the target's surface. Thus, the smooth paths of the combined LQR and APF can aid automated pointing.

As a first implementation of this automated pointing, the inspector's +y face was controlled to point in the direction of the velocity vector, and the -z face was controlled to face the target surface. The -z face contains the tank, which will emulate the camera. First, the velocity of the inspector is determined in laboratory coordinates from the SPHERES sensors. Then, the quaternions needed to point the satellite in the direction of that velocity are found using the method in Reference [24]. The Euler angle rotations that would point the +y face to the velocity vector are found using Equations 4.2-4.4.

$$\theta = \tan^{-1}\left(\frac{V_z}{V_y}\right) + \pi, \quad V_z < 0 \tag{4.2}$$

$$\theta = \tan^{-1}\left(\frac{V_z}{V_y}\right) - \pi, \quad V_z > 0 \tag{4.3}$$

$$\Phi = \frac{\pi}{2} + \tan^{-1}\left(\frac{V_x}{\sqrt{V_y^2 + V_z^2}}\right) \tag{4.4}$$

where V_x, V_y, and V_z are the coordinates of the velocity vector in the ISS test-volume coordinate system. Equations 4.2 and 4.3 find the angle Θ that will rotate the SPHERE about its x-axis so that its -z face points towards the velocity vector. Next, Equation 4.4 finds the angle Φ that will rotate the SPHERE about its z-axis so that its +y face points towards the velocity vector. After Θ

and Φ are calculated, then the rotation matrix A is found using Equations 4.5- 4.7. Lastly, the quaternions that would give these rotations are found using Equations 4.8-4.11.

$$A_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \tag{4.5}$$

$$A_z = \begin{pmatrix} \cos\Phi & \sin\Phi & 0 \\ -\sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

$$A = A_z A_x \tag{4.7}$$

$$q_1 = \frac{1}{2}\sqrt{1 + A_{11} - A_{22} - A_{33}} \tag{4.8}$$

$$q_2 = \frac{1}{4q_1}(A_{12} + A_{21}) \tag{4.9}$$

$$q_3 = \frac{1}{4q_1}(A_{13} + A_{31}) \tag{4.10}$$

$$q_4 = \frac{1}{4q_1}(A_{23} - A_{32}) \tag{4.11}$$

This pointing algorithm was implemented in the SPHERES MATLAB simulation. The quaternions were commanded and controlled using a SPHERES PD attitude controller. Figure 4-6 shows the results of one maneuver. Note how one face points towards the velocity vector, which is parallel to the navigation-zone surfaces. Also, the tank face points towards the vehicle.

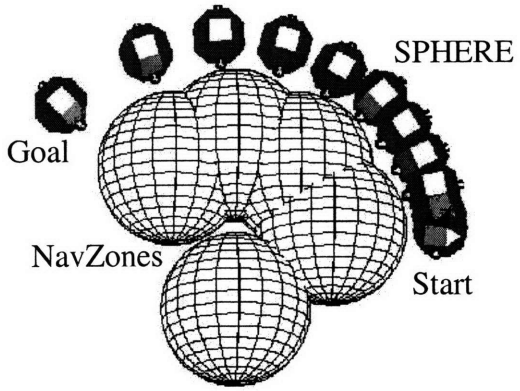


Figure 4-6: Simulation of maneuver with the pointing algorithm.

One of the weaknesses in this pointing algorithm is that when velocities become very small, the noise in the velocity measurements is larger than the commanded velocity. When this is the case, then the velocity magnitude is no longer parallel to the target surface. To handle this aspect of the pointing algorithm, whenever the inspector comes within 15 cm of the target point, its quaternions are set to a default value with the tank face pointing downward in the ISS test volume.

4.5 Waypoints and Active Handling of Local Minima

During an inspection operation, the pilot of the satellite will need the ability to actively set waypoints for the satellite to follow. An algorithm for waypoint following was designed, simulated, implemented on SPHERES, and tested. For a series of maneuvers, the locations of waypoints are commanded. Then the NavZ algorithm plans the path and controls the satellite to maneuver through those waypoints.

Initially, an on-board counter is set to zero and the first waypoint is set as the goal point. LQR plans a path to that waypoint with APF modifying that path to go around the NavZones. During each one-second control cycle, the variable r_{goal} that is used in Equation 3.6 is checked. If r_{goal} is less than 6.0 cm, then the body of the SPHERE is touching the waypoint. During the control cycle, if r_{goal} is less than 6.0 cm, then the counter is increased by a value of one. A second later, the control cycle is run again. If the counter increases to two, then the SPHERE has been touching the waypoint for more than one second. At this moment, the counter is reset to

zero, and the next waypoint is set as the goal point. Each waypoint is set as the goal point in turn, until the inspector has maneuvered through all of the waypoints.

This waypoint algorithm was implemented in the MATLAB simulation, and Figure 4-7 shows a maneuver that includes two waypoints. The satellite first flies to the waypoint at the bottom of the test volume, then flies up to a second waypoint on the right side of the test volume.

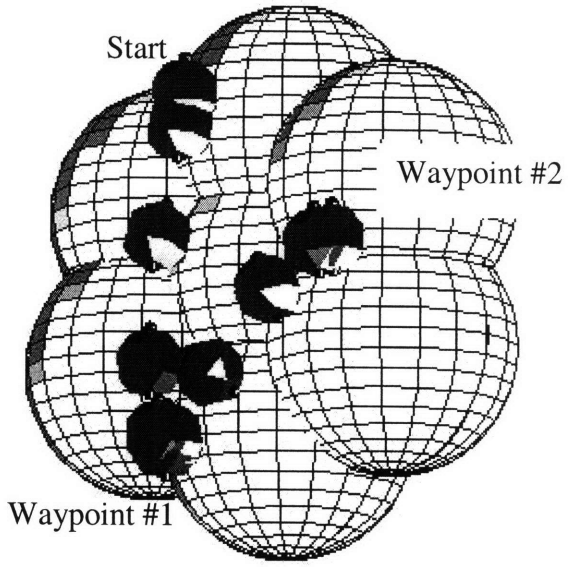


Figure 4-7. Snapshots of a satellite performing two maneuvers to two different waypoints in simulation.

After simulating this algorithm, it was tested on the SPHERES table. Because the forces of the NavZ controller are very small, the commanded thruster firings are not large enough to overcome the friction of the table. However, the objective of this test was to check that the counter incremented properly, and that the goal point switched when the SPHERE reached a waypoint. Thus, the movement of the satellite was assisted by gently pushing it in the directions that the thrusters were firing. When the satellite reached a waypoint, the telemetry showed that the counter increased until it reached two. Then the waypoint was changed and the counter reset as expected. The satellite successfully went through five waypoints during this test.

This waypoint algorithm can also be used to handle local minima within the NavZ controller. One of the weaknesses of some APF path planners is that there are local minima. A local minimum occurs when the LQR commanded accelerations are canceled by the APF accelerations. If the remaining LQR forces are smaller than the minimum thruster firing force, then the satellite cannot implement those forces and will remain stationary. In the NavZ controller, this only

happens if the satellite is inside a concave surface shaped by the navigation zones, or if it is on the exact opposite side of a navigation zone from the target point. The pilot of the inspector satellite can actively handle these local minima by setting an additional waypoint. This approach is similar to the wall-following algorithm in Reference [25], in that the satellite follows the wall until it gets out of the local minimum. The robot in Reference [25] remains completely autonomous by choosing a semi-random direction to leave the local minimum. On the other hand, the NavZ utilizes the knowledge of the human pilot to add a waypoint. Thus a human who is in-the-loop chooses the direction that the satellite leaves the local minimum.

4.6 Getting Out of a Navigation Zone

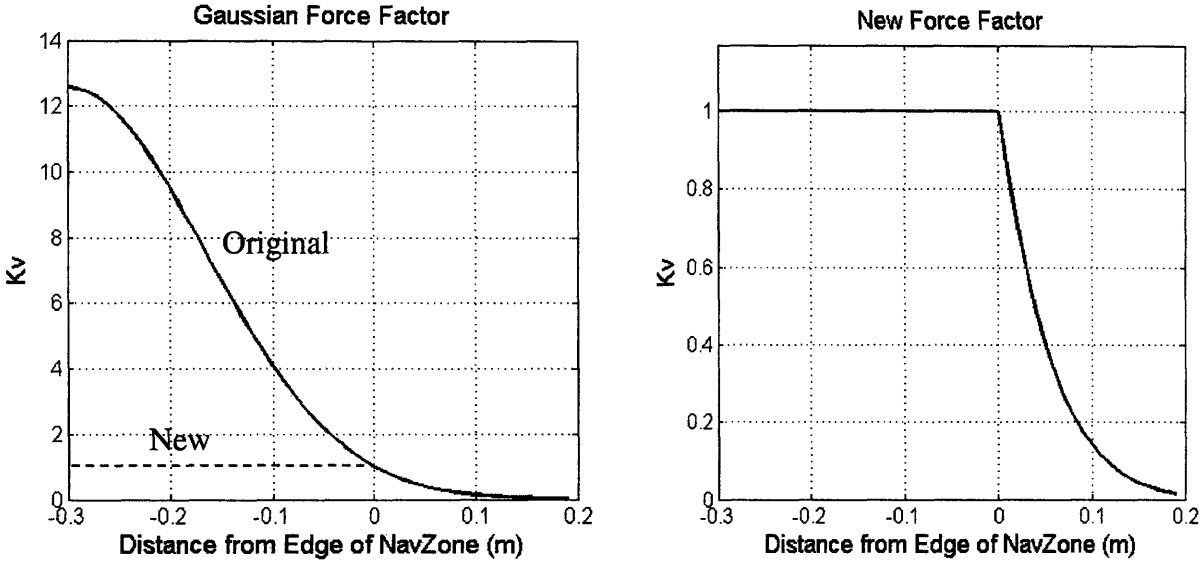
In the first ISS test of the NavZ algorithm, the SPHERE began its maneuver inside a NavZone. Because the strength of the APF forces increase exponentially as the satellite gets closer to the NavZone, the satellite was pushed very rapidly away from the NavZone. The algorithm's original design was for a mission in which satellites approach each other from a distance, and it was not designed to handle the case when a satellite begins inside the NavZone. However, in inspection operations it is possible to accidentally set the navigation zones such that the satellite is inside of them.

For example, an inspector could be located at four meters away from the target surface, and then it is commanded to fly a maneuver at 5 meters above the target surface. In this instance, the inspector is inside the NavZones. The correct procedure would be to relocate the inspector to five meters, then begin the NavZ maneuver. However, the algorithm should be designed to automatically handle a situation in which the satellite is inside a NavZone.

To improve the NavZ algorithm for handling this case, the values of the APF forces are modified if the inspector is inside a zone. Instead of continuing to increase exponentially inside the zone, the APF forces remain constant, set to their level on the edge of the zone. Also, the LQR forces are set to zero until the inspector has successfully cleared the NavZone. With this design, APF forces moderately push it out of the NavZone, and then the LQR & APF control comes online once it has escaped the zone.

To alter the APF forces, Equation 3.10 was modified so that the force factor k_v levels off and remains constant when the satellite is inside a NavZone. Figure 4-8a shows the original k_v over a range of distances outside and inside the NavZone. Figure 4-8b shows the new k_v that levels off

inside the NavZone. For these graphs, the stopping distance in Equation 3.10 has been set to 5.0 cm, and the NavZone size has been set to 0.3 m.



a) Old force factor b) New force factor

Figure 4-8. APF force factor, k_v, over a range of distances from the edge of a NavZone.

This APF algorithm revision was implemented in the MATLAB simulation. The inspector starting position and NavZones were set to the same locations as the ISS test discussed in Section 3.1.4. As shown in Figure 4-9, the satellite is pushed out of the navigation zone in a slower and better-controlled fashion than the maneuver seen in the first ISS test.

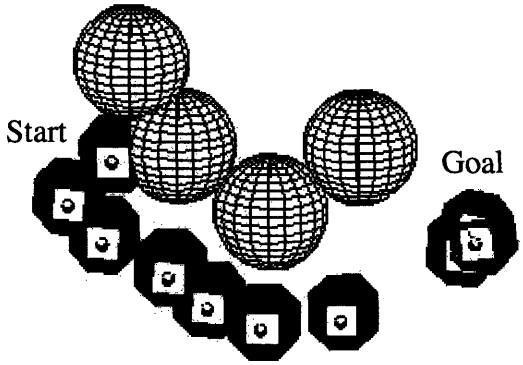


Figure 4-9. Snapshots of a simulation in which the satellite maneuvers out of a NavZone using the new force factor distribution.

4.7 Replanning Path Due to Previously Unknown Obstacles

One useful quality of the NavZ path planner is that it can handle navigation without a-priori knowledge of the target spacecraft's geometry or with the emergence of unexpected obstacles. As discussed in Chapter 3, the inspector satellite does not have APF accelerations from a NavZone until it is within its influence zone. Thus, it adjusts its path online once it "sees" a new feature point.

Figure 4-10 shows a simulation in which there is an obstacle (OBS) that sticks out from the rest of the navigation zones. When the satellite approaches the top of the vehicle, it enters the influence zone of this protruding obstacle. At this point, it "sees" the obstacle, and the APF alters the path around the obstacle.

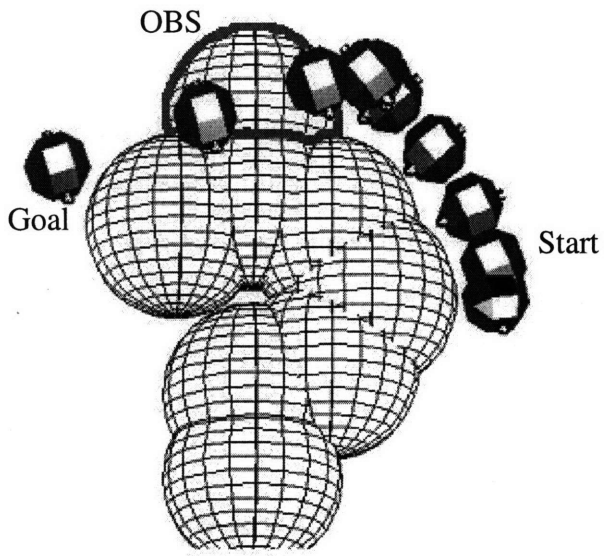


Figure 4-10. Simulation of an inspector satellite maneuvering around an unexpected obstacle.

There are several situations in which this aspect of NavZ is useful. For example, mission control could suspect that a CEV solar panel has not deployed correctly. As long as the computer-vision algorithm notices the new locations of the feature points, the NavZ algorithm places NavZones at those new locations. Another example is the inspection of an unknown satellite. An inspector approaches a satellite with an unknown configuration, the computer-vision system determines feature points, then the NavZ places NavZones around those feature points. Thus, NavZ can handle unknown spacecraft shapes or unknown obstacles. Navigation in

these situations is not limited by NavZ, but is limited by the ability of the computer-vision to accurately identify feature point locations. The next chapter will discuss emulating computer-vision and some of its weaknesses.

4.8 Summary

After the first flight test of the NavZ algorithm aboard the ISS, several improvements were incorporated into NavZ. First, the size of the influence zones was adjusted so that the satellite would closely follow the edge of the NavZones. With this improvement, the pilot of the inspector satellite can now expect the satellite to fly approximately at a height equal to the NavZone size. Another issue discovered during the first flight test was that the satellite moves out of a NavZone too rapidly. Thus the forces inside NavZones were adjusted so that the satellite leaves the zone in a slower manner.

Later, several user choices were added to the satellite operation. The ability for the pilot to change a maneuver's speed and height was implemented and simulated. Additionally, waypoint commands were incorporated into the algorithm. With these improvements to NavZ, the inspector satellite pilot now has more control over the flight.

Lastly, the ability for the algorithm to adapt to previously unknown obstacles was simulated. This situation may be useful if the target spacecraft has an unknown shape, or if its shape has changed unexpectedly. For example, the CEV's expected shape could be distorted if one of the solar panels did not deploy properly.

With these new developments for NavZ, the algorithm is now more capable for use in inspection operations. After these developments were completed, some improvements to the emulation of the computer-vision information were incorporated into the NavZ simulations. These emulation improvements are discussed in Chapter 5. Also, several of these developments were integrated together into flight-tests for the ISS. These tests are discussed in Chapter 6.

5 Emulating Vision Based Navigation (VBN)

For the NavZ algorithm, one of the key sensors is a camera system used for computer vision. The block diagram in Figure 5-1 shows how these sensors fit into the simplified inspector system model. For vision-based navigation (VBN), the camera system would locate the feature points on the inspected vehicle. As discussed in Section 1.3, these feature points are then the locations of the navigation zones. While the NavZ algorithm was designed and implemented, other members of the SPHERES team were developing a computer-vision system for the satellites. The goal was to develop the camera and control system designs in parallel, so that they would interface with each other. To ease the integration of the NavZ with the evolving computer vision system, the information received from the cameras was emulated. Section 5.2 discusses the initial VBN-data emulation that was used for NavZ. This emulation was used for the simulations and tests in Chapters 3 and 4. Then Section 5.3 discusses an improvement to this emulation in which the SPHERES navigation beacons are placed at the locations of the feature points in simulation. Section 5.3 also discusses the position uncertainties that are introduced because of this beacon placement. Section 5.4 discusses another improvement to the VBN emulation, in which a position uncertainty of 0.02 meters is incorporated into the NavZ algorithm. Lastly, Section 5.5 shows how the NavZ algorithm can adjust and still work when the computer-vision loses track of a feature point.

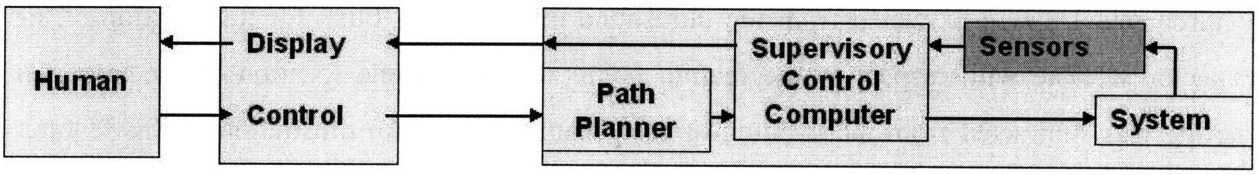


Figure 5-1. The camera system used for VBN is a part of the Sensors block in this inspector satellite system model.

5.1 Navigation Requirements for the Computer Vision System

The NavZ algorithm was developed with some assumptions about the information received from the computer-vision system. These assumptions were defined after several iterations of exchanging information and discussions with the team working on the computer-vision system.

Based on these assumptions, the following requirements were established for the computer vision system so that the NavZ would interface with it:

0. The computer vision system will identify feature points on the target spacecraft's surface.
1. The computer vision system will determine the distance of the inspector satellite from feature points.
2. The computer vision system will determine the location of the inspector satellite with respect to feature points using a reference coordinate system defined below.
3. The computer vision system will determine the orientation of the inspector satellite with respect to feature points.

For testing inspection algorithms on the ISS, the laboratory coordinate system would be the reference. And for an inspector satellite, the reference coordinate system would be that of the target spacecraft.

5.2 Initial Computer Vision Emulation Using SPHERES Ultrasound

During the first implementation of NavZ with SPHERES, the computer-vision information was emulated using the SPHERES ultrasound system. The locations of the feature points were predetermined and coded into the controller. Using the ultrasound system, the satellite location was estimated in the ISS laboratory frame. Then, the information for computer-vision requirements 1-3 was extracted from the ultrasound information. First, for the position of the inspector satellite with respect to the feature points, the ultrasound location of the inspector satellite was subtracted from the location of each feature point. An illustration of this feature-point vector extraction is shown in Figure 5-2. Second, the distance to each feature point was determined by finding the magnitude of each of those vectors.

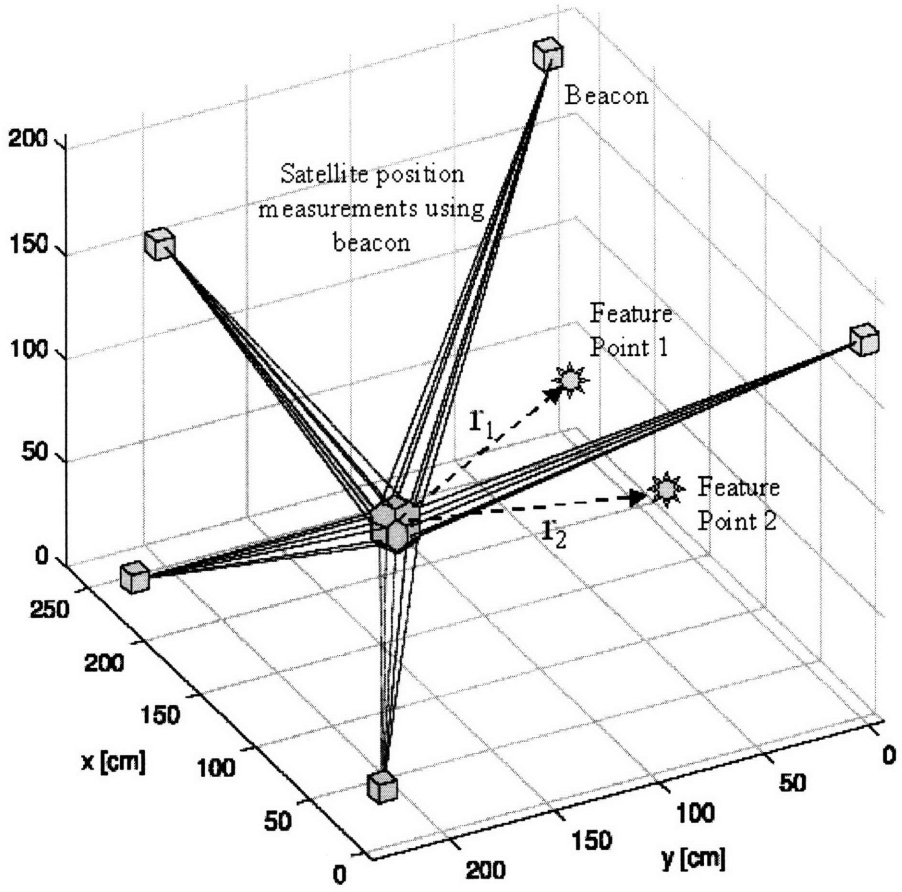


Figure 5-2. Satellite position vectors determined by the ultrasound beacons. The vectors from the satellite to the feature points are then extracted from this data.

All of the results in the previous sections of this document utilized this method. The performance of the satellite maneuvers has shown a position error of less than 2.0 mm. This small error is due to the 2.0-millimeter position uncertainty of the SPHERES ultrasound estimators. However, most computer-vision algorithms have different levels of position and attitude uncertainties than ultrasound systems. To introduce some of the position uncertainty that computer-vision exhibits, the emulation was improved by placing the SPHERES navigation beacons at the feature points in simulation. This development is discussed in Section 5.3

5.3 Placement of Beacons at Feature Points

The next step towards improving the emulation was placing beacons at the feature points in the simulation. VBN measures the locations of the feature points relative to the body frame of the inspector satellite. This aspect of VBN can be mapped into the ultrasound system by placing

the beacons at feature point locations. Then the satellite only receives position information from those points.

In the SPHERES simulation, a surface was designed as shown in Figure 5-3. Three beacons were placed in the center of the NavZones, to emulate feature points. Also, two additional beacons were placed in front of and behind the NavZones as shown in the figure. All of the beacons were rotated in the simulation so that they faced downwards.

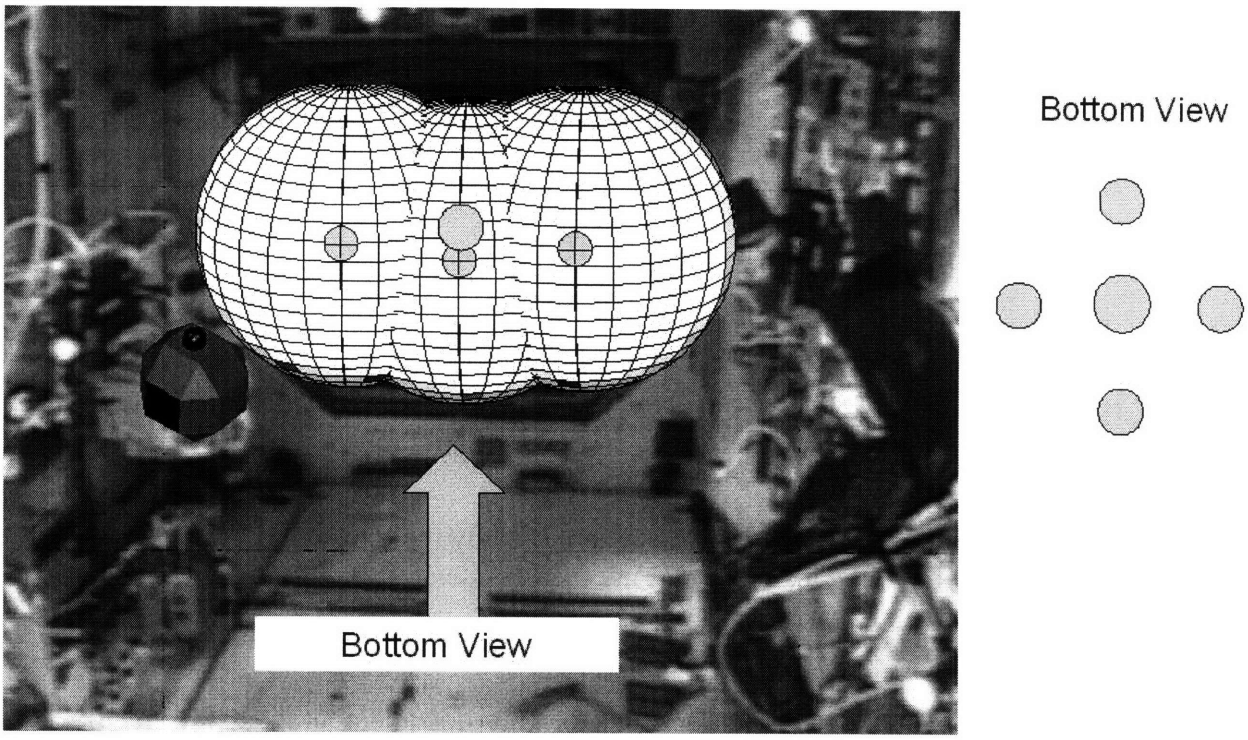


Figure 5-3. Simulation design, with beacons located inside the NavZones. The bottom view illustrates the additional two beacons in front of and behind the NavZones.

This simulation was run first using the SPHERES global estimator with the beacons located at the edges of the test volume. Then, the simulation was run with the beacons at the feature points. Figure 5-4 shows the path of the inspector satellite. The simulated telemetry results in Figure 5-5 and Figure 5-6 show that performance of the satellite in both simulations is very similar. In these graphs, the dashed lines are the x, y, and z positions estimated by the satellite in the simulation, and the solid lines are the actual positions of the satellite in the simulation. The estimator converged and the satellite was able to navigation around the zones. Note how the errors between estimated and actual position of the satellite are slightly larger when the beacons are placed at the feature points than when they are placed at the edges of the test volume.

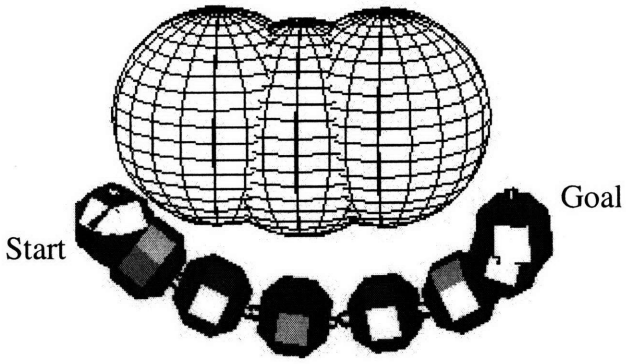


Figure 5-4. Path of satellite maneuver for simulation of emulated VBN.

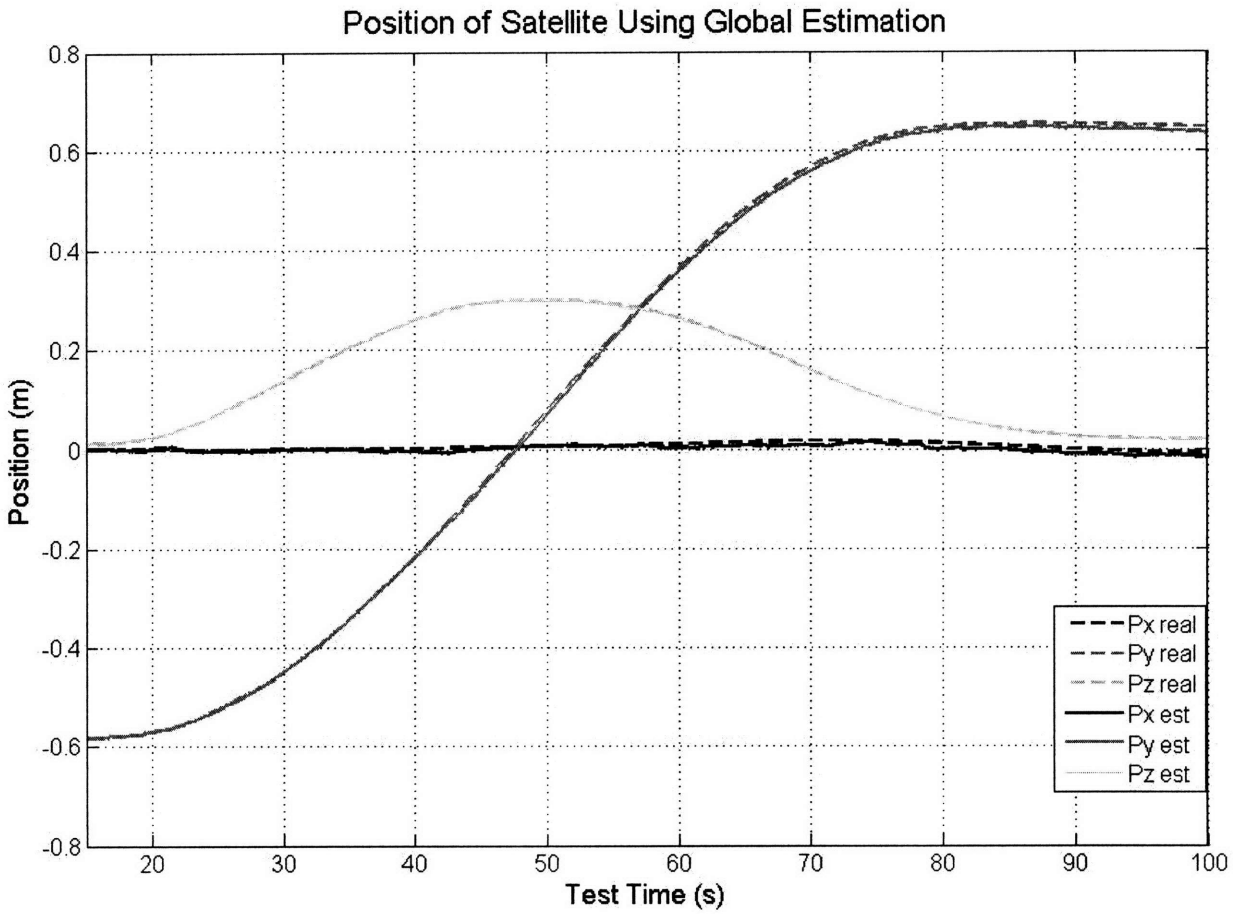


Figure 5-5. Simulated position of the satellite as it maneuvered using the SPHERES global metrology in simulation.

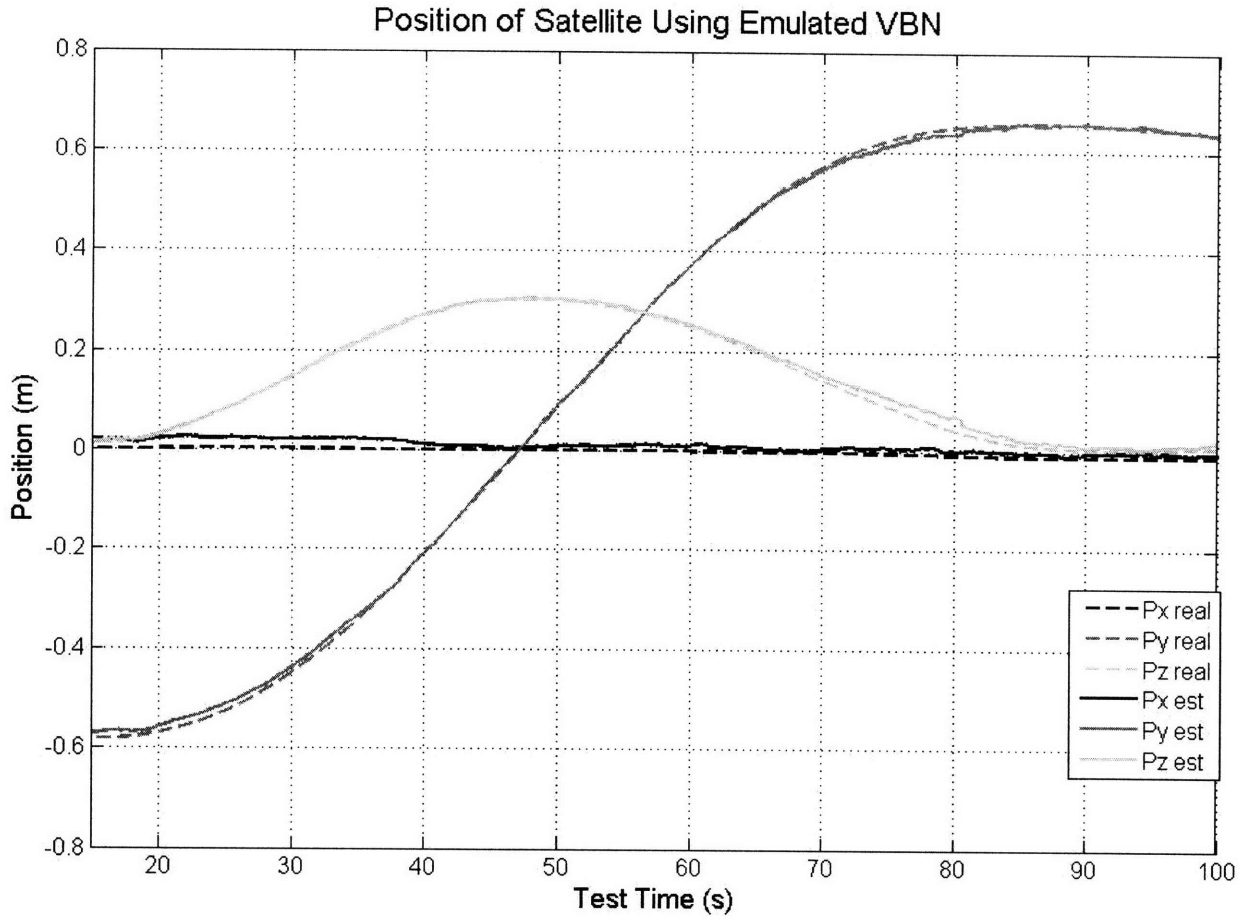


Figure 5-6. Simulated position of the satellite as it maneuvered with the beacons at the feature points.

Looking at the geometry of this new beacon set-up can help explain why the estimation errors are slightly larger in Figure 5-6 than in Figure 5-5. Because the beacons are placed on a flattened surface when they are at the feature points, error in the estimation of shear translation is introduced. Shear translation is the movement of the inspector satellite parallel to the target spacecraft's surface.

With the ultrasound system, any movement in shear translation must be measured through changes in the ranges from the beacons. Because the ultrasound beacon has a position uncertainty of 2 mm, this uncertainty introduces a shear translation uncertainty. The purpose of the following calculations is to find the minimum change in shear-translation position that can be detected by the SPHERES ultrasound system.

In Figure 5-7, L_1 is the range from the satellite to a feature point. When the satellite moves $\Delta\delta$, its new range to the feature point is the variable L_2. During this maneuver, the range changes by ΔL. Equation 5.1 shows the relationship between L_1, L_2, and ΔL. Rearranging the equation and taking the Taylor Expansion gives Equation 5.2. Then, rearranging the terms gives Equation 5.3 for ΔL, which is the change in the range due to the shear-translational movement. Finally, the Equation 5.4 gives $\Delta\delta$. If ΔL is the set to equal the 2.0-mm minimum detectable range change, then $\Delta\delta$ is the minimum detectable shear-translation change.

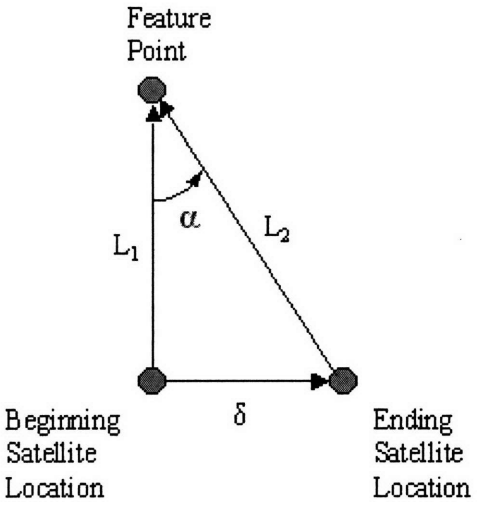


Figure 5-7. Diagram of the geometry used for calculations of the shear translational error using SPHERES ultrasound system.

$$L_2^2 = L_1^2 + \Delta\delta^2 \tag{5.1}$$

$$\frac{L_2}{L_1} = \sqrt{1+\left(\frac{\Delta\delta}{L_1}\right)^2} \approx 1 + \frac{1}{2}\left(\frac{\Delta\delta}{2}\right)^2 \tag{5.2}$$

$$\Delta L = L_2 - L_1 \approx \frac{1}{2}\frac{\Delta\delta^2}{L_1} \tag{5.3}$$

$$\Delta\delta \approx \sqrt{2\,\Delta L\, L_1} \tag{5.4}$$

Using Equation 5.4, if ΔL is set to the range uncertainty of 2.0 millimeters and L_1 is set to 0.5 meters, the shear translation uncertainty $\Delta\delta$ is approximately 4.5 centimeters. Table 5.1 shows a few ranges, and their corresponding shear translation uncertainties.

Table 5.1. Shear translation uncertainty due to range error at several ranges from a feature point

Range from Beacon L_1 (mm)	Ultrasound Range Uncertainty ΔL (mm)	Shear Translation Uncertainty δ (mm)
100	2	10
500	2	45
1000	2	63

Next, the relationship between the range resolution and the attitude resolution will be discussed. A SPHERE has four ultrasound sensors on each face located 50 mm apart from each other. An assumption in the following calculations is that whenever a sensor has moved so that its change of range to a beacon is over 2.0 millimeters, then the system detects that change. In Figure 5-8, the satellite changes its pitch by an angle of φ. The angle φ is related to ΔL in Equation 5.5. If ΔL is set to 2.0 millimeters, then φ is the angular resolution of the satellite during pitch and yaw maneuvers. The satellite must rotate by at least φ degrees before its estimator can determine that it is rotating. Thus, the angular resolution of pitch and yaw rotations is approximately 2.3 degrees. Pitch and yaw rotations are along axes that do not point towards the beacon or feature point.

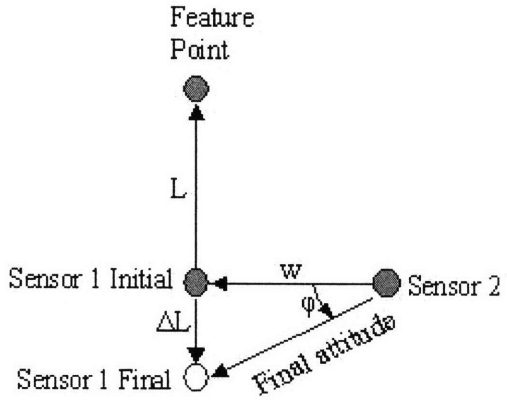


Figure 5-8. Diagram of the geometry used for calculations of the pitch and yaw rotational resolution using the SPHERES ultrasound system.

$$\varphi_{\text{pitch, yaw}} \approx \frac{\Delta L}{w} = \frac{2\,\text{mm}}{50\,\text{mm}} = 0.04\,\text{radians} = 2.3^\circ \tag{5.5}$$

For roll around the axis which runs between the satellite and the feature point, no change in the roll angle will give a ΔL change in the range of the sensors. Thus, the angular resolution for this pure roll situation is undefined.

$$\varphi_{\text{roll}} = \infty \tag{5.6}$$

As a shown from the calculations in this section, placing the beacons at the feature points in the simulation introduces a new set of position and rotational uncertainties other than the original 2.0-millimeter uncertainty. These range, pitch, yaw, roll, and shear-translation resolutions developed in this section will be used in the next section to improve the emulation of VBN. Also, because the estimator still worked with this beacon modification, this simulation was then modified further to emulate VBN errors.

5.4 Incorporating VBN Errors into Emulation

When a computer-vision system is used, it has different position and attitude uncertainties than those of an ultrasound system. These uncertainties are affected by how the VBN system works. Figure 5-9 illustrates a basic computer-vision algorithm, in which a tetrahedral feature point is mapped onto a camera-image plane to determine the movement of the satellite. The points are shown as only filling one pixel for illustrative purposes, but would fill multiple pixels in reality. As the satellite moves to a closer range, the images spread out, and the motion is determined when the points enter the adjoining pixels. Likewise, during rotation maneuvers, the points move along the camera image plane as shown in Figure 5-9 (d) and (e).

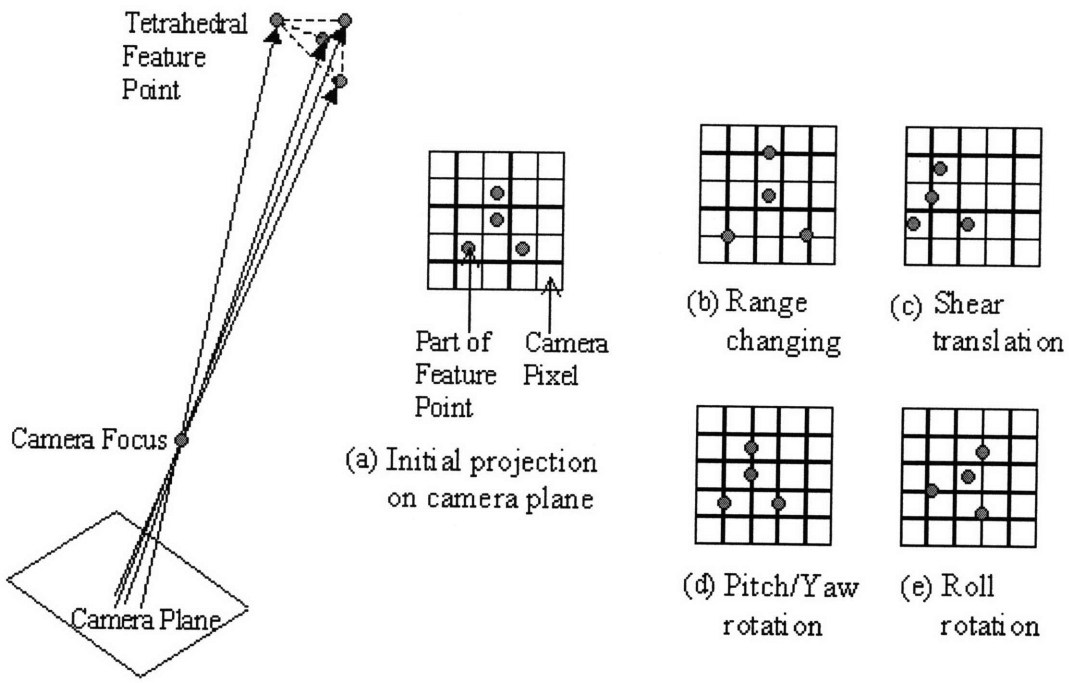


Figure 5-9. Illustration of how rotational and translational satellite movement maps into a camera image.

The algorithms have a greater challenge detecting shear translation, because it looks very similar to pitch/yaw rotation, except that there is image distortion. Partially because of this ambiguity between pitch/yaw rotation and shear translation, these measurements in satellite movement are not as accurate as those of range and roll. Figure 5-10 illustrates how the accuracies of position and attitude measurements compare between ultrasound and VBN. In general, the VBN range errors are larger than the two-millimeter ultrasound accuracy [26]. Meanwhile, the changes in attitude using VBN (particularly roll) are easier to detect than those using ultrasound. Looking at Figure 5-9, very small changes in attitude will move the pixel-locations of features, thus making rotation easy to detect.

Overall, the computer vision is strong at determining attitude, because of the way that attitude-change maps into the pixels. Its weakness is distance, because it has to extrapolate distance information from the pixel mapping. However, the ultrasound system's strength is in determining position, because this is what it initially measures with its sensors. Its weakness is attitude, because it has to extrapolate attitude information based upon the distance measurements.

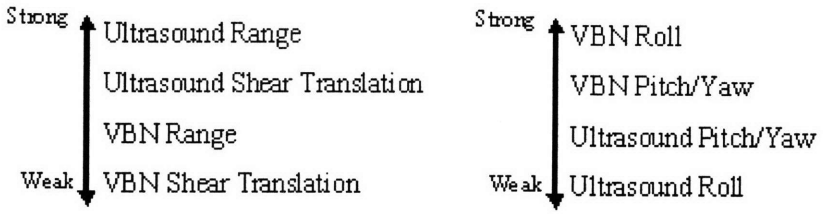


Figure 5-10. Ranking the strengths and weaknesses of detecting changes in position and rotational movement using ultrasound or computer vision.

To improve the emulation of VBN data while still using the SPHERES ultrasound system, the uncertainty in the VBN distance measurements can be artificially inserted into the positioning algorithm. Fortunately, one can take very accurate position data and add uncertainty. Thus, adding position uncertainties similar to those seen using computer vision can improve the emulation of the VBN.

In the VBN emulation for the NavZ controller, position uncertainties were incorporated. The values of these uncertainties were determined empirically. An algorithm was chosen that was similar to one that could be used for CEV inspection operations. For inspecting the CEV, two-dimensional feature-point markers may be preferred over three-dimensional markers. In such a case, the markers would not stick out from the vehicle's body. An algorithm developed by Coelho, Campos, and Kumar uses 2-D markers shown in Figure 5-11 [27].

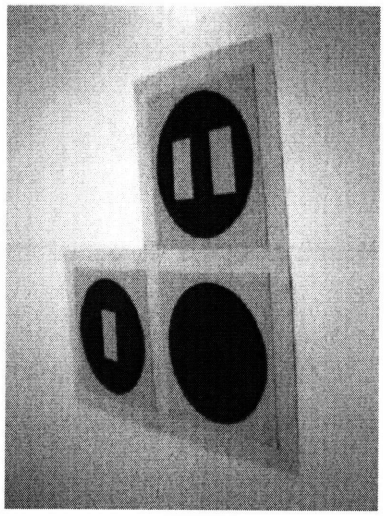


Figure 5-11. Two-dimensional marker for computer-vision navigation designed and tested by Coelho, Campos, and Kumar at 0.5-meter distances [27]

Coelho, Campos, and Kumar measured the accuracy of the algorithm in hardware through nine tests. Three orientations of the marker were tested. During each test, the camera system

was placed at a distance between 0.45 and 0.66 meters from the marker. The range errors that were measured were between -2 cm and 2 cm [27].

Because this algorithm uses 2-D markers and was tested at distances similar to those used by inspection operations, the data about its position uncertainties has the order of magnitude that will probably be seen when SPHERES uses an actual computer vision system. Thus, these position uncertainties of 2 cm were incorporated into the NavZ emulation.

First, the variable 'uncert' that goes into Equation 3.12 was changed from 0.002 to 0.02 meters. Then, a random uncertainty that ranges between -2 cm and 2 cm was added to the measurement of r_o, which is then used in Equation 3.10 to determine the affect of the APF. The same maneuver was performed as in Figure 5-4. Figure 5-12 shows the simulated position telemetry of this satellite maneuver with this emulated VBN uncertainty. The dashed lines in Figure 5-12 are the states graphed in Figure 5-6 for comparison. These simulation results show that this new emulation does introduce a greater position error, but the inspector still successfully maneuvers to the desired location.

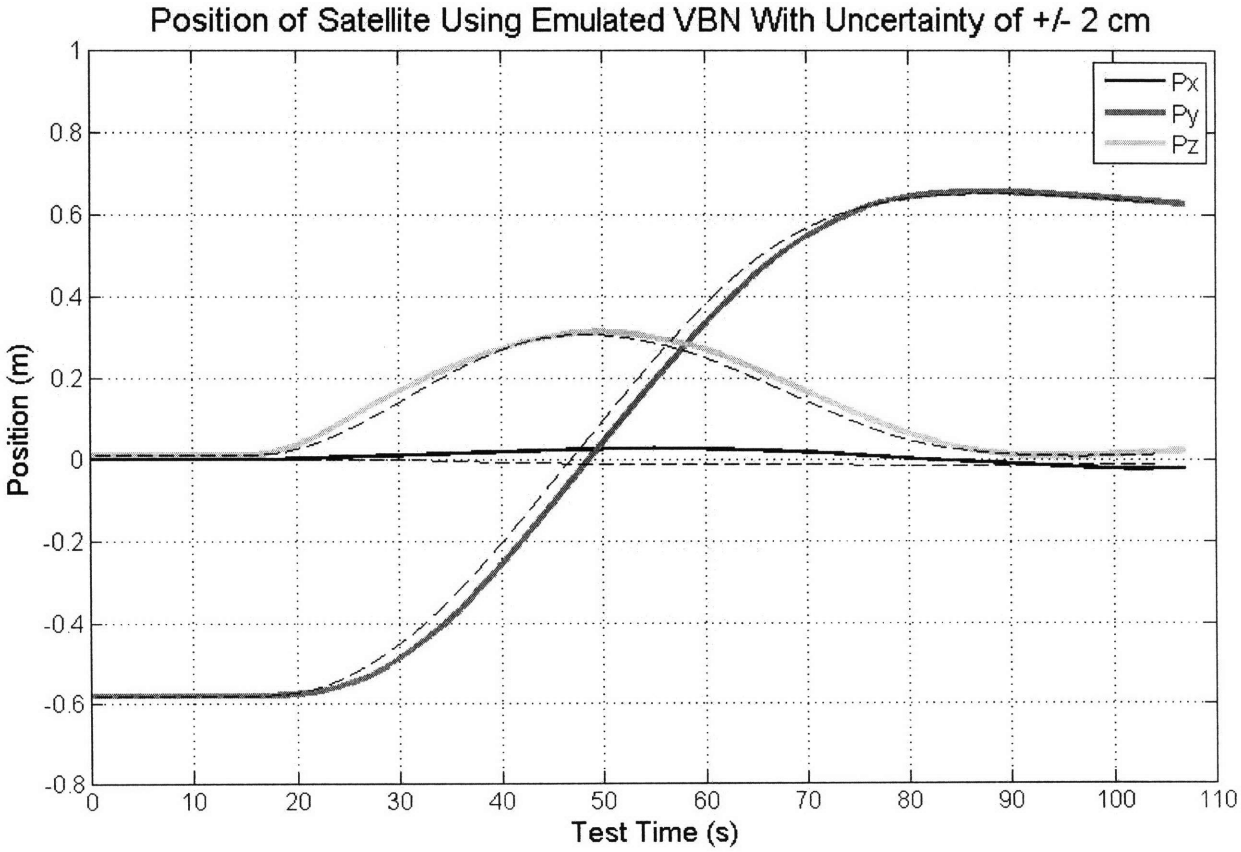


Figure 5-12. Simulated maneuver of satellite with an emulated VBN uncertainty of 2.0 centimeters.

5.5 Actively Handling Failure of Feature-Point Detection

Another aspect of VBN that was not initially emulated is that feature points are not always detected. Based on the information in Reference [26], a computer-vision system can miss or loose track of a feature point. This aspect of computer-vision is one of its weaknesses. Because the NavZ determines the path in real-time for every control cycle, it can adapt if a feature point cannot be detected. Especially in the case of the CEV, with which the shape of the vehicle is known, a parallel computer process can be used to determine if a feature point is missing. Then a NavZone can be added where a feature point should be located. Or, the NavZones surrounding the missing feature point could be stretched into ellipses and connect together to cover the hole in the mesh. Also, in the case of a satellite that does not have a shape known a-priori, a parallel process could be used to detect holes in the mesh. Then, the surrounding NavZones can be stretched so that they connect together.

This idea of stretching the NavZones to cover missing feature points was implemented in simulation. Figure 5-13 shows the beginning of the satellite maneuver, in which all of the adjacent feature points are detected. Forty seconds into the maneuver, a feature point in the middle of the mesh disappears, and the other NavZones are stretched so that they connect to each other. Figure 5-14 shows that the satellite completes the maneuver with the re-shaped NavZones.

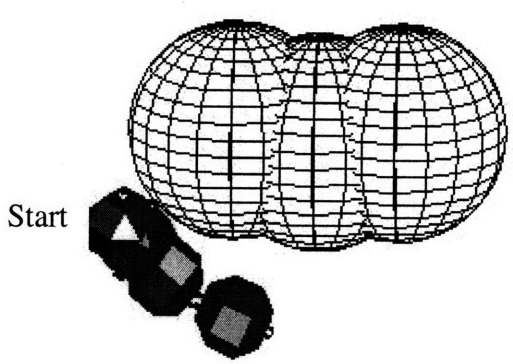


Figure 5-13. Simulation results in which satellite sees three feature points through first 40 seconds of maneuver.

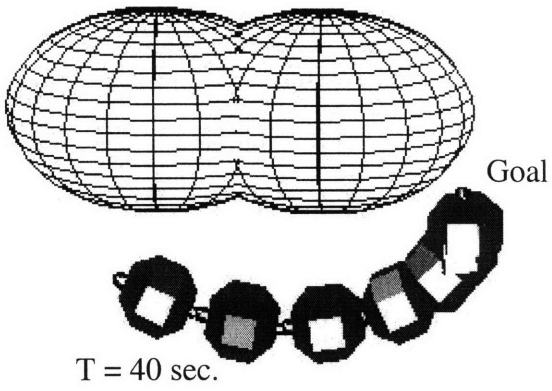


Figure 5-14. Simulation results after 40 seconds of the maneuver, in which the satellite has lost the detection of the center feature point.

5.6 Summary

In the future, the NavZ algorithm will interface with a computer-vision system. In order to facilitate this integration, the emulation of the computer-vision data was improved. In all the previous development of NavZ, the ranges to feature points were extracted from the surrounding ultrasound system. For the first improvement to this emulation, the beacons were placed at the location of the feature points in simulation. This adaptation was one step closer to emulating VBN, because the position information was received directly from the location of the feature points, and a shear translation uncertainty of 4 cm was introduced. Second, a position uncertainty of 2 cm was added to the range-to-feature-point measurements in simulation. This uncertainty was based empirically on the computer-vision tests done by Coelho, Campos, and Kumar [27]. These first two emulation improvements were simulated in the SPHERES MATLAB simulation, and the NavZ algorithm performed well with the satellite navigating along a mesh of NavZones.

Lastly, a simulation was performed of an operations situation in which the camera does not detect a critical feature point. In this situation, the NavZones of the adjacent feature points were adjusted into ellipses to cover the hole in the surface mesh. This simulation showed the adaptability of the NavZ algorithm, and one possible approach towards improving this feature-point-detection weakness in computer-vision navigation.

6 Integrated Tests and 3-D Navigation Around CEV

After the developments discussed in Chapter 4 were simulated, they were integrated onto the SPHERES hardware for two final ISS tests. These tests were designed to perform two complete scenarios of operations inspecting the CEV. Sections 6.1 and 6.2 discuss the design of these tests. Then Section 6.4 shows a simulation of one of the operations in Satellite Tool Kit, including a scale-model of the CEV.

6.1 Integrated Test One

This test is designed to incorporate NavZ maneuvers, waypoints, ellipsoid potential functions, speed change, and height change. Also, this single test demonstrates two different mission scenarios.

Mission 1: The astronauts and mission-control team suspect a problem at a specific location on the CEV surface. The communications antenna has potentially deployed improperly. Inspection will give valuable information for planning an EVA to repair the antenna. An inspector satellite is deployed for this mission. The flight team sends pre-planned waypoints that the inspector follows to the location of concern. Then, the flight team sends a series of real-time commands that will obtain different images of that location. To support this scenario, the NavZones in Integrated Test One are placed in a mesh that has a 5.0-meter-diameter curvature, similar to the CEV hull. One NavZone is placed so that it protrudes from the mesh curvature. This NavZone imitates the location of an antenna.

Mission 2: The MIT students and researchers on the SPHERES team suspect that during the last few months of ISS construction, a new object has been installed that encroaches into the test volume. This object is a large white box. During the previous test session, a SPHERE bumped into it, which notified the team of its existence. The team needs to know the location of this box so that they can avoid collisions with it in the future. One student designs a test in which the SPHERE performs inspection maneuvers along the inside wall of the ISS, and moves in closer to fly around the trespassing box. Figure 6-1 shows a picture of this object. To support both of these mission scenarios with the same test, the mesh described for Mission 1 is placed along the

port wall of the ISS test volume. The NavZone that protrudes from the mesh curvature is placed at the location of the white box. Thus, this protruding NavZone represents both the location of the white box and the location of the CEV antenna.

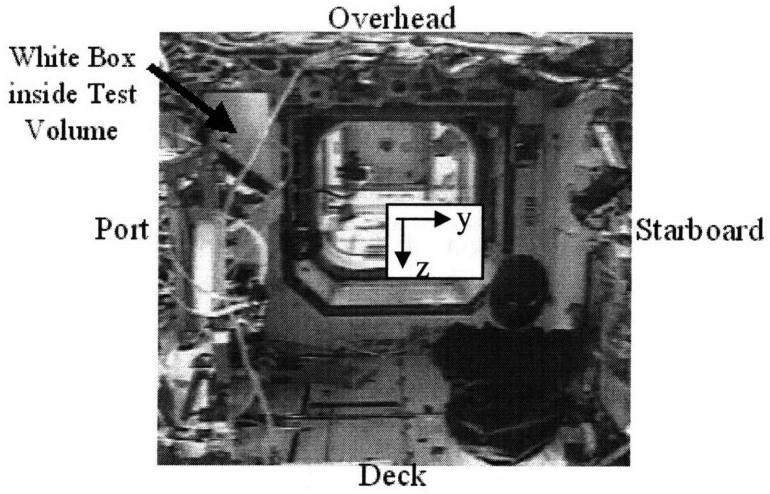
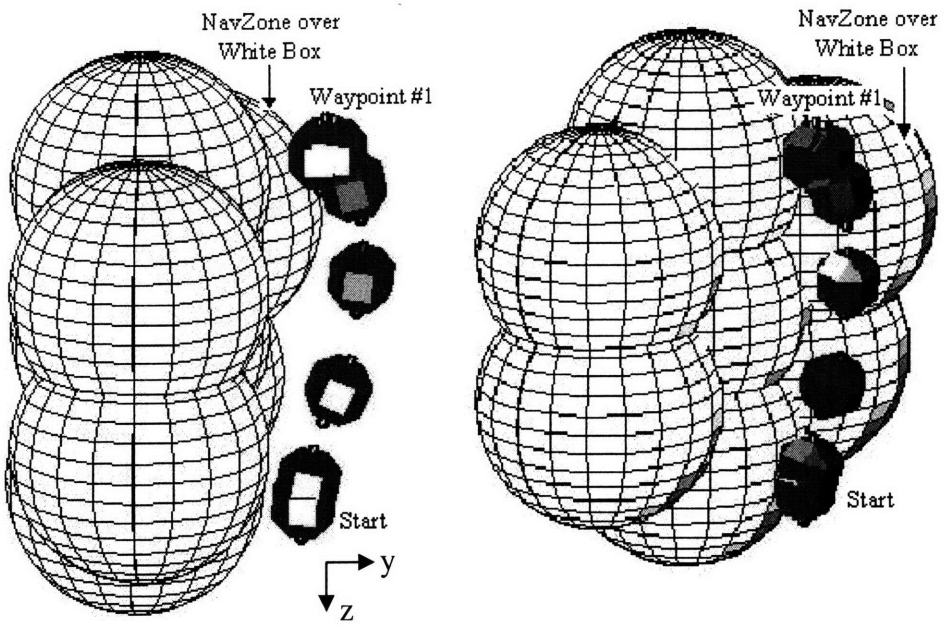


Figure 6-1: The ISS test volume with the encroaching white box located on its port side.

6.1.1 Design and Simulation

For this test, the satellite does three maneuvers that are sequentially closer to the box/antenna. During the first maneuver, the satellite does an inspection fly-by at a height of 0.43 meters and a v_{max} of 0.5 m/s. This v_{max} is larger than it was in Section 3.2.3, so the satellite will maneuver faster than it did in the first ISS flight test. The maneuver starts near the deck of the ISS, and it ends at the first waypoint located near the overhead. In Figure 6-2, the results are shown from the simulation of this maneuver. A snapshot of the position and orientation of the SPHERE is shown for every 10-second interval of test-time. Figure 6-2a shows a front view of the maneuver, from the same viewpoint as the ISS picture in Figure 6-1. In Figure 6-2b, the image has been rotated 45° to give a 3-D viewpoint of the maneuver. As discussed before, this NavZone mesh outlines the surface of a cylinder with a 5-meter diameter, to imitate the surface of the CEV. The NavZone that covers the box is the one that protrudes from the others in the figure. The simulation images in Figure 6-2 show how the first maneuver also curves with the surface.

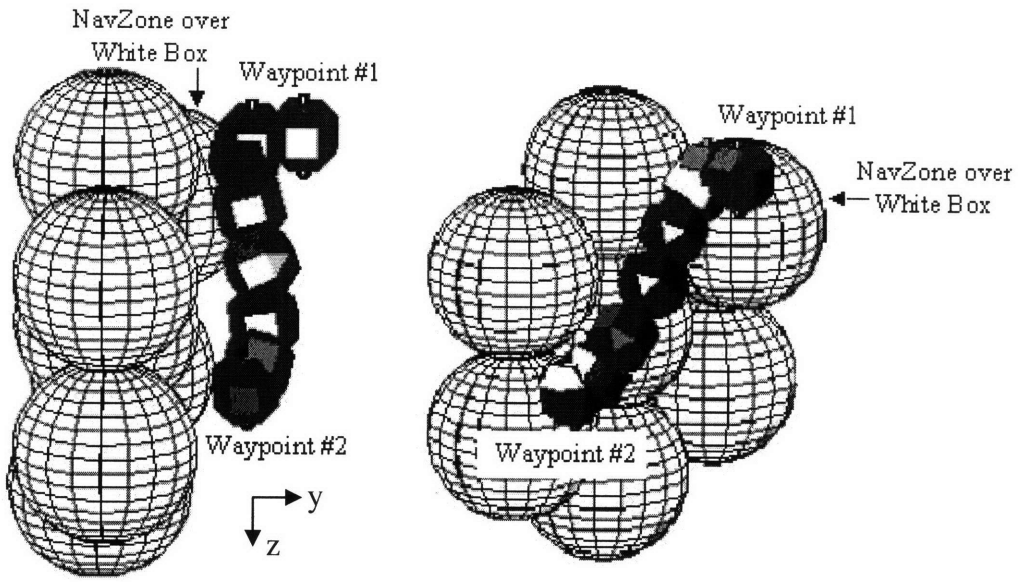


a) Front View b) 3-D View

Figure 6-2: Simulation of the satellite flying the first maneuver.

While the satellite did this first pass, its pilots have noticed the box/antenna that they want to inspect more closely. A new height of 0.27 meters is set, so that the satellite flies closer to the surface. Then, the next waypoint is placed close to the box. The SPHERES PD controller is used to reposition the inspector satellite to the new 0.27-meter height, and then the NavZ is used for maneuvering to waypoint two. The images in Figure 6-3 show a simulation of the second maneuver. Note how the NavZones in Figure 6-3 are smaller than those in Figure 6-2.

After reaching the second waypoint, the pilots want to fly the inspector even closer to the surface and to pass directly over the box/antenna. A new height of 0.08 meters is set. The NavZones shapes are changed to ellipsoids to provide the smaller height but still cover the wall. Again, the SPHERES PD controller is used to reposition to the new 0.08-meter height above the surface. Then, the NavZ is used for maneuvering to waypoint three. Figure 6-4 shows this final maneuver. Note how this final maneuver requires the NavZ to plan a path that avoids the box's NavZone. Additional viewpoints are provided for this maneuver to show how the path wraps around the box.



a) Front View b) 3-D View

Figure 6-3: Simulation of the satellite flying the second maneuver.

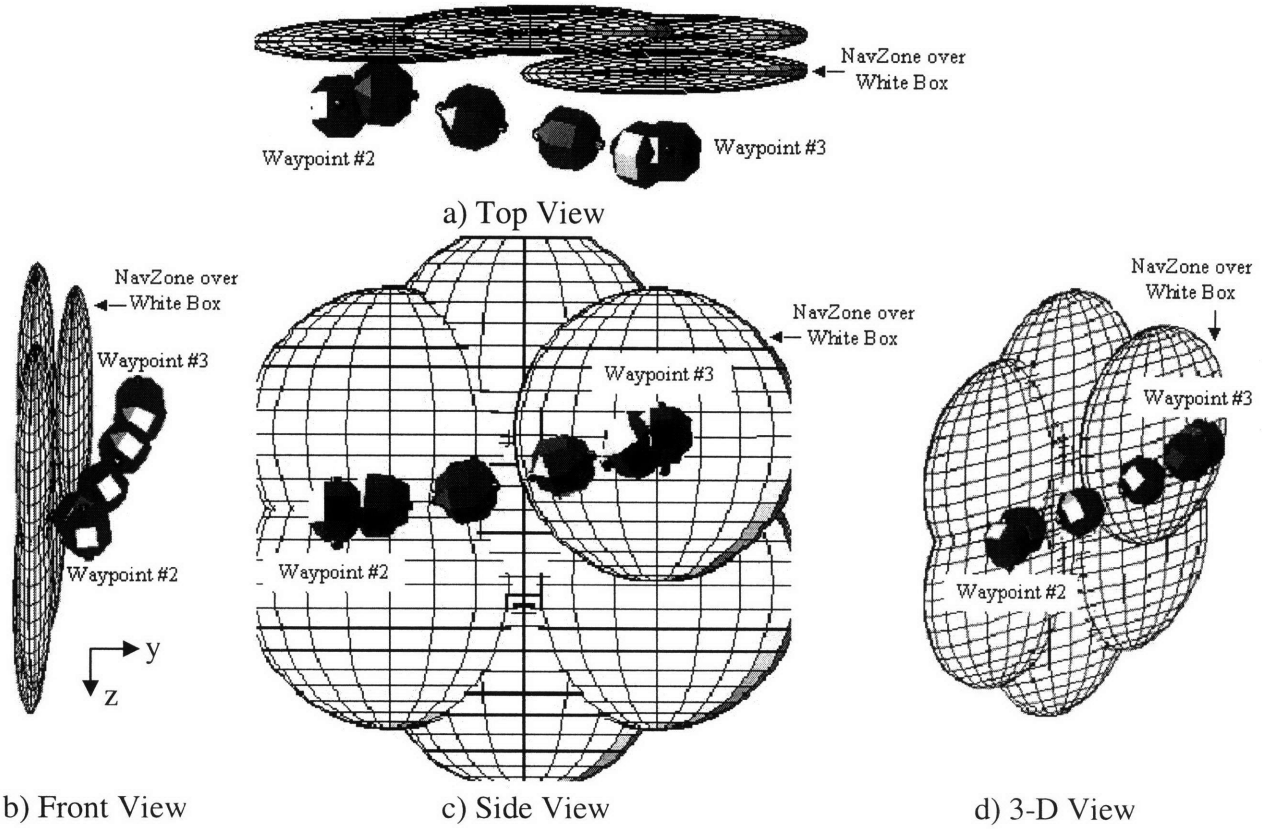


Figure 6-4: Simulation of the satellite flying the third maneuver.

The simulated position and velocity telemetry of the satellite are graphed in Figure 6-5. The graphed position and velocity data are results from the SPHERES MATLAB simulation. Waypoint markers were added to the graph to show the beginning and end of different maneuvers. Also, the approximate lower-limit of *Py* due to the NavZones is shown in Figure 6-5. This limit was determined by calculating the width of the NavZones in ten locations, and placing the limit markers in the graph by hand. The entire operation required about four minutes to execute. During the first maneuver, the satellite swings wide of the limit by approximately 0.15 meters. This behavior happens because the satellite flies close to the protruding NavZone, thus entering the edge of its influence zone and seeing an extra APF contribution. Based on this telemetry and the 3-D visualizations, the satellite successfully performs the desired maneuvers in this simulation without entering a NavZone.

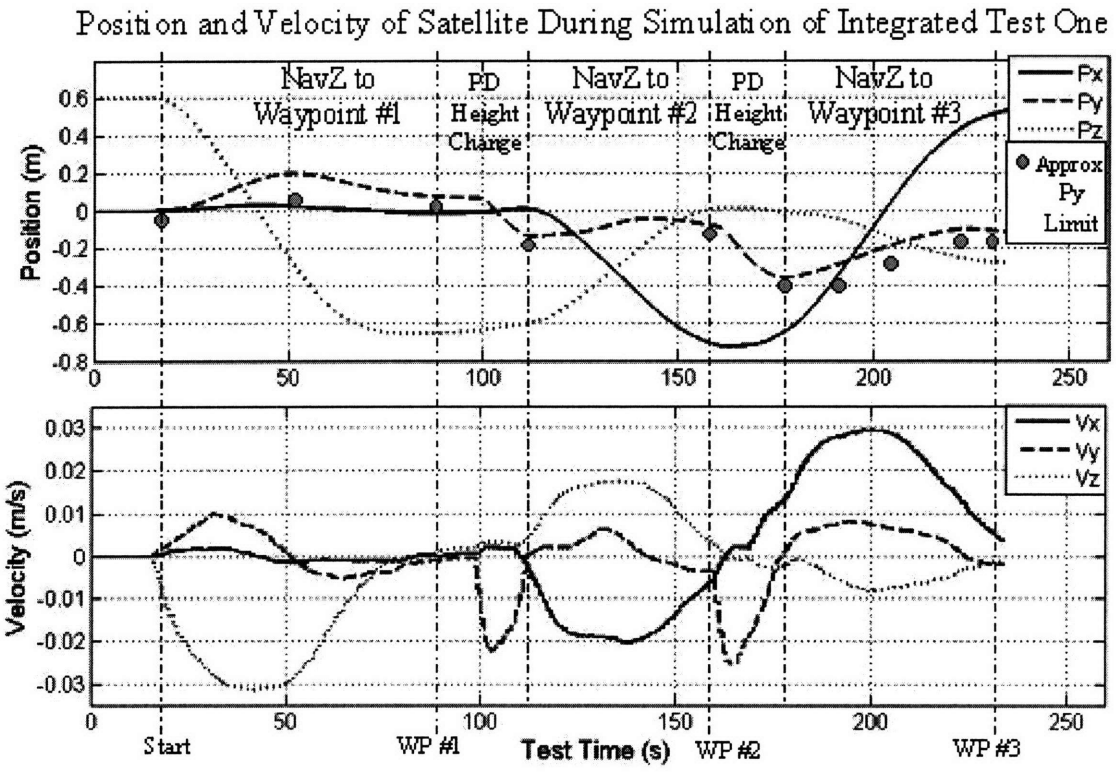


Figure 6-5: Simulated position and velocity telemetry for Integrated Test One.

6.2 Integrated Test Two

This second test is designed to incorporate NavZ maneuvers, waypoints, and pointing. The mission scenario is an automated operation for inspecting the CEV.

Mission: An inspector satellite is commanded to do an automated inspection of the CEV. Flying at a constant height above the CEV hull, the satellite maneuvers from its docking location, around a solar panel, and around the crew module. During all of these maneuvers, the satellite automatically points its camera face towards the CEV hull. The NavZones are placed to form the outline of a 1/10 scale model of the CEV, as shown in Figure 6-6.

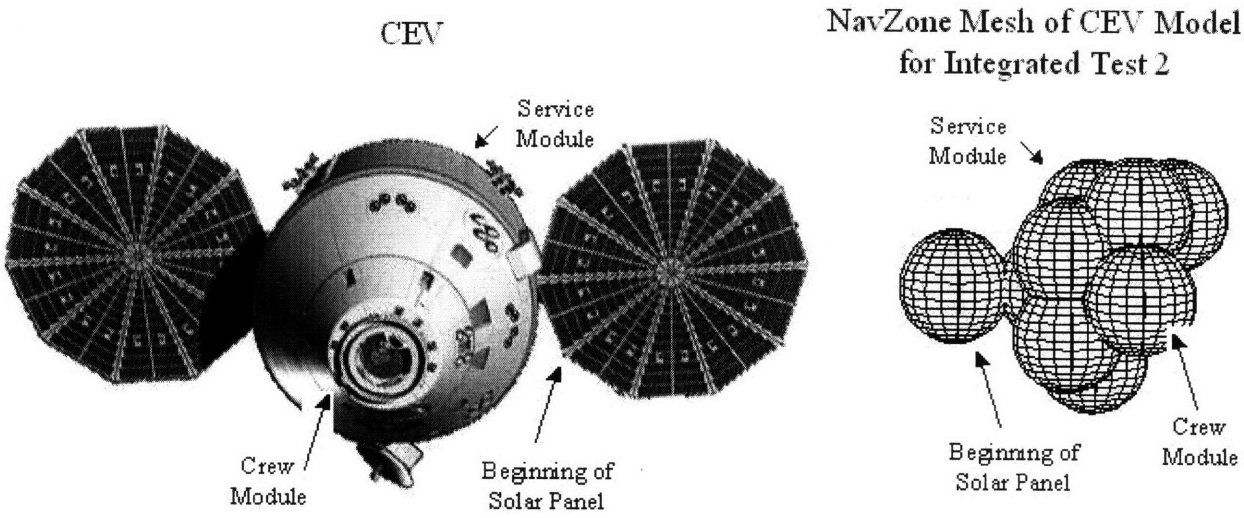
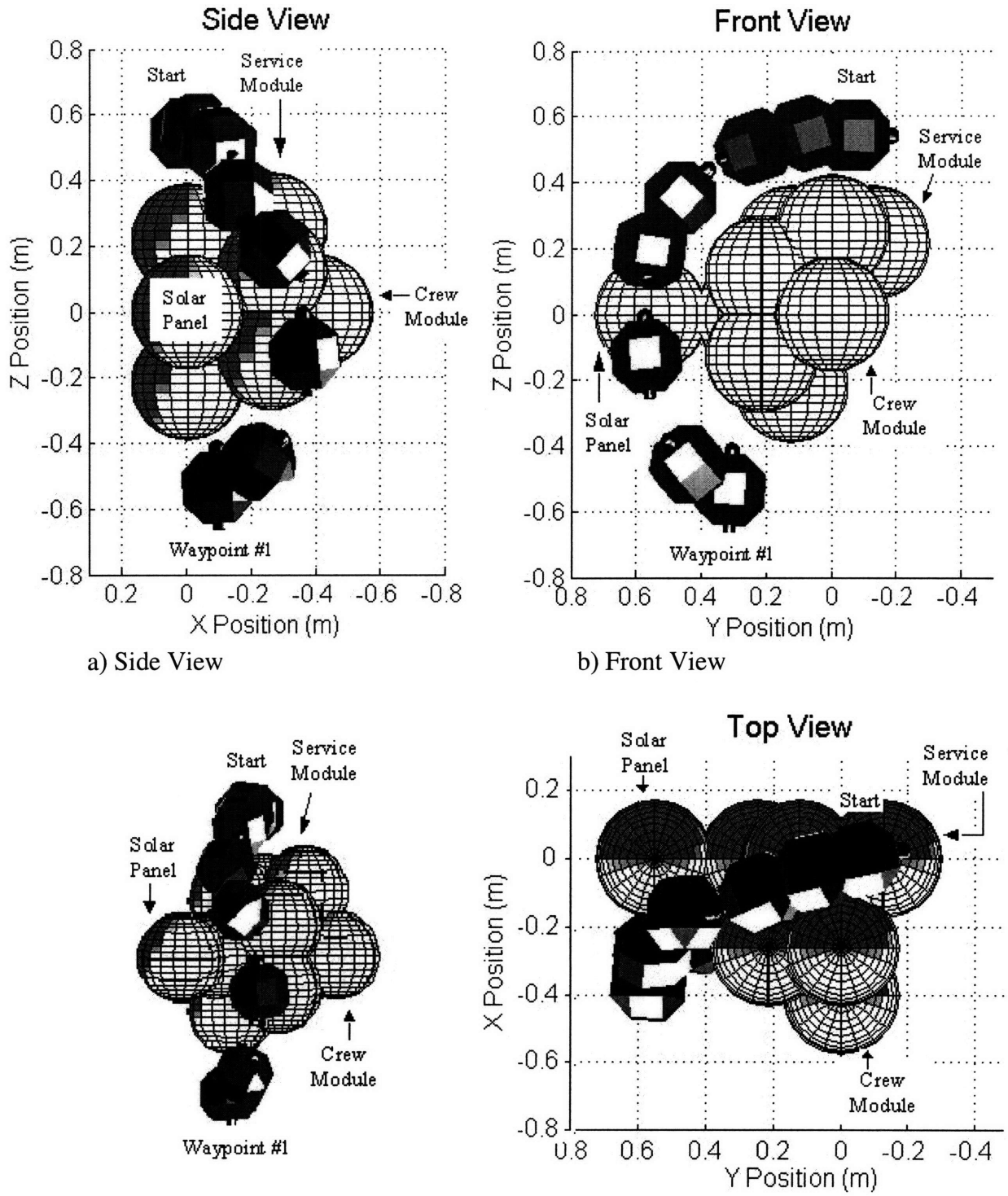


Figure 6-6: Comparison of CEV and the 1/10 scale model of the CEV used for Integrated Test 2.

6.2.1 Design and Simulation

For the first maneuver, the satellite navigates from one side of the CEV model to the other side. In the middle of the flight, it approaches the virtual solar panel and must fly around it. Figure 6-7 shows the simulation results of the first maneuver. Snapshots of the position and orientation of the satellite are shown for intervals of 20 seconds. Based on these images, the satellite successfully maneuvers around the NavZones while pointing one face towards the mesh surface.



a) Side View

b) Front View

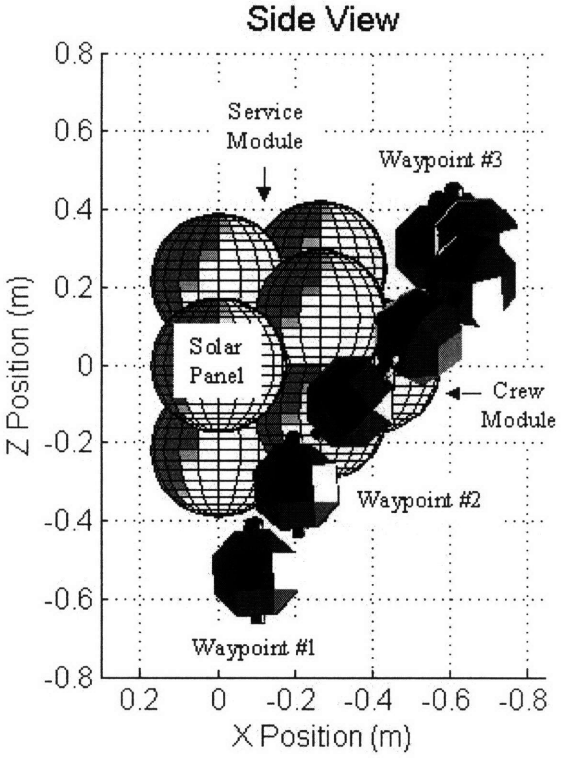
c) 3-D View

d) Top View

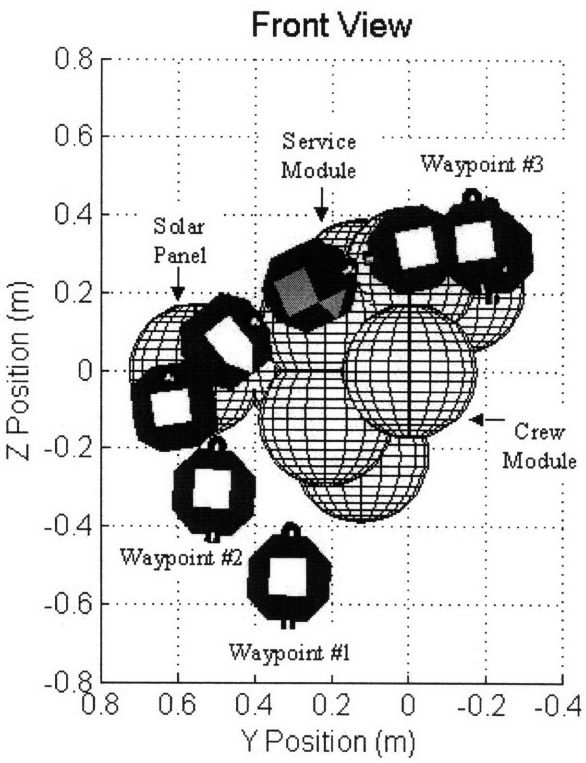
Figure 6-7: Simulation of the satellite flying the first maneuver.

After reaching the first waypoint, the satellite maneuvers to a second waypoint near the solar panel using the SPHERES PD controller. Then, it navigates to a third waypoint on another side

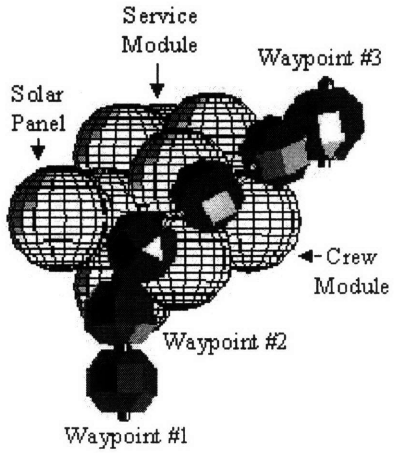
of the CEV model using the NavZ controller. This third maneuver takes it around the virtual crew module. Figure 6-8 shows the simulation results of the second and third maneuvers. Snapshots of the position and orientation of the satellite are shown for intervals of 10 seconds.



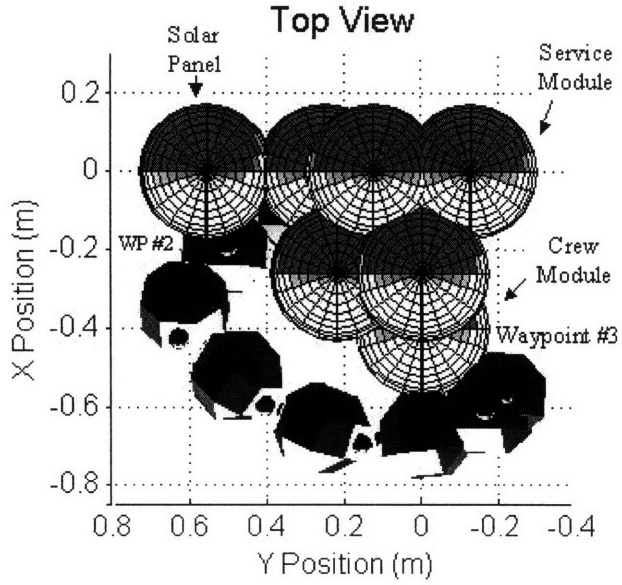
a) Side View



b) Front View



a) 3-D View



b) Top View

Figure 6-8: Simulation of the satellite flying the second and third maneuvers.

Figure 6-9 shows the simulated position and velocity telemetry of the satellite. Based on this telemetry and the above graphs of the maneuvers, the simulated satellite successfully navigated through the waypoints without entering any NavZones, and pointed its camera face towards the virtual CEV surface.

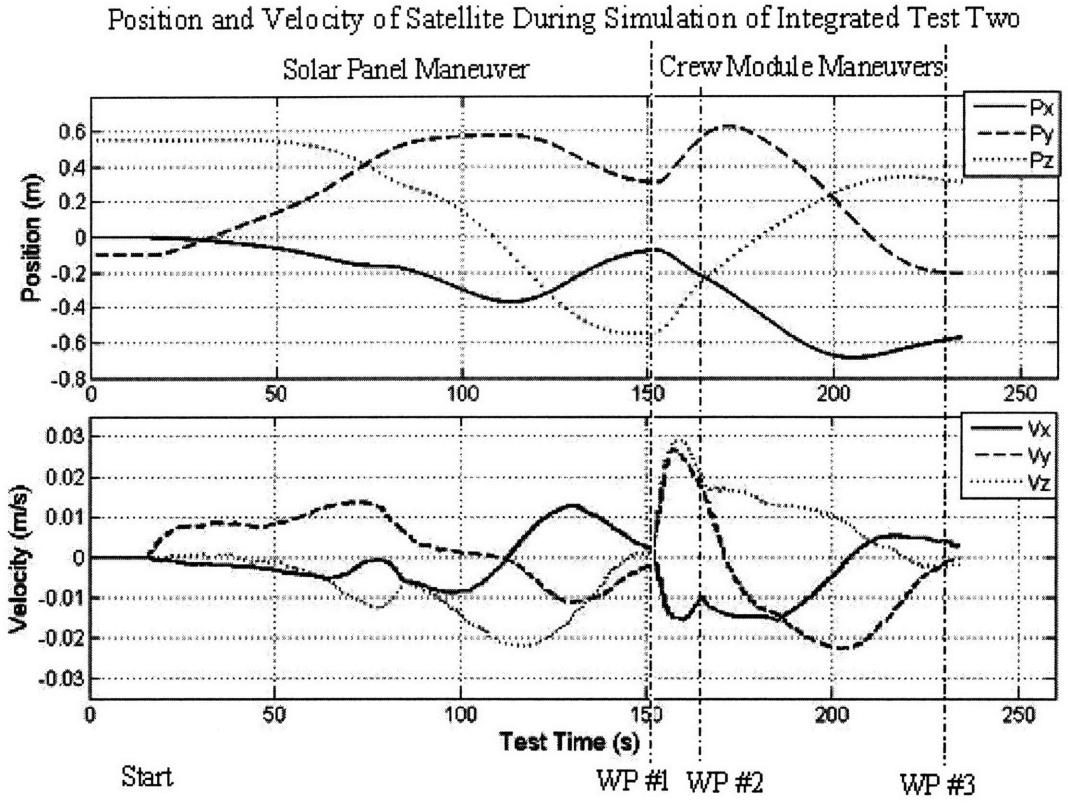


Figure 6-9. Simulated position and velocity telemetry from Integrated Test Two

6.3 Flight Results for Integrated Tests One and Two

These tests were implemented on the SPHERES hardware. The SPHERES team has scheduled for these tests to be conducted in the next ISS test-session. Unfortunately, NASA was unable to schedule the test-session in time for this thesis. The tests will still be performed, and the results will be included in a conference paper. Because the simulation was successful at predicting the behavior of the SPHERE for the flight test discussed Section 3.1.4, and the simulation successfully recreated the behavior of the flight test in Section 3.2.3, it is expected that these tests will also perform like their simulations.

6.4 Simulation of 3-D Inspection of CEV

The final development of NavZ for this research was to implement Integrated Test One into a 3-D simulation of CEV inspection. For this simulation, the STK interface developed by the NPS and introduced in Section 3.1.2 was utilized. Figure 6-10 shows snapshots of the first maneuver of the mission. Then Figure 6-11 shows the third maneuver of Integrated Test 1. In this case, the satellite is inspecting the thruster system.

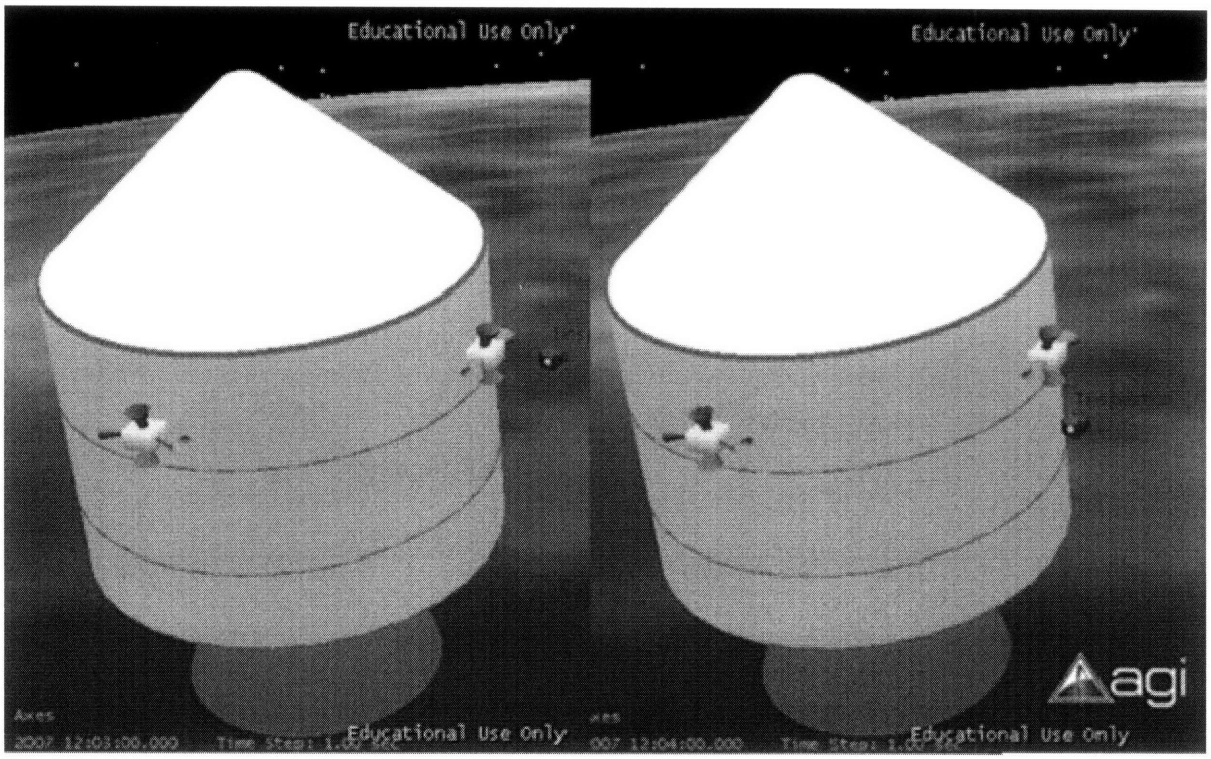


Figure 6-10. First maneuver around the CEV from Integrated Test One, visualized in STK.

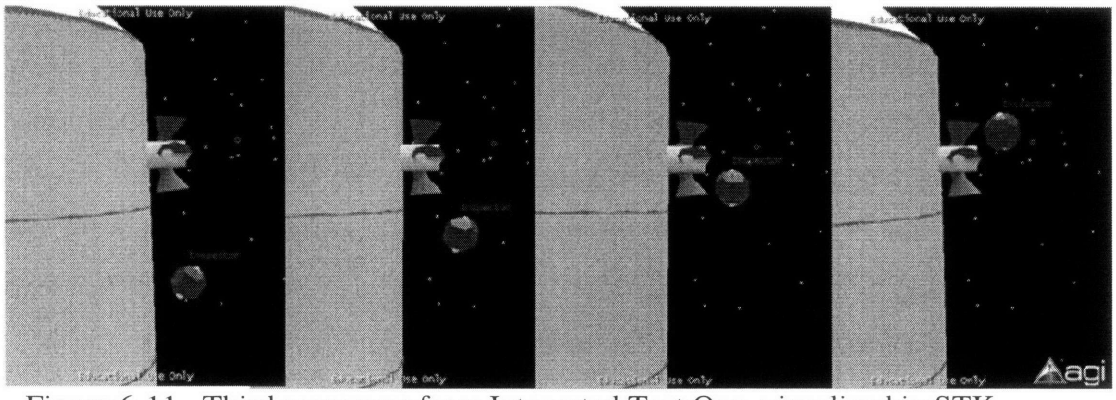


Figure 6-11. Third maneuver from Integrated Test One, visualized in STK.

In the launch vehicle industry, STK is utilized to simulate the launch for NASA TV. This resource could also be used for inspection operations. Before the inspection, an STK interface could show maneuvers before they are implemented. During the operation, STK could display the location of the inspector satellite for its pilot. Thus, a visualization similar to this STK simulation could be used as a display tool for an inspector pilot. This display fits into the inspection system design shown in Figure 6-12.

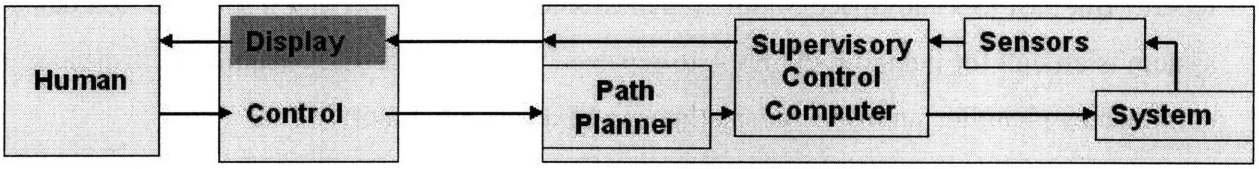


Figure 6-12. Block diagram of inspector control system.

6.5 Summary

As a final development of the research for this thesis, several of the improvements of the NavZ algorithm introduced in Chapter 4 were combined into two integrated tests. These tests were designed to simulate two CEV inspection scenarios. From the results of the simulations, the adaptability of the NavZ algorithm and the capability of the SPHERE satellite are apparent. The NavZ algorithm was successfully adapted to operate with waypoints, ellipsoid potential functions, speed change, height change, and an autonomous inspection-pointing algorithm. Then, in the simulation of the first integrated test, the SPHERE satellite is able to navigate around a full-scale virtual model of a portion of the CEV wall. In the second simulation, the SPHERE satellite navigates around a 1/10-scale virtual model of the CEV.

In the near future, these two tests will be performed inside the ISS. They are scheduled in the next SPHERES flight-test session. Because the simulation was successful at predicting the behavior of the SPHERE for the flight test discussed Section 3.1.4, and the simulation successfully recreated the behavior of the flight test in Section 3.2.3, the flight test results are expected to look like the simulation results. The results of these tests will be discussed in another paper.

Finally, one of the integrated tests was implemented in an STK simulation to visualize a CEV inspection operation. This kind of visualization is potentially helpful for operations planning and real-time situational awareness.

7 Conclusions

As the CEV approaches its defining moment in history, as it gets ever closer to flying humans back to the Moon, its need for portable inspection techniques becomes ever more pressing. In the CEV's current design, it can only support contingency EVA's for inspection and repair. But past ISS and Space Shuttle repair missions have shown that inspection of a damaged system is crucial for planning the EVA to repair it. To assist the CEV team with inspection and contingency planning, an inspector satellite can be an essential tool.

Chapter 1 of this thesis presents the idea of using a miniature satellite to inspect the CEV for damage while flying in formation. Two of the main objectives of this thesis research are to develop a new inspector-satellite controller and flight-test it on the SPHERE satellites. The baseline LQR and APF controller designed by Dr. McCamish was implemented on SPHERES, and then it was redesigned to plan maneuvers around meshes of NavZones using emulated computer-vision. Named the NavZ, this controller provides the smooth-wall-following behavior that has been useful with Space Shuttle OBSS inspections, but has not been exhibited in previous inspector-satellite designs.

The initial development of the NavZ algorithm is discussed in Chapter 3, crowned by successful flight tests of both the baseline and NavZ algorithms. The flight test results show that these algorithms can be successfully implemented on a satellite for obstacle avoidance. Also, the NavZ results revealed potential improvements to the NavZ algorithm that were then implemented in Chapter 4. The developments in this chapter satisfied objectives 1b, 2, 5, 6, 7, and 8 from Section 1.3.

Throughout Chapter 4, the NavZ algorithm underwent several improvements. The developments in this chapter satisfied objectives 1a, 1b, 2, 4, 5, 6, and 8 from Section 1.3. The influence zones of the NavZ were adjusted so that the satellite would closely follow the NavZone edges. Also, the APF was adjusted so that the satellite exited NavZones in a controlled manner. The capabilities for waypoint handling, speed adjustment, and height adjustment were incorporated into NavZ and simulated. Lastly, the ability of the NavZ to adapt and navigate around a previously unknown obstacle was simulated. All these developments after the first

flight test strengthened the NavZ algorithm, evolving and maturing it towards becoming a flight-worthy algorithm for real inspection operations.

During this development, the information that would be received from a computer-vision system was emulated. To pave the way towards interfacing the NavZ with actual computer-vision hardware, Chapter 5 explored the uncertainties of computer-vision information and improved the emulation. The developments in this chapter satisfied objective 3 from Section 1.3. The simulation results in Chapter 5 show that the NavZ can potentially adapt to uncertainties of position and losses of feature points.

Lastly, several of the NavZ improvements discussed in Chapter 3 were combined into two final ISS tests with virtual scale models of the CEV. This development satisfied objectives 1a, 1b, 2, 4, 5, 6, 8, and 9 from Section 1.3. The designs and simulation results of these tests are discussed in Chapter 6. The flight results for these tests have not been received yet, but the tests are scheduled for the next SPHERES session aboard the ISS. However, simulation shows that the final algorithms exhibit the desired autonomous waypoint, wall-following, and pointing behaviors. STK visualizations show these maneuvers around a full-scale CEV.

In conclusion, all of these NavZ developments have compounded into an algorithm that has the potential to be utilized for future inspection operations. Also, an inspection algorithm was successfully implemented and flight-tested on a SPHERE satellite, and the SPHERES testbed has proven very useful for the development of inspector-satellite algorithms.

7.1 Future Work

In the future, there are several potential developments to improve the design of inspection operations and the design of the NavZ algorithm.

First, the final ISS tests discussed in Chapter 6 will be run in the next test session. The results from these tests will be analyzed and included in another paper.

Second, a key implementation step will be to integrate NavZ with an actual computer-vision system. The SPHERES team can complete its development of the camera system and interface it with the NavZ controller. Then, this integration can be tested on the SPHERES laboratory table. For useful testing with computer-vision in 2-D, the NavZ controller needs to be adapted so that it provides forces large enough to overcome the friction of the table. For microgravity testing, the camera hardware will need to go through the qualification process for operation inside the ISS.

Third, detailed methods for dealing with the loss of feature points could be developed for NavZ. One method was introduced in Section 5.5 of this thesis, but it is just an initial demonstration of the adaptability of the NavZ. With the lighting conditions in space, finding feature points can be difficult, and the computer-vision information may need to be supplemented with measurements from other sensors. For example, laser range sensors or relative GPS may provide additional navigation information.

Fourth, there are several analyses that would aid the future design of inspection operations. One study could develop the best sequence of maneuvers that will inspect the entire surface of the CEV. This study could determine the maneuver sequence and heights above the CEV surface that would provide the quickest full-surface inspection. This is an interesting problem, because a larger height above the CEV could give a wider view, but would require more fuel and farther maneuvering distances. This study would also provide information about how long a full-surface inspection would take using an inspector satellite.

Fifth, the SPHERES team can redesign some of the SPHERES systems so that they can operate in the space environment. An internal preliminary study about what needs to be redesigned has already been performed. Further development would pave the way towards utilizing the SPHERES as inspector satellites.

Sixth, the SPHERES payload of inspection sensors can be developed. The type of inspection sensors for this payload should be determined. Also, the interface between the SPHERES and the inspection sensor payload can be defined.

Seventh, the inspector systems and operations design could be adapted to provide global situational awareness of a worksite during an EVA.

Last, but not least, in the future the SPHERE satellites and/or NavZ algorithm could be used for CEV inspection.

Appendix A SPHERES Controller Code

This code is run every control cycle, and calls the NavZ function.

```
#include "gspInspection.h"
#include "pads_internal.h"
#include "comm.h"
#include "control.h"
#include "ctrl_position.h"
#include "ctrl_attitude.h"
#include "est_StateProp.h"
#include "prop.h"
#include <string.h>
#include "find_state_error.h"
#include "ctrl_mix.h"
#include "system.h"
#include "gsutil_thr_times.h"
#include "math_matrix.h"
#include "ctrl_NPS_Inspection.h"
#include "std_includes.h"
#include <math.h>

static unsigned char cDocked;
static unsigned int waypoint;
static state_vector ctrlStateTarget;
static float ctrlStateMesh[3][9];
static float initialPosChaseInertial[3];

// Variables used for all satellites' controllers
state_vector ctrlStateChaserNPS, ctrlStateTargetSatNPS, ctrlStateObstacleNPS;
float delTNPS=0.0f;
unsigned int tstampNPS=0, indexNPS=0, min_pulseNPS=10, convTimeNPS=15000;
float duty_cycle_NPS = 20.0f;
float comp_delay_factor = 4.0f; // Twice the percentage of the control period
reserve for computation (need to be twice since mixer outputs pulses centered
in the thrusting window)
float stepsize = 1.0f;
const unsigned int maneuverSequenceNPS[] = {1, 2, 3};
state_vector ctrlStateErrorNPS;
float ctrlControlNPS[6];
prop_time firing_timesNPS;
extern const float KPattitudePID;
extern const float KIattitudePID;
extern const float KDattitudePID;
extern const float KPpositionPID;
extern const float KIpositionPID;
extern const float KDpositionPID;
extern const float KPattitudePD;
extern const float KDattitudePD;
```

```
extern const float KPpositionPD;
extern const float KDpositionPD;
dbg_short_packet DebugVecSignedShortNPS;
dbg_ushort_packet DebugVecUnsignedShortNPS;
dbg_ushort_packet DebugVecUnsignedShortNPS2;
dbg_float_packet DebugVecFloatNPS;
comm_payload_soh sohNPS;
float timecheck;

void gspInspection_InitTest(unsigned int test_number)
{
extern state_vector initState;

// Reset PID controllers
ctrlAttitudePIDinit();
ctrlPositionPIDinit();

// Reset static vectors
cDocked = 0;
waypoint = 1;
memset(ctrlStateTarget, 0, sizeof(state_vector));
ctrlStateTarget[QUAT_4] = 1.0f;
memset(initialPosChaseInertial, 0, sizeof(float)*3);
memset(ctrlStateMesh,0,sizeof(float)*3*4);

padsEstimatorInitWaitAndSet(initState, padsInertialBufferCapacity(), 200,
205, PADS_INIT_THRUST_INT_ENABLE, PADS_BEACONS_SET_1TO9);

ctrlPeriodSet(1000);

// Quaternions of the reference frame
memset(ctrlStateTargetSatNPS, 0, sizeof(state_vector));
ctrlStateTargetSatNPS[QUAT_1] = sqrt(2.0f)/2.0f;
ctrlStateTargetSatNPS[QUAT_2] = -sqrt(2.0f)/2.0f;
ctrlStateTargetSatNPS[QUAT_3] = 0.0f;
ctrlStateTargetSatNPS[QUAT_4] = 0.0f;
memset(ctrlStateObstacleNPS, 0, sizeof(state_vector));

switch (test_number)
{
case 1:
// Mesh feature point locations for Test 1
ctrlStateMesh[0][0] = 0.5f;
ctrlStateMesh[1][0] = -0.3585f-0.18f;
ctrlStateMesh[2][0] = 0.2776f;
ctrlStateMesh[0][1] = 0.5f;
ctrlStateMesh[1][1] = -0.56f+0.15f;
ctrlStateMesh[2][1] = -0.2776f;

ctrlStateMesh[0][2] = 0.0f;
ctrlStateMesh[1][2] = -0.56f;
ctrlStateMesh[2][2] = 0.0f;
ctrlStateMesh[0][3] = 0.0f;
ctrlStateMesh[1][3] = -0.3985f-0.18f;
ctrlStateMesh[2][3] = 0.5198f;
ctrlStateMesh[0][4] = 0.0f;
ctrlStateMesh[1][4] = -0.56f;
```

```
ctrlStateMesh[2][4] = -0.5198f;

ctrlStateMesh[0][5] = -0.5f;
ctrlStateMesh[1][5] = -0.3585f-0.18f;
ctrlStateMesh[2][5] = 0.2776f;
ctrlStateMesh[0][6] = -0.5f;
ctrlStateMesh[1][6] = -0.3585f-0.18f;
ctrlStateMesh[2][6] = -0.2776f;

ctrlStateMesh[0][7] = 0.5f;
ctrlStateMesh[1][7] = -0.56f;
ctrlStateMesh[2][7] = -0.7889f;

ctrlStateMesh[0][8] = 0.5f;
ctrlStateMesh[1][8] = -0.56f;
ctrlStateMesh[2][8] = -0.7889f;
break;
case 2:
// Mesh feature point locations for Test 2
ctrlStateMesh[0][0] = 0.0f;
ctrlStateMesh[1][0] = -0.125f+0.25f;
ctrlStateMesh[2][0] = 0.2165f;
ctrlStateMesh[0][1] = 0.0f;
ctrlStateMesh[1][1] = 0.25f;
ctrlStateMesh[2][1] = 0.0f;
ctrlStateMesh[0][2] = 0.0f;
ctrlStateMesh[1][2] = -0.125f+0.25f;
ctrlStateMesh[2][2] = -0.2165f;

ctrlStateMesh[0][3] = -0.26f;
ctrlStateMesh[1][3] = -0.0335f+0.25f;
ctrlStateMesh[2][3] = 0.125f;
ctrlStateMesh[0][4] = -0.26f;
ctrlStateMesh[1][4] = -0.0335f+0.25f;
ctrlStateMesh[2][4] = -0.125f;

ctrlStateMesh[0][5] = -0.26f;
ctrlStateMesh[1][5] = -0.0f;
ctrlStateMesh[2][5] = 0.25f;
ctrlStateMesh[0][6] = 0.0f;
ctrlStateMesh[1][6] = 2.0f*0.153f+0.25f;
ctrlStateMesh[2][6] = 0.0f;
ctrlStateMesh[0][7] = 0.0f;
ctrlStateMesh[1][7] = -0.375f+0.25f;
ctrlStateMesh[2][7] = 0.2165f;

ctrlStateMesh[0][8] = -0.4f;
ctrlStateMesh[1][8] = 0.0f;
ctrlStateMesh[2][8] = 0.0f;
break;
}
}

void gspInspection_Control(unsigned int test_number, unsigned int test_time,
unsigned int maneuver_number, unsigned int maneuver_time)
{
```

```
unsigned int i;
unsigned int iobs;
unsigned int waypointnum, LQR=1, ellipsoid=0;
float g2b[3][3], b2g[3][3]; // global to body rotation matrix
float r_to_chaser_inertial[3];
float v_to_chaser_inertial[3];
float r_to_obstacle_inertial[3];
float r_to_obstacle_copy[3][9];
float v_to_obstacle_inertial[3];
float r_to_chaser_body_temp[3];
float v_to_chaser_body_temp[3];
float r_to_goal_body_temp[3];
float v_to_goal_body_temp[3];
float r_to_obstacle_body_temp[3];
float v_to_obstacle_body_temp[3];
float r_rsw[3];
float v_rsw[3];
float r_goal[3];
float r_to_obstacle_rsw[3];
//float v_to_obstacle_rsw[3];
float r_norm;
float v_norm;
float g_norm;
//float r_to_obstacle_norm;
//float v_to_obstacle_norm;
float L, Lmax, a_max, dim, uncert, OBS_DoR, V_max, range, dock_limit, G;
extern const float VEHICLE_LIN_ACCEL;
int nObs, sats, Chase;
float OBSiRSW[3][9];
float iRSW[3];
float fRSW[3];
float iRSWTemp1[3];
float iRSWTemp2[3];
extern const float VEHICLE_MASS;
float a_tot[3];
float a_body[3];
float F_body[3];
float theta, phi, angle;
float Am1[3][3], Am2[3][3], Am3[3][3], Am[3][3], Amtemp[3][3];
float pi = 3.14159f;

// make a zero-offset index
indexNPS = test_number-1;

memset(ctrlControlNPS,0,(sizeof(float))*6);
memset(&firing_timesNPS,0, sizeof(prop_time));
memset(ctrlStateErrorNPS,0,sizeof(state_vector));
ctrlStateErrorNPS[QUAT_4] = 1.0f;
memset(DebugVecSignedShortNPS,0,sizeof(dbg_short_packet));
memset(DebugVecUnsignedShortNPS,0,sizeof(dbg_ushort_packet));
memset(DebugVecUnsignedShortNPS2,0,sizeof(dbg_ushort_packet));
memset(DebugVecFloatNPS,0,sizeof(dbg_float_packet));

// Timecheck measures how long control code takes to run
timecheck = sysSphereTimeGet();

// Get states from global estimator and propagate them to actual time
```

```
delTNPS = (sysSphereTimeGet() - padsStateGet(ctrlStateChaserNPS))*0.001f;
if (delTNPS>0.0f)
statePropDelTPss(delTNPS, ctrlStateChaserNPS);

// Get states of the other Spheres and propagate them to actual time
// Added so that it is one satellite test, hardcoding states of obstacle and
target

if (maneuver_number > 1)
// Turn OFF global metrology
padsGlobalPeriodSet(SYS_FOREVER);

switch (maneuver_number)
{
case 1: // EKF convergence period
if (test_time>=convTimeNPS)
{
// The chaser satellite holds its initial attitude until Man 3 starts
// (until it gets the attitude of the target)
memcpy(&ctrlStateTarget[QUAT_1], &ctrlStateChaserNPS[QUAT_1],
sizeof(float)*4);

// Change maneuver
ctrlManeuverTerminate();
}

// Save debug data
for (i=0;i<13;i++)
DebugVecSignedShortNPS[1+i] = (short) (ctrlStateErrorNPS[i]*10000.0f);

break;

case 2: // Move into position

switch (test_number)
{
case 1:
// Starting position for Test 1
ctrlStateTarget[POS_X] = 0.0f;
ctrlStateTarget[POS_Y] = 0.0f;
ctrlStateTarget[POS_Z] = 0.6f;
break;
case 2:
// Starting position for Test 2
ctrlStateTarget[POS_X] = 0.0f;
ctrlStateTarget[POS_Y] = -0.1f;
ctrlStateTarget[POS_Z] = 0.55f;
break;
}
// Set quaternions of satellite to those of the reference frame
ctrlStateTarget[QUAT_1] = ctrlStateTargetSatNPS[QUAT_1];
ctrlStateTarget[QUAT_2] = ctrlStateTargetSatNPS[QUAT_2];
ctrlStateTarget[QUAT_3] = ctrlStateTargetSatNPS[QUAT_3];
ctrlStateTarget[QUAT_4] = ctrlStateTargetSatNPS[QUAT_4];

// Compute state errors
findStateError(ctrlStateErrorNPS, ctrlStateChaserNPS, ctrlStateTarget);
```

```
// Attitude controller
ctrlAttitudeNLPIDwie(KPattitudePID, KIattitudePID, KDattitudePID,
KPattitudePID, KIattitudePID, KDattitudePID, KPattitudePID, KIattitudePID,
KDattitudePID, 1.0f, ctrlStateErrorNPS, ctrlControlNPS);

// Run both position and attitude controllers
// Position controller
ctrlPositionPDgains(KPpositionPD, KDpositionPD, KPpositionPD, KDpositionPD,
KPpositionPD, KDpositionPD, ctrlStateErrorNPS, ctrlControlNPS);


if (maneuver_time>=30000)
{
// Store the initial position states
memcpy(&initialPosChaseInertial[POS_X], &ctrlStateChaserNPS[POS_X],
sizeof(float)*3);

// Change maneuver
ctrlManeuverTerminate();
}

// Save debug data
for (i=0;i<13;i++)
DebugVecSignedShortNPS[1+i] = (short) (ctrlStateErrorNPS[i]*10000.0f);

break;

case 3: // Roll and dock using NPS controller

switch (test_number)
{
case 1:
// Waypoint locations for Test 1
waypointnum = 5;  // Number of waypoints
nObs = 7;         // Number of obstacles in mesh
switch (waypoint)
{
case 1:
      LQR = 1;    // LQR+APF is run for maneuvering to this waypoint
      dim = 0.85f;  // Dimension of Obstacle (spherical diameter)
      ctrlStateTargetSatNPS[POS_X] = 0.0f;  // Location of waypoint
      ctrlStateTargetSatNPS[POS_Y] = 0.02f;
      ctrlStateTargetSatNPS[POS_Z] = -0.6f;
      break;
case 2:
      LQR = 0;    // PID is run for maneuvering to this waypoint
      ctrlStateTarget[POS_X] = 0.0f;
      ctrlStateTarget[POS_Y] = -0.23f-0.07f+0.153f;
      ctrlStateTarget[POS_Z] = -0.6f;
      ctrlStateTargetSatNPS[POS_X] = 0.0f;
      ctrlStateTargetSatNPS[POS_Y] = -0.23f-0.07f+0.153f;
      ctrlStateTargetSatNPS[POS_Z] = -0.6f;
      break;
case 3:
      LQR = 1;
      dim = 0.54f;
```

```
      ctrlStateTargetSatNPS[POS_X] = -0.7f;
      ctrlStateTargetSatNPS[POS_Y] = -0.23f-0.07f+0.153f;
      ctrlStateTargetSatNPS[POS_Z] = 0.0f;
      break;
case 4:
      LQR = 0;
      ctrlStateTarget[POS_X] = -0.7f;
      ctrlStateTarget[POS_Y] = -0.3585f-0.18f+0.09f+0.153f;
      ctrlStateTarget[POS_Z] = 0.0f;
      ctrlStateTargetSatNPS[POS_X] = -0.7f;
      ctrlStateTargetSatNPS[POS_Y] = -0.3585f-0.18f+0.09f+0.153f;
      ctrlStateTargetSatNPS[POS_Z] = 0.0f;
      break;
case 5:
      LQR = 1;
      dim = 0.08f;
      ellipsoid = 1;
      ctrlStateTargetSatNPS[POS_X] = 0.5f;
      ctrlStateTargetSatNPS[POS_Y] = -0.56f+0.09f+0.153f+0.15f;
      ctrlStateTargetSatNPS[POS_Z] = -0.2776;
      break;
}
break;
case 2:
// Waypoint locations for Test 2
waypointnum = 3;
nObs = 9;          // Number of obstacles in mesh
dim = 2.0f*0.153f;  // Dimension of Obstacle (spherical diameter)
switch (waypoint)
{
case 1:
      LQR = 1;
      ctrlStateTargetSatNPS[POS_X] = -0.1f;
      ctrlStateTargetSatNPS[POS_Y] = 0.3f;
      ctrlStateTargetSatNPS[POS_Z] = -0.48f;
      break;
case 2:
      LQR = 0;
      ctrlStateTarget[POS_X] = -0.25f;
      ctrlStateTarget[POS_Y] = 0.57f;
      ctrlStateTarget[POS_Z] = -0.23f;
      ctrlStateTargetSatNPS[POS_X] = -0.25f;
      ctrlStateTargetSatNPS[POS_Y] = 0.57f;
      ctrlStateTargetSatNPS[POS_Z] = -0.23f;
      break;
case 3:
      LQR = 1;
      ctrlStateTargetSatNPS[POS_X] = -0.55f;
      ctrlStateTargetSatNPS[POS_Y] = -0.18f;
      ctrlStateTargetSatNPS[POS_Z] = 0.28f;
      break;
}
break;
}

quat2matrixIn(g2b, &ctrlStateTargetSatNPS[QUAT_1]);
for (iobs=0; iobs<nObs; iobs++)
```

```
{
for (i=0;i<3;i++)
{
      ctrlStateObstacleNPS[i] = ctrlStateMesh[i][iobs];

      //  The following vectors are expressed in the inertial frame
      r_to_chaser_inertial[i] = ctrlStateChaserNPS[i] -
ctrlStateTargetSatNPS[i];
      v_to_chaser_inertial[i] = ctrlStateChaserNPS[i+3] -
ctrlStateTargetSatNPS[i+3];
      r_to_obstacle_inertial[i] = ctrlStateObstacleNPS[i] -
ctrlStateTargetSatNPS[i];
      r_to_obstacle_copy[i][iobs] = r_to_obstacle_inertial[i];
      v_to_obstacle_inertial[i] = ctrlStateObstacleNPS[i+3] -
ctrlStateTargetSatNPS[i+3];
}

// The following vectors are expressed in the target's body frame
mathMatVecMult(r_to_chaser_body_temp, (float**)g2b, r_to_chaser_inertial, 3,
3);
mathMatVecMult(v_to_chaser_body_temp, (float**)g2b, v_to_chaser_inertial, 3,
3);
r_to_goal_body_temp[0] = r_to_chaser_body_temp[0];
r_to_goal_body_temp[1] = r_to_chaser_body_temp[1];
r_to_goal_body_temp[2] = r_to_chaser_body_temp[2];
memcpy(v_to_goal_body_temp, v_to_chaser_body_temp, sizeof(float)*3);
mathMatVecMult(r_to_obstacle_body_temp, (float**)g2b, r_to_obstacle_inertial,
3, 3);
mathMatVecMult(v_to_obstacle_body_temp, (float**)g2b, v_to_obstacle_inertial,
3, 3);

// The following vectors are expressed in the RSW frame
r_rsw[0] = -r_to_chaser_body_temp[2];
r_rsw[1] = r_to_chaser_body_temp[0];
r_rsw[2] = -r_to_chaser_body_temp[1];
v_rsw[0] = -v_to_chaser_body_temp[2];
v_rsw[1] = v_to_chaser_body_temp[0];
v_rsw[2] = -v_to_chaser_body_temp[1];
r_goal[0] = -r_to_goal_body_temp[2];
r_goal[1] = r_to_goal_body_temp[0];
r_goal[2] = -r_to_goal_body_temp[1];
r_to_obstacle_rsw[0] = -r_to_obstacle_body_temp[2];
r_to_obstacle_rsw[1] = r_to_obstacle_body_temp[0];
r_to_obstacle_rsw[2] = -r_to_obstacle_body_temp[1];
//v_to_obstacle_rsw[0] = -v_to_obstacle_body_temp[2];
//v_to_obstacle_rsw[1] = v_to_obstacle_body_temp[0];
//v_to_obstacle_rsw[2] = -v_to_obstacle_body_temp[1];

// The following are the norms to the previous vectors
r_norm = mathVecMagnitude(r_rsw, 3);
v_norm = mathVecMagnitude(v_rsw, 3);
g_norm = mathVecMagnitude(r_goal, 3);
//r_to_obstacle_norm = mathVecMagnitude(r_to_obstacle_rsw, 3);
//v_to_obstacle_norm = mathVecMagnitude(v_to_obstacle_rsw, 3);

OBSiRSW[0][iobs] = r_to_obstacle_rsw[0];
OBSiRSW[1][iobs] = r_to_obstacle_rsw[1];
```

```
OBSiRSW[2][iobs] = r_to_obstacle_rsw[2];
}

////// Plug in Shawn's NPS controller, which should output
////// a three element vector of forces F = [Fx; Fy; Fz]

// Physical parameters:
L = 2.0f*0.107f;        // SPHERES dimension along docking port
Lmax = 2.0f*0.153f;     // with CO2 tank
a_max= (duty_cycle_NPS-comp_delay_factor)*0.01f*VEHICLE_LIN_ACCEL;     // Max
accel over a control period (the -4 accounts for computational time)

// Obstacle:
uncert = 0.02f;  //position uncertainty
OBS_DoR = (dim+Lmax+uncert)/2.0f;  // Minimum approach range


// Simulation Parameters:
V_max = 0.5f;                          // Max velocity desired
sats = 2; // number of active SPHERES (Target & Chase)
range = 2.0f;                          // ISS within 2 meters
dock_limit = 0.002f;                   // within 2 mm accuracy
G = Lmax;

// Chase spacecraft:
Chase = 1;                             //only one Chase SPHERES
iRSWTemp1[0] = initialPosChaseInertial[0] - ctrlStateTargetSatNPS[0];
iRSWTemp1[1] = initialPosChaseInertial[1] - ctrlStateTargetSatNPS[1];
iRSWTemp1[2] = initialPosChaseInertial[2] - ctrlStateTargetSatNPS[2];
mathMatVecMult(iRSWTemp2, (float**)g2b, iRSWTemp1, 3, 3);
iRSW[0] = -iRSWTemp2[2];       // Convert to RSW frame
iRSW[1] = iRSWTemp2[0];        // Convert to RSW frame
iRSW[2] = -iRSWTemp2[1];       // Convert to RSW frame
fRSW[0] = 0.0f;                        // Chase final position
fRSW[1] = 0.0f;
fRSW[2] = 0.0f;

if (LQR < 1)       // If not doing NPS LQR, do PD position control
{
findStateError(ctrlStateErrorNPS, ctrlStateChaserNPS, ctrlStateTargetSatNPS);
ctrlPositionPDgains(KPpositionPD, KDpositionPD, KPpositionPD, KDpositionPD,
KPpositionPD, KDpositionPD, ctrlStateErrorNPS, ctrlControlNPS);
}
else  // Run modified NPS controller for inspection
{
ctrlNpsSimple_Inspection(a_tot, v_norm, r_norm, v_rsw, r_rsw, r_goal, g_norm,
sats, Chase, iRSW, fRSW, range, V_max, a_max, dock_limit, nObs, Lmax, G, L,
OBSiRSW, OBS_DoR, stepsize, ellipsoid, r_to_obstacle_copy);
a_body[0] = a_tot[1];
a_body[1] = -a_tot[2];
a_body[2] = -a_tot[0];
F_body[0] = VEHICLE_MASS*a_body[0];
F_body[1] = VEHICLE_MASS*a_body[1];
F_body[2] = VEHICLE_MASS*a_body[2];

// The forces need to be expressed in the inertial frame
// Use the following to convert from the target body frame to
```

```
// the inertial frame, if needed
quat2matrixOut(b2g, &ctrlStateTargetSatNPS[QUAT_1]);

// The vector ctrlControlNPS must contain the computed forces in
// the inertial frame:
mathMatVecMult(&ctrlControlNPS[FORCE_X], (float**)b2g, F_body, 3, 3);
}

// If doing PD control or at waypoint, set quarternions to reference frame
if (LQR < 1 || g_norm <=75.0f*dock_limit)
{
ctrlStateTarget[QUAT_1] = ctrlStateTargetSatNPS[QUAT_1];
ctrlStateTarget[QUAT_2] = ctrlStateTargetSatNPS[QUAT_2];
ctrlStateTarget[QUAT_3] = ctrlStateTargetSatNPS[QUAT_3];
ctrlStateTarget[QUAT_4] = ctrlStateTargetSatNPS[QUAT_4];
}
else  // If doing NPS LQR, then point one satellite face in direction of
velocity
{
// Determine first and second rotation angles (theta and angle)
theta = atan(ctrlStateChaserNPS[VEL_Z]/ctrlStateChaserNPS[VEL_Y]);
switch (test_number)
{
case 1:
      if (ctrlStateChaserNPS[VEL_Y] > 0)
      {
      if (ctrlStateChaserNPS[VEL_Z] < 0)
            theta = theta + pi;
      else
            theta = theta - pi;
      }
    if (ctrlStateChaserNPS[VEL_Z] <= 0)
      angle = -pi/2.0f;
    else
      angle = pi/2.0f;
    break;
case 2:
      if (ctrlStateChaserNPS[VEL_Y] > 0)
      {
      if (ctrlStateChaserNPS[VEL_Z] < 0)
            theta = theta + pi;
      else
            theta = theta - pi;
      }
    angle = -pi/2.0f;
    break;
}
// Determine third rotation angle (phi)
phi = pi/2.0f +
atan(ctrlStateChaserNPS[VEL_X]/sqrt(ctrlStateChaserNPS[VEL_Y]*ctrlStateChaser
NPS[VEL_Y]+ctrlStateChaserNPS[VEL_Z]*ctrlStateChaserNPS[VEL_Z]));
// Rotation matrices Am1, Am2, Am3
memset(Am1, 0, sizeof(Am1));
Am1[0][0] = 1.0f;
Am1[1][1] = cos(theta);
Am1[1][2] = sin(theta);
Am1[2][1] = -sin(theta);
```

```
Am1[2][2] = cos(theta);
memset(Am2, 0, sizeof(Am2));
Am2[0][0] = cos(phi);
Am2[0][1] = sin(phi);
Am2[1][0] = -sin(phi);
Am2[1][1] = cos(phi);
Am2[2][2] = 1.0f;
memset(Am3, 0, sizeof(Am3));
Am3[0][0] = cos(angle);
Am3[0][2] = -sin(angle);
Am3[1][1] = 1.0f;
Am3[2][0] = sin(angle);
Am3[2][2] = cos(angle);
mathMatMatMult((float **)Amtemp, (float **)Am2, (float **)Am1, 3, 3, 3);
mathMatMatMult((float **)Am, (float **)Am3, (float **)Amtemp, 3, 3, 3);
// Find the quaternions
ctrlStateTarget[QUAT_1] = 1.0f/2.0f*sqrt(1.0f+Am[0][0]-Am[1][1]-Am[2][2]);
ctrlStateTarget[QUAT_2] =
1.0f/4.0f*(Am[0][1]+Am[1][0])/ctrlStateTarget[QUAT_1];
ctrlStateTarget[QUAT_3] =
1.0f/4.0f*(Am[0][2]+Am[2][0])/ctrlStateTarget[QUAT_1];
ctrlStateTarget[QUAT_4] = 1.0f/4.0f*(Am[1][2]-
Am[2][1])/ctrlStateTarget[QUAT_1];
}

// Compute state errors
findStateError(ctrlStateErrorNPS, ctrlStateChaserNPS, ctrlStateTarget);

// Attitude controller
ctrlAttitudeNLPIDwie(KPattitudePID, KIattitudePID, KDattitudePID,
KPattitudePID, KIattitudePID, KDattitudePID, KPattitudePID, KIattitudePID,
KDattitudePID, 1.0f, ctrlStateErrorNPS, ctrlControlNPS);

// If at a waypoint, either move to next waypoint or terminate test
if (g_norm<=30.0f*dock_limit)
{
cDocked++;
if (waypoint < waypointnum)
{
      if (cDocked>=2)
      {
            waypoint++;  // Move to next waypoint
            cDocked = 0;
      }
}
else  // Termination condition for satellite
{
      if (cDocked>=5)
            ctrlTestTerminate(TEST_RESULT_NORMAL);
}
}

// Save debug data
DebugVecSignedShortNPS[1]  = (short) (r_rsw[0] * 10000.0f);
DebugVecSignedShortNPS[2]  = (short) (r_rsw[1] * 10000.0f);
DebugVecSignedShortNPS[3]  = (short) (r_rsw[2] * 10000.0f);
DebugVecSignedShortNPS[4]  = (short) (v_rsw[0] * 10000.0f);
```

```
DebugVecSignedShortNPS[5]  = (short) (v_rsw[1] * 10000.0f);
DebugVecSignedShortNPS[6]  = (short) (v_rsw[2] * 10000.0f);
DebugVecSignedShortNPS[7]  = (short) (r_goal[0] * 10000.0f);
DebugVecSignedShortNPS[8]  = (short) (r_goal[1] * 10000.0f);
DebugVecSignedShortNPS[9]  = (short) (r_goal[2] * 10000.0f);
DebugVecSignedShortNPS[10] = (short) (iRSW[0] * 10000.0f);
DebugVecSignedShortNPS[11] = (short) (iRSW[1] * 10000.0f);
DebugVecSignedShortNPS[12] = (short) (iRSW[2] * 10000.0f);
DebugVecSignedShortNPS[13] = (short) (OBSiRSW[0][0] * 10000.0f);
DebugVecSignedShortNPS[14] = (short) (OBSiRSW[1][0] * 10000.0f);
DebugVecSignedShortNPS[15] = (short) (OBSiRSW[2][0] * 10000.0f);

break;

default:
break;
}

// Recording the time that the control code took to run
timecheck = sysSphereTimeGet()-timecheck;

// For all maneuvers but the first one
if (maneuver_number>1)
{
// Run attitude and position controllers
// Mixer
ctrlMixWLoc(&firing_timesNPS, ctrlControlNPS, ctrlStateChaserNPS,
min_pulseNPS, duty_cycle_NPS-comp_delay_factor, FORCE_FRAME_INERTIAL);

// Re-enable IR flashes for state estimator
padsGlobalPeriodSetAndWait(200, ((unsigned int)(duty_cycle_NPS*10.0f))+5);

DebugVecUnsignedShortNPS2[8] = 1;

// Actuates the thrusters
propSetThrusterTimes(&firing_timesNPS);
}

// Timeout to terminate test
if (test_time>=330000)
ctrlTestTerminate(TEST_RESULT_TIMEOUT);

DebugVecSignedShortNPS[0]  = (short) (test_time * 0.01f);
commSendPacket(COMM_CHANNEL_STL,GROUND,sysIdentityGet(),
COMM_CMD_DBG_SHORT_SIGNED, (unsigned char *) DebugVecSignedShortNPS, 0);

DebugVecUnsignedShortNPS[0] = (unsigned short) (test_time * 0.01f);
dbgThrusterTimesPackFull(&DebugVecUnsignedShortNPS[1], firing_timesNPS);
commSendPacket(COMM_CHANNEL_STL,GROUND,sysIdentityGet(), COMM_CMD_THR_TIMES,
(unsigned char *) DebugVecUnsignedShortNPS, 0);

DebugVecUnsignedShortNPS2[0] = (unsigned short) (test_time * 0.01f);
DebugVecUnsignedShortNPS2[1] = waypoint;
DebugVecUnsignedShortNPS2[2] = cDocked;
DebugVecUnsignedShortNPS2[3] = (unsigned short) (LQR);
DebugVecUnsignedShortNPS2[4] = (unsigned short) (ctrlStateTarget[QUAT_2]*
10000.0f);
```

```
DebugVecUnsignedShortNPS2[5] = (unsigned short) (ctrlStateTarget[QUAT_3]*
10000.0f);
DebugVecUnsignedShortNPS2[6] = (unsigned short) (ctrlStateTarget[QUAT_4]*
10000.0f);
DebugVecUnsignedShortNPS2[7] = (unsigned short) (g_norm * 100.0f);
commSendPacket(COMM_CHANNEL_STL,GROUND,sysIdentityGet(),
COMM_CMD_DBG_SHORT_UNSIGNED, (unsigned char *) DebugVecUnsignedShortNPS2, 0);

DebugVecFloatNPS[0] = (float) test_time * 0.01f;
for (i=0;i<6;i++)
DebugVecFloatNPS[1+i] = ctrlControlNPS[i]*1000.0;
DebugVecFloatNPS[7] = timecheck;
commSendPacket(COMM_CHANNEL_STL,GROUND,sysIdentityGet(), COMM_CMD_DBG_FLOAT,
(unsigned char *) DebugVecFloatNPS, 0);

}
```

Appendix B NavZ Code Modified From MSCPC

Below is the NavZ function that is called in the control code in Appendix A. This NavZ code was modified from the original MSCPC code, which was designed by the NPS and implemented on SPHERES by Dr. McCamish and the SPHERES team.

```
/*
* ctrl_NPS_Inspection.c
*
* This file contains a library of position controllers developed by
* Major Shawn B. McCamish, Ph.D. candidate at the Naval PostGraduate
* School.  It has been modified by Christine Edwards.
*
* MIT Space Systems Laboratory
* http://ssl.mit.edu/spheres/
*
* Copyright 2007 Massachusetts Institute of Technology and Naval
* PostGraduate School.
*
* Last modified 2008-04-18
*/

#include "ctrl_NPS_Inspection.h"
#include "ctrl_LQR.h"
#include <string.h>
#include "math_matrix.h"
#include <math.h>

//Arbitrary spacecraft # limitation of 7
#define NUMSC_MAX   7

//Arbitrary obstacle # limitation of 14
#define NUMOBS_MAX 14

void ctrlNpsSimple_Inspection(float *a_tot, float v_norm, float r_norm, float
*v_rsw, float *r_rsw_passed, float *r_goal, float g_norm_passed, int sats,
int chase, float *iRSW, float *fRSW, float range, float V_max, float a_max,
float dock_limit, int nObs, float Lmax, float G, float L, float
OBSiRSW[3][9], float OBS_DoR, float stepsize, int ellipsoid, float
r_to_obstacle_inertial[3][9])
{
//LQR/APF Controller algorithm developed by Shawn McCamish
//from Naval Postgraduate School (NPS) from 2006-2007
//converts RSW pos and velocity of chase into RSW accleration commands

float r_rswt[3], r_rsw[NUMSC_MAX][3], g_normt, g_norm[NUMSC_MAX], a_rsw[3];
float ofsetg[NUMSC_MAX][3], Klqr[3][6], ChatterL;
float A[6][6], B[6][3], Qlqr1, bravo, bravo_den, Qlqr2;
```

```
float Rlqr1, Qlqr[6][6], Rlqr[3][3], Nlqr[6][3], state[6];
float posobsat[NUMSC_MAX][3], rsat[NUMSC_MAX], vobsatmag[NUMSC_MAX];
float vrepsmag[NUMSC_MAX], aobsatmag[NUMSC_MAX], arepsmag[NUMSC_MAX];
float Vreps[NUMSC_MAX], kreps[NUMSC_MAX], vdesreps[3], adesreps[3];
float areps[3], a_desreps[3], stopdist, OBS_Dsat, OBS_Ds, stdevs, wids;
float ksat_vatt, ksat_aatt, v_offsat[NUMOBS_MAX][3];
float krepo[NUMOBS_MAX], Vrepo[NUMOBS_MAX], vdesrepo[3], adesrepo[3];
float a_desrepo[3], vobsmag[NUMOBS_MAX], aobsmag[NUMOBS_MAX];
float vrepomag[NUMOBS_MAX], arepomag[NUMOBS_MAX], arepo[3];
float posobs[NUMOBS_MAX][3], robs[NUMOBS_MAX], stopdisto, OBS_Do;
float stdevo, wido, ko_aatt;
int i, iobsat, iso, iobs;
float *_Klqr, *_r_rsw;
float OBSiRSW_copy[3];
float a, b, c, lambda, beta, OBS_DoR_x, OBS_DoR_y, OBS_DoR_z;
//    float r_to_obstacle_mag;
//    float r_to_obstacle_copy[3], posobsivec[3];

_Klqr = (float *)Klqr;
_r_rsw = (float *)r_rsw;

//initialize variables:
memset(r_rswt, 0, sizeof(float)*3);
memset(r_rsw, 0, sizeof(r_rsw));
memcpy(r_rsw, r_rswt, sizeof(float)*3);
memcpy(&r_rsw[1][0], r_rsw_passed, sizeof(float)*3);
g_normt = 0.0f;
memset(g_norm, 0, sizeof(float)*NUMSC_MAX);
memcpy(&g_norm, &g_normt, sizeof(float));
memcpy(&g_norm[1], &g_norm_passed, sizeof(float));
memset(a_rsw, 0, sizeof(float)*3);
memset(a_tot, 0, sizeof(float)*3);
memset(ofsetg, 0, sizeof(ofsetg));
for (i=0;i<3*6;i++)
_Klqr[i]=1.0f;
ChatterL=0.3f;          //chatter limiter and may also improve plum problems
ofsetg[1][0] = iRSW[0] - fRSW[0];
ofsetg[1][1] = iRSW[1] - fRSW[1];
ofsetg[1][2] = iRSW[2] - fRSW[2];
////Linear Dynamics:
memset(A, 0, sizeof(A));
A[0][3]=1.0f;
A[1][4]=1.0f;
A[2][5]=1.0f;
memset(B, 0, sizeof(B));
B[3][0]=1.0f;
B[4][1]=1.0f;
B[5][2]=1.0f;

//// Attractive LQR for each Chase s/c (1 through sats-1):
if (g_norm[chase]>=dock_limit)
{
//// LQR Parameters
if (g_norm[chase]>=ChatterL)  //avoids chatter and divide by zero problems
{
Qlqr1=1.0f/g_norm[chase];      //normalize by norm relative range
bravo=g_norm[chase];
```

```
}
else
{
Qlqr1=1.0f/ChatterL;
bravo=ChatterL;
}
bravo_den=(mathVecMagnitude(&ofsetg[1][0],3)/(range))*V_max;
Qlqr2=bravo/(bravo_den*bravo_den);  //normalize by max relative velocity
memset(Qlqr, 0, sizeof(Qlqr));      //weight control err
Qlqr[0][0] = Qlqr1;
Qlqr[1][1] = Qlqr1;
Qlqr[2][2] = Qlqr1;
Qlqr[3][3] = Qlqr2;
Qlqr[4][4] = Qlqr2;
Qlqr[5][5] = Qlqr2;
Rlqr1=bravo/(a_max*a_max);        //normalize by max thruster acceleration
memset(Rlqr, 0, sizeof(Rlqr));      //weight control effort
Rlqr[0][0] = Rlqr1;
Rlqr[1][1] = Rlqr1;
Rlqr[2][2] = Rlqr1;
memset(Nlqr, 0, sizeof(Nlqr));
//// Calculate Gain Matrix for V=(x'Qx)+(u'Ru)+(2x'Nu) :
ctrlLQRSimple(Klqr, A, B, Qlqr, Rlqr, Nlqr);
//// Note: Klqr is mostly a constant 3x6 matrix:
state[0] = -1*r_goal[0];
state[1] = -1*r_goal[1];
state[2] = -1*r_goal[2];
state[3] = -1*v_rsw[0];
state[4] = -1*v_rsw[1];
state[5] = -1*v_rsw[2];
mathMatVecMult(a_rsw, (float **)Klqr, state, 3, 6);
}             // Attractive LQR within docking limit

//// Initialize Vectors and Matrix for Repulsive from obstacles:
memset(posobsat, 0, sizeof(posobsat));
memset(rsat, 0, sizeof(float)*NUMSC_MAX);
memset(vobsatmag, 0, sizeof(float)*NUMSC_MAX);
memset(vrepsmag, 0, sizeof(float)*NUMSC_MAX);
memset(aobsatmag, 0, sizeof(float)*NUMSC_MAX);
memset(arepsmag, 0, sizeof(float)*NUMSC_MAX);
memset(Vreps, 0, sizeof(float)*NUMSC_MAX);
memset(kreps, 0, sizeof(float)*NUMSC_MAX);
memset(vdesreps, 0, sizeof(float)*3);
memset(adesreps, 0, sizeof(float)*3);
memset(areps, 0, sizeof(float)*3);
memset(a_desreps, 0, sizeof(float)*3);
for (iobsat=0;iobsat<sats;iobsat++)  //loop through other spacecraft
{
////position vector from obsat toward chase (dir of rep)
iso=3*iobsat;
posobsat[iobsat][0]=_r_rsw[chase*3]  -_r_rsw[iso];
posobsat[iobsat][1]=_r_rsw[chase*3+1]-_r_rsw[iso+1];
posobsat[iobsat][2]=_r_rsw[chase*3+2]-_r_rsw[iso+2]; //chase wrt Others
rsat[iobsat]=mathVecMagnitude(&posobsat[iobsat][0],3);            //range
from Others
stopdist=(v_norm*v_norm)/(4*a_max);
```

```
OBS_Dsat=(2)*(Lmax+stopdist);          //sphere of influence of obstacle
Do>=radius
OBS_Ds=(2)*(G+stopdist);      //sphere of influence of Target spacecraft
if (iobsat!=chase)                    //as long as chase not self
{
if ( (rsat[iobsat]<=OBS_Dsat) && (g_norm[chase]>=(g_norm[iobsat]-Lmax)) &&
(g_norm[chase]>=(rsat[iobsat]-Lmax/2)) )
{
vobsatmag[iobsat] = -1.0f/rsat[iobsat]*mathVecInner(&posobsat[iobsat][0],
v_rsw, 3);
if (vobsatmag[iobsat]>0)
vrepsmag[iobsat]=vobsatmag[iobsat];
else
vrepsmag[iobsat]=0;
////attractive acceleration magnitude toward Obstacle:
aobsatmag[iobsat] = -1.0f/rsat[iobsat]*mathVecInner(&posobsat[iobsat][0],
a_rsw, 3);
if (aobsatmag[iobsat]>0)
arepsmag[iobsat]=aobsatmag[iobsat];
else
arepsmag[iobsat]=0;
if (iobsat==0) // if the Target spacecraft
{
stdevs=(OBS_Ds)/3;          //width of meters (numerator)
wids=2*stdevs*stdevs;        //2*variance (stdev^2)
Vreps[iobsat]=(float)( (exp(-1.0f*rsat[iobsat]*rsat[iobsat]/wids)-exp(-
1.0f*OBS_Ds*OBS_Ds/wids)) / (exp(-1.0f*Lmax*Lmax/wids)-exp(-
1.0f*OBS_Ds*OBS_Ds/wids)) );
if (rsat[iobsat]>=((L+L)/2))     //boundry implementation
{
ksat_vatt=Vreps[iobsat];
ksat_aatt=(float)exp(-fabs(rsat[iobsat]-Lmax));
vdesreps[0]=-1.0f*ksat_vatt*v_rsw[0];
vdesreps[1]=-1.0f*ksat_vatt*v_rsw[1];
vdesreps[2]=-1.0f*ksat_vatt*v_rsw[2];
areps[0]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
areps[1]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
areps[2]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
}
else
{
ksat_vatt=1;
ksat_aatt=1;
vdesreps[0]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
vdesreps[1]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
vdesreps[2]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
areps[0]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
areps[1]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
areps[2]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
}     //boundary of Target spacecraft for repulsive gain
adesreps[0]=vdesreps[0]/stepsize-areps[0];
adesreps[1]=vdesreps[1]/stepsize-areps[1];
adesreps[2]=vdesreps[2]/stepsize-areps[2];
//end     //if Target is an actual spacecraft/object with repulsion
}
else    //if not the Target spacecraft
{
```

```
stdevs=(OBS_Dsat)/3;          //width in meters (numerator)
wids=2*stdevs*stdevs;        //2*variance (stdev^2)
Vreps[iobsat]=(float)( (exp(-1.0f*rsat[iobsat]*rsat[iobsat]/wids)-exp(-
1.0f*OBS_Dsat*OBS_Dsat/wids)) / (exp(-1.0f*Lmax*Lmax/wids)-exp(-
1.0f*OBS_Dsat*OBS_Dsat/wids)) );
if (rsat[iobsat]>=((L+L)/2))       //boundry implementation
{
ksat_vatt=Vreps[iobsat]*(1.0f-(float)exp(-1.0f*g_norm[chase]));
ksat_aatt=(1.0f-(float)exp(-1.0f*g_norm[chase]))*(float)exp(-
1.0f*(float)fabs(rsat[iobsat]-Lmax));
vdesreps[0]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
vdesreps[1]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
vdesreps[2]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
areps[0]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
areps[1]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
areps[2]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
}
else
{
ksat_vatt=Vreps[iobsat];
ksat_aatt=Vreps[iobsat];
vdesreps[0]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
vdesreps[1]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
vdesreps[2]=ksat_vatt*vrepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
areps[0]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][0];
areps[1]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][1];
areps[2]=-1.0f*ksat_aatt*arepsmag[iobsat]/rsat[iobsat]*posobsat[iobsat][2];
}     //boundary of Target spacecraft for repulsive gain
adesreps[0]=vdesreps[0]/stepsize-areps[0];
adesreps[1]=vdesreps[1]/stepsize-areps[1];
adesreps[2]=vdesreps[2]/stepsize-areps[2];
}         //target is different then other spacecraft
}
else        //if not in the range of influence for repulsion
{
Vreps[iobsat]=0;
kreps[iobsat]=0;
memset(vdesreps, 0, sizeof(float)*3);
memset(adesreps, 0, sizeof(float)*3);
}         //if within region of influence of spacecraft
}
else        //if refering to self do nothing
{
Vreps[iobsat]=0;
kreps[iobsat]=0;
memset(vdesreps, 0, sizeof(float)*3);
memset(adesreps, 0, sizeof(float)*3);
}         //end of if not self loop
a_desreps[0]=a_desreps[0]+adesreps[0];
a_desreps[1]=a_desreps[1]+adesreps[1];
a_desreps[2]=a_desreps[2]+adesreps[2];
}             //loop through other spacecraft
//// Repulsive Potential due to obstacles:
//// Initialize Vectors and Matrix for Repulsive from obstacles:
memset(v_offsat, 0, sizeof(v_offsat)); //index must be obs+1 to ensure
connection
memset(krepo, 0, sizeof(float)*NUMOBS_MAX);
```

```
memset(Vrepo, 0, sizeof(float)*NUMOBS_MAX);
memset(vdesrepo, 0, sizeof(float)*3);
memset(adesrepo, 0, sizeof(float)*3);
memset(a_desrepo, 0, sizeof(float)*3);
memset(vobsmag, 0, sizeof(float)*NUMOBS_MAX);
memset(aobsmag, 0, sizeof(float)*NUMOBS_MAX);
memset(vrepomag, 0, sizeof(float)*NUMOBS_MAX);
memset(arepomag, 0, sizeof(float)*NUMOBS_MAX);
memset(arepo, 0, sizeof(float)*3);
if (nObs>=1)
{
memset(posobs, 0, sizeof(posobs));
memset(robs, 0, sizeof(float)*NUMOBS_MAX);
for (iobs=0;iobs<nObs;iobs++)
{
posobs[iobs][0] = _r_rsw[chase*3]   -OBSiRSW[0][iobs]; //chase wrt Obs
posobs[iobs][1] = _r_rsw[chase*3+1]-OBSiRSW[1][iobs]; //chase wrt Obs
posobs[iobs][2] = _r_rsw[chase*3+2]-OBSiRSW[2][iobs]; //chase wrt Obs
OBSiRSW_copy[0] = OBSiRSW[0][iobs];
OBSiRSW_copy[1] = OBSiRSW[1][iobs];
OBSiRSW_copy[2] = OBSiRSW[2][iobs];
robs[iobs]=mathVecMagnitude(&posobs[iobs][0],3); //range from Obs
memcpy(&v_offsat[iobs][0], v_rsw, sizeof(float)*3);
stopdisto=(mathVecMagnitude(&v_offsat[iobs][0],3)*mathVecMagnitude(&v_offsat[
iobs][0],3))/(4*a_max);

if (ellipsoid>0)
{
a = 0.54f;
b = 0.08f;
c = 0.54f;
lambda =
atan(a/b*r_to_obstacle_inertial[1][iobs]/r_to_obstacle_inertial[0][iobs]);
beta =
atan(b/c*r_to_obstacle_inertial[2][iobs]/r_to_obstacle_inertial[1][iobs]/sin(
lambda));
OBS_DoR_x = a*cos(beta)*cos(lambda);
OBS_DoR_y = b*cos(beta)*sin(lambda);
OBS_DoR_z = c*sin(beta);
OBS_DoR =
0.153f+0.01f+sqrt(OBS_DoR_x*OBS_DoR_x+OBS_DoR_y*OBS_DoR_y+OBS_DoR_z*OBS_DoR_z
);
}


OBS_Do=2*(OBS_DoR+stopdisto);          //sphere of influence of obstacle
Do>=radius
if ( (robs[iobs]<=OBS_Do)  && (r_norm>=(mathVecMagnitude(OBSiRSW_copy,3)-
(3*OBS_DoR))) )
{
stdevo=(OBS_Do-OBS_DoR)/3;          //width of OBS_Do meters (numerator)
wido=2*stdevo*stdevo;       //2*variance (stdev^2)
vobsmag[iobs] = -
1.0f/robs[iobs]*mathVecInner(&posobs[iobs][0],&v_offsat[iobs][0],3);
if (vobsmag[iobs]>0)
vrepomag[iobs]=vobsmag[iobs];
else
```

```
vrepomag[iobs]=0;
Vrepo[iobs]=(float)( (exp(-1.0f*robs[iobs]*robs[iobs]/wido)-exp(-
1.0f*OBS_Do*OBS_Do/wido)) / (exp(-1.0f*OBS_DoR*OBS_DoR/wido)-exp(-
1.0f*OBS_Do*OBS_Do/wido)) );
krepo[iobs]=vrepomag[iobs]*Vrepo[iobs];
vdesrepo[0]=krepo[iobs]/robs[iobs]*posobs[iobs][0];
vdesrepo[1]=krepo[iobs]/robs[iobs]*posobs[iobs][1];
vdesrepo[2]=krepo[iobs]/robs[iobs]*posobs[iobs][2];
aobsmag[iobs] = -1.0f/robs[iobs]*mathVecInner(&posobs[iobs][0], a_rsw, 3);
if (aobsmag[iobs]>0)
arepomag[iobs]=aobsmag[iobs];
else
arepomag[iobs]=0;
arepo[0] = -1.0f*arepomag[iobs]/robs[iobs]*posobs[iobs][0];
arepo[1] = -1.0f*arepomag[iobs]/robs[iobs]*posobs[iobs][1];
arepo[2] = -1.0f*arepomag[iobs]/robs[iobs]*posobs[iobs][2];
ko_aatt=Vrepo[iobs];
adesrepo[0]=vdesrepo[0]/stepsize-ko_aatt*arepo[0];
adesrepo[1]=vdesrepo[1]/stepsize-ko_aatt*arepo[1];
adesrepo[2]=vdesrepo[2]/stepsize-ko_aatt*arepo[2];
}
else
{
Vrepo[iobs]=0;
krepo[iobs]=0;
memset(vdesrepo, 0, sizeof(float)*3);
memset(adesrepo, 0, sizeof(float)*3);
}         //if within region of influence of obstacle
a_desrepo[0]=a_desrepo[0]+adesrepo[0];
a_desrepo[1]=a_desrepo[1]+adesrepo[1];
a_desrepo[2]=a_desrepo[2]+adesrepo[2];
}             //looping through obstacles
}                 //if obstacles

// These equations used for the original NPS design
//    a_tot[0] = a_rsw[0]+a_desreps[0]+a_desrepo[0];
//    a_tot[1] = a_rsw[1]+a_desreps[1]+a_desrepo[1];
//    a_tot[2] = a_rsw[2]+a_desreps[2]+a_desrepo[2];

// These equations used for the NavZ design
a_tot[0] = a_rsw[0]+a_desrepo[0];
a_tot[1] = a_rsw[1]+a_desrepo[1];
a_tot[2] = a_rsw[2]+a_desrepo[2];
}
```

Appendix C LQR Code

Below is the LQR function that is called by the NavZ function.

```
/*
 * ctrl_LQR.c
 *
 * This file contains a library of LQR solvers.  BalanceMatrix, ElmHes, and
HQR functions
 * from the "Numerical Recipes in C" 2nd Edition.
 *
 * MIT Space Systems Laboratory
 * http://ssl.mit.edu/spheres/
 *
 * Copyright 2007 Massachusetts Institute.
 *
 * Last modified 2007-10-25
 */

#include "ctrl_LQR.h"
#include <string.h>
#include "math_matrix.h"
#include "math_LQR.h"

void ctrlLQRSimple(float Klqr[3][6], float A[6][6], float B[6][3], float
Qlqr[6][6], float Rlqr[3][3], float Nlqr[6][3])
{
      float Rinv[3][3], BinvR[6][3], BinvRBt[6][6], H[12][12], L[13], wi[13];
      float **HH;
      float K1,K2;
      int i, j;

      memset(Rinv, 0, sizeof(Rinv));

      // Calculate the inverse of R (R is diagonal positive definite)
      Rinv[0][0] = 1.0f/Rlqr[0][0];
      Rinv[1][1] = 1.0f/Rlqr[1][1];
      Rinv[2][2] = 1.0f/Rlqr[2][2];

      // Finding B*inv(R)*B'
      mathMatMatMult((float **)BinvR, (float **)B, (float **)Rinv, 6, 3, 3);
      mathMatMatTransposeMult((float **)BinvRBt, (float **)BinvR, (float
**)B, 6, 3, 6);

      // Create the Hamiltonian matrices
      // H = [A -B*Rinv*B';-Q -A']
      for (i=0;i<6;i++)
      {
            for (j=0;j<6;j++)
```

```
            {
                H[i][j]=A[i][j];
                H[i][j+6]=-BinvRBt[i][j];
                H[i+6][j]=-Qlqr[i][j];
                H[i+6][j+6]=-A[j][i];
            }
        }

        // Balance matrix
        HH = mathConvertMatrix(&H[0][0],1,2*6,1,2*6);
        mathBalanceMatrix(HH,2*6);

        // Hessenberg reduction
        mathElmHes(HH,2*6);

        // Find closed loop pole locations (saved in L)
        mathHQR(HH,2*6,(float *)L,(float *)wi);

        // Matrices simplified to solve using Ackermann's Formula
        // Gains are:
        K1 = L[2]*L[4];
        K2 = -L[2]-L[4];

        Klqr[0][0] = K1;
        Klqr[1][0] = 0.0;
        Klqr[2][0] = 0.0;

        Klqr[0][1] = 0.0;
        Klqr[1][1] = K1;
        Klqr[2][1] = 0.0;

        Klqr[0][2] = 0.0;
        Klqr[1][2] = 0.0;
        Klqr[2][2] = K1;

        Klqr[0][3] = K2;
        Klqr[1][3] = 0.0;
        Klqr[2][3] = 0.0;

        Klqr[0][4] = 0.0;
        Klqr[1][4] = K2;
        Klqr[2][4] = 0.0;

        Klqr[0][5] = 0.0;
        Klqr[1][5] = 0.0;
        Klqr[2][5] = K2;

}
```

Bibliography

[1] Nolet, S., Saenz-Otero, A., Miller, D. W., and Fejzic, A., "SPHERES Operations Aboard the ISS: Maturation of GN&C Algorithms in Microgravity," *30th Annual AAS Guidance and Control Conference*, No. AAS 07-042, Breckenridge, Colorado, February 2007.

[2] Kong, E. M. C., Otero, A. S., Nolet, S., Berkovitz, D. S., Miller, D. W. and Sell, S. W., "SPHERES as a Formation Flight Algorithm Development and Validation Testbed: Current Progress and Beyond," *Proceedings of the 2nd International Symposium on Formation Flying Missions and Technologies*, Washington DC, September 2004.

[3] Kong, E. M., Hilstad, M. O., Nolet, S. and Miller, D. W., "Development and Verification of Algorithms for Spacecraft Formation Flight Using the SPHERES Testbed: Application to TPF," *Proceedings of SPIE, Vol. 5491, New Frontiers in Stellar Interferometry*, October 2004, pp. 308-319.

[4] Shawn B. McCamish, "Distributed Autonomous Control of Multiple Spacecraft During Close Proximity Operations, " PhD dissertation, Electrical Engineering Dept., Naval Postgraduate School, Monterey, CA, 2007.

[5] McCamish, S. B., Romano, M., and Yun, X., "Autonomous Distributed Control Algorithm for Multiple Spacecraft in Close Proximity Operations," *AIAA Guidance, Navigation and Control Conference*, Hilton Head, SC, August 2007.

[6] McCamish, S. B., Romano, M., and Yun, X., "Autonomous Distributed LQR/APF Control Algorithm for Multiple Small Spacecraft during Simultaneous Close Proximity Operations," *Proceedings of the 21st Annual AIAA/USU Conference on Small Satellites*, Logan, UT, August 2007.

[7] McCamish, S. B., and Romano, M., "Simulations of Relative Multiple-Spacecraft Dynamics and Control by MATLABSimulink and Satellite Tool Kit," *AIAA Modeling and Simulation Technologies Conference*, Hilton Head, SC, August 2007.

[8] McCamish, S.B., Romano, M., Nolet, S., Edwards, C.M., and Miller, D.W., "Ground and Space Testing of Multiple Spacecraft Control During Close-Proximity Operations." *AIAA Guidance, Navigation, and Control Conference*, Honolulu, Hawaii, August 2008.

[9] Roger, A.B., and McInnes, C.R., "Safety Constrained Free-Flyer Path Planning at the International Space Station," *AIAA Journal of Guidance, Navigation, and Dynamics*, Vol. 23, No. 6, 2000, pp. 971-979.

[10] Fredrickson, S.E., Duran, S., and Mitchell, J.D., "Mini AERCam Inspection Robot for Human Space Missions," number AIAA 2004-5843, *AIAA Space 2004 Conference and Exhibit*, San Diego, California, September 2004.

[11] Hiltz, M., Ruta, K., Boyle, K., and Terra, L., "Space Shuttle Inspection and Repair Boom Sensor System for Return to Flight," *55th International Astronautical Congress*, Vancouver, Canada, 2004.

[12] Greaves, S., Boyle, K, and Doshewnek, N., "Orbiter Boom Sensor System and Shuttle Return to Flight: Operations Analyses," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, California, 2005.

[13] Aziz, S., "Lessons Learned From the Robotics Operations on STS-114 (Return to Flight)," number AIAA 2006-5955, *AIAA SpaceOps 2006 Conference*, 2006.

[14] Choset, H., et all, "Path Planning and Control for AERCam, a Free-flying Inspection Robot in Space," *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*, Detroit, Michigan, May 1999.

[15] Nolet, S., "The SPHERES navigation system: from early development to on-orbit testing," number AIAA-2007-6354, *AIAA Guidance, Navigation and Control Conference and Exhibit*, Hilton Head, South Carolina, August 2007.

[16] Nolet, S., Kong, E., and Miller, D. W., "Design of an algorithm for autonomous docking with a freely tumbling target," *Modeling, Simulation, and Verification of Space-based Systems II*, P. Motaghedi, Vol. 5799, SPIE, 2005, pp. 123-134.

[17] Nolet, S. and Miller, D. W., "Autonomous docking experiments using the SPHERES testbed inside the ISS," *Sensors and Systems for Space Applications*, edited by R. T. Howard and R. D. Richards, Vol. 6555, SPIE, 2007.

[18] Nolet, S., Kong, E., and Miller, D. W., "Autonomous docking algorithm development and experimentation using the SPHERES testbed," *Spacecraft Platforms and Infrastructure*, edited by P. Tchoryk, Jr. and M. Wright, Vol. 5419, SPIE, 2004, pp. 1-15.

[19] Nolet, S., "Development of a Guidance, Navigation and Control Architecture and Validation Process Enabling Autonomous Docking to a Tumbling Satellite," Sc. D. thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2007.

[20] Enright, J., Hilstad, M., Saenz-Otero, A. and Miller, D., "The SPHERES Guest Scientist Program: collaborative science on the ISS," *Proceedings of the 2004 IEEE Aerospace Conference*, 6-13 March 2004.

[21] Microsoft Visual C++, Microsoft Corporation, URL: http://msdn2.microsoft.com/en-us/visualc/default.aspx [cited 31 January 2008].

[22] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P., *Numerical Recipes in C, The Art of Scientific Computing*, 2nd Edition, Cambridge University Press, Cambridge, U.K., 1992.

[23] MATLAB and Simulink, The MathWorks Incorporated, URL: http://www.mathworks.com/ [cited 18 September 2006].

[24] Wertz, J.R., *Spacecraft Attitude Determination and Control*, Vol. 73, D. Reidel Publishing Company, Boston, U.S., 1985.

[25] Yun, X., and Tan, K., "A Wall-Fallowing Method for Escaping Local Minima in Potential Field Based Motion Planning," *Proceedings of the International Conference on Advanced Robotics* (ICAR), July 1997, pp. 421-426.

[26] Se, S., Jasiobedzki, P., Wildes, R., "Stereo-Vision Based 3D Modeling of Space Structures," *Sensors and Systems for Space Applications*, edited by R.T. Howard and R.D. Richards, Vol.6555, SPIE, 2007, pp. 1-13.

[27] de Coelho, L.X., Campos, M.F.M., and Kumar, V., "Computer Vision-Based Navigation for Autonomous Blimps," *Proceedings of the International Symposium on Computer Graphics, Image Processing, and Vision*, Rio de Janeiro, October 1998, pp. 287-294.

[28] Hilstad, M., "A Multi-Vehicle Testbed and Interface Framework for the Development and Verification of Separated Spacecraft Control Algorithms," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2002.