DEVELOPMENT OF A GEOSPATIAL DATA-SHARING METHOD FOR UNMANNED VEHICLES BASED ON THE JOINT ARCHITECTURE FOR UNMANNED SYSTEMS (JAUS)

By

CARL PRESTON EVANS, III

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2005

“It is difficult to say what is impossible, for the dream of yesterday is the hope of today and the reality of tomorrow.”

—Dr. Robert H. Goddard

To my grandparents, Lottie and James Patterson, for their never-ending, unconditional love, and support of me. Because of them, I have been able to see and do things that they were unable to. In the words of the great Albert Einstein, I have seen farther because they have allowed me to stand on their shoulders.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my committee chair (Dr. Carl D. Crane III) for his patience with me during the process of finishing this work, and for giving me the great opportunity to study and do my master's research at the University of Florida's Center for Intelligent Machines and Robotics. Dr. Crane and I have had a professional relationship since 2000, when we met at a Joint Architecture for Unmanned Systems (JAUS) working group meeting. I look forward to a continued professional relationship as we continue to research the vast field of autonomous systems. Thanks also go to my other committee members (Drs. John Schueller and Christopher Neizrecki) for their valuable contributions to this study.

Thanks also go out to Mr. Dan Deguire, Project Manager in Foster-Miller's Design and Systems Integration (DSI) Group. I had the honor of working for Dan for two-and-a-half years, starting as a co-op student and ending as a Design Engineer. While at Foster-Miller, Dan funded my first foray into obstacle detection and autonomous navigation (my undergraduate senior design project). I appreciate the friendship that we continue to share.

Special thanks go out to the Center for Intelligent Machines and Robotics (CIMAR) Team CIMAR; competitors in the inaugural Defense Advanced Research Projects Agency (DARPA) Grand Challenge held in March 2004. I particularly those who were on (or otherwise contributed to work of) the perception team: Mel Torrie,

Sarah Gray, Kristopher Klingler, Charles Smith, Sanjay Solanki, Danny Kent, Erica Zawodny-McArthur, and Donald McArthur.

Thanks also go out to the members of the JAUS World Model Subcommittee for making each of my meetings feel like a thesis defense. Their valuable suggestions helped to shape this document, particularly Chapter 4.

I thank Chad Tobler (a friend and biker buddy, late in my years at CIMAR) for keeping me sane and for providing valuable insight during the writing of this study.

Thanks also go to my great friend Matthew Bokach from the School of Natural Resources and Environment at the University of Florida. I thank him for being there for me in many ways both personally and professionally. I especially appreciate his help with this study. From the day that he told me that two points of equal latitude, longitude, and elevation are not always coincident, I have learned much from him.

Of course I must thank my new coworkers (Todd, Parag, Patrick, and Mark) at Applied Perception, Inc. (API), for giving me a hard time for taking so long to finish this study, but more importantly for their kind words of support. I look forward to many successful years with them at API.

Last but certainly not least, thanks go to the Air Force Research Laboratory at Tyndall Air Force Base, FL, for funding this work. For over a decade, they have been supporters of the work done by the talented group of roboticists at the University of Florida's Center for Intelligent Machines and Robotics. Without their support, this study could not have been completed. I am thankful for their support, and I look forward to continued work with them in the future.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DEVELOPMENT OF A GEOSPATIAL DATA-SHARING METHOD FOR UNMANNED VEHICLES BASED ON THE JOINT ARCHITECTURE FOR UNMANNED SYSTEMS (JAUS)

By

Carl Preston Evans, III

August 2005

Chair: Carl D. Crane, III
Major Department: Mechanical and Aerospace Engineering

A task performed almost effortlessly by humans, perception is perhaps one of the most difficult tasks for autonomous vehicles. While substantial research has been done to develop these technologies, few studies have examined ways for multiple heterogeneous unmanned systems to cooperate in their perception tasks. Our study examined ways to model both perceived and *a priori* geospatial information, and formatting these data so that they can be used by the growing unmanned systems community.

We introduce a perception system model, consisting of distributed "smart" sensors. This system of sensors was developed for the Team CIMAR entry into the inaugural DARPA Grand Challenge autonomous vehicle competition held in March 2004. The Smart Sensor Architecture proved to be a power method of distributing the possessing of sensor data to systems developed by engineers who best knew a particular sensor modality. By standardizing the logical, transport, and electrical interfaces, the smart sensor architecture developed into a powerful world modeling method.

We also investigated current geospatial data-modeling methods used in the unmanned systems and geodetic information systems (GIS) communities. Our study determined the commonalities among current methods and resulted in a first-generation geospatial data-sharing standard for unmanned systems compliant with the Joint Architecture for Unmanned Systems (JAUS).

CHAPTER 1
INTRODUCTION AND RESEARCH PROBLEM

1.1 Introduction

Imagine a world without languages, with no standard methods of communicating with other people. Imagine a world in which an individual or a small group of individuals had a language completely different from that of other individuals or groups. Image also that even the most primitive methods of communicating required an interpreter. It would be unreasonable to expect two people to be able to come together and (with minimal effort) understand each other. This is the state of the world in the unmanned systems community. Developing a common language for unmanned systems is not trivial. However, as unmanned systems become more commonplace and gain the ability to interoperate and ultimately collaborate, a standard communications method or language must be developed.

As it has with a number of technological innovations throughout recent history, the United States Department of Defense (DoD) is helping to revolutionize the unmanned systems community by pushing the development of a standard communications method for all future DoD unmanned systems. Recognizing the increased acquisition and maintenance costs for a growing fleet of unmanned systems with proprietary interfaces, the Office of the Secretary of Defense chartered the Joint Architecture for Unmanned Ground Systems (JAUGS) Working Group to address these concerns. The JAUGS Working Group was tasked with developing an initial standard for interoperable unmanned ground systems. In 2002, the charter of the JAUGS Working Group was

modified such that their efforts would extend to all unmanned systems, not only ground systems. The standard was therefore renamed Joint Architecture for Unmanned Systems (JAUS).

Unmanned systems are becoming increasingly popular. In fact, large U.S. government acquisition programs such as Future Combat Systems (FCS) and Man Transportable Robotic Systems (MTRS) show that unmanned systems are here to stay. The Future Combat Systems (FCS) program is an ambitious multi-billion-dollar program with a goal of integrating autonomous, semi-autonomous, and tele-operated systems into the battlefield of tomorrow. Man Transportable Robotic Systems (MTRS) is a large multi-million-dollar program that requires a large number of tele-operated unmanned systems for use in the task of explosive ordnance disposal (EOD). Both the FCS and MTRS programs require systems that can communicate with one another (operator control units to vehicle or inter-vehicle) using a shared language. This language (JAUS) is the subject of our study.

Currently, JAUS supports tele-operation; and, to an extent, primitive levels of semi-autonomy. Technological innovations in the areas of sensors, sensor processing and fusion, perception, and intelligence have advanced robotics so much that demands that were not long ago far-fetched are fast becoming a reality. To this end, JAUS must adapt to meet the growing requirements of the semi-autonomy and autonomy camps of the unmanned systems community. Types of autonomous behaviors are as numerous as human behaviors. However, for unmanned systems, the next step in the natural progression beyond tele-operation is the development of assisted tele-operation and autonomous navigation and obstacle avoidance.

It is as difficult to define a new all-encompassing language for unmanned systems as it would be for humans. In the context of a particular mission, however, it is possible to develop a syntax that can be used to communicate relevant information. By initially limiting the scope of JAUS and incrementally adding functionality, a robust language is being built.

The focus of our study was on allowing JAUS-based unmanned systems to share geospatial data. These geospatial data are needed to support the tasks of obstacle detection, obstacle avoidance, and path planning among multiple JAUS subsystems. The concept of the world model helps to put this work into perspective. Meystel and Albus [18] defined the world model as

> the intelligent system's best estimate of the state of the world. The world model includes a database of knowledge about the world, plus a database management system that stores and retrieves information. The world model also contains a simulation capability that generates expectations and predictions. The world model provides answers to requests about the present, past, and probable future states of the world. The world model provides this information service to the behavior generation system element in order to make intelligent plans and behavioral choices. It provides information to the sensory processing system element to perform correlation, model matching, and model-based recognition of states, objects, and events. It provides information to the value judgment system element to compute values such as cost, benefit, risk, uncertainty, importance, and attractiveness. The world model is kept up to date by the sensory processing system element.

A world model presents unending directions to investigate. No doubt, many such investigations have begun. Our study focuses on the database of knowledge. We examined how the data are stored inside the database, and how databases can share data using a common language. In the context of JAUS, our study presents a first-generation standard for sharing a database of knowledge. Because unmanned systems used in the JAUS community are outdoor vehicles (and because of the desired tasks) these world model databases store geospatial data. A review of the relevant literature formed a solid

basis for creating this standard. We also introduced the implementation of a perception system.

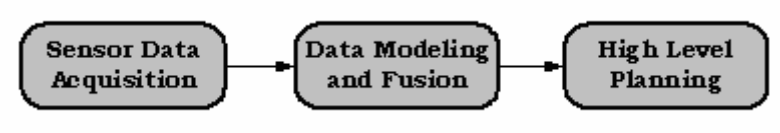

Figure 1-1. Example of the processes leading up to higher-level planning

Geospatial data generated by an unmanned system are only as good as the system's sensors and its sensor fusion and registration methods. Also important is what is done with these geospatial data after they have been fused and registered (high-level planning and intelligent behaviors). These important issues fall outside the scope of our study. Figure 1-1 shows some of the processes leading up to high-level planning. Figure 1-2 shows how our study fits in. The message-set generated by this study will allow different databases to share knowledge among themselves or with higher-level planning processes.

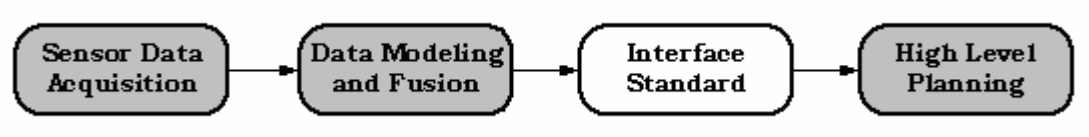

Figure 1-2. How our study fits into the higher-level planning process

1.2 Research Problem

Our study takes its direction from the following research problem.

> Given the experience and knowledge gained from examining current methods of modeling geospatial data within the unmanned systems and geographic information systems (GIS) communities and from implementing a perception system for an unmanned ground system, create a first generation geospatial data-sharing method for unmanned systems. Present this in a format consistent with the Joint Architecture for Unmanned Systems (JAUS) messaging framework.

This is a broad and open-ended topic. However, it must be addressed. As the capabilities of JAUS are extended, being without a method for communicating even the most basic forms of obstacle data would be a severe limitation. The primary purpose of

the standard presented in our study is to support mission planning. However other applications (such as data visualization) also benefit.

The contribution of our study is a first-generation method recommended for sharing data needed by state-of-the-art real-world, unmanned systems. The recommendations may be seen as guidelines for a first attempt (at least within the JAUS community) to allow multiple disparate unmanned systems (from different organizations, with completely different perception implementations) to share data.

CHAPTER 2
REVIEW OF RELEVANT LITERATURE

The most difficult behaviors for unmanned systems are perception and reasoning. Reasoning for an unmanned system is highly dependent on the quality of the estimation of the environment in which the unmanned system operates. This estimation is often used to support higher level behaviors performed by either the unmanned system or a human operator through tele-operation. Each system typically has its own method for modeling and sharing data. As we move towards increased interoperability among unmanned systems from different vendors, work must be done to bridge the gap between different methods of representing sensed data and providing those data to disparate unmanned systems. Again, this is the focus of our study; to provide a first generation standardized method for modeling the environments that unmanned systems operate in and then providing those data to other concerned manned or unmanned systems.

Much work has been done in recent years to move toward true interoperability between unmanned systems. One of the major efforts towards reaching this goal is the Joint Architecture for Unmanned Systems (JAUS). JAUS is a standard that defines the format of messages that travel between unmanned systems. Since it is fast becoming the standard for military unmanned systems, JAUS provides a suitable base upon which to build a first generation world modeling standard for unmanned systems.

2.1 Joint Architecture for Unmanned Systems (JAUS)

The Joint Architecture for Unmanned Systems (JAUS) is a messaging standard being developed with overall goals of reducing life cycle costs, enabling fast integration

of new technologies, and facilitating interoperability amongst heterogeneous unmanned systems. In 1998, the Office of the Secretary of Defense (OSD) chartered the Joint Architecture of Unmanned Ground Systems (JAUGS) Working Group and tasked this working group with developing a common model for messages used for controlling and monitoring processes within unmanned ground systems. Now the Joint Architecture for Unmanned Systems (JAUS), the working group is tasked with expanding the standard to the entire domain of unmanned systems. This group is currently represented by a diverse group of members from government, industry, and academic institutions. By having a wide range of input in developing the standards, JAUS is better prepared for wide acceptance by the unmanned systems community.

2.1.1 Tenets of JAUS

To ensure the flexibility, extensibility, and ultimately the longevity of the emerging JAUS standard, it was developed with four main tenets. These are: technology independence, hardware independence, platform independence, and mission independence [16].

The technology independence of JAUS assures that the messages that compose the JAUS standard as well as the methods for transporting the messages are not dependent on any past, present, or developing standard. For example, many JAUS implementation use the user datagram protocol (UDP) and the internet protocol (IP) for data transmission. Other implementations may, however, use asynchronous serial communications links such as EIA/TIA 232. There may be cases where one communications method is preferred over another. By restricting the dependence on a communications technology, JAUS leaves this decision to the system developer and thus remains very flexible. By defining only the messages to be communicated, JAUS will remain relevant over time.

The Hardware Independence rule is similar to the technology independence requirement. JAUS does not rely on knowledge of the structure of an unmanned system. There are no assumptions about the type of platform or the contents of the platform. So long as a system has adequate hardware to create, receive, process, and respond to the standardized JAUS messages, it is considered to be compliant with the specification.

Platform independence is the third tenet of JAUS. There are no assumptions about the type of systems that will use JAUS. The JAUS standard is just as useful for large tanks as it is for miniature microcontroller based unattended sensors. Surely as systems become more embedded, the read only memory (ROM), random access memory (RAM), and computing resources available decrease. Therefore an embedded system is less likely to be able to support large complex JAUS messages. This is acceptable as JAUS is very flexible with respect to the messages that each system must support. With the exception of a small number of core input and output messages, JAUS allows systems to use only the messages (as well as fields within those messages) that they need to perform their function.

JAUS also does not presuppose that the unmanned systems based on the specification are designed for any particular mission. This is the mission independence tenet of JAUS. By defining a comprehensive message set, it is hoped that JAUS developers can assemble systems that can complete any mission. Surely this is intractable, but with the guidance the diverse membership of the JAUS working group, JAUS has a firm foundation on which to build.

2.1.2 System Structure of JAUS

The Joint Architecture for Unmanned Systems consists of a number of hierarchical elements that work together to form a complete JAUS compliant unmanned system. The

lowest level of abstraction within JAUS is the component. Going up the chain of complexity, a JAUS node consists of multiple components, a subsystem consists of one or more nodes, and a JAUS system consists of one or more subsystems. Figure 2-1 shows the structure of a JAUS system. Not show in this figure is the concept of multiple instances of a component. This feature is included in JAUS to support component redundancy.

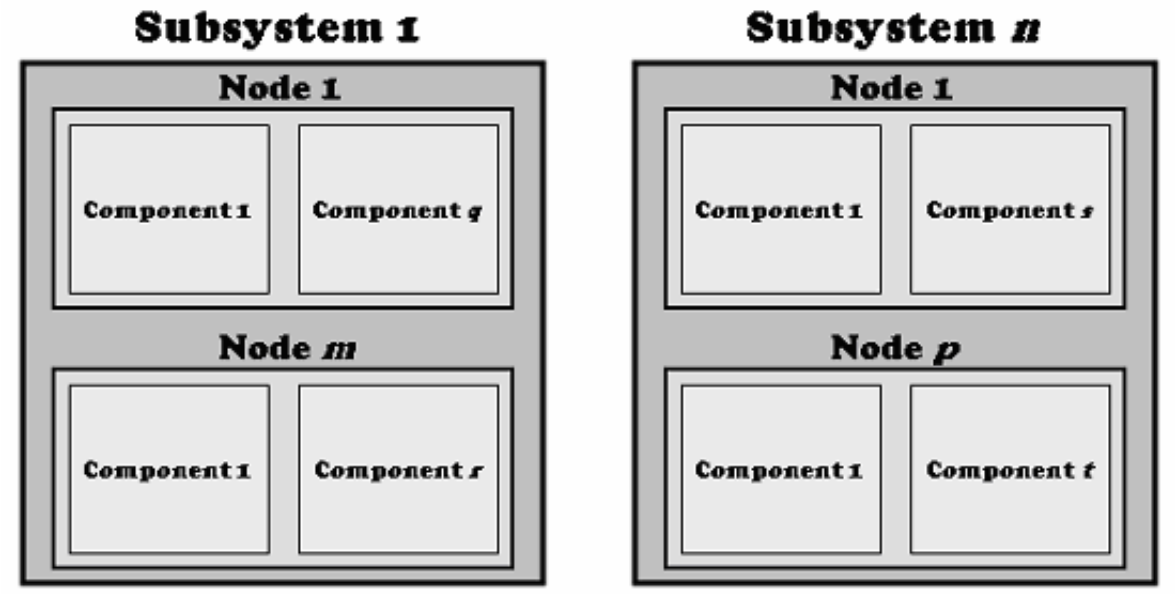

Figure 2-1. System structure of JAUS

The component encapsulates a specific function and the input and output messages necessary to command, control, and monitor the component. For example, the JAUS Primitive Driver component is responsible for the low-level command and control of an unmanned system. It controls and reports current status of the lowest level devices on the platform and reports platform specific data such as platform name and dimensions. Another component, the JAUS Global Pose component, interfaces to a device or a number of devices that are capable of providing the platform with its current global position, orientation, and orientation rate information. These are just two examples of

JAUS components. The JAUS Reference Architecture currently defines 26 components, each with its own specific function. The Reference Architecture allows up to 254 components to operate within a JAUS node.

A node is a single computing entity that consists of one or more JAUS components running in a tightly coupled manner. In this context, tightly coupled implies that the computing entities are not linked by any external connections. Instead, they are connected internally. This could be by function calls or shared memory, for example. If two or more components are to be linked by an external communications medium, they should be considered separate nodes. The JAUS standard currently allows up to 254 nodes within a subsystem.

A subsystem is device that performs a function through the synergy of the component containing nodes within it. There must be at least one node within a subsystem. This node may contain all the components necessary for the subsystem to perform its function. The subsystem may also contain a number of nodes that each provide components necessary for the subsystem to perform its function. The JAUS standard currently allows up to 254 subsystems to operate within a JAUS system.

A system consists of one or more subsystems working together for some useful purpose. This is the highest level within the JAUS hierarchy. JAUS currently does not permit communications between different JAUS systems. Within a system, however, any component, node, or component may communicate with any other component, node, or subsystem.

The hope is that the JAUS standard is generalized enough that it will not inhibit the creativity of the engineers and scientists developing these systems. Of course it is not

possible to account for all possible unmanned system scenarios. Because of this, the JAUS standard has been developed to allow for the development of user-defined components. The idea is that as these user-defined components mature and their usefulness is recognized by the JAUS community, they would be incorporated into the JAUS Reference Architecture. What is most important overall about JAUS is that it standardizes the interface between these components. As unmanned systems become more and more common place, without JAUS or some industry-wide JAUS-like standard, the interoperability issues will only be compounded.

2.1.3 World Model Subcommittee for JAUS

In October 2002, the Joint Architecture for Unmanned Systems Reference Architecture Committee's World Model Subcommittee was established to address the growing need within the unmanned systems community for a messaging architecture that allows multiple heterogeneous unmanned systems to share geospatial data. The task of this subcommittee was to develop the methods to allow modeling and sharing of geospatial data within the JAUS framework. For JAUS, the primary purpose for modeling and sharing of these data is to support the tasks of mission planning and distributed mapping for autonomous systems. JAUS is focused on the practical approach to unmanned systems and therefore so should a JAUS standard for geospatial data modeling.

2.2 Real-Time World Modeling Methods

The field of mobile robotics is generally interested in real time world modeling methods. Typically these world modeling methods support the task of reflexive obstacle avoidance whereby an unmanned system uses an instantaneous view of the environment to effect change in its current mission. For example, an unmanned system may be tasked

to autonomously navigate to a given waypoint without colliding with anything along the way. Similar to a human reflexively reacting to a sudden undesired condition, an unmanned system given this task may reflexively respond to obstacles that appear within the field of view of its sensors. Often these methods require very little modeling or processing of the sensor data. Of paramount concern is the safety of the systems.

It is often desired or necessary to have unmanned systems accumulate a model of the environment in which they operate. This may simply be for the sake of building an accumulative map of the environment or it may be to allow the unmanned system to make a more informed decision should it decide that it needs to modify its current behavior in order to successfully complete its given task. For example, if an unmanned system can perceive the environment at a distance that extends far beyond a range at which the system must act reflexively to maintain the safety of the system, those additional data could be used to reactively re-plan a path that avoids the hazard completely. At the very least, an accumulative model of the environment would provide the system with the ability to, should it have to act reflexively, choose the best long term plan.

Typically both reflexive and reactive obstacle avoidance systems use a tessellated raster grid based data structure to represent the environment. These raster grids are most commonly used for real-time world modeling because it is simple to project the sensor view into a two-dimensional Cartesian grid.

2.2.1 Raster Occupancy Grid

Sensors are all prone to errors that affect the quality of their data. Some of the sensors, such as radar and sonar, have wide fields of view, but very low resolution within their fields of view. To handle the issues of uncertainty and errors in the sensor data, the

concept of the occupancy grid was introduced by Elfes [13]. The raster occupancy grid is a tessellated grid used to accumulate real-time sensor data. A probabilistic model of the data from the sensor is generated and is used to update occupancy probabilities within the raster grid. The grid cell values for the occupancy grid represent the probability that an object exists or does not exist in the area covered by the cell. Updates are made to the patches of the grid that represent the field of view of the sensor. Even though this idea was pioneered by Elfes in 1989, it is still the most common implementation for real-time sensor data accumulation. It is especially useful for supporting the task of obstacle avoidance.

Over the years there have been several extensions on the pioneering work done by Elfes. One extension to Elfes's approach was introduced by Borenstein [6]. Rather than updating a large patch of the occupancy grid within the field of view of the sensor, this method updates a single cell along the major axis of the sensor. Borenstein shows that as the unmanned system traverses an area, this method is cheaper computationally and achieves similar results because the method focuses on using data along the sensor's axis that provides the best data. Novick [22] extended the concept of the raster occupancy grid update method. His approach was to apply a nonhomogenous Markov chain based method to update grid cells. Using this approach, Novick shows that this method is a significant advance in sensor fusion for outdoor vehicles. Both Borenstein and Novick's methods use raster grids to represent their data.

2.2.2 Real Time Terrain Mapping

An extension of the occupancy grid methods is the real-time terrain mapping method. This method attempts to generate a model of the Earth's surface in a tessellated data structure. This two and a half dimensional representation assigns a height to each

grid cell as opposed to an occupancy probability. Crosetto and Crippa [10] presented a method for fusing stereo and radar data to form real-time elevation maps.

2.2.3 Raster Traversability Grid

The traversability grid concept is an extension of the both raster occupancy grid and terrain mapping methods. In this implementation, the value in a grid cell represents the degree to which the area covered by the grid cell is considered drivable by the vehicle. Unlike the previous two methods, occupancy grids and terrain mapping, the traversability method is dependent on vehicle parameters. This is because the concept of traversability is inherently platform dependent. For example, an area occupied by a small rock may be deemed untraversable by a small unmanned system. However, a larger unmanned system confronting the same rock may consider the region less than desirable, but still traversable. Vehicle parameters that are often used traversability determination include the maximum allowable rotation angles of the platform about its three axes. Using a model of the terrain in which an unmanned system operates, it is possible to calculate the pose of the unmanned system along a path given the vehicle's physical parameters.

2.3 A Priori World Modeling Methods

An *a priori* world model data store is one that contains data that were accumulated prior to use by an unmanned system. For example, if an unmanned system maps, this map could be stored for future use by the unmanned system or transferred to another unmanned system to allow it to make mission decisions. This is an example of the use of raster data *a priori*. This is not the typical use of *a priori* within unmanned systems because of possible errors in the map making process. Instead, vector methods are used more frequently for initial data. An example of the use of this modeling method is

presented by Pasha [23]. The model of the world used is based on a polygonal representation as shown in Figure 2-2. In this work, Pasha models an environment in which an unmanned system must operate. The locations of static obstacles are known and can be used during the path planning process. Compounding the problem however, is the presence of numerous radiation sources. Given the obstacles and location and strength of radiation sources, a path plan is computed that most efficiently gets the unmanned system to its desired destination while minimizing its exposure to radiation.

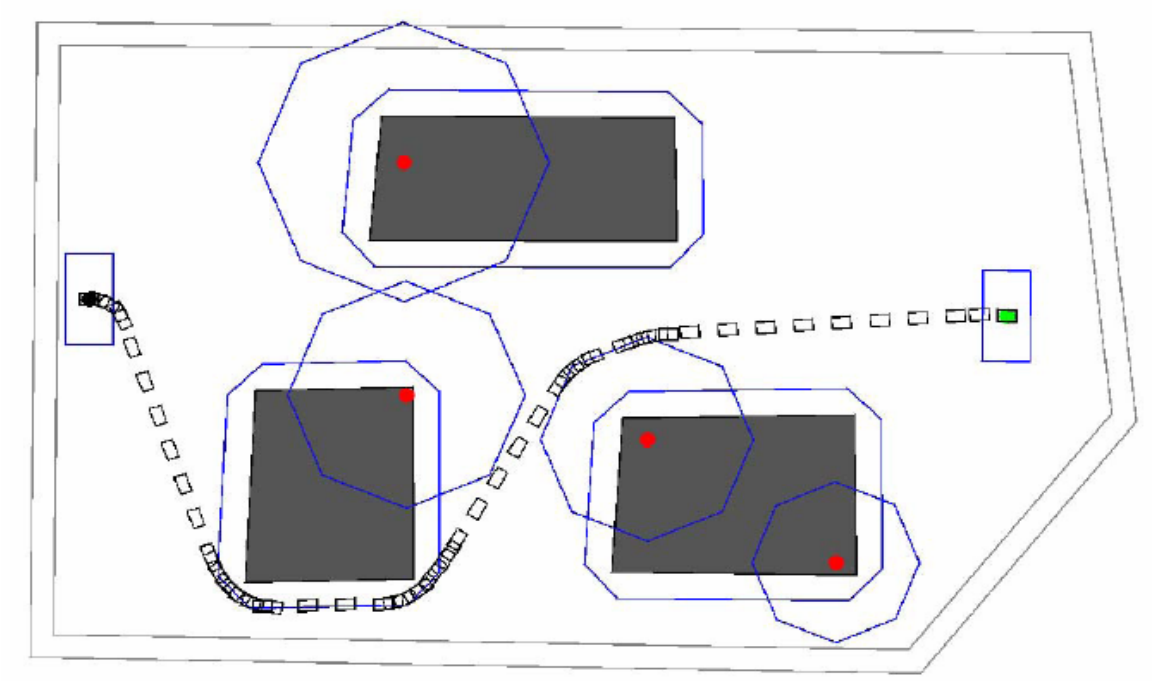

Figure 2-2. Path planning in a bounded radiation environment (Source: A. Pasha, "Path Planning for Nonholonomic Vehicles and Its Application to Radiation Environments," Master of Science Thesis. Department of Mechanical Engineering: University of Florida, 2003, p. 59, Figure 6-9)

2.4 Geographic Modeling Methods

The areas in which unmanned systems operate are typically assumed to be simple planar surfaces. As unmanned systems begin to be introduced into real world outdoor applications, this assumption can not hold.

2.4.1 Global Coordinate Systems

When moving from the laboratory to real world, outdoor applications that cover large distances, the methods presented in Section 2.3 must be modified. Those methods assumed that the unmanned system was operating in a perfectly planar environment; where, in the case of raster data, the cells were square and the coordinate system Cartesian. The Earth is not flat and, therefore, when unmanned systems operate over large distances, they must take the Earth's true shape into consideration.

There are three commonly used models of the Earth's shape. They are actual shape of the Earth's surface, the ellipsoid, and the geoid [7]. Because of the large variations in the Earth's surface, it is difficult to develop a true mathematical model for it. Therefore, the other two methods of modeling, the ellipsoid and the geoid, are typically used.

The ellipsoid is a mathematical model of the shape of the Earth. The ellipsoid (Figure 2-3) is defined by its semi-major and semi-minor axes. Over the years different Ellipsoidal models of the Earth have been established based on the best known shape of the Earth. Currently the most commonly used model is the World Geodetic System as defined in 1984 (WGS84). This model defines the semi-major axis r_1 as 6,378,137.0 meters and the semi-minor axis r_2 as 6,356,752.3 meters [7].

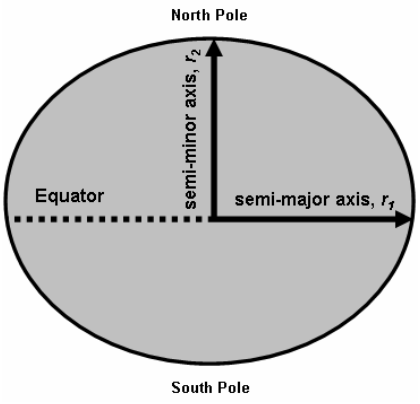

Figure 2-3. Ellipsoidal model of Earth

Once the Earth ellipsoidal model is established, a geographic coordinate system must also be established. Because of the spherical shape of the Earth, a spherical coordinate system is used to define points on the ellipsoid. A point on the ellipsoidal surface is described in spherical coordinates by a latitude value in degrees, a longitude value in degrees, and a height or elevation value in feet or meters. As shown in Figure 2-4, latitude values increase going north and range from -90° at the South Pole, to 0° at the Equator, to 90° at the North Pole. Longitude values start at 0° at the Prime Meridian and range between plus and minus 180°. The values go negative going west and positive going east.

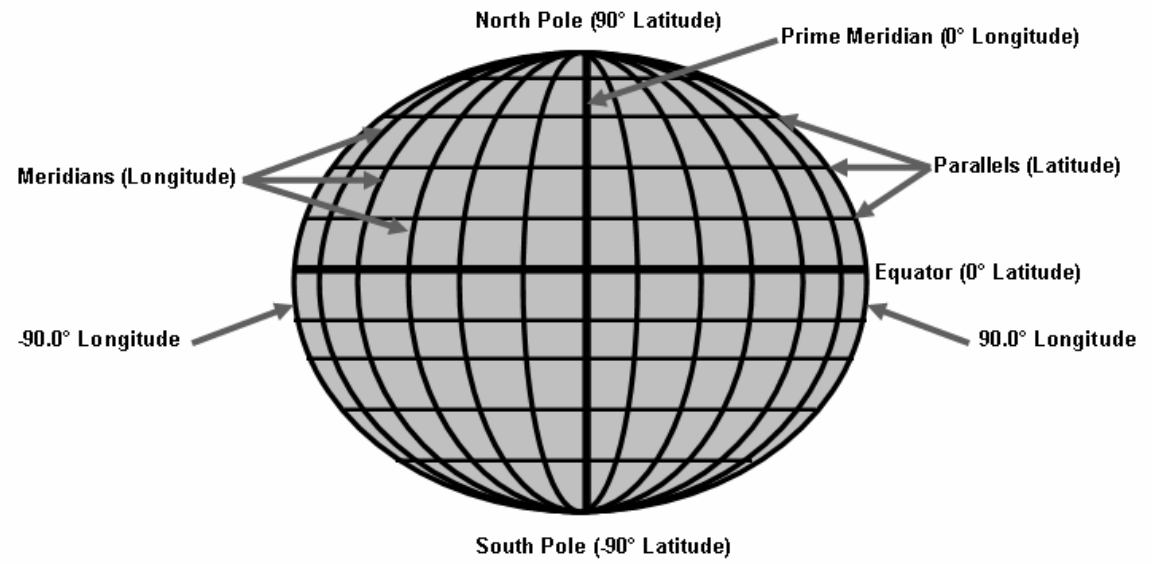

Figure 2-4. Earth-centered global coordinate system

Bolstad [7] describes the geoid as a three-dimensional surface that has a constant pull of gravity at each point. This equipotential surface is important for establishment of a vertical datum. In fact, this surface typically defines what is referred to as mean sea level [36]. If the Earth was covered by only water and no land, gravity would pull the water such that the geoid and the sea level would be the same [36]. This is how mean sea level is defined for land areas that are not near the sea. As with the ellipsoid model,

locations are referenced by latitude, longitude, and elevation. The difference is that the ellipsoid model uses the surface of the ellipsoid to establish elevation whereas this method measures the elevation of the geoid with respect an ellipsoid.

2.4.2 Projected Coordinate Systems

While true global coordinates are expressed as points of latitude, longitude, and elevation, it is more intuitive to model the world in Cartesian coordinates. This is particularly true when extending the methods in Section 2.3 to outdoor applications. In order to use a Cartesian coordinate system, methods have been established to mathematically project global, spherical coordinates onto a rectangular grid.

Because it is not possible to exactly represent this three-dimensional surface in two-dimensions, there are different mathematical projections that preserve different features of the three-dimensional surface. Typical features that are preserved are local shape, area, distance, and true direction. Conformal projections preserve local shape, equal area projections preserve area, equidistant projections preserve distance to some points, and true-direction projections preserve true-course between certain points [14].

Projections are not only classified by the types of features they preserve, they are also classified by the type of method used to create them. The main classifications are: cylindrical, conic, and planar or azimuthal [14].

Cylindrical projections convert from the Earth's three-dimensional spherical coordinate system to a cylindrical coordinate system. After the projection, the cylindrical representation is sliced so that it forms a two-dimensional rectangular representation of the Earth's surface. Conic projections convert from the Earth's three-dimensional spherical coordinate system to a conic coordinate system. After the projection, the conic representation is sliced so that it forms a two-dimensional representation of the Earth's

surface. Planar or azimuthal projections convert from the Earth's three-dimensional spherical coordinate system directly to a planar coordinate system. There are numerous types and variation of each type of projection. Bolstad [7] and ESRI [14] provide qn in-depth discussion of these projections and many of their variations.

2.4.3 Universal Transverse Mercator projection

The Universal Transverse Mercator (UTM) projection is a modification of the cylindrical Mercator projection. This projection is a conformal projection which preserves local shape of objects [14]. It creates minimal distortion of areas, local angles, and distance [14]. Unlike the cylindrical projection shown in Figure 2-5, the UTM projection divides the cylinder into 60 vertical zones. Each UTM zone is exactly 6 degrees of longitude wide and is further divided into north and south parts [7]. The UTM zones each have their own coordinate system which is completely different than the coordinate system of other zones. Because of this, it is difficult to use the UTM projection when traveling between UTM zones. This is rarely a problem with unmanned ground systems because the six degree UTM zones are much larger than what would be reasonably expected for a system of this type to traverse. It may be an issue with unmanned aerial vehicles, but this is something that can be taken into account by the system developers.

What is most attractive about the UTM projection is that it is a projection that is defined globally. In each zone, it is able to maintain shape, area, direction, as well as distance. These are all features that are important for unmanned vehicles during navigation and world modeling tasks. The reader is referred to [7] for a more in-depth discussion of the Universal Transverse Mercator projection, its applications, and limitations.

2.5 Georeferenced World Model Data

The modeling methods presented in Sections 2.3 and 2.4 are dependent on a planar assumption for the environment that the unmanned system operates in. In these applications, the coordinate systems are Cartesian with the origin being based on an arbitrarily chosen local coordinate system. As discussed in Section 2.4, these data can be stored and used as *a priori* data from other unmanned systems. What is more common, however, is to use data from third party sources. The most important

2.5.1 Raster Data Stores

Raster Data Stores are those that provide tessellated grid based geospatial data. Examples of the raster data stores include Digital Elevation Model (DEM), Digital Terrain Elevation Data (DTED), Digital Raster Graphics (DRG), Digital Orthophoto Quadranges (DOQs). This list is by no means exhaustive. There are many more types of raster data stores. Each type of data store provides different types of data at different resolutions possibly using different projections.

Figure 2-5 shows the high-level format of Digital Elevation Model (DEM) data. DEM data use the UTM projection to create a Cartesian coordinate system. These DEM data represent a 2.5D surface. The resolution of DEM data is 30 meters.

2.5.2 Vector Data Stores

Vector Data Stores are those that provide geospatial data that are referenced by points, lines, or vertices of polygons. Types of vector data stores include Digital Line Graphs (DLG), State Soil Geographic (STASGO), and Topologically Integrated Geographic Encoding and Referencing (TIGER). This list is by no means exhaustive. There are many more types of vector data stores available from third parties. Since there is no globally accepted geospatial data standard, each store may use a different format..

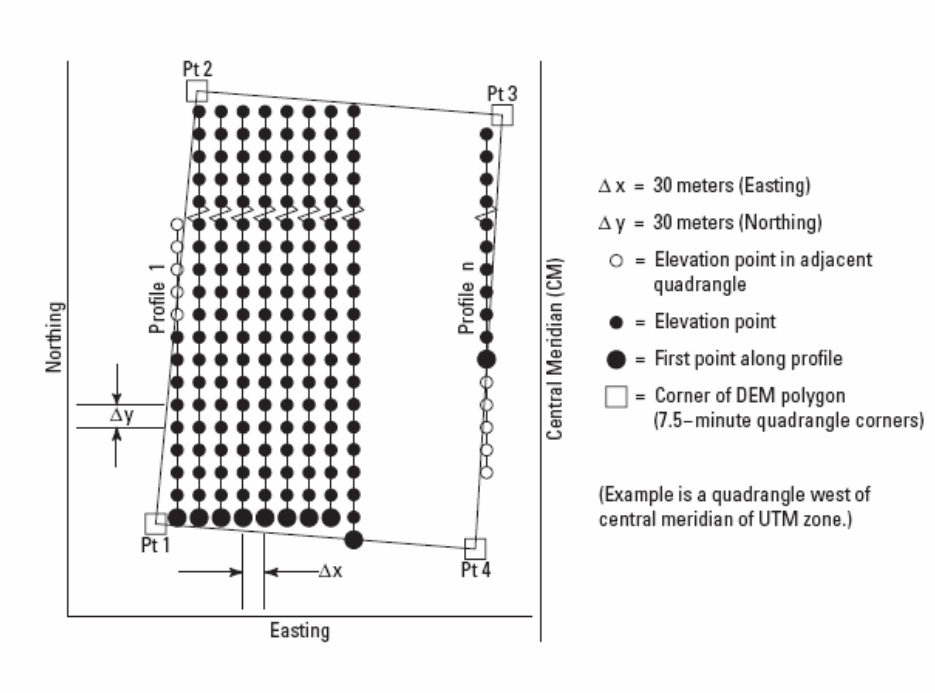

Figure 2-5. Example format of digital elevation model

Figure 2-6 shows an example of Digital Line Graph (DLG) data being extracted from a Digital Orthophoto Quadrangle. The benefit of this extraction is that the resulting DLG vector data size is smaller than the DOQs data size.

2.6 Distributed World Modeling Methods

There are very few major efforts attempting to tackle the difficult task of distributed world modeling. Two of the current efforts are the Spatial Data Transfer Standard (SDTS) and the Geography Markup Language (GML).

2.6.1 Spatial Data Transfer Standard (SDTS)

SDTS is an open standard being developed by the Unites States government for use in geographic information systems. One of the reasons for developing this standard is that there are various types of geospatial data available based of different Earth models and projections with each having different errors associated with them. SDTS seeks to provide a method that allows a complete data transfer with all necessary information

associated with those data needed to incorporate them into other data systems. SDTS specifies the entire process of storing and sharing geospatial data. This ranges from the methods for modeling raster and vector geospatial data down to the way that data are stored in digital files. SDTS is also a very broad standard that is able to support different models of the Earth, different map projections, and different method of modeling the data.

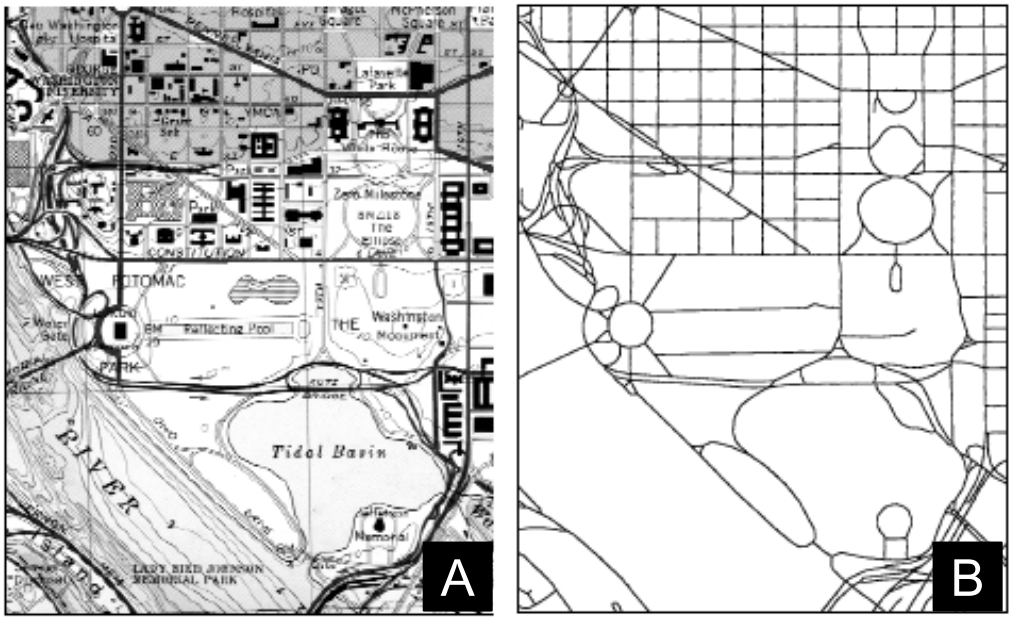

Figure 2-6. An example of USGS source data. A) USGS Orthoimage B) Extracted digital line graph

The SDTS is divided into six profiles that completely define the standard. The first three parts define the logical specification, spatial features, and data encoding, respectively. The other parts are called profiles. Each profile provides instructions for using the base SDTS rules, parts one through three, to different types of geospatial data [32].

Part four of the SDTS standard is the Topological Vector Profile (TVP). This profile allows transfer of geospatial vector data described by vector geometry and

topology. This profile allows data to be geometrically described using points, lines, polygons, as well as combinations of these. The Topological Vector Profile is useful for transferring digital line graph (DLG) data such as those presented in Figure 2-9 [3].

Part five of the SDTS standard is the Raster Profile and Extensions (RPE). This profile supports various types of raster formatted geospatial data. This includes Georeferenced orthoimages, grid formatted terrain data such as DTED and DEM, as well as any type of tessellated geospatial data. RPE does not support data of a higher dimension that two and a half (such as terrain data) [5].

The Last part of the current version of SDTS is the Point Profile. This profile provides support for high precision point data only. While the Topological Vector Profile does support point data, it does not at high enough precision for some applications. The Point Profile supports up to 64 bits of precision whereas the TVP only supports up to 32 bits of precision [4]. All six parts of the SDTS standard combine to form a powerful and comprehensive method for modeling and distributing geospatial data.

2.6.2 Geography Markup Language

The Geography Markup Language (GML) is a broad standard that supports raster and vector data in 2, 2.5, and 3 dimensions. It also supports more types of complex shapes and surfaces than are needed for unmanned system world modeling. It is able to support data based on different projections as well as different Earth models [11].

GML is an extension of the Extensible Markup Language (XML). XML, like the Hypertext Markup Language (HTML) commonly used for transfer of web pages, supports tags that specify the types of data included in the document. For XML the tags are defined by the document creator for the type of data included. HTML specified all of its tags *a priori*. Also unlike HTML, XML and subsequently GML, does not mix the

data content with the formatting of the content. For GML, the descriptors (or tags) are geospatial data related. While XML provides a very loose structure for the types of data described, GML places restrictions on XML by specifying the methods for geometrically modeling the data. If GML based system developers associate different attributes with the geospatial data types, they will at the very least be able to understand each other's data at a geometric level [17].

Both SDTS and GML are both adequate methods for modeling geospatial data and sharing those data, but they are not exactly appropriate for JAUS based unmanned systems. Of the two, GML is more appropriate since it is based on the powerful XML standard which is designed for real-time transfer. By defining additional XML tags, it is possible to make data store modifications in real-time rather than on a per XML document basis. The downside of GML is that it is all ASCII text based and requires extra characters to support its extensibility. Because some of the tags are many characters long, this translates to additional bandwidth being used for the support characters.

SDTS is not appropriate for use with unmanned systems where bandwidth utilization should be minimized. Because SDTS transfers are to be all self-contained with all necessary data included, this is not suitable for real-time data transfer. A real-time world modeling message set should support the ability to make individual changes to the data store in real-time rather than requiring changes to be transmitted via an updated version of all the data in the data store.

This work is interested in using the power of the JAUS infrastructure to support distributed world modeling. Since JAUS defines the structure of its messages *a priori*,

beyond its 16-byte header, JAUS does not require any other bytes to support its infrastructure. All of the data after the JAUS header are values for the field described in the JAUS message definition. Rather than incorporating a completely different, non-optimal standard into JAUS for world modeling, the world modeling standard builds on the framework developed by the JAUS Working Group.

CHAPTER 3
SMART SENSORS

While setting out to develop a standard for modeling the various types of geospatial data presented in the preceding chapter, a distributed set of world models was developed. These world models were tightly coupled to their associated sensors and therefore were initially considered to be smart sensors.

3.1 Smart Sensor Architecture

The smart sensor architecture was originally developed for the perception system in the Team CIMAR NaviGATOR which is represented Figure 3-1. The NaviGATOR was developed as an entry to the 2004 Defense Advanced Research Projects Agency (DARPA) Grand Challenge. Held in March of 2004, the DARPA Grand Challenge was a first of its kind unmanned ground vehicle (UGV) competition. The thrust of this challenge was to develop a UGV that could autonomously navigate and avoid obstacles over the approximately 140 miles from Barstow, California to Primm, Nevada - crossing the Mojave Desert.

Team CIMAR consisted of graduate students and engineering staff from the University of Florida's Center for Intelligent Machines and Robotics (CIMAR) and Logan, Utah based Autonomous Solutions, Inc.

Recognizing the power and flexibility afforded by the use of JAUS, Team CIMAR used it throughout the NaviGATOR and therefore developed, at the time, one of the only completely autonomous systems based on the Joint Architecture for Unmanned Systems (JAUS). The exception to this was the Navigator's perception system where the

messages that defined the smart sensor messaging architecture were only loosely modeled after the JAUS standard and the JAUS World Modeling Subcommittee's forthcoming draft message set which is presented in Chapter 4 of this document.

The smart sensor architecture is a networked system of distributed, modular, heterogeneous sensor units that all use a common messaging and network interface to share data. Each smart sensor processes data specific to its associated sensor modality and determines region traversability using a suitable traversability metric as determined by the sensor system developer. These geospatial traversability data are shared within the perception system and provided to higher level planning components to allow them to make intelligent decisions such as obstacle avoidance.

Figure 3-1. Team CIMAR NaviGATOR DARPA Grand Challenge entry vehicle

The smart sensor units are considered "smart" because they not only process their sensor data, they also provide a logically redundant interface to other components within the system. The impetus behind the creation of this smart sensor architecture was to allow sensing system implementers to develop their sensing technologies independent of one another and then have them, with minimal effort, seamlessly integrate their work to form a robust perception system. The JAUS-like messaging infrastructure and logical redundancy of the smart sensors afforded this flexibility. Even though their implementations and sensor modalities are different, these sensor units are logically redundant in that their messaging interfaces are identical [19]. The idea was that each sensor implementer best knew how to process and register their own sensor data. Rather than relying on a probabilistic model of the sensor to homogenize the sensor data on one system, this implementation expects the sensor data to be homogenized before they are fused. Once their data were available, the smart sensors would publish the data to a central component, the smart sensor arbiter, whose responsibility would be to fuse the data from all of the smart sensors.

The output of the smart sensors is a measure of region traversability cost. This cost is based on a sensor-specific traversability metric being applied to the data from a physical obstacle detection sensor. Behaviors of this type, associating a cost to an attribute based on a metric, are called value judgment [18]. Smart sensor developers were permitted to use any sensor modality that presented data that could be processed to provide sufficient traversability value judgment. For this implementation, these included stereo vision, stationary laser measurement system, and monocular vision based smart sensors all developed by researchers at the University of Florida. A continuously rolling

laser measurement system based smart sensor was developed at Autonomous Solutions, Inc.

While there are duplicate sensor types, the implementation of the associated smart sensor makes the data from the sensors quite unique. For example, the stereo camera and monocular cameras use the same sensing modality however the difference is in the implementation of the smart sensors. The stereo camera data are processed so that they, through the use of image rectification and correlation, provide a sparse three-dimensional representation of the environment within the field of view of the cameras. Traversability is determined by considering the stereo data as real-time terrain data and applying value judgment. The implementation of the monocular camera based smart sensor utilized color and cluster affinity in RGB-space to classify image pixels that belonged to traversable surfaces.

Once the individually developed smart sensors were completed, a predefined messaging architecture was used to transmit the traversability data within the perception system. In order to support true interoperability, however, electrical and transport layer issues also had to be addressed. These issues will be address later in this chapter.

3.2 Smart Sensor Architecture Components

There are three major types of components that make up the perception system's smart sensor architecture. These are the smart sensor, smart sensor arbiter, and reactive planner components. Figure 3-2 shows the perception system components as well as the component interconnects.

3.2.1 Smart Sensor Component

The smart sensor is a modular perception system component that provides an interface between a physical sensor and the smart sensor network. It encapsulates a

physical sensor, the hardware necessary to process the sensor data, a method for determining region traversability from the processed sensor data, a standardized messaging interface, and a communications link.

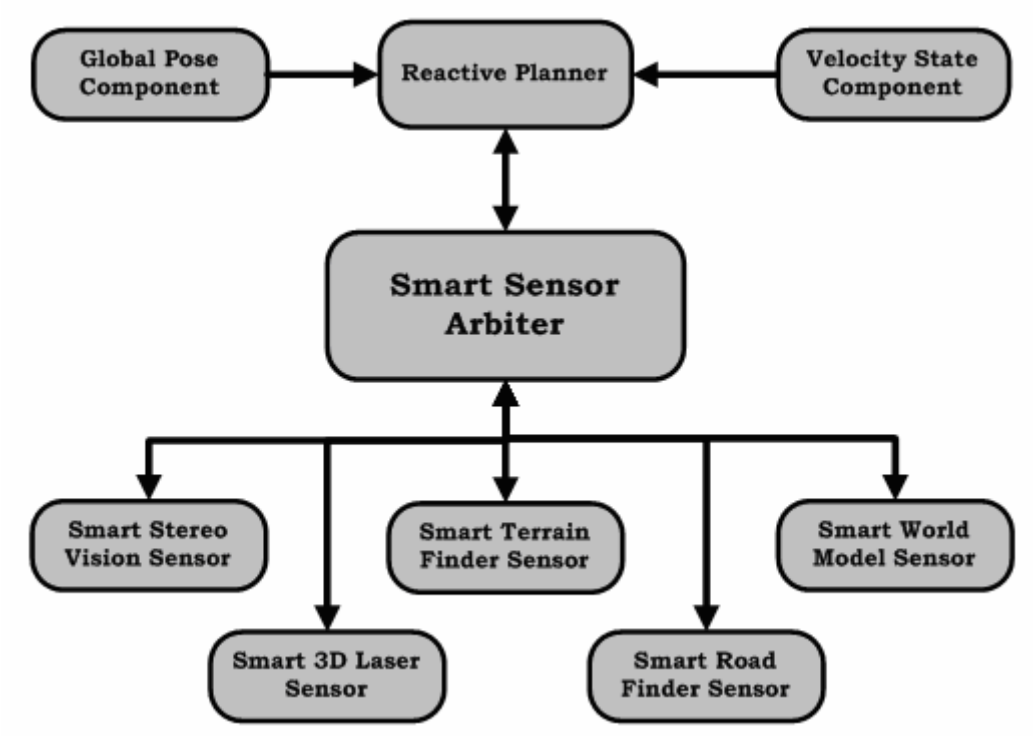

Figure 3-2. Organization of the smart sensor-based perception system

A smart sensor is modular in that it shares the same logical interface with all other smart sensors. With the exception of a single field in the message header, the source component identification number, the output format of each smart sensor is identical to that of all other smart sensors. This allows any smart sensor to seamlessly replace any other.

Internally, the smart sensor maintains a tessellated traversability grid of a size specified by the predefined range and resolution of the grid. As with the occupancy and traversability grids introduced in Chapter 2, this grid maintains a fixed orientation and

remains vehicle centered. In this implementation, the grid maintains a north-east orientation.

As the vehicle moves, the grid is translated in discrete steps to compensate for the vehicle's movement. The translation of the vehicle is determined from the previous and current positions of the vehicle as provided by a global positioning system (GPS) providing coordinates in the WGS84 coordinate system. The coordinates are projected from global to Cartesian coordinates using the Universal Transverse Mercator (UTM) projection. The difference in position, in meters East and North of the origin, is converted to a translation of grid rows and columns. To assure that the vehicle is always centered in the center cell of the traversability grid, the grid dimensions, rows and columns, are required to be odd.

The geospatial traversability data are registered by using the vehicle's orientation to project the sensor data into the two-dimensional traversability grid. As the vehicle translates and rotates, changes to the traversability grid are monitored. As the values of cells change, the updated values are transferred to other systems to provide grid synchronization.

3.2.2 Smart Sensor Arbiter Component

The smart sensor arbiter has the responsibility of fusing data from the smart sensors and, through the synergy of the different sensor modalities, providing a better model of the world to the reactive planner component.

In a complete smart sensor system, the arbiter component is the hub of all data traffic from the smart sensors. As it receives traversability updates from the smart sensors, it immediately fuses the updated data with that from previous sensor updates. Generally, the method used to fuse the traversability data from the sensors is not

specified and is left to the implementer. What is important is that the interface to the arbiter is consistent with the smart sensor message set and that the arbiter's grid resolution is the same as the smart sensors'. Maintaining a grid of equal size as the smart sensors is not required as it may be desirable to have a grid that extends well beyond the bounds of all of the smart sensor grids. This allows the arbiter to maintain a larger local memory of the area perceived by all the smart sensors. In a system with multiple subsystems, this functionality could be used for collaborative mapping of large areas.

The smart sensor arbiter also includes a virtual component - the Region Clutter sensor. This component provides a very fast indication of the saturation of non-traversable areas within the unmanned system's immediate vicinity. This feature gives the higher level planning components information that allows it to modify the vehicle's speed as it encounters cluttered areas. By modifying the system's travel speed, there may be adequate time to generate a plan to negotiate the non-traversable regions.

The smart sensor arbiter also shares the same logical interface as the smart sensors. This allows smart sensor based perception systems to use a single smart sensor without the smart sensor arbiter or multiple smart sensors with the arbiter. This flexibility is an asset especially in the development and debugging processes.

3.2.3 Reactive Planner Component

Within the smart sensor based perception system, higher level obstacle avoidance and vehicle travel speed control is the responsibility of the reactive planner component. The reactive planner component, using the JAUS communications network, receives the position, orientation, and orientation rates of the platform from the JAUS Global Pose and Velocity State components. The reactive planner component then uses the smart sensor architecture messaging interface and the smart sensor network to transmit the

position and orientation updates to the smart sensors. The same network is used to receive the smart sensors' traversability data.

As it is receiving traversability grid updates from either the arbiter or smart sensors, the reactive planner continuously searches for the optimal, lowest cost path through the accumulated traversability data. The output of the reactive planner is a modified path plan for the unmanned system to execute.

3.3 Smart Sensor Messaging Architecture

In order to support the development of the components of the perception system, a standardized messaging interface was defined. Its use was mandated for all components participating in the smart sensor based perception system. This messaging interface was to a large degree based on the methodologies and messages established by JAUS.

3.3.1 Smart Sensor Architecture Message Header

To support message identification, routing, and transfer, a modified version of the standard 16 byte JAUS message header was created. The JAUS header supports more functionality than needed by the smart sensor architecture. Therefore the majority of the bytes within the header would not be needed. Because of the volume of data transferred within the smart sensor system, any savings of would be beneficial. Therefore the JAUS header was reduced so that all unnecessary header fields were removed. Table 3-1 shows the format of the official JAUS message header. Since the smart sensors communicated on their own network, this optimization had no effect on the JAUS based NaviGator network.

The smart sensor development team made several assumptions about the data transfer process in order to justify the reduction in header size. They are as follows:

- Smart sensors are all contained within the same subsystem

- Smart sensors are single component nodes
- Smart sensors have distinct component identification numbers
- Smart sensors have only one instance
- Smart sensor message types are unidirectional
- Smart sensors use the same version of the interface control document
- Smart sensors do not use service connections
- Smart sensors do not require message acknowledgment
- Smart sensors transmit messages of the same priority

Table 3-1. Standard JAUS sixteen byte message header

Field #	Field Description	Size (Bytes)
1	Message Properties	2
2	Command Code	2
3	Destination Instance ID	1
4	Destination Component ID	1
5	Destination Node ID	1
6	Destination Subsystem ID	1
7	Source Instance ID	1
8	Source Component ID	1
9	Source Node ID	1
10	Source Subsystem ID	1
11	Data Control (bytes)	2
12	Sequence Number	2
	Total Bytes	**16**

The smart sensors are all contained within the same subsystem and the subsystem does not communicate spatial data to any other subsystem, therefore the fields for the destination and source subsystem identification numbers (fields 6 and 10, respectively) would be equal and would always remain static. Removing these fields reduces the required header size by 2 bytes.

The message header fields 5 and 9, node identifiers, may be removed due to the assumption that the smart sensors are single component nodes and that each smart sensor has its own component identification number. Because of these assumptions each component identification number must be coupled to one and only one node identification number. Therefore specifying both the component and node identifiers

would be redundant. The removal of the destination and source node identifier fields saves an additional two bytes. Therefore, the smart sensor components are addressed only by component identification numbers.

Within the smart sensor system there is no redundancy of smart sensor implementations, therefore the header's instance identification fields, 3 and 7, would never be used. Removal of these fields results in a savings of 2 bytes.

It is important to note that this redundancy assumption is specific to the smart sensor implementation where each individual smart sensor has its own component identification number. In a true JAUS system, this would not necessarily be the case. They would be treated as redundant components since each smart sensor is just another instance of the same.

Messages traveling down stream from the reactive planner component to the arbiter and smart sensors are position and orientation update messages. Messages traveling upstream from the smart sensors to the arbiter and reactive planner are cell update messages. The exception to this rule is the arbiter clutter sensor component, which will be discussed in greater detail later in this chapter. Because message types are unidirectional and there is *a priori* knowledge of the system configuration, this assumption removes the need for the source component identification number and the message command code; a combined savings of three bytes.

The message properties field in the JAUS header provides important information to the receiving JAUS component. This includes the version of JAUS Reference Architecture message set used to create the attached message as well as message type, acknowledgement, and priority information. The last four assumptions have a direct

impact on this message field by making it useless. The assumption that the smart sensors are not using service connections also removes the need for the Sequence Number field of the JAUS header. Combined, these four assumptions result in a savings of four bytes.

The cumulative savings produced by the nine assumptions presented above is 13 bytes. In a system with a relatively large amount of bandwidth or less frequent raster geospatial data transfers, the thirteen-byte savings may not seem significant. Since the message header must be attached to each message, however, when there is a large volume of data, as can be expected within the smart sensor architecture, the aggregate savings can be substantial. The final three-byte header is show in Table 3-2.

Table 3-2. Smart sensor architecture message header

Field #	Field Description	Size (Bytes)
1	Source Component ID	1
2	Data Control (bytes)	2
	Total Bytes	**3**

One of the strengths of JAUS is that the messages are develop completely separate from, and are not at all dependent on, the message header. Because of this, the smart sensor messages that will be introduced may be transmitted using any message header that can be used to properly route the messages to their intended destination. Again, the header size reduction presented in this section was made primarily for the purpose of saving bandwidth and computing resources.

3.3.2 Smart Sensor Architecture Message Set

The NaviGATOR's perception system, consisting of the components of the smart sensor architecture, provides a unique method for transferring and synchronizing raster formatted geospatial traversability data. The structure of the messages within this

architecture is based on the JAUS Reference Architecture message set. They were designed to support the interoperability, extensibility, and logical redundancy required of the smart sensor architecture.

Figure 3-3 shows the minimum number of component types needed for a complete smart sensor system. It is considered complete because all of the core components are present; the reactive planner, arbiter, and smart sensor. It is minimal because only one smart sensor is present. In fact, this system is not particularly useful because the arbiter and the smart sensor's internal traversability grid representations would be exactly the same. Therefore, the arbiter should be used when there is more than one smart sensor present.

Using the logical redundancy provided by the smart sensor architecture, a more efficient implementation of a single sensor based system is shown in Figure 3-4. This showcases the power of the logically redundant interface as a smart sensor may replace the arbiter or any other smart sensor.

Three types of messages are used in the smart sensor based perception system:

- Report vehicle state
- Report traversability grid updates
- Report region clutter index

The Report Vehicle State message communicates vehicle position, orientation, and orientation rate information. Updates to the smart sensor or smart sensor arbiter traversability grid are transmitted through the use of the Report Traversability Grid Updates Message. The Report Region Clutter Index message transmits an indication of the saturation of non-traversable areas in the immediate vicinity of the vehicle. This is primarily used to limit the speed of the vehicle while in cluttered areas.

3.3.2.1 Report vehicle state message

The Report Vehicle State message, consisting of vehicle position, orientation, and orientation rate updates, is a combination of the JAUS Code 4402h: Report Global Pose and Code 4404h: Report Velocity State messages. The position of the platform is given in latitude, and longitude in accordance with the WGS84 standard. The orientation and orientation rates are with respect to the vehicles coordinate system as defined by JAUS (Figure 3-5).

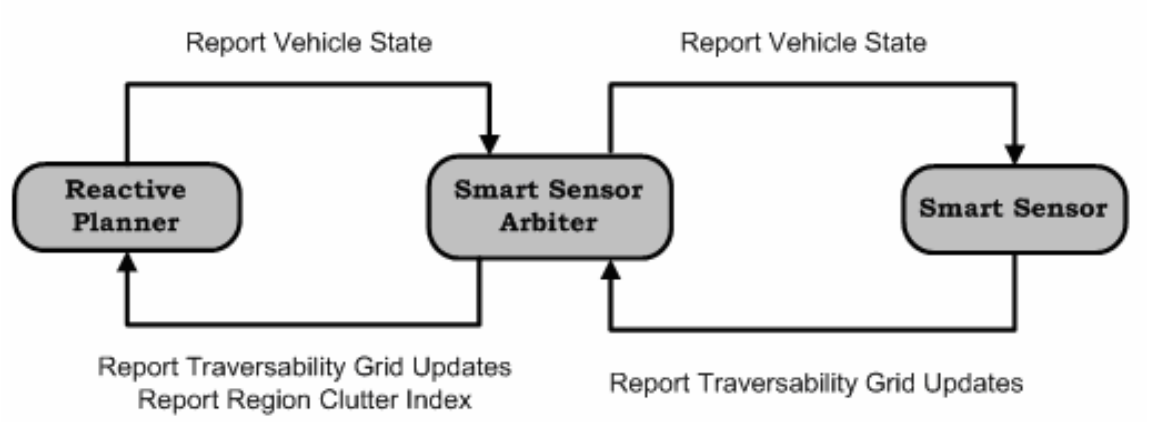

Figure 3-3. Minimally complete smart sensor-based perception system consisting of one instance of the core component.

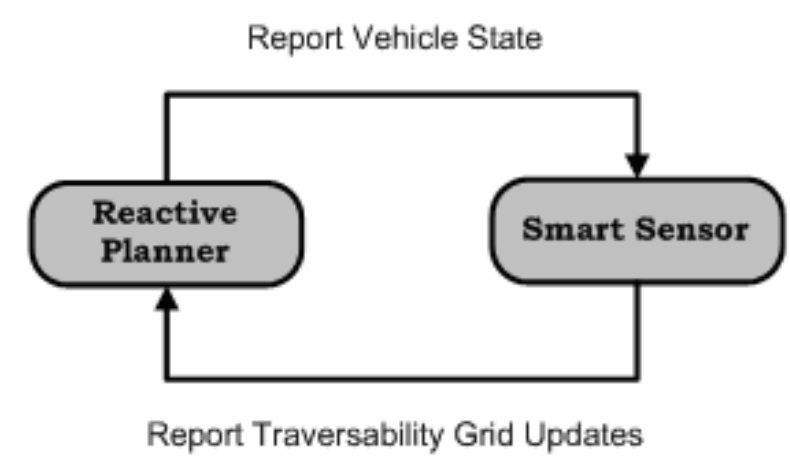

Figure 3-4. Single sensor implementation of smart sensor-based perception system consisting of a single smart sensor synchronizing data with the reactive planner.

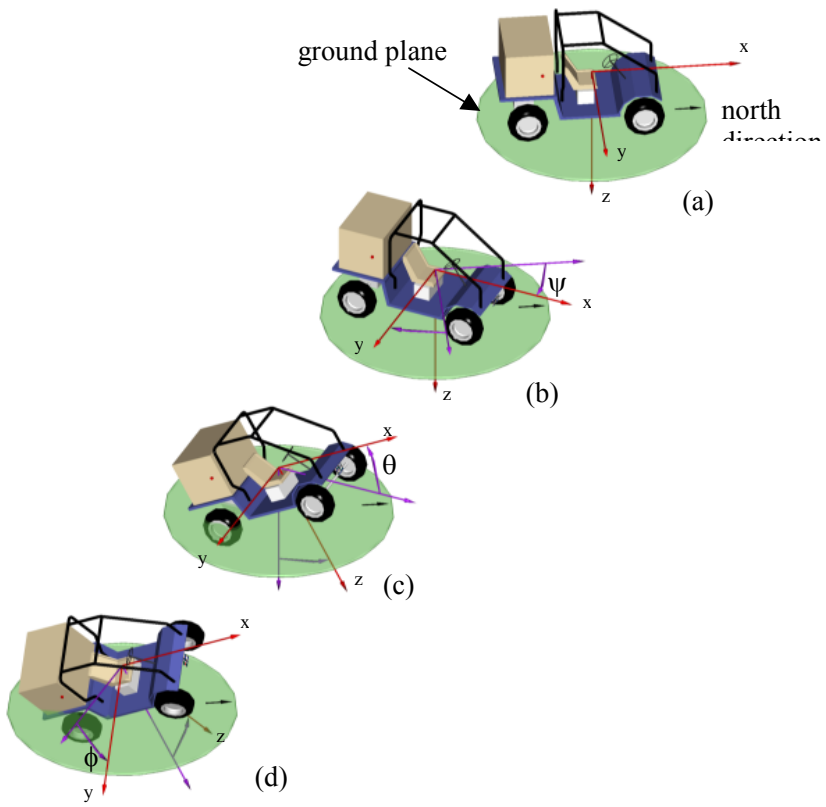

Figure 3-5. Unmanned system coordinate system defined by JAUS

To allow message types of variable size where only the desired data are transmitted, JAUS provides a presence vector. This presence vector is an *n*-byte bit field with flags indicating which optional fields are present in a JAUS message. For the smart sensor implementation, only the latitude, longitude, roll, pitch, yaw, roll rate, pitch rate, and yaw rate fields are needed from the JAUS Report Global Pose and Report Velocity State messages. The remaining fields are not included in the transmitted message. Since these unneeded fields have been removed from this message contraction and there is *a priori* knowledge of the structure of this message, the presence vector was also removed.

The benefit of this approach is that the Report Global Pose and Report Velocity State messages do not have to be sent separately with the 16 byte JAUS header attached to each. An additional benefit to this approach is that the position, orientation, and

orientation rate fields are synchronized; i.e. the message includes an instantaneous reading of both the position and orientation data.

This Report Vehicle State message, Table 3-3, is 20 bytes in length, 23 bytes including the message header. Fields 1 and 2 contain the latitude and longitude, respectively, as scaled integers. Fields 3 through 5 contained the vehicle orientation and fields 6 through 8 contain the orientation rates.

Table 3-3. Smart sensor architecture's report vehicle state message

Field #	Name	Type	Units	Interpretation
1	Latitude (WGS 84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2	Longitude (WGS 84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
3	ϕ (Roll)	Short Integer	Radians	Scaled Integer Lower Limit = $-\pi$ Upper Limit = π
4	θ (Pitch)			
5	ψ (Yaw)			
6	Roll Rate	Short Integer	Radians per Second	Scaled Integer Lower Limit = -32.767 Upper Limit = 32.767
7	Pitch Rate			
8	Yaw Rate			

3.3.2.2 Report traversability grid update message

The Report Traversability Grid Update message provides a synchronization mechanism between the multiple distributed traversability grids. This functionality is event driven and based on updates to the smart sensors' traversability grids. When a change is made to a traversability grid, the change is transmitted to the destination component to synchronize the two grids. By making the process event driven, bandwidth utilization is reduced over transmitting the entire traversability grid, especially when there are only a small number of changes to the traversability grid.

The Report Traversability Grid Updates message is shown in Table 3-4. The first two fields of the message are a latitude and longitude position stamp. This position

stamp represents the point with which the cell update values are referenced; the current location of the vehicle at the time the sensor data was processed. Following the position stamp is a series of cell update three-tuples. Each three-tuple represents the traversability grid update as an updated cell row, column, and traversability value.

The traversability grid cell update values use the entire numeric range of a byte, 0 to 255, to represent the traversability of the region represented by the cell. A value of 127 corresponds to an unknown traversability. As the value approaches zero, exclusive of zero, the cell classification become more and more non-traversable. Conversely as the value approaches 255, the classification is more traversable. The grid cell value zero is reserved exclusively for the world model corridor data which is used to constrain the search for the lowest cost path through the traversability grid.

This message allows all changes to be transmitted in one message, provided that the total message data size is less than the 65527 bytes that the smart sensor architecture header permits. This limit is determined by the UDP/IP transport layer's limit on the maximum number of payload data bytes that may be transmitted in a single transaction [24]. Should the Report Traversability Grid Update message exceed 65527 bytes, it should be broken into separate messages. These separate messages should have the same latitude and longitude position stamp values as the first cell update message. This latitude and longitude position stamp is very important as it defines the origin of the cell changes.

The total number of three-tuple cell updates being transmitted may be inferred from the header data bytes field by subtracting the eight bytes required by the position stamp and dividing the remainder by three.

3.3.2.3 Report region clutter index message

Within the perception system, there is a need to allow higher level components, particularly the Reactive Planner, to know the degree of saturation of non-traversable areas local to the vehicle. The purpose of this is to allow the UGV to reduce its speed to allow it to successfully negotiate the traversable regions. To support this, another pseudo-component was developed. This component, the Region Clutter Sensor, is embedded in the arbiter. It simply provides a fast assessment of the percentage of cells in a specified area that are classified as non-traversable. This message is sent to the reactive planner, which has the responsibility for determining how to react to this notification. Ideally, the reactive planner converts the clutter percentage to a recommended vehicle speed and transmits this to the Global Path Segment Driver component using the JAUS Code 040Ah: Set Travel Speed message.

Table 3-4. Smart sensor architecture's report traversability grid updates message.

Field #	Name	Type	Units	Interpretation
1	Latitude (WGS 84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2	Longitude (WGS 84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
3	Cell Update 1 Row	Byte	N/A	
4	Cell Update 1 Column	Byte	N/A	
5	Cell Update 1 Value	Byte	N/A	0 – Reserved for World Model 1 … 126 – Non Traversable 127 – Unknown (Initial value for cells) 128 … 255 – Traversable 1=completely non-traversable 255=completely traversable

Table 3-4. Continued

Field #	Name	Type	Units	Interpretation
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
3n	Cell Update *n* Row	Byte	N/A	
3n + 1	Cell Update *n* Column	Byte	N/A	
3n + 2	Cell Update *n* Value	Byte	N/A	Same as field 5

The area covered by the Region Clutter Sensor is not specified in the Smart Sensor Architecture Interface Control Document (ICD). This is a system specific parameter and is therefore left to the system implementer. A system traveling at high speed may need to monitor a large area whereas a slower or smaller system may need to monitor a smaller area.

Table 3-5. Smart sensor architecture's report region clutter index message.

Field #	Name	Type	Units	Interpretation
1	Clutter Index	Byte	Percent	Scaled Byte Lower Limit = 0 Upper Limit = 100 Percentage clutter in specified area

3.3.3 Smart Sensor Architecture Network Communications

The smart sensor data are transferred within the perception system via the user datagram protocol (UDP) running on top of the Internet protocol (IP). This combination of user datagram protocol and the Internet protocol will be referred to as UDP/IP. UDP/IP provides a connectionless, unreliable communications link between systems. The term unreliable is in some respects a misnomer because UDP/IP can provide a quality connection. Unlike the Transmission Control Protocol, UDP does not have any

checks to assure receipt of data. It relies on the host application to do the checking. For example, the JAUS header provides a message acknowledgement flag that requests that the receiving component notify the sending component of receipt of a message. If the sending component does not respond in a set period of time, as per the JAUS RA, the sending component retries up to three times and then terminates transmission. If a JAUS implementation used UDP/IP, then this functionality would help assure reliable communications.

The smart sensor system is set up *a priori* under the assumption that the minimum number of system components are present and that they are online and in the ready state. It was developed such that each component commences transfer of the supported messages to the appropriate component directly after initialization. This may be considered transmission of unsolicited responses to repeated data queries (*sans* the queries) or as an unsolicited JAUS service connection. The UDP/IP transport layer supports this functionality. UDP/IP is connectionless and therefore does not require that the destination component be present or a link established in order for data to be sent within the system. The popular alternative to UDP/IP, TCP/IP, generally requires that a socket connection be established between two or more systems before data can be sent.

To route data to the smart sensors, internet protocol (IP) addresses had to be defined. To allow the IP addresses to be determined dynamically based on the destination of the message, an IP addressing convention was established. Since each smart sensor has a unique component ID, the component ID was used as the last octet of the IP address. The first three octets of the IP address were established *a priori*. For example: 192.168.1.***component_id*** is an example configuration where the first three

octets are the defined and the component ID is used as the last octet. Table 3-6 presents a list of smart sensor components and their associated component identification numbers. A standard UDP/IP port was also designated.

The network interface between all components within the perception system was wired Ethernet capable of providing data transfer at rates of up to 100 Megabits per second.

Table 3-6. Smart sensor components and their component identification numbers

Smart Sensor	Component ID
Reactive Planner	10
Smart Sensor Arbiter	11
Smart 3D Laser Sensor	21
Smart Stereo Vision Sensor	22
Smart Terrain Finder Sensor	23
Smart Road Finder Sensor	24
Smart World Model Sensor	25
Region Clutter Sensor	127

3.4 Smart Sensor Implementation

While the smart sensor architecture was originally developed for the Team CIMAR NaviGATOR, final testing and verification took place on the Center for Intelligent Machines and Robotics' Navigation Test Vehicle 2 (NTV2) shown in Figure 3-6. The implementation of the smart sensor units at CIMAR exploited the commonality between implementations.

3.4.1 Abstraction of Smart Sensor Core Functionality

Because of the considerable amount of implementation overlap, the CIMAR smart sensor system was designed so that all developers build their smart sensors on top of a common base implementation that contained the core smart sensor functionality. This approach saved a considerable amount of time because testing and debugging of the main

base sensor implementation occurred independent of development of the sensors. The interface to this system was made into a clean application programmer's interface (API).

This API handles communications, grid synchronization, and all other low level smart sensor tasks. The system designer has the responsibility of processing the sensor specific data to determine traversability, placing that data in a grid of the proper range and resolution, and using the smart sensor API to publish the new data to concerned components within the system. Figure 3-7 shows the high-level conceptual separation between the two functions. The power of this approach is that it allows new implementations of sensors to come online in very short order.

3.4.2 Base Smart Sensor

The base smart sensor encapsulates all low-level functionality common to all smart sensors. This functionality includes:

- Allocating memory for a local traversability grid
- Receiving position and orientation updates via UDP/IP
- Transforming data from sensor coordinates to grid coordinates
- Shifting the traversability grid to keep it vehicle centered
- Monitoring traversability grid updates
- Synchronizing traversability grid updates via UDP/IP

This functionality leaves to the operator the task of solely proving an instantaneous local traversability grid from their sensor data. They initialize their smart sensors using the API. The base smart sensor has two thread that run concurrently with the sensor interface specific thread. Figure 3-8 shows a call graph for all functions within the base smart sensor.

The traversability data are registered in the grid by utilizing the platform orientation data. Upon startup, the base smart sensor spawns a thread (Figure 3-9) to handle

asynchronous position updates from either the smart sensor arbiter or directly from the reactive planner component

Figure 3-6. Center for Intelligent Machines and Robotics Navigation Test Vehicle 2.

The rotations Ψ, θ, and ϕ, as shown in Figure 3-5, as well as the sensor's offset from the vehicle's coordinate system are used in the homogenous transformation of the data from the sensor's coordinate system to the grid coordinate system. Equation 3-1 shows the compound transformations necessary for this. The x_{offset}, y_{offset}, and z_{offset} values all represent the offset of the sensor coordinate system from the vehicle's coordinate system. It is assumed that sensor is aligned such that there is no rotational difference between the two coordinate systems, only translation. The x_{sensor}, y_{sensor}, and z_{sensor} values represent the coordinates of a point as read from the sensor in the sensor's coordinate

system; $x_{vehicle}$, $y_{vehicle}$, and $z_{vehicle}$ are the coordinates of the point after transformation to the vehicle coordinate system.

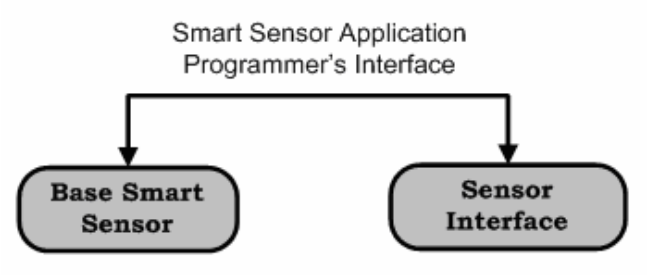

Figure 3-7. Smart sensor implementation - abstraction of low-level smart sensor functionality

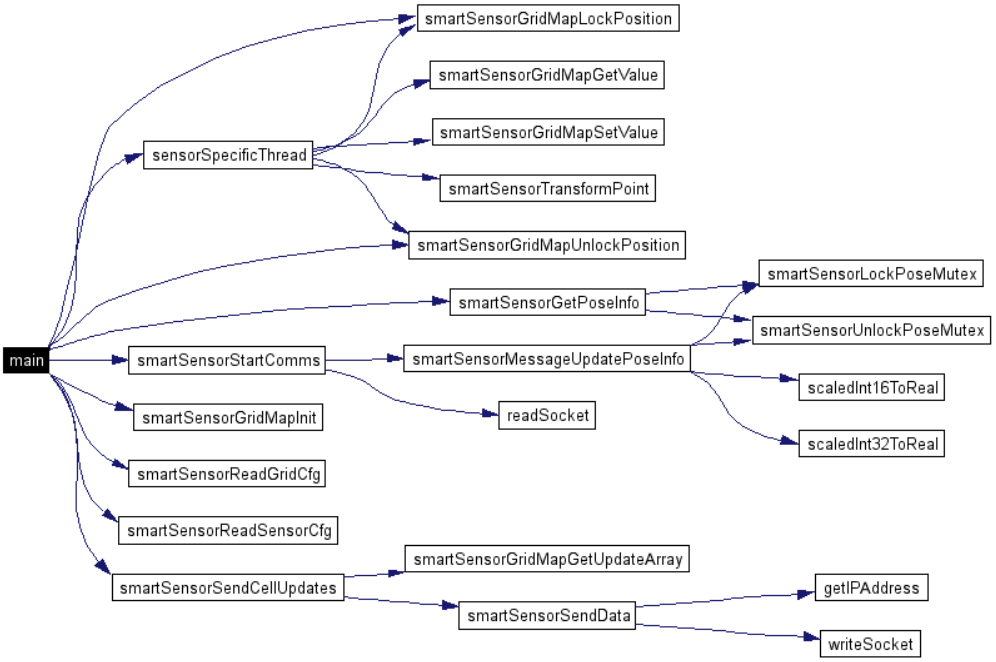

Figure 3-8. Call graph for all functions within the base smart sensor API.

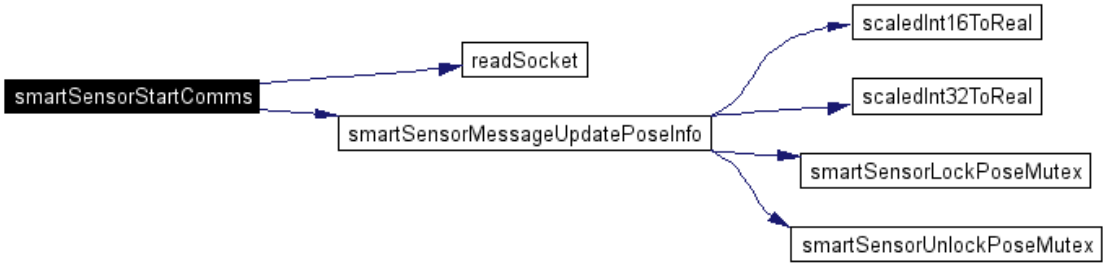

Figure 3-9. Call graph for smart sensor communications receive thread

$$\begin{bmatrix} x_{vehicle} \\ y_{vehicle} \\ z_{vehicle} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_{offset} \\ 0 & 1 & 0 & y_{offset} \\ 0 & 0 & 1 & z_{offset} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{sensor} \\ y_{sensor} \\ z_{sensor} \\ 1 \end{bmatrix} \quad (3\text{-}1)$$

To synchronize the position of the grid map, the existing cell data are shifted such that the vehicle is always located in the center of the raster grid. The benefit of shifting the cell data is that it provides a limited short-term memory of the area directly local to the vehicle.

It is assumed that the position and orientation data are fairly accurate and precise. If they are not, proper data registration will not be attained. Research is currently taking place to find ways of handling this problem, but this is outside the scope of this work. This is not an issue within this system because all smart sensors use the same position and orientation updates. Therefore any errors introduced due to loss of position system precision or accuracy will be present in all of the smart sensors' data.

Once the grid has been shifted, the sensor specific data may be entered into the traversability grid as if it were a local traversability sensor, i.e. no global position or orientation data. When the **smartSensorTransformPoint()** function is called, it handles converting the data from sensor coordinates to vehicle coordinates and finally to world coordinates. This transformation is shown in Equation 3-1.

Once updates have been made to the base smart sensor-based traversability grid, the main thread causes the grid to be checked for changes. These changes are transmitted via the **smartSensorSendCellUpdates()** function, as shown in Figure 3-12, and its access to the UDP/IP transport layer.

The next section details the implementation of a stereo vision based smart sensor. While the method described in this section is specific to the stereo vision system, the

power of the smart sensor approach is that this sensor specific interface is abstracted out. This means that as long as a sensor implementer uses the same grid parameters, interfacing to the base smart sensor will be trivial.

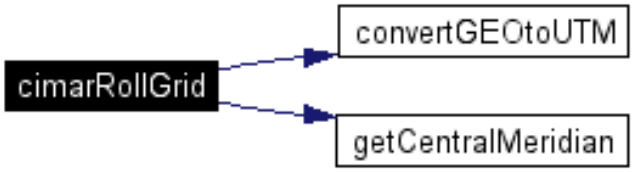

Figure 3-10. Call graph for function that determines number of rows and columns to shift the traversability grid based on the current and previous positions

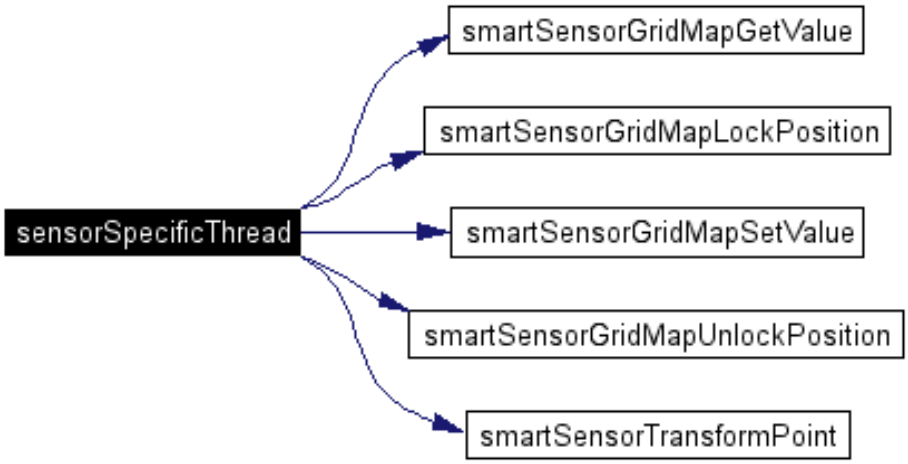

Figure 3-11. Call graph for sensor specific interface thread.

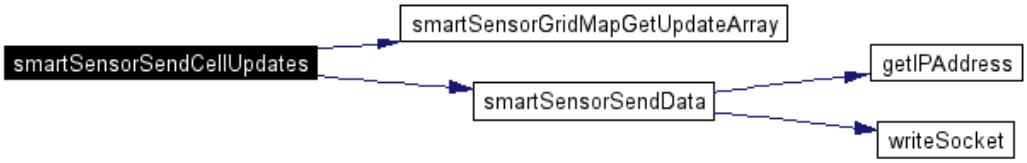

Figure 3-12. Call graph for function used to detect changes in the traversability grid and transmit these changes to the smart sensor arbiter

3.5 Smart Stereo Vision Sensor Implementation

Like all CIMAR smart sensors, the smart stereo vision sensor builds on the base smart sensor module. It is based on the Videre Design STH-MD1-C stereo vision camera system and the SRI Small Vision System.

3.5.1 Stereo vision Hardware

The Videre Design STH-MD1-C, shown in Figure 3-13, is a high resolution, wide baseline stereo vision camera system. It consists of two CMOS imagers and an IEE1394 (Firewire) interface for transferring the digital images to the computer doing the stereo processing.

3.5.2 Stereo Vision Software

To handle the tasks of camera calibration, image rectification, and stereo correlation, the SRI Small Vision System (SVS) is used. SVS provides an application programmer's interface to its internal implementation of the functions necessary for stereo processing [35]. This system is available for both Linux and Windows based systems. Figure 3-14 shows a rectified stereo image pair from the Videre system. The output of SVS's processing is shown below the stereo pair. In this image brighter pixels correspond to smaller distances. Conversely, darker pixels correspond to larger distances as calculated by stereo correlation.

3.5.3 Smart Stereo Vision Sensor

The base smart sensor handles all of the low level functionality of the smart sensor. Because of this, the smart stereo vision sensor has to only provide an instantaneous indication of the region traversability within the area local to the vehicle. To handle the tasks of camera calibration, image rectification, and stereo correlation, the SRI small vision system (SVS) is used.

A check of the range resolution was done at the range specified by the Team CIMAR perception team. The following equation relates the range resolution to the camera parameters as:

$$\Delta r = \frac{r^2}{b \cdot f} \cdot d \tag{3-2}$$

where Δr is the resolution at range r, b is the baseline of the stereo vision camera system, f is the focal length of the camera lenses, and d is the smallest disparity perceivable by the stereo vision system. For this sensor system's STH-MD1, the baseline was 200 millimeters, the focal length was 12.5 millimeters, and the smallest disparity perceivable was 0.46875e-3 millimeters. A graph of range versus range resolution is shown in Figure 3-16 [35]. As can be seen in the figure, at a range of the 30 meters, the range resolution is approximately 17 mm – not a problem at all considering that the grid resolution is constant at 0.5 meters per cell.

Figure 3-13: Videre Design STH-MD1-C stereo camera head (left)

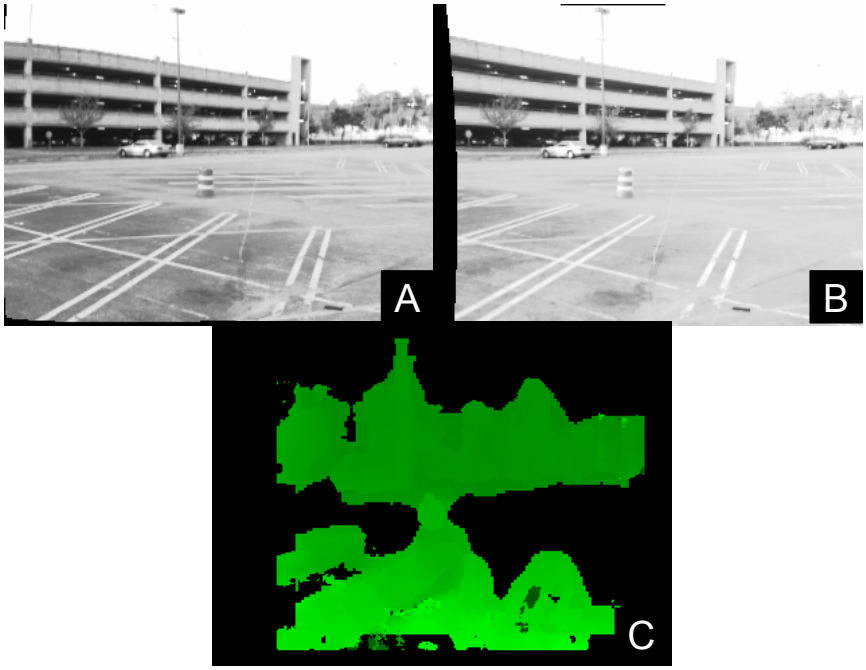

Figure 3-14: Source data and results from stereo correlation. A) Left image. B) Right image. C) Disparity image.

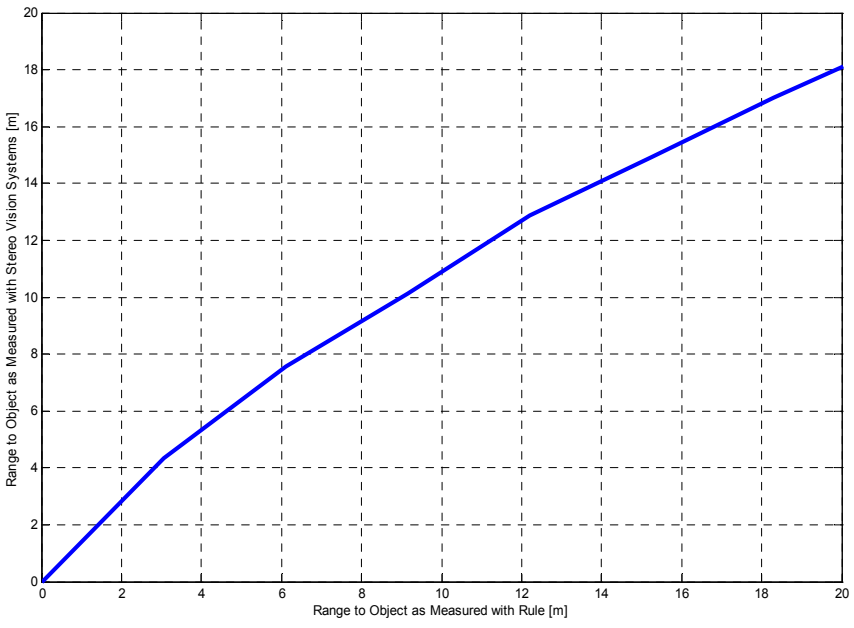

Figure 3-15. Graph of range determined from stereo vision system vs. actual measured range

Region traversability value judgment is based on an assessment of the three dimensional data provided by the stereo vision system. A method for fast obstacle

classification based on allowable slope is presented in [15]. This is shown in Equation 3-3.

$$\frac{\left(z_k - z_g\right)}{\left(x_k - x_g\right)^2 + \left(y_k - y_g\right)^2 + \left(z_k - z_g\right)^2} \geq \sin^2(\alpha) \tag{3-3}$$

In this equation (x_g, y_g, and z_g) represent the coordinates of a known ground point and (x_k, y_k, and z_k) represent the coordinates of sensed point in space. The maximum allowable angle is represented by α. This method analyses each point within the sensor data to determine whether or not it represents a traversable region.

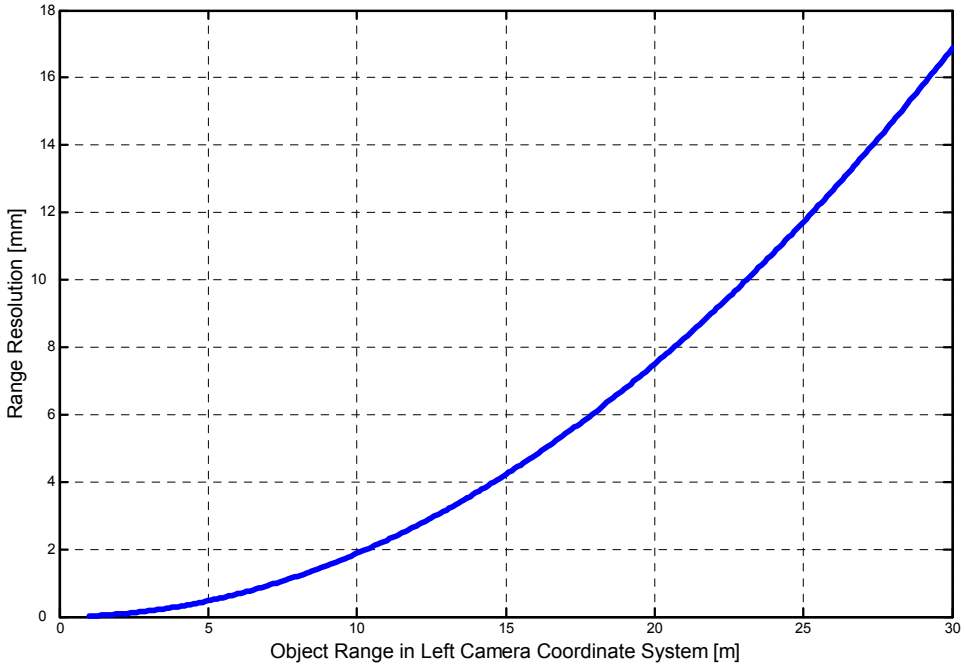

Figure 3-16. Plot of range resolution vs. range for the Videre Design STH-MD1-C with 12.5mm focal length lenses

In this work Hong et al. [15] also show that it is possible for an object to fail this test, but still be an obstacle because of the object's height. The following test, Equation 3-4, checks for this condition, by considering the height of the object above the ground

plane. If an object is too tall for the vehicle to drive over, then it is classified as an obstacle. The constant H in Equation 3-4 sets this threshold.

$$\left|z_k - z_g\right| < H \tag{3-4}$$

Because the smart stereo vision system is based on accumulated instantaneous sensor readings, the ground point used in (3.3) is the origin of the vehicle projected onto the plane defined by the intersection of the vehicle's tires and the ground plane. By establishing this point as the origin of the vehicle's coordinate system, the terms x_g, y_g, and z_g drop out of the equation. Therefore for the instantaneous sensor reading, the obstacle check is based on Equation 3-5.

$$\frac{z_k}{x_k^{\,2} + y_k^{\,2} + z_k^{\,2}} \geq \sin^2(\alpha) \tag{3-5}$$

The base smart sensor API is used to perform the conversion from three-dimensional world coordinates to two dimensional grid coordinates. Because the base smart sensor has access to the current position and orientation of the vehicle, the offsets of the vehicle and sensor coordinate systems, and the range and resolution of the traversability grid, it is able to provide the smart stereo vision system a transformation from world coordinates directly to grid coordinates.

To update data within the local traversability grid, a method for updating the traversability grid was established based on the work by [20]. This implementation of a local occupancy grid uses a simpler approach for grid updating. They based their updated method on the observation that stereo errors are systemic and are not easily modeled probabilistically. This is because stereo vision systems are dependent of the visual properties of the environment. For example, as lighting and texture conditions

change, the performance of the stereo matching process may improve or degrade. Because of this a probabilistic model of the stereo vision system may not be the same as under ideal conditions. Similar to the grid cell properties established in Section 3.3.2.2, Murray and Little's [20] method uses a one byte per cell representation with an unknown state represented by the value 127.

$$\textbf{IF } i \in \text{TRAVERS}(r) \textbf{ THEN } \text{G}(i) = \text{G}(i)+K_t \textbf{ ELSE } \text{G}(i) = \text{G}(i)-K_{nt} \qquad (3\text{-}6)$$

An extension of their work was developed for use in the smart stereovision system traversability grid. This is method updates cells based on Equation 3-6. As in Murray and Little's implementation, *i* represents a location within the grid –in this case the traversability grid, *r* is a reading from the stereo vision sensor, **TRAVERS()** is the operator that determines if the sensor reading represents a traversable point, **G(*i*)** is the traversability grid value at location *i*, K_t and K_{nt} are constants used to, respectively, increment and decrement the traversability cell value. The addition of K_t and K_{nt} is a departure from Murry and Little's approach where a single constant is used. By having separate constants, emphasis can be placed on either maintaining clean data with slower response times for detecting obstacles or vise versa. To bias one approach over the other, the associated incrementing constant is made larger than the other. Otherwise the constants should be equal.

3.6 Use of Obstacle Detection and Free Space Sensors

As mentioned previously, the purpose of the Smart Sensor Architecture is to generate a model of the world local to the vehicle to support the task of obstacle avoidance. Since all sensor modalities do not provide high-resolution data within their

fields of view, an important distinction is made between different types of sensors. They are classified as free space detectors, obstacle detectors, or a combination of both.

While highly accurate with respect to presence within a zone, the issue is that when trying to use this for obstacle detection, the entire zone would have to be classified as an obstacle because of the lack of granularity in the sensor field. Rather than considering the radar unit an obstacle detector, it is viewed as a free space detector. If the radar unit indicates that there is no object in a particular zone, it can be assumed that the entire zone is clear. It follows that if the radar until indicates that all zones are clear then there is a fast, computationally inexpensive method of classifying the entire sensor field of view as clear and traversable. The associated cells are updated to correspond to this classification. Sensors with high resolution such as the stereo vision system or the LADAR sensor presented in chapter two can be used to detect free space as well as objects.

3.7 Smart Sensor Arbiter Implementation

The smart sensor arbiter also builds on the base smart sensor module. The initial implementation of the arbiter is minimalist. Because position and orientation updates from the JAUS network are sent through the smart sensor arbiter down to all of the smart sensor components, the arbiter knows its current position. As grid cell updates come in from the smart sensors, each message has a latitude and longitude position stamp indicating the origin of the cell updates. Using the Universal Transverse Mercator projection, the latitude and longitude based coordinate values are converted to Cartesian coordinates within a UTM zone. The arbiter then does the same conversion using the coordinates of the vehicle's current location. The difference in the vehicle position and

the origin of the grid cell updates is converted to an offset of grid coordinates (rows and columns). This offset is simply applied to each grid cell update.

This approach is acceptable in this situation because, while they are distributed systems, all smart sensors use position data from single position system. If each smart sensor were on a different subsystem with independent position systems, then this approach would have to be modified because of accuracy and precision issues.

As the data within the smart sensors' grids change, they transmit corresponding traversability grid updates. To fuse the data from the smart sensors, the arbiter uses the method shown in Equation 3-7.

$$G(i) = \sum_{n=1}^{num\_smart\_sensors} w_n \cdot cell\_update(row_i - row\_offset, col_i - col\_offset)_n \qquad (3\text{-}7)$$

where G(i) is the value of the fused traversability grid cell at the grid position *i*. The constant w_n represents a weight associated with data from smart sensor *n*.

Consider the case of a smart sensor system consisting of stereo vision based smart sensor and a RADAR based smart sensor. If the RADAR is considered a free space detector and is limited to that traversable range for cell updates, then when the RADAR unit classifies a pixel as free, it is highly probable that the RADAR's data would always be more accurate than the stereo vision system, which is subject to the systemic errors discussed by [20]. Therefore weighing the RADAR data more than the stereo data would put more emphasis on the high quality RADAR data. This is only true when the RADAR is used as a free space detector. If the RADAR were used as an obstacle detector, then it would adversely affect the quality of the fused data from the stereovision system.

The Region Clutter Sensor is a quasi-component embedded in the smart sensor arbiter. This component simply applies a non-traversable region threshold to the values

within a region of the traversability grid. As the saturation of non-traversable regions increases, the vehicle makes the appropriate changes in velocity necessary to allow successful negotiation of area. This component is only present in the arbiter, so when a smart sensor replaces the arbiter, this functionality is lost.

CHAPTER 4
JAUS WORLD MODEL KNOWLEDGE STORES

The previous chapter presented a detailed description of the smart sensor architecture and the implementation of a stereovision based smart sensor unit. The smart sensor architecture message set defines a standard logical interface that allows data to be shared between the components of a perception system. While this interface is acceptable to meet the synchronization requirements of the smart sensor architecture's traversability grids, it is not general enough to support the sharing of data based on the raster and vector modeling methods presented in Chapter 2. Building on the reviewed literature as well as on lessons learned from developing the smart sensor architecture, this chapter introduces standard modeling and input/output methods for world model knowledge stores.

A world model knowledge store is to be the central geospatial data store for a JAUS component, node, subsystem, or system. The knowledge store provides only geospatial data storage and access methods. Therefore, no processing or higher level functionality should be provided by the knowledge store. It is the most primitive world modeling component and forms the foundation for all future world model components. These future components will extend the world modeling capabilities of JAUS by providing functions such as value judgment, simulation, prediction, etc. as described by Mystel [18].

Similar to the smart sensors in Chapter 3, the world model knowledge stores are envisioned as location independent, modular JAUS components. Because of this, it is possible to have multiple subsystems accumulating data in a global world model knowledge store or to have individual subsystems accumulate data in their own world model knowledge stores and then have synchronization of those stores. The data within these stores may be either persistent or volatile. This will not be specified as it is an implementation issue.

4.1 Observations and Recommendations

Chapter 2 showed that there is a considerable amount of commonality between the numerous types of data that are available in *a priori* data stores and the types of data that may be accumulated in real-time. The main two classes of data types are raster and vector data. Raster data may consist of elevation, geo-referenced orthoimages, density maps, occupancy grids, traversability grids, etc. Vector data may consist of digital road maps, polygon maps, etc. By enforcing some constraints, it is possible to distill these data into a common format that may be used by unmanned systems community.

A number of key observations were made about the current methods for accessing and sharing real time and *a priori* geospatial data.

- The level of complexity of modeling methods designed for *a priori* data-sharing (such as SDTS and GML) is well beyond what is necessary for JAUS based unmanned systems.
- There are a number of different projections that may be used to transform data from geodetic coordinates to a two or two and a half dimensional surface.
- Current world modeling methods are, at their core, based on either raster or vector primitives.
- For real-time world modeling on unmanned vehicles, raster methods are used most often.

- Both vector and raster modeling methods are commonly used in *a priori* data stores.
- Most *a priori* data stores include metadata which have extra information about the stored data.

Therefore it is recommended that at a minimum the initial JAUS World Model standard should:

- Provide the ability for JAUS based subsystems, nodes, and/or components to share geospatial data with minimal complexity.
- Allow developers some degree of flexibility within the constraints of the standard.
- Specify a map projection and horizontal and vertical datums to be used within the knowledge stores.
- Allow for use and transfer of *a priori* and real-time raster and vector data.
- Provide a mechanism to allow distinguishing between different types of geospatial data.
- Provide a means for saving and sharing information about the geospatial data within the knowledge store.
- Meet the standard JAUS requirements for definition of new components.

A JAUS World Model Knowledge Store standard should not be concerned with the method of modeling data internal to the system, but with how the data are formatted and presented to other JAUS components that use or store geospatial data. The work done on the smart sensor architecture as well as past experiments with JAUS interoperability has shown that as component interfaces become more complex, it becomes increasingly difficult to achieve true interoperability. The approach with the message set presented herein is to develop a method of sharing the data at the most primitive levels. Complexity has been limited so as to provide to the many organizations that make up the JAUS Working Group a more acceptable and undemanding initial standard. As the geospatial data-sharing requirements of the group change, so too will the standard.

Standards inherently impose limitations and this must be accepted. However, standards that are too restrictive run the risk of losing of support. Therefore the JAUS World Model Knowledge Store standard is developed to be as flexible as reasonably possible. The messages are also designed to be as extensible as is possible within the JAUS framework. This standard and all future world modeling component standards should be considered living documents that are able to quickly change to meet the needs of system developers. The JAUS working group's review process will assure that the changes that are made are only those that are applicable to the group as a whole.

Geospatial data transferred from different systems must use the same map projections, ellipsoidal Earth model, and horizontal and vertical datum. For the global coordinates, JAUS specifies that all systems use the World Geodetic System 1984 (WGS84). The map projection will be the Universal Transverse Mercator Projection. Vertical measurements will be based on the vertical datum as established by the ellipsoidal model of the Earth. Since most of the systems will be operating in the United States, the horizontal with be the North American Datum as established in 1983 (NAD83).

Chapter 2 showed that real-time world modeling methods typically use tessellated raster data structures and *a priori* world modeling methods use ether raster or vector data structures. Therefore a message set has been developed to support two types of knowledge stores: the World Model Raster Knowledge Store and the World Model Vector Knowledge Store.

The World Model Raster Knowledge Store provides a method for storage and sharing of raster formatted geospatial data within a JAUS system. Many unmanned

systems with perception systems utilize a form of the local occupancy grid as introduced by Elfes [13]. The local occupancy grid is implemented as a tessellated geo-referenced grid. The World Model Raster Knowledge Store is a generalization of such a local occupancy grid. It is desired to have this knowledge store support most types of raster data. These include binary image, grey scale images, RGB images, digital elevation model (DEM) data, traversability, occupancy, etc. Typically an occupancy grid stores a value corresponding to a truth metric in each cell. When raster data are stored such that each cell represents a height at that location (such as DEM data), this is referred to as two and a half (2.5) dimensions [21].

Storage and sharing of spatial data such as points, lines, polylines, or polygons is supported by the World Model Vector Knowledge Store. These vector formatted spatial data provides a number of benefits. The primary benefit of such a system in the context of JAUS is that it requires significantly less bandwidth to transmit data as compared to the raster store. This method therefore can reduce the storage requirements within the system.

A feature class represents a categorization of types of spatial data. For example, occupancy, free space, objects, roads, terrain, building, etc. all represent distinct feature classes. A geo-referenced, orthoimage may also represent a feature class. It may be more intuitive to consider these feature classes as different layers of geospatial data within the knowledge store. This is important because it allows different types of spatial data to be handled separately. Predefined feature classes will eventually be defined by the JAUS World Model Subcommittee in the interest of true world model interoperability. Since it is not possible to define all types of feature classes *a priori*, a

sizeable amount of space has been set aside for user defined feature classes. While this does have an adverse effect on interoperability, this is mitigated by having system developers provide each other with a data dictionary when they wish to interoperate. The data dictionary is simple a description of which types of data correspond to a feature class identifier. Even with the predefined feature classes, when testing interoperability system developers must establish the data types that they are using within the knowledge store. It is possible that this exchange could be handled during the discovery process provided in the forthcoming JAUS dynamic configuration and registration extensions

To allow dissemination of information about a feature class, the world model framework provides for storage and transfer of feature class metadata. In this context, metadata is simply text that provides general information about the data within a particular feature class. Initially the metadata are developed to be human readable text in a format specified by the user. Bolstad [7] gives an introduction to metadata and discusses the Content Standard for Digital Geospatial Metadata. Just as this is considered only a guideline for the GIS community, it is considered only a guideline for the JAUS community. Initially these metadata are designed to be human readable and not used in any distributed computations that may be performed on these data.

The local request identifier (LRID) is a single-byte numerical identifier attached to certain classes of messages originating outside of the world model knowledge store. This feature allows synchronization of messages and their associated response. This is important because even though requests to the knowledge store may be synchronized, there is no guarantee that the responses will be synchronized. By attaching the LRID, the requesting component will be able to internally synchronize any asynchronous responses.

4.1.1 Raster and Vector Object Representation

This section describes the raster and vector objects as they should be formatted in the JAUS messages that define the input and outputs of the knowledge stores. Special attention must be made to assure that these conventions are followed by all components sending data to or receiving data from the world model knowledge stores.

The data within a raster knowledge store should always maintain a north-east orientation. Raster data in the knowledge store should be geo-referenced by defining their origin as a single point described by the intersection of a line of latitude and a line of longitude (WGS84). The grid parameters also include the number of rows and columns and the grid resolution. While a grid cell is specified as a point, that point covers an area equal to the grid resolution squared. A Cartesian coordinate system is established at the geo-referenced point. The Cartesian coordinates of the grid cells are derived from use of the Universal Transverse Mercator projection. The grid cells may also be referenced by their row and column offset from the origin point. Figure 4-1 shows the format of a layer of raster data. While cells may have negative row and column values with respect to the grid origin, when transmitting rectangular grid data (e.g. images, DTED), the origin of the raster data must be the point that defines the cell whose column coordinate is equal to the column coordinate of the western most cells and whose row coordinate is equal to the row coordinate of the southern most cells. Therefore when transmitting a rectangular array of raster data, there will be no cell values with coordinates less than zero.

For the vector knowledge store, objects are represented as points, lines and polylines, and polygons. The coordinates of these points are defined by a point of latitude and longitude (WGS84). Polylines and polygons may consist of up to 65535 vertices. Figure 4-2 shows the format of these vector objects. Rather than assigning

these points Cartesian coordinates with respect to an arbitrarily chose datum, each vertex is expressed as a point of latitude and longitude.

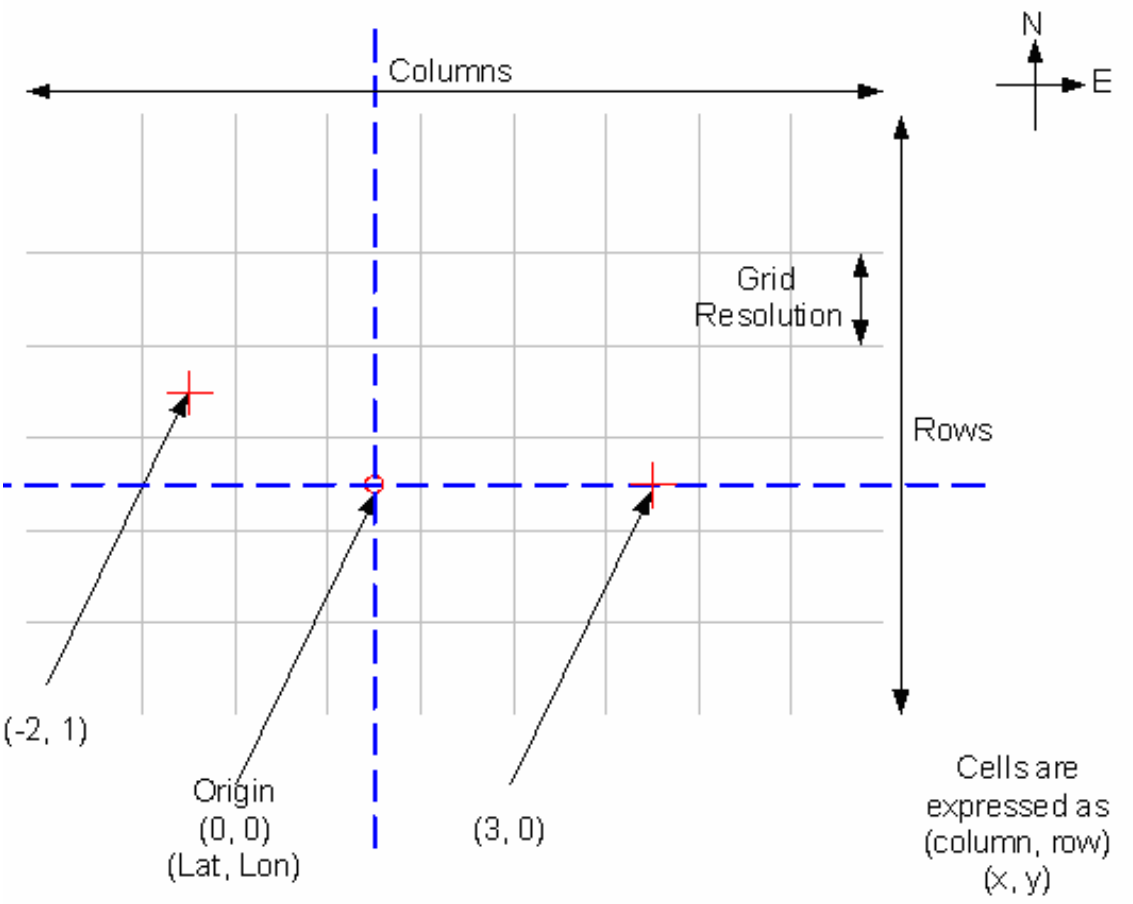

Figure 4-1. Definition of raster grid parameters and coordinate system

The vector objects on the right of Figure 4-2 have a buffer parameter. The buffer parameter establishes a radial region around each vector object vertex and connects the radial regions of two or more radial regions by drawing lines at their tangents. The area within these radial regions and tangent lines are considered to be within the vector object's buffer zone. This feature allows a region to be established in proximity to the vector objects. For example, United States Geological Survey (USGS) road data is presented in vector form representing the center-line of such roads. It may be useful to do a search within the perimeter along a particular route defined in the USGS digital line

graph data. For simple cases, it may be possible to generate a polygonal representation of the area around the road. Establishing this polygon will require transmitting the coordinates of each of its vertices. As the problem scales up, this method becomes very inefficient. A better solution to this problem would be to determine the route using the USGS digital line graph data and assign a region buffer to each line segment. The region buffer is defined as an offset distance in meters. The spatial buffer is established by defining a radius from each point on the vector object. For many cases, this buffer will be a simple offset with the exception of point objects and along non-smooth contours. Figure 4-2 shows these cases. If the system designer requires finer control over this region, they may define the buffer using the aforementioned polygonal representation.

4.2 World Model Knowledge Store Message Set

The following sections present the initial draft message set for the first two JAUS World Modeling components. This message set is based on a review of the current methods of modeling spatial and geospatial data as presented in Chapter 2. These methods are distilled into their most basic form and codified into a standard consistent with the JAUS framework.

4.2.1 JAUS Core Input and Output Message Sets

Support for the JAUS core message set is required by the current version of the JAUS Reference Architecture (RA). The JAUS Core Message Set consists of the following messages:

- Code 0001h: Set component authority
- Code 0002h: Shutdown
- Code 0003h: Standby
- Code 0004h: Resume
- Code 0005h: Reset
- Code 0006h: Set emergency

- Code 0007h: Clear emergency
- Code 0008h: Create service connection
- Code 0009h: Confirm service connection
- Code 000Ah: Activate service connection
- Code 000Bh: Suspend service connection
- Code 000Ch: Terminate service connection

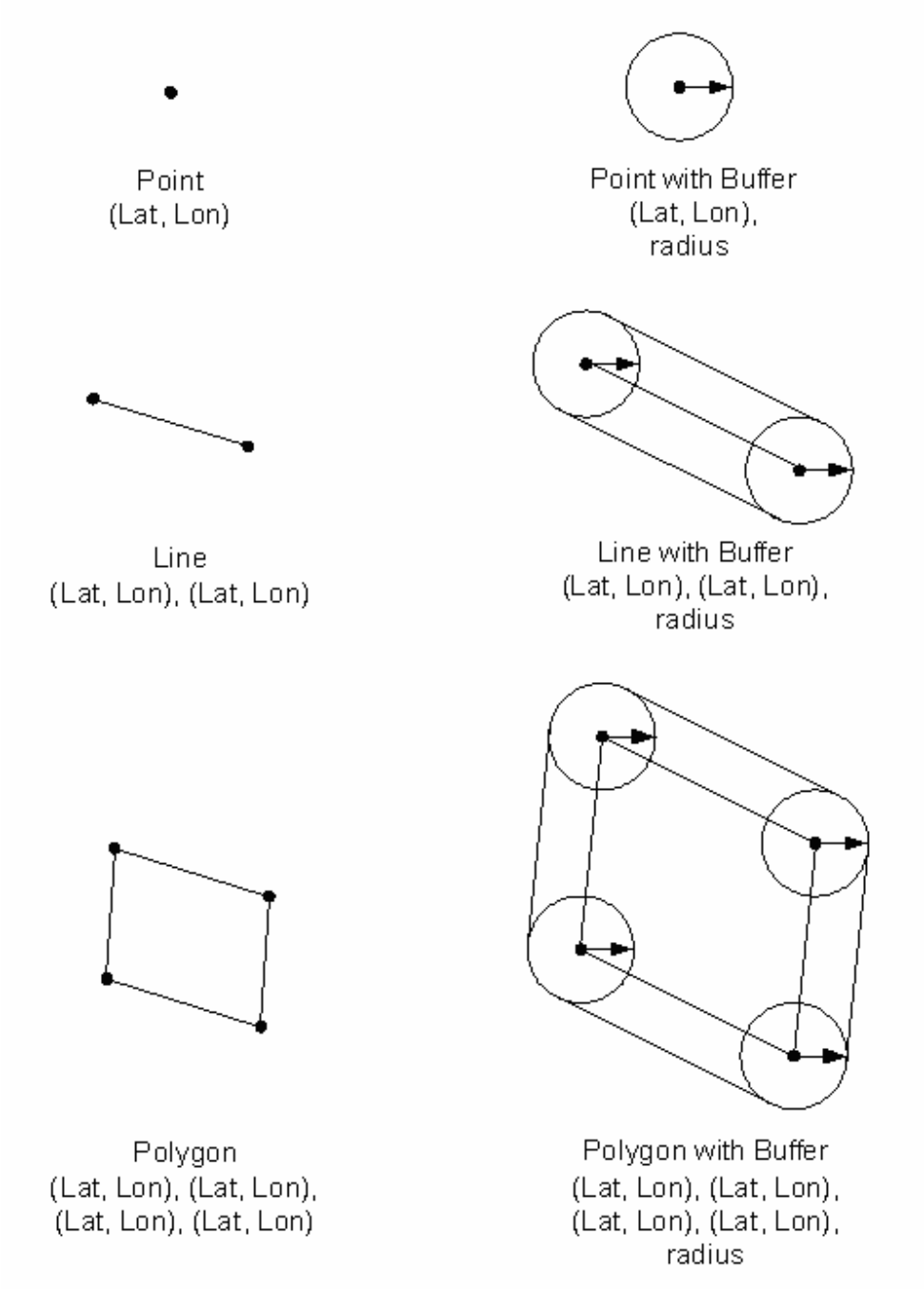

Figure 4-2. Definition of vector objects and parameters

While the JAUS RA does require that these messages be accepted by all components, there is no requirement that components have an action associated with each input message. Because the expected behavior of components while in each state is somewhat ambiguous, they will be defined for the world model knowledge stores. So too will the message that are required to have a response.

The world model knowledge stores should have an appropriate response to the following messages:

- Code 0002h: Shutdown
- Code 0003h: Standby
- Code 0004h: Resume
- Code 0005h: Reset
- Code 0009h: Confirm service connection
- Code 000Ah: Activate service connection
- Code 000Bh: Suspend service connection
- Code 000Ch: Terminate service connection
- Code 2002h: Query component status
- Code 4002h: Report component status

The Code 0002h: Shutdown message should cause the receiving knowledge store to immediately terminate all data transfer upon receipt. If the knowledge store is responding to a query, it should immediately terminate the flow of data and transmit the Code F405h: Report Raster Knowledge Store Data Transfer Termination or the Code F424h: Report Vector Knowledge Store Data Transfer Termination message to the component whose query response was interrupted and any components with outstanding requests. Upon termination of all data transfer, the world model should execute its specific shutdown routine and then halt. It should no longer respond to any data requests and should require a hard reset in order to resume operation.

The Code 0003h: Standby message should cause the receiving knowledge store to respond as if it had received the Code 0002h: Shutdown message. The exception is that

the knowledge store should not halt. It should respond only to the Code 0004h: Resume and Code 0005: Reset messages. Upon resumption to the ready state, the knowledge store should resume normal operations. It should not resume any suspended query responses.

The Code 0005h: Reset message should cause the receiving knowledge store to immediately terminate the transfer and processing of any data. The knowledge store should transmit to all components with outstanding requests or data transfers the Code F405h: Report Raster Knowledge Store Data Transfer Termination or the Code F424h: Report Vector Knowledge Store Data Transfer Termination message. The knowledge store should them immediately restart and return to the ready state. Terminated data transfers should not resume.

The Codes 0009h: Confirm Service Connection, 000Ah: Activate Service Connection, 000Bh: Suspend Service Connection, 000Ch: Terminate Service Connection, 2002h: Query Component Status and 4002h: Report Component Status messages should all invoke that typical JAUS response associated with their receipt.

4.2.2 Raster Knowledge Store Input Message Set

In the following subsections are the messages that define the input to the raster version of the world model knowledge store. These command, query, and event setup class messages are transmitted in order to initiate an appropriate inform or event notification class message output. These outputs messages are defined in Section 4.2.3.

The inputs to the raster knowledge store are:

- The JAUS core input message set
- Code F000h: Create raster knowledge store object
- Code F001h: Set raster knowledge store feature class metadata
- Code F002h: Modify raster knowledge store object (cell update)

- Code F003h: Modify raster knowledge store object (grid update)
- Code F004h: Delete raster knowledge store objects
- Code F200h: Query raster knowledge store objects
- Code F201h: Query raster knowledge store feature class metadata
- Code F202h: Query raster knowledge store bounds
- Code F600h: Raster knowledge store event notification request
- Code F601h: Raster knowledge store bounds change event notification request
- Code F005h: Terminate raster knowledge store data transfer

4.2.2.1 Code F000h: Create raster knowledge store object

The Code F000h: Create Raster Knowledge Store Object message (Table 4-1) is used to create and initialize a layer of feature class data within the raster knowledge store. In order for data to be added to the feature class, the feature class layer must first be created.

The origin of the raster grid must be geo-referenced by specifying its origin in fields 4 and 5 as a point of latitude and longitude. Extents of the layer must also be specified as a number of rows and columns in fields 7 and 8. Both the data types that describe the number of rows and columns and the cell attribute type are variable and must also be specified in fields 6 and 11, respectively. The grid cell resolution is also specified in field 9. Because this message is used to create a feature class layer, the feature class must be specified using field 10.

This message has a single optional field (field 11). Inclusion of this optional field is determined from the state of bit zero in the message presence vector (Table 4-2). If the bit zero is set, then the value in field 12 shall be used to initialize all cells within the feature class.

When the feature class layer is initialized using this message, the data are filled in the grid on a row by row basis starting at the southern most row and moving north. It is filled beginning at the southwestern most point moving east.

Table 4-1. Create raster knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Message Properties	Byte	N/A	Bit Field 0: Request confirmation of object creation 1 – 7: Reserved
2	Message Properties	Byte	N/A	Bit Field 0: Request confirmation of object creation
3	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component
4	Origin Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
5	Origin Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
6	Raster Data Row and Column Data Type	Byte	N/A	Enumeration 0: Byte 1: Reserved 2: Reserved 3: Reserved 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved
7	Raster Grid Update Rows	Varies (See field 4)	Grid Cells	
8	Raster Grid Update Columns	Varies (See field 4)	Grid Cells	
9	Cell Resolution	Float	Meters	
10	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
11	Raster Cell Data Type	Byte	N/A	Eumeration Same format as field 6
12	Initial Value for Raster Grid Cells	Varies (see field 11)	N/A	

Table 4-2. Presence vector for create raster knowledge store objects message

Vector to Data Field Mapping for Above Command								
Vector Bit	7	6	5	4	3	2	1	0
Data Field	R	R	R	R	R	R	R	12

4.2.2.2 Code F001h: Set raster knowledge store feature class metadata

As described in Section 4.1, metadata are data about data. The Code F001h: Set Raster Knowledge Store Feature Class Metadata (Table 4-3) message allows a user to create, modify, and erase feature class metadata. At the present time the format of these metadata is not specified. It is left to the system designer to develop a convention for doing this. Initially these data are to be used by the human operators. In the future a schema may be defined so as to provide a standard metadata format that may be parsed and the data used by unmanned systems without human intervention.

Table 4-3. Set raster knowledge store feature class metadata message format

Field #	Name	Type	Units	Interpretation
1	Metadata Options	Byte	N/A	Enumeration 0: Append 1: Prepend 2: Overwrite 3 – 254: Reserved 255: Erase All
2	Feature Class	Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
3	Number of String Characters	Unsigned Short Integer	N/A	0 … 65,535 This field should be equal to zero only when Field 1 is equal to 255 (Erase All)
4	Metadata	String	N/A	Variable length string

4.2.2.3 Code F002h: Modify raster knowledge store object (cell update)

The Code F002h: Modify Raster Knowledge Store Object (Cell Update) message (Table 4-4) is used to change data within a raster knowledge store feature class layer.

This message can only be used on a layer that has been created within the raster knowledge store. This method is specified as a cell update version because it allows modification of the raster grid on a cell by cell basis. This message has no optional fields.

The origin of the raster grid cell updates must be geo-referenced by specifying its origin in fields 2 and 3 as a point of latitude and longitude. Both the data types that describe the update row and column and cell attribute are variable and must also be specified in fields 4 and 7, respectively. The grid cell update resolution is also specified in field 5. Because this message is used to modify a feature class layer, the feature class must be specified using field 6. The data type for the field that specifies the number of cell updates included in the message (field 9) is also variable and is defined in field 8. Each cell update is a three-tuple representing the cell update's row, column, and update attribute value.

Table 4-4. Modify raster knowledge store object (cell update) message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component
2	Origin Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
3	Origin Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
4	Raster Data Row and Column Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved

Field #	Name	Type	Units	Interpretation
5	Cell Resolution	Float	Meters	
6	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 - Reserved
7	Raster Cell Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 19: RGB (3 Bytes) 10 – 255: Reserved
8	Data Type for Number of Cell Updates	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved
9	Number of Cell Updates	Varies (see field 8)	N/A	
10	Raster Cell Update *1* Row	Varies (see field 4)	N/A	
11	Raster Cell Update *1* Col	Varies (see field 4)	N/A	
12	Raster Cell Update *1* Data	Varies (see field 7)	Varies with Feature Class	
…				
…				
…				

Table 4-4. Continued

Field #	Name	Type	Units	Interpretation
3n + 7	Raster Cell Update n Row	Varies (see field 4)	N/A	
3n + 8	Raster Cell Update n Col	Varies (see field 4)	N/A	
3n + 9	Raster Cell Update n Data	Variable (see field 7)	Varies with Feature Class	

4.2.2.4 Code F003h: Modify raster knowledge store object (grid update)

The Code F003h: Modify Raster Knowledge Store Object (Grid Update) message (Table 4-5) is similar to the Code F002h: Modify Raster Knowledge Store Object (Cell Update) message in that it permits change of grid cell values. It differs from that method in that rather than transmitting single cell updates, an entire rectangular patch of cells is updated. As the number of cells that need to be modified increases, this method becomes more efficient than the cell update method.

The origin of the raster grid update must be geo-referenced by specifying its origin in fields 2 and 3 as a point of latitude and longitude. Both the data types that describe the update row and column and cell attribute are variable and must also be specified in fields 4 and 9, respectively. Fields 5 and 6 specify the number of rows and columns of raster grid updates being transmitted. The grid cell update resolution is also specified in field 7. Because this message is used to modify a feature class layer, the feature class must be specified in field 8.

Table 4-5. Modify raster knowledge store object (grid update) message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component

Table 4-5. Continued

Field #	Name	Type	Units	Interpretation
2	Origin Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
3	Origin Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
4	Raster Data Row and Column Data Type	Byte	N/A	Enumeration 0: Byte 1: Reserved 2: Reserved 3: Reserved 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved
5	Raster Grid Update Rows	Varies (See field 4)	Grid Cells	
6	Raster Grid Update Columns	Varies (See field 4)	Grid Cells	
7	Cell Resolution	Float	Meters	
8	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 - Reserved
9	Raster Cell Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 19: RGB (3 Bytes) 10 – 255: Reserved
10	Raster Cell Update 1	Varies (see field 9)	N/A	
11	Raster Cell Update 2	Varies (see field 9)	N/A	
…				

Table 4-5. Continued

Field #	Name	Type	Units	Interpretation
9 + n	Raster Cell Update n	Varies (see field 9)	N/A	
10 + n	Raster Cell n+1	Varies (see field 9)	N/A	

4.2.2.5 Code F004h: Delete raster knowledge store objects

The Code F004h: Delete Raster Knowledge Store Object message (Table 4-6) is used to free all resources allocated to a feature class layer within the raster knowledge store. In order to resume accumulation of data within the deleted feature class, the feature class layer must be recreated using the Create Raster Knowledge Store Object message. The message allows a single feature class or all feature classes to be deleted in one message.

Table 4-6. Delete raster knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Byte	N/A	See mapping table below
2	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component
3	Number of Feature Classes	Byte	N/A	
4	Feature Class 1	Short Integer	N/A	Enumeration 0 … 65,534 – See Feature Class Table 65,535 – ALL
…	…	…	…	…
3 + n	Feature Class n	Short Integer	N/A	Enumeration 0 … 65,534 – See Feature Class Table 65,535: Reserved

4.2.2.6 Code F200h: Query raster knowledge store objects

The Code F200h: Query Raster Knowledge Store Objects message (Table 4-7) provides access to data within the raster knowledge store. Field 1 of this message is the

message presence vector (Table 4-8). The optional fields in this message are fields 4, 5, and 6. Field 2 is the Query Response Properties bit field. When bit zero is clear, the response to the query should only include the number of records that would be returned. When bit one is set, the query response shall be the Code F402h: Report Raster Knowledge Store Objects (Cell Update) message. Otherwise, the Code F403h: Report Raster Knowledge Store Objects (Grid Update) message shall be sent. Field 3 is the message Local Request Identifier. This field allows synchronization of message responses. Field 4 is the Raster Query Resolution. This field allows the querying component to specify the cell resolution to be used in the response to the query. If this resolution does not match the native resolution of the queried knowledge store, then the knowledge store should either sub-sample or interpolate the data to obtain the desired resolution. This field is optional. Field 5 specifies a specific feature class to be queried. This field is optional. If a feature class is not specified, then the query should be done on all feature classes within the knowledge store. Fields 6 through 9 specify two points of latitude and longitude that limit the range of the query. These fields are optional. If presence vector bit two is set, then fields 6 through 9 shall all be included. Otherwise, they should not.

Table 4-7. Query raster knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short Integer	N/A	See mapping table below
2	Query Response Properties	Byte	N/A	Bit Field 0: Only return number of responses that would be transmitted 1: Return cell update 3 tuples or raster scan (active low) 2 – 7: Reserved

Table 4-7. Continued

Field #	Name	Type	Units	Interpretation
3	Local Request ID	Byte	N/A	Request identifier to be used when returning data to requesting component
4	Raster Query Resolution	Float	Meters	
5	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – All Feature Classes
6	Query Region Point 1 Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
7	Query Region Point 1 Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
8	Query Region Point 2 Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
9	Query Region Point 2 Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

Table 4-8. Presence vector for query raster knowledge store objects message

Vector to Data Field Mapping for Above Command								
Vector Bit	7	6	5	4	3	2	1	0
Data Field	R	R	R	R	R	6	5	4

4.2.2.7 Code F201h: Query raster knowledge store feature class metadata

The Code F201h: Query Raster Knowledge Store Feature Class Metadata message (Table 4-9) should cause the Raster Knowledge Store to reply to the requestor with the Code F402h: Report Raster Knowledge Store Feature Class Metadata. There is a single

field associated with this message. This field specifies the feature class metadata to return in the reply. There is also an option to return metadata for all feature classes present in the queried raster knowledge store.

Table 4-9. Query raster knowledge store feature class metadata message format

Field #	Name	Type	Units	Interpretation
1	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – All

4.2.2.8 Code F202h: Query raster knowledge store bounds

The Code F202h: Query Raster Knowledge Store Bounds message (Table 4-10) is used to request the spatial extents of a single feature class or of all feature classes within a raster knowledge store. The knowledge store should respond with the Code F404h: Report Raster Knowledge Store Bounds message. The bounds are represented by two points the represent the rectangular region that just covers all of the data within the feature class layer or layers.

Table 4-10. Query raster knowledge store bounds message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Request identifier to be used when returning data to requesting component
2	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – All Feature Classes

4.2.2.9 Code F600h: Raster knowledge store event notification request

The Code F660h: Raster Knowledge Store Event Notification Request message is used to establish an event triggered query within the knowledge store. Therefore, this

message is formatted exactly the same as the Code F200h: Query Raster Knowledge Store Objects message. That message should be referenced for the format of this message. Whenever the criteria established in this message are met, depending on the query response field of the event notification request, the raster knowledge store should transmit either the Code F800h: Raster Knowledge Store Event Notification (Cell Update) message or the Code F801h: Raster Knowledge Store Event Notification (Grid Update) message.

4.2.2.10 Code F601h: Raster knowledge store bounds change event notification request

The Code F601h: Raster Knowledge Store Bounds Change Event Notification Request message is used to establish an event triggered response to notify the requesting component of when the data in a feature class extends past the bounds of the data when the initial request was sent. When the extents of the data change, the raster knowledge store will transmit the Code F802: Raster Knowledge Store Bounds Change Event Notification message.

4.2.2.11 Code F005h: Terminate raster knowledge store data transfer

This Code F005h: Terminate Raster Knowledge Store Data Transfer message is a command class message that should cause the raster knowledge store to immediately terminate the transfer of all current and outstanding data destined to the requesting component. Upon termination, the raster knowledge store should send the requestor the Code F405h: Report Raster Knowledge Store Data Transfer Termination message.

4.2.3 Raster Knowledge Store Output Message Set

In the following subsections are the messages that define the output of the raster version of the world model knowledge store. These inform and event notification class

messages are transmitted in response to the command, query, and event setup class of input messages presented in Section 4.2.2.

The outputs of the raster knowledge store are:

- The JAUS core output message set
- Code F400h: Report raster knowledge store object creation
- Code F401h: Report raster knowledge store feature class metadata
- Code F402h: Report raster knowledge store objects (cell update)
- Code F403h: Report raster knowledge store objects (grid update)
- Code F404h: Report raster knowledge store bounds
- Code F800h: Raster knowledge Store Event Notification (cell update)
- Code F801h: Raster knowledge store event notification (grid update)
- Code F802h: Raster knowledge store bounds change event notification
- Code F405h: Report raster knowledge store data transfer termination

4.2.3.1 Code F400h: Report raster knowledge store object creation

The Code F400h: Report Raster Knowledge Store Object Creation message (Table 4-11) is used to confirm creation of raster objects in the raster knowledge store. This message is sent only when an object creation message is requested by setting bit zero in the Code F000h: Create Raster Knowledge Store Object message. If this bit is set, this message will be transmitted and the local object identifier (field 1) is set to the value sent with the Code F000h: Create Raster Knowledge Store Raster Object message.

Table 4-11. Report raster knowledge store object creation message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Local request identifier sent by creating component

4.2.3.2 Code F401h: Report raster knowledge store feature class metadata

The Code F401h: Report Raster Knowledge Store Feature Class Metadata message (Table 4-12) allows access to feature class metadata stored within raster knowledge store. It is transferred in response to the Code F201h: Query Raster Knowledge Store Feature

Class Metadata message. If the query message requests all feature classes, a separate message should be sent for each feature class.

These metadata are entered using the Code F001h: Set Raster Knowledge Store Feature Class Metadata message.

Table 4-12. Report raster knowledge store feature class metadata message format

Field #	Name	Type	Units	Interpretation
1	Feature Class	Short Integer	N/A	Enumeration 0 … 65,535 - See Feature Class Table
2	Number of String Characters	Unsigned Short Integer	N/A	0 … 65,535
3	Metadata	String	N/A	Variable length string

4.2.3.3 Code F402h: Report raster knowledge store objects (cell update)

The Code F402h: Report Raster Knowledge Store Objects (Cell Update) message (Table 4-13) is sent in direct response to a Code F200h: Query Raster Knowledge Store Objects message if and only if bit two of the bit field in message field two is set. Otherwise, the Code F403h: Report Raster Knowledge Store Objects (Grid Update) message is transmitted. If bit one of field two of the Code F200h: Query Raster Knowledge Store Objects message is set, then only the first two fields of this message shall be transmitted. Field 1 of this message is Local Request Identifier sent with the query that initiated this report message. Field 2 notifies the receiving component of the number of records included in the report message. Fields 3 and 4 establish the geodetic origin (latitude and longitude) of the cell updates included in the message. Both the data types that describe the update row and column and cell attribute are variable and are specified in fields 5 and 8, respectively. Field 6 is the resolution of the raster grid

updates reported in the message. Field 7 is the feature class that raster data are assigned to.

Table 4-13. Report raster knowledge store objects (cell update) message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Request identifier sent with initial request
2	Number of Responses	Unsigned Short Integer	N/A	0 … 65,535 Number of Responses Included on this Report Message
3	Origin Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
4	Origin Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
5	Raster Data Row and Column Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved
6	Cell Resolution	Float	Meters	
7	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 - Reserved
8	Raster Cell Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 19: RGB (3 Bytes) 10 – 255: Reserved

Table 4-13. Continued

Field #	Name	Type	Units	Interpretation
9	Data Type for Number of Cell Updates	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved
10	Number of Cell Updates	Varies (see field 8)	N/A	
11	Raster Cell Update 1 Row	Varies (see field 5)	N/A	
12	Raster Cell Update 1 Col	Varies (see field 5)	N/A	
13	Raster Cell Update 1 Data	Varies (see field 8)	Varies with Feature Class	
…				
…				
…				
3n + 8	Raster Cell Update n Row	Varies (see field 5)	N/A	
3n + 9	Raster Cell Update n Col	Varies (see field 5)	N/A	
3n + 10	Raster Cell Update n Data	Varies (see field 8)	Varies with Feature Class	

4.2.3.4 Code F403h: Report raster knowledge store objects (grid update)

The Code F403h: Report Raster Knowledge Store Objects (Grid Update) message (Table 4-14) is sent in direct response to a Code F200h: Query Raster Knowledge Store Objects message if and only if bit two of the bit field in message field two is clear. Otherwise, the Code F402h: Report Raster Knowledge Store Objects (Cell Update)

message is transmitted. If bit one of field two of the Code F200h: Query Raster Knowledge Store Objects message is set, then only the first two fields of this message shall be transmitted.

Field 1 of this message is Local Request Identifier sent with the query that initiated this report message. Field 2 notifies the receiving component of the number of records included in the report message. Fields 3 and 4 establish the geodetic origin (latitude and longitude) of the cell updates included in the message. Both the data types that describe the update row and column and cell attribute are variable and are specified in fields 5 and 10, respectively. Fields 6 and 7 represent the number of rows and columns of grid update cells. Field 8 is the resolution of the raster grid updates reported in the message. Field 9 is the feature class that raster data are assigned to.

Table 4-14. Report raster knowledge store objects (grid update) message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Request identifier sent with initial request
2	Number of Responses	Unsigned Short Integer	N/A	0 … 65,535 Number of responses (objects) included on this report message
3	Origin Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
4	Origin Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
5	Raster Data Row and Column Data Type	Byte	N/A	Enumeration 0: Byte 1: Reserved 2: Reserved 3: Reserved 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7 – 255: Reserved

Table 4-14. Continued

Field #	Name	Type	Units	Interpretation
6	Raster Grid Update Rows	Varies (See field 5)	Grid Cells	
7	Raster Grid Update Columns	Varies (See field 5)	Grid Cells	
8	Cell Resolution	Float	Meters	
9	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
10	Raster Cell Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 19: RGB (3 Bytes) 10 – 255: Reserved
11	Raster Cell Update 1	Varies (see field 10)	N/A	
12	Raster Cell Update 2	Varies (see field 10)	N/A	
…				
10 + n	Raster Cell Update n	Varies (see field 10)	N/A	
11 + n	Raster Cell n+1	Varies (see field 10)	N/A	

4.2.3.5 Code F404h: Report raster knowledge store bounds

The Code F404h: Report Raster Knowledge Store message format is shown in Table 4-15. This message reports the Raster Knowledge Store bounds as a response to the Query Knowledge Store Bounds message. In this message, the raster knowledge

store returns the two geographic points that represent the extents of the data within a feature class layer or all feature class layers.

Table 4-15. Report raster knowledge store bounds message format

Field #	Name	Type	Units	Interpretation
1	Southwest Point Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2	Southwest Point Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
3	Northeast Point Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
4	Northeast Point Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

4.2.3.6 Code F800h: Raster knowledge store event notification (cell update)

The Code F800h: Raster Knowledge Store Event Notification (Cell Update) message is an event triggered message that is sent in response to the Code F600h: Raster Knowledge Store Event Notification Request message. When bit two of the bit field in that message field two is set, this message is transmitted when the conditions specified in the event notification request are met. The format of this message is identical to that of the Code F402h: Report Raster Knowledge Store Objects (Cell Update) message.

4.2.3.7 Code F801h: Raster knowledge store event notification (grid update)

The Code F801h: Raster Knowledge Store Event Notification (Grid Update) message is an event triggered message that is sent in response to the Code F600h: Raster Knowledge Store Event Notification Request message. When bit two of the bit field in that message field two is clear, this message is transmitted when the conditions specified

in the event notification request are met. The format of this message is identical to that of the Code F403h: Report Raster Knowledge Store Objects (Grid Update) message.

4.2.3.8 Code F802h: Raster knowledge store bounds change event notification

The Code F802h: Raster Knowledge Store Bounds Change Event Notification message is an event triggered message that is sent in response to the Code F601h: Raster Knowledge Store Bounds Change Event Notification Request message. It is transmitted to the requesting component each time the spatial extents of a feature class or feature classes (as specified in the event notification request message) change. The format of this message is identical to that of the Code F404h: Report Raster Knowledge Store Bounds message.

4.2.3.9 Code F405h: Report raster knowledge store data transfer termination

The Code F405h: Report Raster Knowledge Store Data Transfer Termination message notifies other JAUS components that data that were being transferred or were going to be transferred to them has been stopped. This message is sent in response to the Code F005h: Terminate Raster Knowledge Store Data Transfer message. It is also sent whenever data transfer is interrupted due to a change in the component state as discussed in Section 4.2.1.

4.2.4 Vector Knowledge Store Input Message Set

Below are the messages that define the input methods to the vector version of the knowledge store.

Inputs:

- The JAUS core input message set
- Code F020h: Create vector knowledge store objects
- Code F021h: Set vector knowledge store feature class metadata
- Code F022h: Delete vector knowledge store objects
- Code F220h: Query vector knowledge store objects

- Code F221h: Query vector knowledge store feature class metadata
- Code F222h: Query vector knowledge store bounds
- Code F620h: Vector knowledge store event notification request
- Code F621h: Vector knowledge store bounds change event notification request
- Code F023h: Terminate vector knowledge store data transfer

4.2.4.1 Code F020h: Create vector knowledge store objects

The Code F020h: Create Vector Knowledge Store Objects message (Table 4-16) is used to add objects to the Vector Knowledge Store. This message allows multiple vector objects to be created using a single message.

Field 1 of this message is the presence vector (Table 4-17). When multiple objects are created using the same message, the presence vector shall apply to all objects. Because there is a single presence vector associated with this message, all objects within this message shall use this presence vector. Field 2 of this message is the creation message properties. If bit zero is set, then the knowledge store shall return the Code F420h: Report Vector Knowledge Store Object(s) Creation message with the local request identifier specified in field 3. The data type that describes the vector objects' attributes is variable and is specified in fields 4. Field 5 indicates the number of vector objects included in the message. Fields 6 begins the definition of a single vector object. The vector objects is defined by its type (point, line, or polygon), the number of feature classes that it is assigned to, an attribute for each feature class, followed by the global coordinates of the vertices of the object. These fields are repeated for each object created using this message. Again, the presence vector applies to each vector object.

Table 4-16. Create vector knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Byte	N/A	See mapping table below

Table 4-16. Continued

Field #	Name	Type	Units	Interpretation
2	Message Properties	Byte	N/A	Bit Field 0: Request confirmation of object creation 1 – 7: Reserved
3	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component
4	Feature Class Attribute Data Type for Vector Objects	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 9: RGB (3 Bytes) 10 – 255: Reserved
5	Number of Objects	Unsigned Short Integer		0, reserved 1 … 65,535
6	Object *1* Type	Byte	N/A	Enumeration 0: Point 1: Line 2: Polygon 3 – 255: Reserved
7	Object *1* Buffer	Float	Meters	
8	Object *1* Number of Feature Classes	Byte	N/A	
8	Object *1* Feature Class *1*	Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
…	…	…	…	…
	Object *1* Feature Class *m*	Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved

Table 4-16. Continued

Field #	Name	Type	Units	Interpretation
	Object *1* Feature Class Attribute *1*	Varies (see field 4)	Varies with Feature Class	
…	…	…	…	…
	Object *1* Feature Class Attribute *m*	Varies (see field 4)	Varies with Feature Class	
	Object *1* Point *1* Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
	Object *1* Point *1* Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
	…	…	…	…
	Object *1* Point *n* Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
	Object *1* Point *n* Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
	Object *p* Type	Byte	N/A	Enumeration 0: Point 1: Line 2: Polygon 3 – 255: Reserved
	Object *p* Buffer	Float	Meters	
	Object *p* Number of Feature Classes	Byte	N/A	
	Object *p* Feature Class *1*	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 – See Feature Class Table 65,535 – Reserved
…	…	…	…	…

Table 4-16. Continued

Field #	Name	Type	Units	Interpretation
	Object *p* Feature Class *m*	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
	Object *p* Feature Class Attribute *1*	Varies (see field 4)	Varies with Feature Class	
…	…	…	…	…
	Object *p* Feature Class Attribute *m*	Varies (see field 4)	Varies with Feature Class	
	…	…	…	…
	Object *p* Point *r* Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
	Object *p* Point *r* Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

Table 4-17. Presence vector for create vector knowledge store objects message

Vector to Data Field Mapping for Above Command								
Vector Bit	7	6	5	4	3	2	1	0
Data Field	R	R	R	R	R	R	R	7

4.2.4.2 Code F021h: Set vector knowledge store feature class metadata

As described in Section 4.1, metadata are data about data. The Code F021h: Set Vector Knowledge Store Feature Class Metadata (Table 4-18) message allows a user to create, modify, and delete feature class metadata. At the present time the format of these metadata is not specified. It is left to the system designer to develop a convention for doing this. Initially these data are to be used by the human operators. In the future a schema may be defined so as to provide a standard metadata format that may be parsed and the data used by unmanned systems without human intervention.

Table 4-18. Set vector knowledge store feature class metadata message format

Field #	Name	Type	Units	Interpretation
1	Metadata Options	Byte	N/A	Enumeration 0: Append 1: Prepend 2: Overwrite 3 – 254: Reserved 255: Erase all metadata for given feature class
2	Feature Class	Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
3	Number of String Characters	Unsigned Short Integer	N/A	0 … 65,535 This field should be equal to zero only when Field 1 is equal to 255 (Erase All)
4	Feature Class Metadata	String	N/A	Variable length string

4.2.4.3 Code F022h: Delete vector knowledge store objects

The Code F022h: Delete vector knowledge store objects message (Table 4-19) allows the deletion of objects from the vector knowledge store. This message allows multiple vector objects to be deleted using a single message.

Field 1 of this message is the presence vector (Table 4-20). Fields 5 and 6 are the only optional fields in this message. When they are included, they further limit the scope of the deletion. Field 2 of this message is the Local Request Identifier. Field 3 identifies the type of region that will be used to select the objects to delete. The number of vertices for this region is specified in field 4. Field 5 indicates the size of the region buffer to use with this message. Fields 7 begins the definition of vertices of the object deletion region.

Table 4-19. Delete vector knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Byte	N/A	See mapping table below
2	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component
3	Region Type	Byte	N/A	Enumeration 0: Point 1: Line 2: Polygon 3 – 255: Reserved
4	Number of Region Points	Short Integer	N/A	0: Reserved 1 … 65,535
5	Region Buffer	Float	Meters	
6	Feature Class	Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – ALL
7	Deletion Region Point 1 Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
8	Deletion Region Point 1 Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
	…	…	…	…
	…	…	…	…
2n + 5	Deletion Region Point n Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2n + 6	Deletion Region Point n Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

Table 4-20. Presence vector for delete vector knowledge store objects message

Vector to Data Field Mapping for Above Command								
Vector Bit	7	6	5	4	3	2	1	0
Data Field	R	R	R	R	R	R	6	5

4.2.4.4 Code F220h: Query vector knowledge store objects

The Code F220h: Query Vector Knowledge Store Objects message (Table 4-21) allows the access to objects within the vector knowledge store. Field 1 of this message is the presence vector (Table 4-22). Fields 6, 7, and 8 are the only optional fields in this message. When these fields are included, they further limit the scope of the query. Field 2 is a presence vector used to set the query response properties. If bit zero is clear, then the response shall only include the first three fields of the Code F422h: Report Vector Knowledge Store Objects message. Field 3 of this message is the Local Request Identifier. Field 4 identifies the type of region that will be used to limit the query. The number of vertices for this region is specified in field 5. Field 6 indicates the size of the region buffer to use with this message. Fields 8 begins the definition of vertices of the object query region. If this field is not present, the query scope shall be the entire knowledge store.

Table 4-21. Query vector knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short Integer	N/A	See mapping table below
2	Local Request ID	Byte	N/A	Request identifier to be used when returning data to requesting component
3	Query Properties	Byte	N/A	Bit Field 0: Only return number of responses that would be transmitted 1 – 7: Reserved

Table 4-21. Continued

Field #	Name	Type	Units	Interpretation
4	Region Type	Byte	N/A	Enumeration 0: Point 1: Line 2: Polygon 3 – 255: Reserved
5	Number of Region Points	Unsigned Short Integer	N/A	0, reserved 1 … 65,535
6	Region Buffer	Float	Meters	
7	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – All Feature Classes
8	Query Region Point 1 Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
9	Query Region Point 1 Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
2n + 6	Query Region Point n Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2n + 7	Query Region Point n Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

Table 4-22. Presence vector for query vector knowledge store objects message

Vector to Data Field Mapping for Above Command								
Vector Bit	7	6	5	4	3	2	1	0
Data Field	R	R	R	R	R	8	7	6

4.2.4.5 Code F221h: Query vector knowledge store feature class metadata

The Code F221h: Query Vector Knowledge Store Feature Class Metadata message (Table 4-23) should cause the Vector Knowledge Store to reply to the requestor with the

Code F422h: Report Vector Knowledge Store Feature Class Metadata. There is a single field associated with this message. This field specifies the feature class metadata to return in the reply. There is also an option to return metadata for all feature classes present in the queried raster knowledge store.

Table 4-23. Query vector knowledge store feature class metadata message format

Field #	Name	Type	Units	Interpretation
1	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – All

4.2.4.6 Code F222h: Query vector knowledge store bounds

The Code F222h: Query Vector Knowledge Store Bounds message (Table 4-24) is used to request the spatial extents of a single feature class or of all feature classes within a vector knowledge store. The knowledge store should respond with the Code F424h: Report Vector Knowledge Store Bounds message. The bounds are represented by two points the represent the rectangular region that just covers all of the data within the feature class layer or layers.

Table 4-24. Query raster knowledge store bounds message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Request identifier to be used when returning data to requesting component
2	Feature Class	Unsigned Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – All Feature Classes

4.2.4.7 Code F620h: Vector knowledge store event notification request

The Code F620h: Vector Knowledge Store Event Notification Request message is used to establish an event triggered query within the knowledge store. Therefore, this

message is formatted exactly the same as the Code F220h: Query Vector Knowledge Store Objects message. That message should be referenced for the format of this message. Whenever the criteria established in this message are met, the raster knowledge store should transmit the Code F820h: Vector Knowledge Store Event Notification message with the appropriate data attached.

4.2.4.8 Code F621h: Vector knowledge store bounds change event notification request

The Code F621h: Vector Knowledge Store Bounds Change Event Notification Request message is used to establish an event triggered response to notify the requesting component of when the data in a feature class extends past the bounds of the data when the initial request was sent. When the extents of the data change, the raster knowledge store will transmit the Code F821h: Vector Knowledge Store Bounds Change Event Notification message.

4.2.4.9 Code F023h: Terminate vector knowledge store data transfer

This Code F023h: Terminate Vector Knowledge Store Data Transfer message is a command class message that should cause the vector knowledge store to immediately terminate the transfer of all current and outstanding data destined to the requesting component. Upon termination, the raster knowledge store should send the requestor the Code F424h: Report Vector Knowledge Store Data Transfer Termination message.

4.2.5 Vector Knowledge Store Output Message Set

Below are the messages that define the output methods for the vector version of the world model knowledge store.

Outputs:

- The JAUS core output message set
- Code F420h: Report vector knowledge store object(s) creation

- Code F421h: Report vector knowledge store feature class metadata
- Code F422h: Report vector knowledge store objects
- Code F423h: Report vector knowledge store bounds
- Code F820h: Vector knowledge store event notification
- Code F821h: Vector knowledge store bounds change event notification
- Code F424h: Report vector knowledge store data transfer termination

4.2.5.1 Code F420h: Report vector knowledge store object(s) creation

The Code F420h: Report Vector Knowledge Store Object Creation message (Table 4-25) is used to confirm creation of objects in the vector knowledge store. This message is sent only when an object creation message is requested by setting bit zero in the Code F020h: Create Vector Knowledge Store Object message. If this bit is set, this message will be transmitted and the local object identifier (field 1) is set to the value sent with the Code F020h: Create Vector Knowledge Store Raster Object message.

Table 4-25. Report vector knowledge store object(s) creation message format

Field #	Name	Type	Units	Interpretation
1	Local Request ID	Byte	N/A	Local request identifier sent by creating component

4.2.5.2 Code F421h: Report vector knowledge store feature class metadata

The Code F421h: Report Vector Knowledge Store Feature Class Metadata message (Table 4-26) allows access to feature class metadata stored within raster knowledge store. It is transferred in response to the Code F221h: Query Vector Knowledge Store Feature Class Metadata message. If the query message requests all feature classes, a separate message should be sent for each feature class. These metadata are entered using the Code F021h: Set Vector Knowledge Store Feature Class Metadata message.

Table 4-26. Report vector knowledge store feature class metadata message format

Field #	Name	Type	Units	Interpretation
1	Feature Class	Short Integer	N/A	Enumeration 0 … 65,535 See Feature Class Table

Table 4-26. Continued

Field #	Name	Type	Units	Interpretation
2	Number of String Characters	Unsigned Short Integer	N/A	0 … 65,535
3	Feature Class Metadata	String	N/A	Variable length string

4.2.5.3 Code F422h: Report vector knowledge store objects

The Code F422h: Report Vector Knowledge Store Objects message (Table 4-27) is sent in direct response to a Code F220h: Query Vector Knowledge Store Objects message.

Field 1 is a presence vector that informs the receiving component as to whether or not data are included with the message. If bit zero is set, then data should be expected after message field 3. Field 2 of this message is Local Request Identifier sent with the query that initiated this report message. Field 3 indicates the number of vector objects included in the message. The data type that describes the vector objects' attributes is variable and is specified in fields 4. Fields 5 begins the definition of a single vector object. The vector objects is defined by its type (point, line, or polygon), the number of feature classes that it is assigned to, an attribute for each feature class, followed by the global coordinates of the vertices of the object. These fields are repeated for each object reported in this message.

Table 4-27. Report vector knowledge store objects message format

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Byte	N/A	Bit Field Bit 0: If data are present after field 3, this bit should be set. This is based on the parameters in the received Query Vector Knowledge Store Objects Message. Bits 1-7: Reserved

Table 4-27. Continued

Field #	Name	Type	Units	Interpretation
2	Local Request ID	Byte	N/A	Request identifier to be used when returning confirmation to requesting component
3	Number of Objects	Unsigned Short Integer		0, reserved 1 … 65,535
4	Object 1 Type	Byte	N/A	Enumeration 0: Point 1: Line 2: Polygon 3 – 255: Reserved
5	Object 1 Buffer	Float	Meters	
6	Object 1 Feature Class	Short Integer	N/A	Enumeration 0 … 65,534 - See Feature Class Table 65,535 – Reserved
7	Object 1 Feature Class Attribute Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 9: RGB (3 Bytes) 10 – 255: Reserved
8	Object 1 Feature Class Attribute	Varies (see field 4)	Varies with Feature Class	
9	Number of Points for Object 1	Unsigned Short Integer		0, reserved 1 … 65,535
10	Object 1 Point 1 Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90

Table 4-27. Continued

Field #	Name	Type	Units	Interpretation
11	Object 1 Point 1 Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
…	…	…	…	…
…	…	…	…	…
2m + 8	Object 1 Point m Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2m + 9	Object 1 Point m Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
2m + 10	Object n Type	Byte	N/A	Enumeration 0: Point 1: Line 2: Polygon 3 – 255: Reserved
2m + 11	Object n Buffer	Float	Meters	
2m + 12	Object n Feature Class	Short Integer	N/A	Enumeration 0 … 65,535 See Feature Class Table
2m + 13	Object n Feature Class Attribute Data Type	Byte	N/A	Enumeration 0: Byte 1: Short Integer 2: Integer 3: Long Integer 4: Unsigned Short Integer 5: Unsigned Integer 6: Unsigned Long Integer 7: Float 8: Long Float 9: RGB (3 Bytes) 10 – 255: Reserved
2m + 14	Object n Feature Class Attribute	Varies (see field 4)	Varies with Feature Class	

Table 4-27. Continued

Field #	Name	Type	Units	Interpretation
2m + 15	Number of Points for Object n	Unsigned Short Integer		0, reserved 1 … 65,535
2m + 16	Object n Point 1 Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2m + 17	Object n Point 1 Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180
…	…	…	…	…
…	…	…	…	…
	Object n Point k Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
	Object n Point k Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

4.2.5.4 Code F423h: Report vector knowledge store bounds

The Code F423h: Report Vector Knowledge Store Bounds message format is shown in Table 4-28. This message reports the bounds as a response to the Query Vector Knowledge Store Bounds message. In this message, the knowledge store returns the two geographic points that represent the extents of the data within a feature class layer or all feature class layers.

Table 4-28. Report vector knowledge store bounds message format

Field #	Name	Type	Units	Interpretation
1	Southwest Point Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
2	Southwest Point Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

Table 4-28. Continued

Field #	Name	Type	Units	Interpretation
3	Northeast Point Latitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -90 Upper Limit = 90
4	Northeast Point Longitude (WGS84)	Integer	Degrees	Scaled Integer Lower Limit = -180 Upper Limit = 180

4.2.5.5 Code F820h: Vector knowledge store event notification

The Code F820h: Vector Knowledge Store Event Notification message is an event triggered message that is sent in response to the Code F620h: Vector Knowledge Store Event Notification Request message. The format of this message is identical to that of the Code F422h: Report Vector Knowledge Store Objects message.

4.2.5.6 Code F821h: Vector knowledge store bounds change event notification

The Code F821h: Vector Knowledge Store Bounds Change Event Notification message is an event triggered message that is sent in response to the Code F621h: Vector Knowledge Store Bounds Change Event Notification Request message. It is transmitted to the requesting component each time the spatial extents of a feature class or feature classes (as specified in the event notification request message) change. The format of this message is identical to that of the Code F423h: Report Vector Knowledge Store Bounds message.

4.2.5.7 Code F424h: Report vector knowledge store data transfer termination

The Code F424h: Report Vector Knowledge Store Data Transfer Termination message notifies other JAUS components that data that were being transferred or were going to be transferred to them has been stopped. This message is sent in response to the Code F025h: Terminate Vector Knowledge Store Data Transfer message. It is also sent

whenever data transfer is interrupted due to a change in the component state as discussed in Section 4.2.1.

The messages presented in the preceding sections present a solution to world modeling within the context of the Joint Architecture for Unmanned Systems (JAUS). The defined messages allow the raster and vector versions of the knowledge store to receive and transmit formatted geospatial data. Because the underlying geometry of most geospatial data is based on raster or vector objects, the JAUS World Model components are able to support most types of geospatial data including those presented in Chapter 2.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Our study focused on standardization of an interface between unmanned systems. Specifically, it focused on standardizing the interface between different types of geospatial data stores within the JAUS framework. In responding to the research problem stated in Chapter 1, a review of relevant literature was done. Next a system of distributed modular sensor processing units was developed. This system of modular sensors is called the Smart Sensor Architecture. Its development laid the foundation for the development of the world model message set presented in Chapter 4. This generic JAUS message set was developed to allow transfer of basic forms of both raster and vector formatted geospatial data.

The interfaces to the world model knowledge stores as introduced in Chapter 4 present a standardized method for communicating geospatial data. The application possibilities of these messages are endless. While it is useful for single unmanned systems for mapping its environment, it is particularly useful for collaborative robotics tasks. For example, an unmanned system with a powerful sensor suite could be used to map an area for obstacles. That map, or world model, could be shared with other unmanned systems to allow them to traverse a region with minimal or no sensors; essentially sharing the resources of another unmanned system.

It should be clear that our study in not a final solution to the question of how to share geospatial data between multiple unmanned systems. It is a first step in a long

process of defining a standard for the growing JAUS community. The consequences of the work presented herein could very well be far reaching indeed. Just imagine a class of unmanned systems with a shared language – a standard method of communicating with other unmanned systems. Unmanned systems that are able to, despite the fact that they were developed by different vendors, interoperate with minimal effort. With JAUS, this is becoming a reality. Our study is a significant contribution to the JAUS Working Group's effort to develop the next generation of intelligent JAUS systems.

5.2 Future Work

Since the problem addressed by our study is open-ended, this work must and most certainly will continue on. There is no single best way to model or share geospatial data. What is important is that all concerned parties reach consensus on how to do this. Therefore the results of our study provides a base upon which the JAUS community can build. As with any new component added to the JAUS Reference Architecture, the component messages presented herein must be vetted by all interested parties within the JAUS community. Only after approval by the JAUS Reference Architecture Committee and the JAUS Working Group as a whole will it be adopted.

Chapter 4 presents two separate methods for modeling and sharing both raster and vector geospatial data. What was not address is how to bridge the two modeling methods. Converting vector data to raster format is trivial. This simply requires the projection of the points along the edges of the vector object into a grid. The grid should be of high enough resolution to accurately represent the vector objects. The more difficult side of this bridge is the conversion from raster to vector. Because transfer and storage of raster data is very expensive, this is of particular importance when it comes sharing data between unmanned systems. Raster data requires a large amount of

bandwidth during transmission. For example, even when transmitting a large amount of raster formatted geospatial data areas have similar values, each cell must still be transmitted. A possible approach to the raster to vector conversion problem is the use of the Level Set Method developed by Sethian in [27]. This approach will be explored as a possible solution to this problem.

Another future extension for the standard presented in Chapter 4 is the addition of support for more projected coordinate systems. For the sake of simplicity, this message set requires all global coordinates to be based on the World Geodetic System 1984 (WGS) and the projected coordinate system to be based on the Universal Transverse Mercator projection. As discussed in Chapter 2, there are benefits to use of different types of projected coordinate systems. UTM is a good general purpose transformation, but system developers may want or need to use another projection that preserves features that are most important to them. The standard should grow to not only support, but allow systems to distinguish and convert of data from different projections.

One of the most often discussed issue with JAUS is that is not a very flexible or extensible architecture. As this document attempts a first step at bridging the GIS and Unmanned Systems communities, it is expected that the World Model subcommittee of JAUS will make an effort to bring in members of the GIS community. There is a wealth of knowledge and contributions to be gained from both the unmanned systems and GIS communities. This is perhaps the most important continuation plan for this work.

REFERENCES

[1] American National Standards Institute, *Spatial Data Transfer Standard (SDTS) Part 1: Logical Specification*. Washington, D.C.: American National Standards Institute, 1997.

[2] American National Standards Institute, *Spatial Data Transfer Standard (SDTS) Part 2: Spatial Features*. Washington, D.C.: American National Standards Institute, 1997.

[3] American National Standards Institute, *Spatial Data Transfer Standard (SDTS) Part 3: ISO 8211 Encoding*. Washington, D.C.: American National Standards Institute, 1997.

[4] American National Standards Institute, *Spatial Data Transfer Standard (SDTS) Part 4: Topological Vector Profile*. Washington, D.C.: American National Standards Institute, 1997.

[5] American National Standards Institute, *Spatial Data Transfer Standard (SDTS) Part 6: Point Profile*. Washington, D.C.: American National Standards Institute, 1998.

[6] American National Standards Institute, *Spatial Data Transfer Standard (SDTS) Part 5: Raster Profile*. Washington, D.C.: American National Standards Institute, 1999.

[7] P. Bolstad, *GIS Fundamentals: A First Text on Geographic Information Systems*. White Bear Lake, MN: Eider Press, 2002.

[8] J. Borenstein, "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots," *IEEE Journal of Robotics and Automation*, vol. 7, no. 3, pp. 278-288, 1991.

[9] K. Brown, *Datums and Projections: A Brief Guide*, United States Geological Survey Center for Biological Informatics. Reston, VA: United States Geological Survey, 1999.

[10] M. Crosetto, B. Crippa, "Optical and Radar Data Fusion for DEM Generation," *IAPRS GIS Between Vision and Applications*, vol. 32, no. 4, pp. 128-134, Stuttgart, Germany, 1998..

[11] S. Cox, P. Daisey, R. Lake, C. Portele, A. Whiteside, *OpenGIS Geography Markup Language Implementation Specification*, OpenGIS Consortium, Inc., 2004.

[12] G. Dudeck, M. Jenkin, *Computational Principles of Mobile Robotics*. New York, NY: Cambridge University Press, 2002.

[13] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22, no. 6, pp. 46-57, 1989.

[14] ESRI, *Understanding Map Projections: GIS by ESRI*, ESRI. Redlands, CA, 2004.

[15] T. Hong, M. Abrams, T. Chang, M. Shneier, "An Intelligent World Model for Autonomous Off-Road Driving." Gaithersburg, MD: NIST Intelligent Systems Division, 2001.

[16] JAUS Working Group, *Joint Architecture for Unmanned Systems (JAUS) Version 3.1, Volume 2*, The Joint Architecture for Unmanned Systems. http://www.jauswg.org, April 2004.

[17] R. Lake, *Enabling the Geo-spatial Web*, Galdos Systems Inc. Vancouver, CA, 2001.

[18] A. Meystel, J. Albus, *Intelligent Systems: Architecture, Design, and Control*. New York, NY: John Wiley & Sons, 2002.

[19] R. Murphy, *Introduction to AI Robotics*. Cambridge, MA: The MIT Press, 2000.

[20] D. Murray, J. Little, "Using Real-Time Stereo Vision for Mobile Robot Navigation," *Autonomous Robots*, vol. 8, no. 2, pp. 151-171, 2000.

[21] O. R. Musin, "Towards 3/4-D GIS." Moscow, Russia: Moscow State University, Department of Cartography and Geoinformatics, 1998.

[22] D. Novick, "Implementation of a Sensor Fusion-Based Object Detection Component for an Autonomous Outdoor Vehicle," Doctor of Philosophy Dissertation. Department of Mechanical Engineering: University of Florida, 2002.

[23] A. Pasha, "Path Planning for Nonholonomic Vehicles and Its Application to Radiation Environments," Master of Science Thesis. Department of Mechanical Engineering: University of Florida, 2003.

[24] J. Postel, *User Datagram Protocol, Request for Comments 768*, USC Information Sciences Institute. Marina del Ray, CA, August 1980.

[25] P. Rigaux, M. Scholl, A. Voisard, *Spatial Databases with Application to GIS*. San Francisco, CA: Morgan Kaufmann, 2002.

[26] J. Rosenblatt, "DAMN: A Distributed Architecture for Mobile Navigation," Doctor of Philosophy Dissertation. Robotics Institute: Carnegie Mellon University, 1997.

[27] J. A. Sethian, Level *Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge, UK: Cambridge University Press, 1999.

[28] Sick, Inc., *LMS200/LMS211/LMS220/LMS221/LMS291 Laser Measurement Systems Technical Description*, Sick, Inc. Minneapolis, MN, 2003.

[29] United States Geological Survey, *U.S. GeoData Digital Line Graphs Fact Sheet*. Reston, VA: United States Geological Survey, 1996.

[30] United State Geological Survey, National Mapping Division, *National Mapping Program Technical Instructions: Part 1 General Standards for Digital Elevation Models*. Reston, VA: United States Geological Survey, 1997.

[31] United States Geological Survey, *Digital Raster Graphics Fact Sheet*. Reston, VA: United States Geological Survey, 1999.

[32] United States Geological Survey, *Spatial Data Transfer Standard (SDTS) Fact Sheet*. Reston, VA: United States Geological Survey, 1999.

[33] United States Geological Survey, *U.S. GeoData Digital Elevation Models Fact Sheet*. Reston, VA: United States Geological Survey, 2000.

[34] United States Geological Survey, *U.S. GeoData Digital Orthophoto Quadrangles Fact Sheet*. Reston, VA: United States Geological Survey, 2001.

[35] Videre Design, *STH-MD1/-C Stereo Head User's Manual*, Videre Design. Menlo Park, CA, 2001.

[36] Wikipedia Contributors, "Sea Level," http://en.wikipedia.org/wiki/Sea_level. Wikipedia: The Free Encyclopedia. (December 15, 2004).

BIOGRAPHICAL SKETCH

Carl P. Evans III was born on May 15, 1978, in Salisbury, Maryland. As a child and teenager, he was very inquisitive and had an appreciation for science and technology (particularly robotics, albeit toy robotics). He was first exposed to programming when his grandmother purchased a Commodore 64 computer for him. The computer came with very little software so, out of necessity, he wrote his own. Building on the vast experience he gained writing Commodore 64 code, Mr. Evans took 2 years of programming in high school. This work culminated in Carl's Operating System (COS) a PASCAL-developed graphical user interface that ran on top of Microsoft DOS.

Motivated by his desire to develop his own version of Short Circuit's Johnny Five, Mr. Evans decided to pursue a degree in engineering. Not knowing which field of engineering (electrical or mechanical) he wanted to study, in 1996 Mr. Evans enrolled in the unique Electromechanical Engineering (BELM) program at Wentworth Institute of Technology in Boston, MA (the only ABET-accredited interdisciplinary Electromechanical Engineering program in the United States). A campus leader, Mr. Evans served as chapter president of ASME and IEEE, Institute President's Host, and Resident Assistant. In 2001, Mr. Evans graduated, cum laude, with his B.S. degree and gave the student commencement address to his fellow graduates.

As part of the WIT Electromechanical Engineering program's graduation requirements, Mr. Evans worked two co-op semesters, to supplement his studies with practical experience. He earned the honor of being an engineer assistant at Foster-Miller,

Inc., in Waltham, MA. At FMI, the range of Mr. Evans' work was as broad as his education. As a student, he earned the level of respect and responsibility granted to Staff Engineers. Upon graduation, Mr. Evans was granted a double-promotion to the position of Design Engineer, bypassing the position of Staff Engineer. To this day, his contributions live on at Foster-Miller and are helping in the current "war" on terrorism.

In January of 2002, Mr. Evans joined Dr. Carl Crane at the University of Florida's Center for Intelligent Machines and Robotics. While at the University of Florida, he was a member of the Unmanned Systems research team, funded by the Air Force Research Laboratory (Panama City, FL). He was a member of the CIMAR DARPA Grand Challenge team and led the perception team. In August of 2005, he graduated with his Master of Science degree.

A Commonwealth of Massachusetts' Engineer Intern since 2001, Mr. Evans plans to sit for the Professional Engineers' (PE) exam with a concentration in Electrical Engineering. Mr. Evans is a first-generation college graduate. He owes all that he has achieved to his loving grandparents, Lottie and James Patterson. His future educational plans include attending law school with an ultimate goal of one day becoming a politician. In June 2004, Mr. Evans joined Applied Perception, Inc. (API) in Wexford, PA as a Senior Engineer. At API he will continue his involvement with JAUS and the development of perception systems, world modeling methods, and collaborative technologies for unmanned systems.