THE JOINT ARCHITECTURE FOR UNMANNED SYSTEMS

Reference Architecture Specification

Volume II, Part 1

Architecture Framework

Version 3.3

June 27, 2007

TABLE OF CONTENTS

Title **Page**

TABLE OF FIGURES

1 DOCUMENT ORGANIZATION

1.1 Overview

The Joint Architecture for Unmanned Systems (JAUS) is the architecture defined for use in the research, development and acquisition of Unmanned Systems. The overall scope of JAUS is defined in three separate volumes. Volume I is the JAUS Domain Model, Volume II is the JAUS Reference Architecture, and Volume III is the JAUS Document Control Plan. The Reference Architecture Specification is comprised of three parts: Architecture Framework, Message Definition, and Message Set.

1.1.1 Volume I: JAUS Domain Model

The Domain Model (DM) is the model of both known and potential operational requirements that must be supported by the Unmanned System. Two different organizations are defined within the Domain Model: the Acquirer and the Developer. The Acquirer is the organization that determines the operational requirements of the Unmanned System. The Developer is the organization responsible for building the Unmanned System, as defined by the Acquirer.

It is the intent of JAUS that the DM be used to model all requirements, known and anticipated, of the Unmanned Systems. The DM is a tool used by the Developer to understand the requirements of the Acquirer. The Domain Model document is written in the language of the Acquirer.

1.1.2 Volume II: JAUS Reference Architecture

The Reference Architecture (RA) is the technical specification that the Developer shall use to implement JAUS on the Unmanned System. It is also the document that will be used by the Acquirer to assess technical compliance. Therefore, the Reference Architecture document is written in the language of scientists and engineers.

The main purpose of the RA is to describe all functions and messages that shall be employed to design new components. In addition, the RA describes all messages currently defined and the rules that govern messaging. Messaging is the only accepted method used to communicate between components.

In terms of chronological development, capabilities described in the DM will always precede those that appear in the RA. This is because the RA documents only those requirements in the DM that have already gone through a technical evaluation and acceptance process. This process is performed by an organization through the development of models and simulations or through the development of a prototype. In any event, all components defined in the RA can be traced back to a DM capability, whereas the reverse is not always true.

1.1.2.1 Reference Architecture Part 1: Architecture Framework

Part 1, Architecture Framework, of the Reference Architecture specification provides a description of the structure of JAUS based systems. This document serves as the primary mapping of Domain Model requirements to the JAUS message set. Although message specifics are not stated in the Architecture Framework, guidance for the use of messages to achieve Domain Model specified capabilities is provided in the component discussions.

1.1.2.2 Reference Architecture Part 2: Message Definition

Part 2, Message Definition, of the Reference Architecture specification specifies the JAUS specific protocol for transmission of JAUS messages. This document specifies the JAUS message header and the types of JAUS messages. The specification focuses on the rules for messaging as opposed to the domain specific semantics contained in the message set itself.

1.1.2.3 Reference Architecture Part 3: Message Set

Part 3, Message Set, of the Reference Architecture specification specifies the domain specific messages and their exact content. The Message Set is intended to for use within unmanned systems either with or without the preceding two parts of the Reference Architecture Specification.

1.1.3 Volume III: JAUS Document Control Plan

The Document Control Plan (DCP) defines the process used to identify and track requested changes to accepted JAUS documentation. The process for controlling changes to the JAUS documents is based on commercial standards body processes. The DCP defines the roles by committees and its members in the change process as well as providing a historical record of changes and their rationales.

1.2 Applicable Documents

1.2.1 Joint Technical Architecture

The Joint Technical Architecture (JTA) is a set of standards that supports interoperability and data interchange among military systems. The reference for the JTA document is:

Department of Defense Joint Technical Architecture (JTA), Version 3.1, March 2000.

1.2.2 International System of Units

JAUS mandates the use of the International System of Units as specified in NIST Special Publication 330, 1991 Edition, "The International System of Units (SI)," Barry N. Taylor, National Institute of Standards and Technology. The URL for this publication is

http://physics.nist.gov/Document/sp330.pdf

2 INTRODUCTION

2.1 JAUS Purpose

The purpose of JAUS is to support the acquisition of Unmanned Systems by providing a mechanism for reducing system life-cycle costs. This is accomplished by providing a framework for technology reuse/insertion. JAUS defines a set of reusable ***"components"*** and their interfaces. These reusable components not only reduce the maintenance costs of a system, but also dramatically reduce the development costs of any follow-on system(s). Reuse allows a component developed for one Unmanned System to be readily ported to another Unmanned System or to be easily replaced when technological advances.

Technology insertion is achievable when the architecture is designed to be both modular and scaleable. Components that are deemed necessary for the mission of the Unmanned System may be inserted simply by bundling.

JAUS defines components for all classifications of Unmanned Systems from remote control toward autonomous, regardless of application. As a particular system evolves, the architecture is already in place to support more advanced capabilities.

2.2 Technical Constraints

Technical constraints are imposed on JAUS to ensure that the architecture is applicable to the entire domain of Unmanned Systems - now and in the future. The constraints are:

- Platform Independence
- Mission Isolation
- Computer Hardware Independence
- Technology Independence

2.2.1 Platform Independence

Analysis has shown that Unmanned Systems will be based on a variety of mission requirements, including but not limited to surveillance, reconnaissance, force protection, combat, security, and emergency response. In order for JAUS compliant components to be interoperable, no assumptions about the underlying vehicle are made.

2.2.2 Mission Isolation

The purpose of Unmanned Systems is to gather information, alter the state of the environment, or both. The set of these sensing and effecting tasks are called missions. The purpose of this isolation is the anticipation that many Developers will build their systems to support a variety of missions, possibly with removable mission modules.

2.2.3 Computer Hardware Independence

The growth in the computer industry has been enormous over the past 20 years and there are no indications that the growth will slow down. Future Unmanned Systems must be able to capitalize on commercial advancements in computing and sensor technology. The issue is two-fold:

- First, a single Unmanned System must be able to evolve over its product life cycle to accommodate new missions and greater degrees of autonomy. An architecture that imposes a specific hardware implementation reduces the opportunity to take advantage of future technical advancements.
- Second, each Unmanned System Developer should have the flexibility to design a computer hardware architecture that meets that particular system's requirements. Computer hardware that is appropriate for one Unmanned System may not be appropriate for another.

JAUS must maintain computer hardware independence in order to be applicable to all Unmanned Systems.

2.2.4 Technology Independence

The final technical constraint is technology independence. This constraint is similar to computer hardware independence but focuses more on the technical approach rather than the computer hardware. For example, an Unmanned System may use a visual system for numerous purposes. A visual system may provide feedback to the Unmanned Systems operator to support tele-operated driving. A visual system may also perform obstacle detection, target detection, target identification, target tracking, landmark recognition, and so forth. The technology of edge detection is often used in visual systems to perform these functions. However, there may be other techniques besides visual systems and edge detection that could perform obstacle detection. Active Laser Detection and Ranging (LADAR) is one possibility. The point is that there may be multiple technical solutions to a problem. An architecture that is built around a particular technology solution may eliminate a superior alternative.

2.3 JAUS Nomenclature

In the language of JAUS, a number of terms are used to delineate position within the overall hierarchy of the system and must therefore, be well understood. These terms describe the different levels of the architecture and define the required internal hierarchical sub-grouping.

• System	A system is a logical grouping of subsystems. The system definition provides a functional grouping for the full robotic or unmanned capability. This grouping includes all human interface subsystems and unmanned subsystems common with robotic and unmanned applications.
• Subsystem	A subsystem performs one or more unmanned system functions as a single localized entity within the framework of the System. A subsystem shall provide one or more communication command and control capabilities. A mobile subsystem shall execute mobility commands as a single unit and retain a defined center of gravity relative to all articulations and payloads.
• Node	A JAUS Node defines a distinct processing capability within a subsystem. A node retains a set of coherent functions and shall provide a node manager component to manage the flows and controls of JAUS message traffic.
• Component	A component provides a unique functional capability for the unmanned system. JAUS messages are defined with respect to these capabilities so that context in command and control is provided. A JAUS component resides wholly within a JAUS Node.
• Instance	Duplication and redundancy of JAUS Components are provided by Component Instances. All Components are uniquely addressable using Subsystem, Node, Component and Instance Identifiers.

- Message — A JAUS message is comprised of the message header and associated data fields as defined within this document.

2.4 Definitions

Message definitions within JAUS use the following terms to define various fields. The following definitions provide clarity and guidance for the understanding of the terms.

RMS:	The root-mean-square (RMS), often used as a synonym for the standard deviation of a variant ***X***, is the square root of the mean squared value of ***x*** or $\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}$ for a discrete distribution, and $\sqrt{\frac{\int P(x)x^2dx}{\int P(x)dx}}$ for a continuous distribution. Physical scientists often use the term root-mean-square as a synonym for standard deviation when they refer to the square root of the mean squared deviation of a signal from a given baseline or fit. RMS is a mean value and not an instantaneous measurement.
Position RMS:	The position RMS provides a means of determining the error associated with a reported position. This value is measured over time and therefore not representative of the error any particular position value. It provides a statistical measure of the magnitude of the possible error in 3 dimensions. This value is reported in meters.
Attitude RMS:	The attitude RMS provides a means of determining the error associated with a reported attitude. This value is measured over time and therefore not representative of the error any particular attitude value. It provides a statistical measure of the magnitude of the possible error.
Velocity RMS:	The velocity RMS provides a means of determining the error associated with a reported speed. This value is measured over time and representative of the error associated with the manner in which speed is recorded and reported. It provides a statistical measure of the magnitude of the possible error.
Presence Vector:	JAUS provides for variable length messages. These messages either have repeating data or have a mixture of required and optional data fields. The Presence Vector is used to indicate which of the optional data fields are included. Presence Vector bits are set to one (1) to indicate the optional

field is present. The bit is set to zero (0) to indicate the field is not present. Reserved bits shall be set to zero (0).

Data Validity: The Data Validity field in JAUS messages indicates if the originator of the message had sufficient information to populate the fields indicated accurately. This field is independent of the Presence Vector. This field typically uses an identical mapping as the Presence Vector when a Presence Vector is defined. A set bit ('1') indicates data validity where a clear bit ('0') indicates a possibly invalid field.

3 SYSTEM TOPOLOGY

Section 2.3 introduced several JAUS hierarchical terms: System, Subsystem, Node, Component and Instance. A graphical representation of how each element fits into the JAUS system topology is shown below in Figure 3-1.

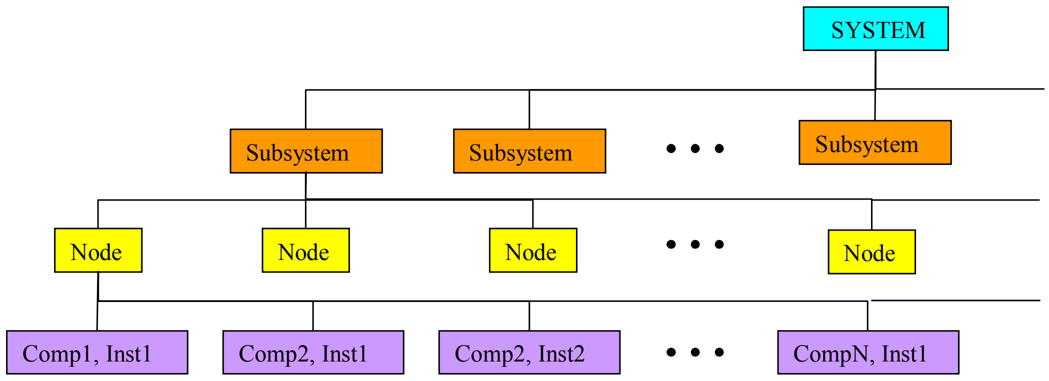


Figure 3-1 - JAUS System Topology

Another way to show how JAUS elements can be assembled to create a JAUS system is shown in Figure 3-2:

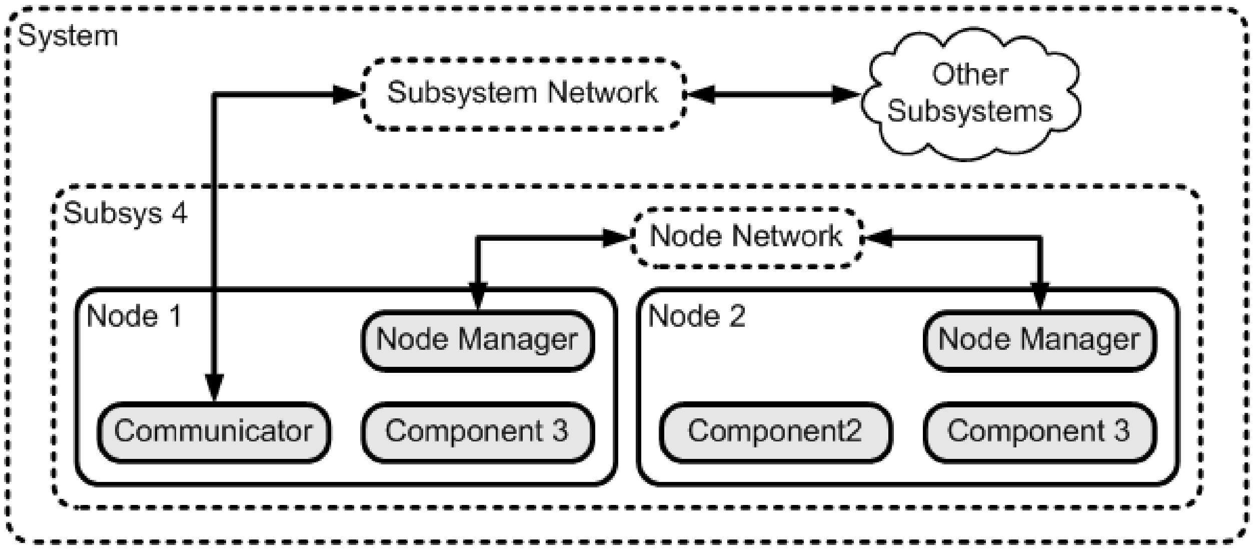


Figure 3-2 - JAUS System Communications Topology

3.1 The System Level

A System is a logical grouping of one or more subsystems. A system would normally be grouped in such a manner as to gain some cooperative advantage between the constituent subsystems. An example system might group the following subsystems:

- An operator control unit (OCU)
- One or more signal repeater subsystems
- One or more vehicle subsystems working towards a common goal

3.2 The Subsystem Level

A subsystem is an independent and distinct unit. The unit may consist of any number of computer nodes and software components necessary to support its functional requirements. The overall collection of nodes and components in the subsystem, are the parts of the unit that shall comply with the JAUS RA. Examples of subsystems include:

- An operator control unit,
- An unmanned ground mobility platform (bulldozer, tractor, ground shuttle, etc.),
- An unmanned air vehicle (fixed wing, rotary wing, missile, etc.),
- An unmanned undersea vehicle (swimmer, crawler, mobile mine, etc.),
- An unmanned surface vehicle (boat, ship, etc.),
- An unmanned stationary sensor (seismic anomaly detector, data repeater, etc.),
- A surveillance system (thermal imager, video camera, etc.),

3.3 The Node Level

A node is composed of all the hardware and software assets necessary to support a well-defined computing capability within the subsystem. A node is often treated as a ***"black-box"*** containing all the hardware and software necessary to provide a complete service. Examples of nodes within a subsystem might be:

- A mobility controller
- A world modeling computer
- A vision processor
- A master controller
- A payload controller

From a hardware perspective, a node consists of the computing and interface resources required to support its software components and their interaction with each other, the rest of the subsystem, and any devices they operate. A node may be implemented on one or more interconnected computing elements as determined by the system engineer and as required to provide its function. A node connects to the devices it operates using the hardware and software techniques required by the device itself and device's own level of computing capabilities, if any. Separate nodes may be interconnected using a wide variety of hardware and software techniques such as serial networks (Ethernet, 1553, CAN, RS232, etc.) or shared bus (e.g. VME, PCI, etc.) or common memory areas. A node is often identified by its existence as a separate entity on a network that interconnects the nodes in a subsystem (e.g., it has a network address).

From a software perspective, a node consists of the software components that run on the node and it is therefore, highly scaleable. A single node could be loaded with all the software necessary to control the whole subsystem, or many nodes could be installed so as to break the job down into more manageable pieces. Detailed configuration is the task of the systems engineer.

3.4 The Component/Instance Level

A component is the lowest level of decomposition in the JAUS hierarchy. A component is a cohesive software unit that provides a well-defined service or set of services. Generally speaking, a component is an executable task or process. All of the components currently supported by JAUS are specified in **Chapter 5**.

Component redundancy is supported, by allowing multiple instances of a component to run on the same node. The address of each instance of a component serves to distinguish between these instances. While instances of the same component share an identical interface they may vary in implementation.

3.5 Topology Simplified

Now that the five most elementary JAUS terms have been defined, a simple and direct statement can be made that gets to the heart of how they work together —

- *A subsystem is composed of component software, distributed across one or more nodes.*

From a systems engineering standpoint, this last statement is significant in that it implies, yet does not specify. Since configurations can be virtually unlimited, node or component interface boundaries are not dictated. This is one of the key aspects of JAUS flexibility.

3.6 Configuration

One of the principal goals of JAUS is to provide a level of interoperability between intelligent systems that has been missing in the past. Towards this end, JAUS defines functional components with supporting messages, but does not impose regulations on the systems engineer that govern configuration.

JAUS does have one absolute, unwavering requirement that can effect configuration. It is:

➢ *To achieve the desired level of interoperability between intelligent computing entities, all messages that pass between JAUS defined components (over networks or via airwaves), shall be JAUS compatible messages.*

The statement above introduces JAUS messages, which are explained in detail in the Reference Architecture Specification Parts 2 and 3. For now, just consider a message as a means of exchanging information. A close examination of this statement reveals that ***"messages"*** pass between ***"components"*** via some communications medium. It is important to understand that this restriction only applies to JAUS messages.

Figure 3-3 presents an example configuration that is intended to explain further. The diagram depicts two JAUS defined components in blue boxes, a user defined component in green, a dedicated hardware device in yellow and a data storage area in brown.

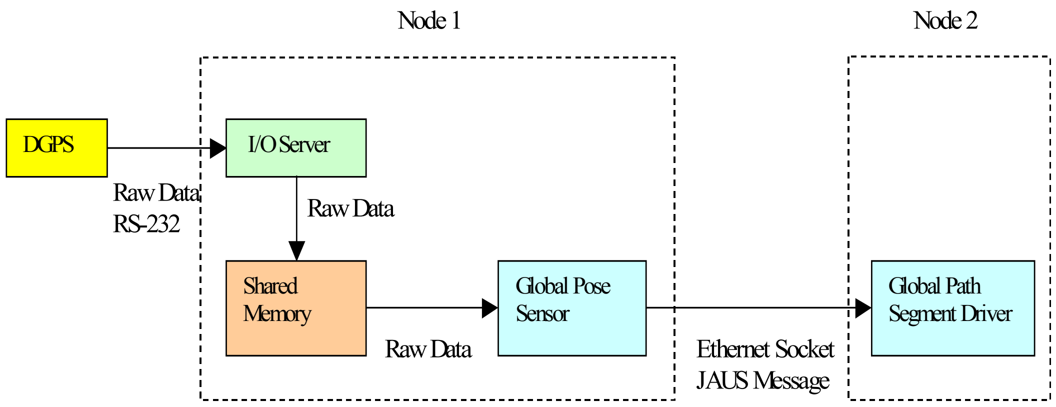


Figure 3-3 - Example Configuration #1

1. Node-1 is configured as a Positioning unit and is dominated by the Global Pose Sensor component. It also supports an I/O server task and a shared memory area. The Differential Global Positioning System (DGPS) is configured as a dedicated device within the node. The DGPS is setup in streaming mode so that positioning data is sent out over RS-232 at some regular interval.
2. Node-2 is configured as a Path Driving unit and is dominated by the Global Path Segment Driver component. It relies on Global Pose Sensor messages to perform its path-tracking task.
3. The I/O server's job is to listen to the RS-232 COM port and send each received data stream to shared memory after it arrives.
4. The Global Pose Sensor component task reads shared memory whenever necessary to get the latest DGPS information.

   Important Note: So far all of the data is in raw (non-JAUS) format because typically dedicated sensors, like the DGPS, do not support JAUS messages.
5. When the Global Pose Sensor component completes its position calculations, it sends a JAUS formatted Global Pose Sensor message to Node-2. This sequence loops continually.

Notice that although the streaming data coming from the DGPS to the I/O server and on to shared memory is not in JAUS format, that the interoperability rule remains unbroken. Only when data is sent between JAUS defined components, must it be formatted into a JAUS compatible message.

Two simpler variations on the example configuration are depicted by Figure 3-4 and Figure 3-5**.** Numerous other configurations are also possible.

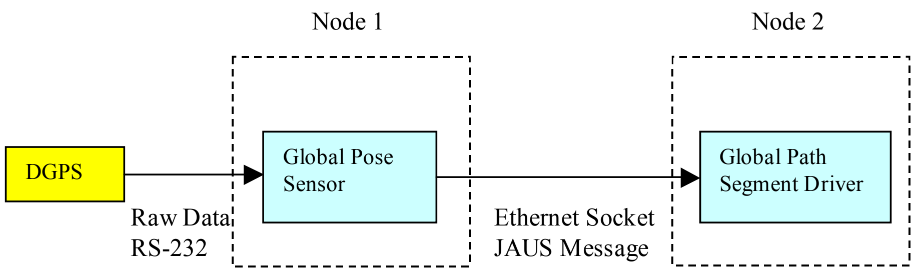


Figure 3-4 - Example Configuration #2

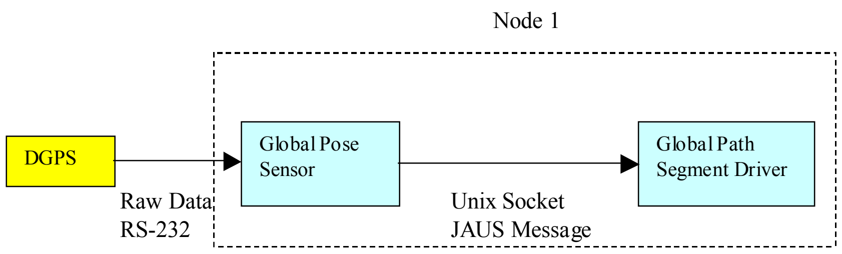


Figure 3-5 - Example Configuration #3

Each of the three configurations will work equally well. Which one is used is entirely the decision of the systems engineer. Configuration is an engineering function and is not mandated by JAUS. As long as the configuration employed does not pose any impediment to JAUS interoperability, the systems engineer is free to employ the most efficient design possible.

3.6.1 A Priori Configuration

Previous versions of JAUS embraced a philosophy whereby system, subsystem, node and component configuration parameters are known or determined before they must be implemented. This so-called "a priori" knowledge imposes certain considerations on the system engineer that must be taken into account during design. Some of the more important matters are as follows:

- Each subsystem and node must be assigned a unique node number manually
- All serial communications must agree on common line control parameters
- Each Ethernet socket and the server process IP address must be assigned

- All user defined tasks must be defined and assigned a user defined component ID
- In multi-vendor projects, vendors must agree upon where node boundaries lie so as not to encounter any functional overlap or deficiency

3.6.2 Dynamic Configuration

This version of JAUS introduces dynamic configuration. See Part 3 for the dynamic configuration messages.

4 COMPONENT SPECIFICATION

4.1 Component Overview

Components represent the lowest level of decomposition within the reference architecture. A component performs a set of operations that have a logical grouping within the context of the domain. Interaction with components is through messages as defined herein. The component's messages constitute its interface. A subsystem is composed of components that interoperate using the defined message set to complete the tasks they are designed to perform.

4.2 Component Identification

The component identification shall be comprised of a component name and identification number.

4.2.1 Component Name

A component shall have a name that is unique to the entire application domain. The naming convention shall follow these guidelines:

4.2.1.1 Component Name Format

The Component Name Format shall be a single character string derived from American Standard Code for Information Interchange (ASCII) with the characters' ASCII values ranging from 48 through 57—numbers zero through nine, 65 through 90—upper case alphabetic, 97 through 122—lower case alphabetic, and 95—the underscore character.

4.2.2 Component Identification Number

Every component has a unique ID number, which is defined by the JAUS specification. Within a subsystem, multiple instances of a component are allowed within a node and/or subsystem. Redundant components must either be on separate nodes or use unique instance

ID's within a node. JAUS views component granularity to be at a macro rather than micro level. JAUS is not a device architecture, so components are not associated with a device; rather, a component defines a function of the subsystem. A component can interface to devices, but this interface is outside the scope of JAUS. Aspects of component execution on their host computing resource such as scheduling, address space protection, fault tolerance, etc., are also outside the scope of this architecture.

4.3 Common Component Behaviors

JAUS defines a core set of messages and states and the behaviors associated with these messages. The core messages and their behaviors form a template for developing JAUS components.

4.4 Component Function Specification

4.4.1 Single Function

A component shall perform a single cohesive function. The scope of a component's function should be the one that has been broken down into a primitive function to the point where it is not advantageous to divide it further.

4.4.2 Bandwidth Minimization

The optimal component function shall be defined such that the subsequent interfaces will minimize communication bandwidth.

5 COMPONENT DEFINITIONS

JAUS is comprised of a series of components with well-defined interfaces. A developer can combine and integrate these components in any manner that is seen fit for a particular application. The JAUS components are grouped as follows:

- Command and control components
- Communications components
- Platform components
- Manipulator components
- Environmental sensor components

The components within each of these groupings are described in the subsequent sections of this document.

5.1 Command and Control Components

The following components provide a mechanism for system integration at the system and subsystem levels. As such, each of the following command and control components is allowed to output any message to any component and to receive any message from any component.

5.1.1 System Commander (ID 40)

Function:

The System Commander coordinates all activity within a given system.

Description:

The System Commander component has the responsibility of performing the multi-subsystem coordination, issuing commands, and querying status for the system operation. The functions of the System Commander component may be performed by humans or by computer resources, or both. When humans perform the System Commander functions, the human computer interface shall be considered part of the System Commander component.

5.1.2 Subsystem Commander (ID 32)

Function:

The Subsystem Commander coordinates all activity within a given subsystem.

Description:

The Subsystem Commander component has the responsibility of performing the mission planning, issuing commands, and querying status for the subsystem operation. The functions of the Subsystem Commander component may be performed by humans or by computer resources, or both. When humans perform commander functions, the human computer interface shall be considered part of the Subsystem Commander component.

5.2 Communications Components

5.2.1 Communicator (ID 35)

Function:

The Communicator component maintains all data links to other subsystems within a system.

Description:

The communicator component allows for a single point of entry to any subsystem (See Figure 5-1). This specification provides flexibility in communications systems design and implementation.

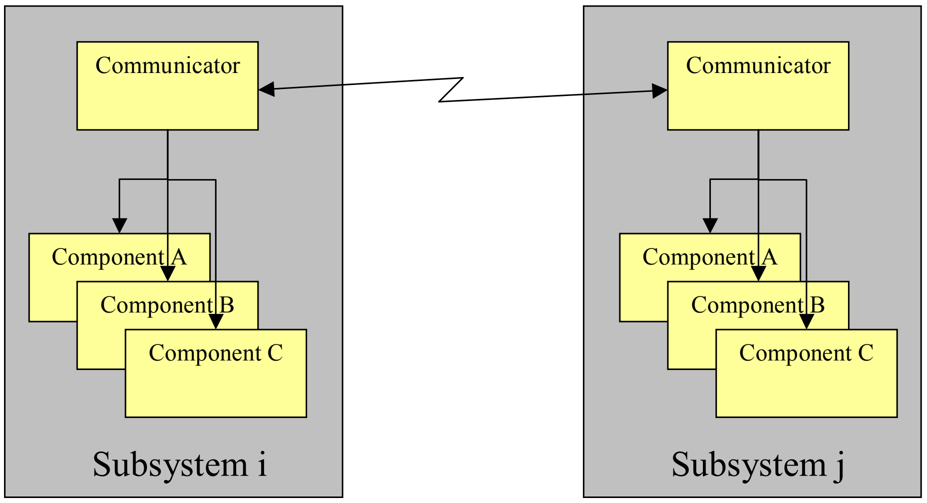


Figure 5-1 Communicators Provide Single Point of Subsystem Entry

5.3 Platform Components

5.3.1 Global Pose Sensor (ID 38)

Function:

The function of the Global Pose Sensor is to determine the global position and orientation of the platform.

Description:

The Report Global Pose message provides the position and orientation of the platform. The position of the platform is given in latitude, longitude, and elevation, in accordance with the WGS 84 standard. Platform orientation is as defined in Part 2 of this document.

5.3.2 Local Pose Sensor (ID 41)

Function:

The function of the Local Pose Sensor is to determine the local position and orientation of the platform.

Description:

The Report Local Pose message provides the position and orientation of the platform relative to a local reference frame. The position of the platform is given in Cartesian coordinates x, y, and z, relative to a defined local reference frame. Platform orientation is as defined in Part 2 of this document.

5.3.3 Velocity State Sensor (ID 42)

Function:

The Velocity State Sensor has the responsibility of reporting the instantaneous velocity of the platform.

Description:

The velocity state of a rigid body is defined as the set of parameters that are necessary to calculate the velocity of any point in that rigid body. Six parameters are required to specify a velocity state of a rigid body in terms of some fixed reference coordinate system. The first three parameters represent the velocity components of a point in the rigid body that is coincident with the origin of the fixed reference. The second three components represent the instantaneous angular velocity components. It is possible to represent the six velocity state parameters as a screw, about which the rigid body is rotating and translating along at that instant.

The reference frame for the velocity state sensor component is selected as a fixed coordinate system that at this instant is co-located with and aligned with the vehicle or system coordinate system. Thus the message data 'velocity x', 'velocity y', and 'velocity z' represents the current velocity of the subsystem's control point at this instant. For example if 'velocity x' has a value of 3 m/sec and 'velocity y' and 'velocity z' are zero, then the vehicle is moving in the forward direction at a velocity of 3 m/sec. The message data 'omega x', 'omega y', and 'omega z' represent the actual rate of change of orientation or angular velocity of the vehicle about its coordinate axes.

5.3.4 Primitive Driver (ID 33)

Function:

The Primitive Driver component performs basic driving and all platform related mobility functions including operation of common platform devices such as the engine and lights.

Description:

This component does not imply any particular platform type such as tracked or wheeled, but describes mobility in six degrees of freedom using a percent of available effort in each direction. Additionally, no power plant (gasoline, diesel, or battery) is implied and the component functions strictly in an open loop manner, i.e., a velocity is not commanded since that requires a speed sensor. Note that the specific actuator commands are not defined by JAUS.

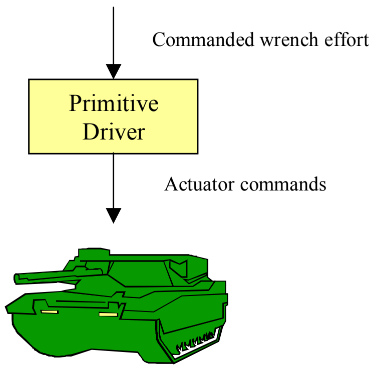


Figure 5-2 Wrench Effort Provides Basic Platform Mobility

5.3.5 Reflexive Driver (ID 43)

Function:

The function of the Reflexive Driver is to modify a commanded wrench effort if its execution would result in motion that compromises platform safety or stability.

Description:

The Reflexive Driver component can be thought of as a filter. It receives a desired motion (Commanded Wrench Effort) and usually just outputs this desired motion (typically to the

Primitive Driver component). However if the input desired motion would cause the platform to collide with an obstructing object or would compromise platform stability, then the Reflexive Driver component would output a modified motion (Modified Wrench Effort). This component provides a low-level protection for the platform and should not be interpreted as providing a high-level re-planning capability. It is designed for operation between a higher-level driver (e.g., Vector Driver, Waypoint Driver, etc. described later) and a Primitive Driver. The output of the component is a wrench effort that is communicated via the Set Wrench Effort message.

The sensors used to determine the platform's safe operation are not specified. Some possible inputs address limits to a platform's speed and/or turning rate and obstacle detection. Following the processing of the inputs, the Reflexive Driver may modify the input wrench to ensure the safety (e.g., avoid a collision) and stability (e.g., prevent a rollover) of the platform.

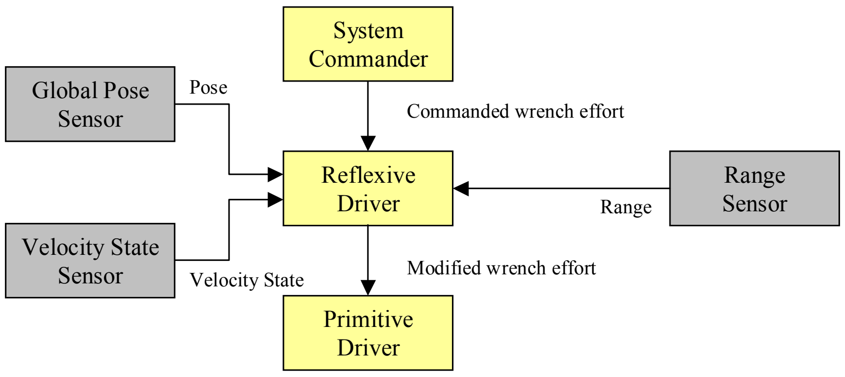


Figure 5-3 Reflexive Tele-Operation Block Diagram

5.3.6 Global Vector Driver (ID 34)

<u>Function</u>:

The function of the Global Vector Driver component is to perform closed loop control of the desired global heading, altitude and speed of a mobile platform.

Description:

The Global Vector Driver component takes the desired heading of the platform as measured with respect to the global coordinate system and the desired speed of the platform. The desired heading angle is defined in a right hand sense about the Z axis of the global coordinate system (the Z axis points downward) where North is defined as zero degrees. The desired Altitude Above Sea Level (ASL) provides a means through which systems capable of flight can be controlled. For ground-based systems, the Altitude ASL field is ignored. The Global Vector Driver component also receives data from the Global Pose Sensor component and the Velocity State Sensor component. This information allows the Global Vector Driver component to perform closed loop control on the platform's global heading, altitude and speed.

The output of the Global Vector Driver component is a wrench message that is typically sent to the Primitive Driver component. This message is interpreted by the Primitive Driver component in order to enact platform motion. The operator, acting as the system commander, could set the desired speed, altitude and heading. This scenario is illustrated in Figure 5-4.

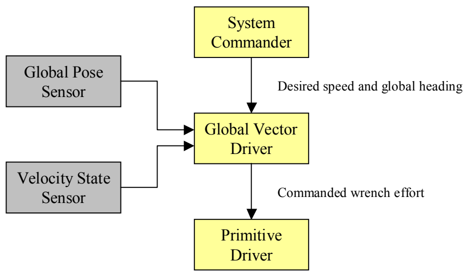


Figure 5-4 Global Vector Driving Block Diagram

Finally, Figure 5-5 depicts a scenario where the commanded wrench effort calculated by the Global Vector Driver is now sent to the Reflexive Driver. This allows the Reflexive Driver to modify this commanded wrench effort in order to maintain vehicle stability, e.g., platform rollover, and safety, e.g., obstacle avoidance.

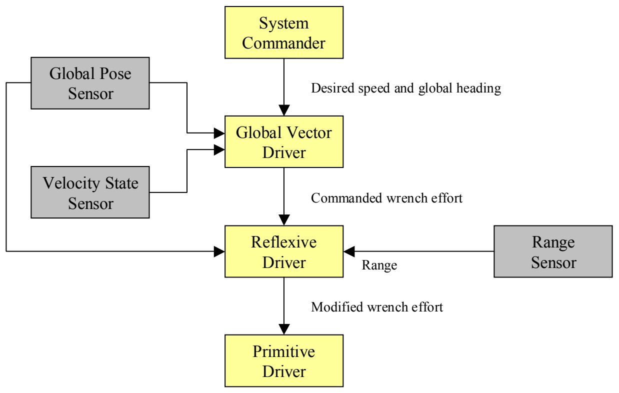


Figure 5-5 Global Vector Driving with Reflexive Driving Block Diagram

5.3.7 Local Vector Driver (ID 44)

Function:

The Local Vector Driver component performs closed loop control of the desired local heading, pitch, roll and speed of a mobile platform.

Description:

The Local Vector Driver component is very similar in function to the Global Vector Driver component, the difference being that the desired heading is defined in terms of a local coordinate system as opposed to the global coordinate system. The Local Vector Driver component takes as input four pieces of information, i.e. the desired heading, pitch and roll of the platform as measured with respect to a local coordinate system and the desired speed of the platform. The desired heading angle is defined in a right hand sense about the Z axis of the local coordinate system (the Z axis points downward) where zero degrees defines a heading that is parallel to the X axis of the local coordinate system. The pitch is the angle about the Y-axis and the roll is the desired angle about the X-axis. The Local Vector Driver component also receives data from the Local Pose Sensor component and the Velocity State Sensor component. This information allows the Local Vector Driver component to perform closed loop control on both the platform's local orientation and speed.

The output of the Local Vector Driver component is a wrench message that is typically sent to the Primitive Driver component. This message is interpreted by the Primitive Driver component in order to enact platform motion. The operator, acting as the system commander, could set the desired speed and local heading. This scenario is illustrated in Figure 5-6.

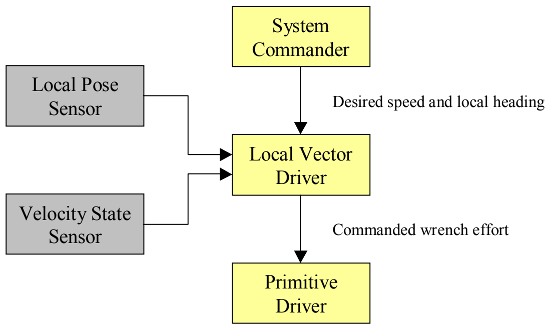


Figure 5-6 Local Vector Driver Block Diagram

Finally, Figure 5-7 depicts a scenario where the commanded wrench effort calculated by the Local Vector Driver is now sent to the Reflexive Driver. This allows the Reflexive Driver to modify this commanded wrench effort in order to maintain vehicle stability, e.g., platform rollover, and safety, e.g., obstacle avoidance.

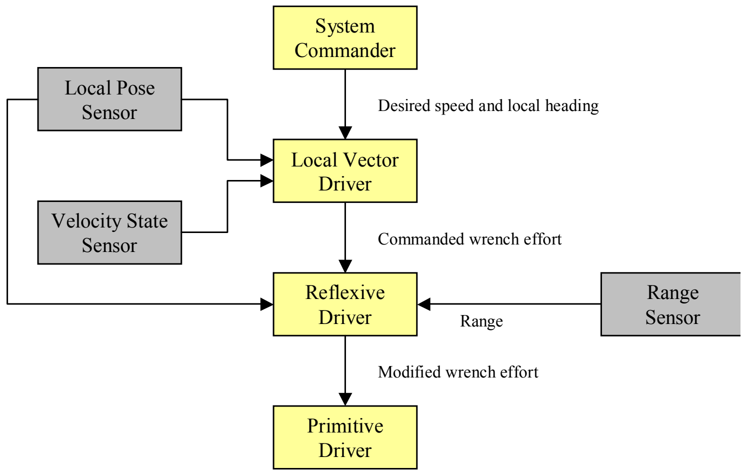


Figure 5-7 Local Vector Driver with Reflexive Driving Block Diagram

5.3.8 Global Waypoint Driver (ID 45)

Function:

The function of the Global Waypoint Driver is to determine the desired wrench of the platform given the desired waypoint(s), travel speed, current platform pose and current velocity state.

Description:

A single waypoint or a set of waypoints is provided via the Set Global Waypoint message. A waypoint consists of the desired position and orientation of the platform. The second input consists of the desired travel speed. The travel speed is set to zero for all transitions from a Standby State to a Ready State. The desired travel speed remains unchanged unless a new Set Travel Speed Message is received. The travel speed may then be changed at any time during waypoint navigation. Once the platform reaches its final waypoint, the component must transition back to a Standby State. The Global Waypoint Driver component will also use the current global pose and velocity state as communicated via the Report Global Pose message and the Report Velocity State message.

The output of the component is a commanded wrench that is communicated via the Set Wrench Effort message.

The two examples shown in **Figure 5-8** and **Figure 5-9** are scenarios where the operator, acting as the System Commander, sets the desired waypoints and travel speed without and with reflexive driving, respectively.

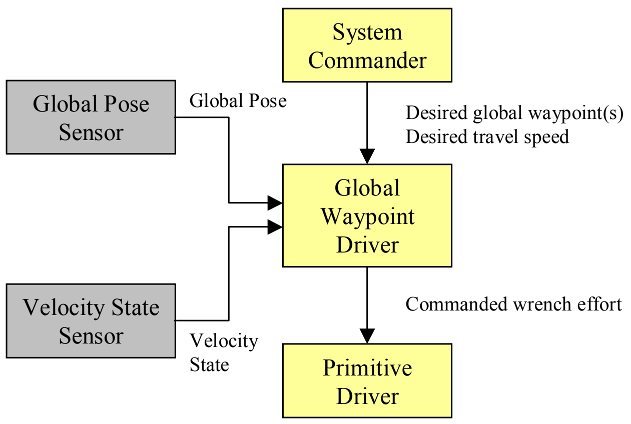


Figure 5-8 Global Waypoint Driver Block Diagram

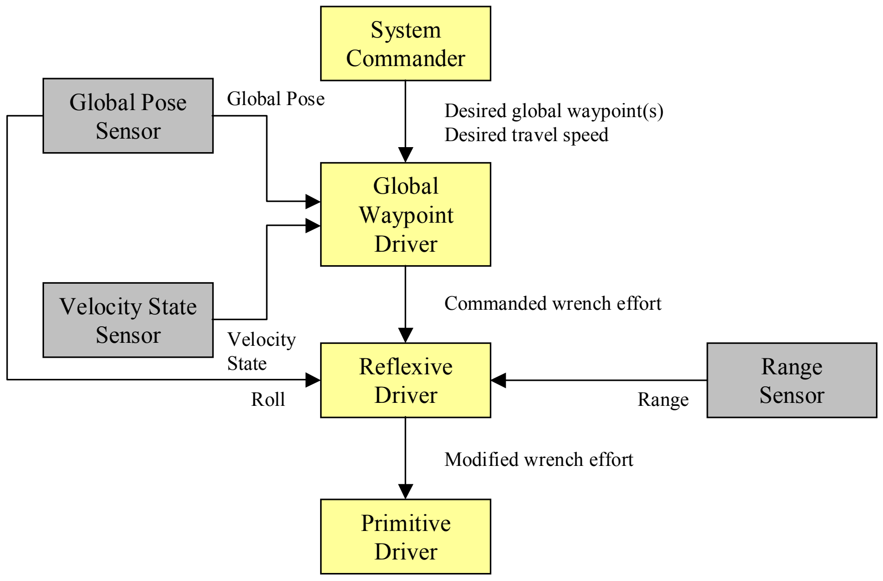


Figure 5-9 Global Waypoint Driving with Reflexive Driving Block Diagram

5.3.9 Local Waypoint Driver (ID 46)

Function:

The function of the Local Waypoint Driver is to determine the desired wrench of the platform given the desired waypoint(s), travel speed, current platform pose and current velocity state.

__Description__:

A single waypoint or a set of waypoints is provided via the Set Local Waypoint message. A waypoint consists of the desired position and orientation of the platform and is defined in terms of a local coordinate system. The second input consists of the desired travel speed. The travel speed is set to zero for all transitions from a Standby State to a Ready State. The desired travel speed remains unchanged unless a new Set Travel Speed message is received. The travel speed may then be changed at any time during waypoint navigation. Once the platform reaches its final waypoint, the component must transition back to a Standby State. The Local Waypoint Driver component will also use the current local pose and velocity state as communicated via the Local Pose message and the Velocity State message.

The output of the component is a desired wrench that is communicated via the Set Wrench Effort message.

The two examples shown in **Figure 5-10** and **Figure 5-11** are scenarios where the operator, acting as the System Commander, sets the desired waypoints and travel speed without and with reflexive driving, respectively.

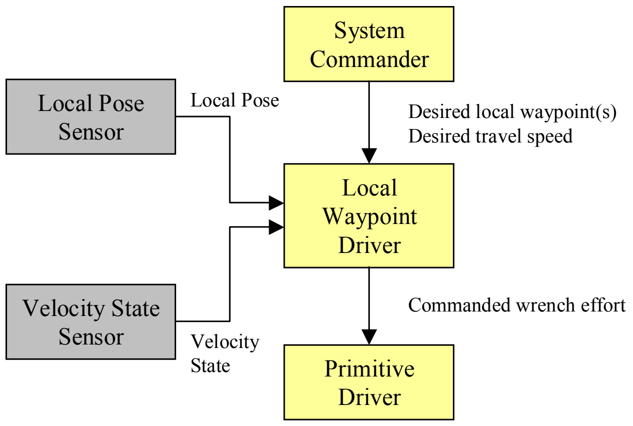


Figure 5-10 Local Waypoint Driving Block Diagram

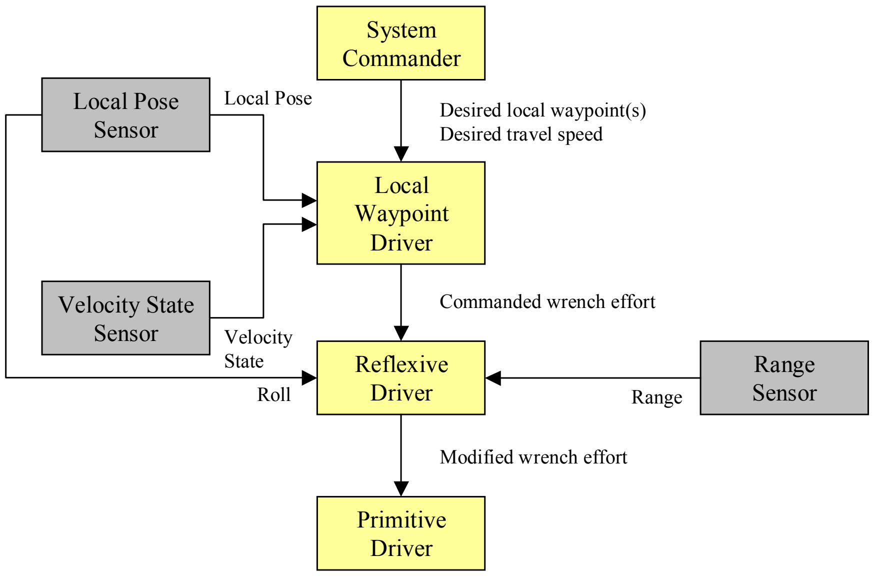


Figure 5-11 Local Waypoint Driving with Reflexive Driving Block Diagram

5.3.10 Global Path Segment Driver (ID 47)

Function:

The function of the Global Path Segment Driver is to perform closed loop control of position and velocity along a path where the path is defined in a generic manner.

Description:

The Global Path Segment Driver differs from the Waypoint Drivers in that the exact path between "waypoints" is strictly defined. A path segment will be defined by specifying the three-dimensional coordinates of three points, $\mathbf{P}_0$, $\mathbf{P}_1$, and $\mathbf{P}_2$ together with one scalar weighting value w_1. The equation of the path is defined as

$$\mathbf{p}(u) = \frac{(1-u)^2\,\mathbf{P}_0 + 2u(1-u)\,w_1\,\mathbf{P}_1 + u^2\,\mathbf{P}_2}{(1-u)^2 + 2u(1-u)\,w_1 + u^2},$$

where $\mathbf{p}$ represents some point on the path segment as u varies from 0 to 1. This equation defines a path segment as a second order polynomial (ellipse, parabola, or hyperbola) that lies in the plane defined by the three points $\mathbf{P}_0$, $\mathbf{P}_1$, and $\mathbf{P}_2$. When u=0, $\mathbf{p}(0) = \mathbf{P}_0$ and when u=1, $\mathbf{p}(1) = \mathbf{P}_2$ and thus the path segment begins at point $\mathbf{P}_0$ and ends at point $\mathbf{P}_2$. The point

P$_1$ is a control point that is not necessarily on the path. Figure 5-12 shows a path segment that is defined by **P**$_0$ = $[1, 2, 0]^T$, **P**$_1$ = $[3, 5, 0]^T$, **P**$_2$ = $[5, 1, 0]^T$, and $w_1 = 1.5$. Figure 5-13 demonstrates the effect of varying the weighting factor w_1. As can be seen in the figure, a larger value of w_1 draws the curve towards the path control point **P**$_1$.

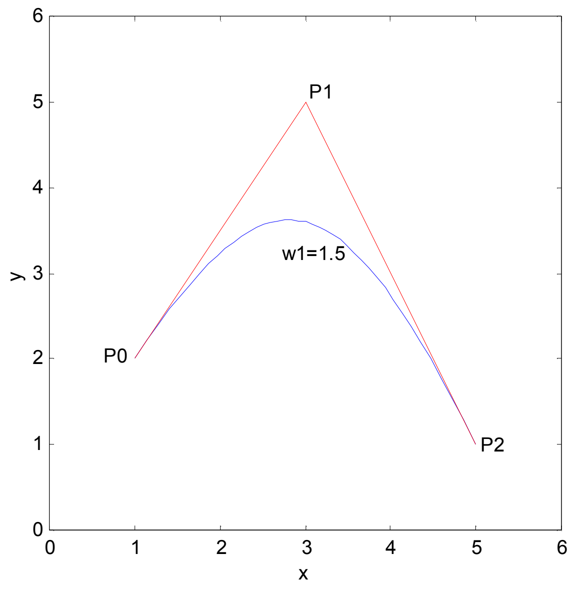


Figure 5-12 - Sample Path Segment with Control Points Shown

One feature of this representation of a path segment is that at the start of the path, the line from **P**$_0$ to **P**$_1$ will be tangent to the curve. Similarly at the end of the path, the line from **P**$_1$ to **P**$_2$ will be tangent to the curve. This feature makes it easy to define subsequent path segments so that there will be continuity at the connection point. Figure 5-14 shows the path segment shown in Figure 5-12 connected to a second path segment defined by the points **P**$_2$ = $[5, 1, 0]^T$, **P**$_3$ = $[6, -1, 0]^T$, and **P**$_4$ = $[8, -1, 0]^T$ with a weighting factor of $w_1 = 2$. As shown in the figure, continuity of slope will occur if the line from **P**$_1$ to **P**$_2$ is collinear with the line from **P**$_2$ to **P**$_3$.

The second input consists of the desired travel speed. The travel speed is set to zero for all transitions from a Standby State to a Ready State. The desired travel speed remains unchanged unless a new Set Travel Speed message is received. The travel speed may then be changed at any time during waypoint navigation. Once the platform reaches its final waypoint, the component must transition back to a Standby State.

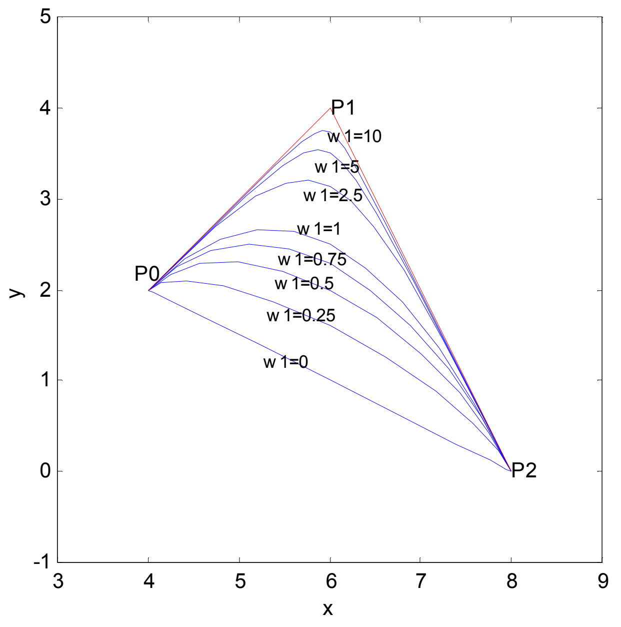


Figure 5-13 - Effect of Weighting Parameter w1 on Curve Shape

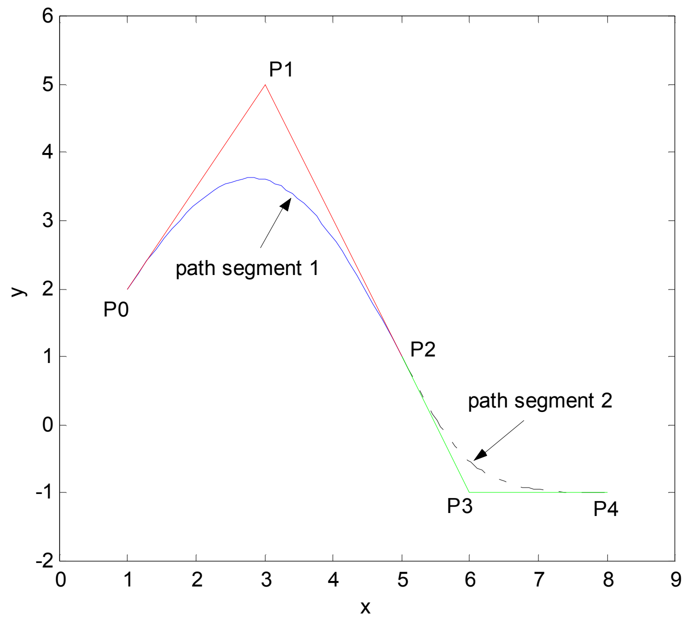


Figure 5-14 - Slope Continuity at Path Segment Connection Point

The two examples shown in **Figure 5-15** and **Figure 5-16** are scenarios where the operator, acting as the System Commander, sets the desired path segments and travel speed without and with reflexive driving, respectively.

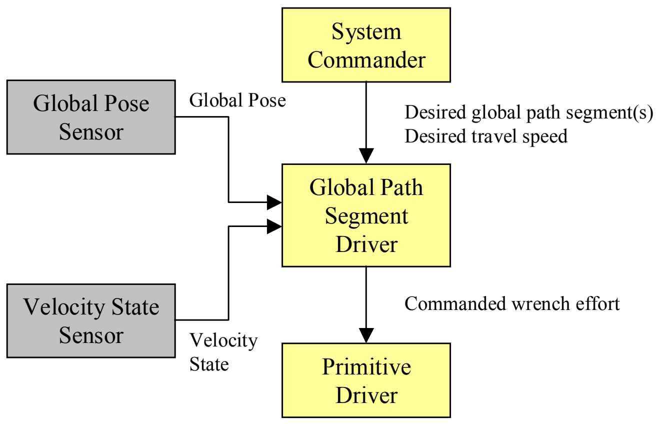


Figure 5-15 Path Segment Driving Block Diagram

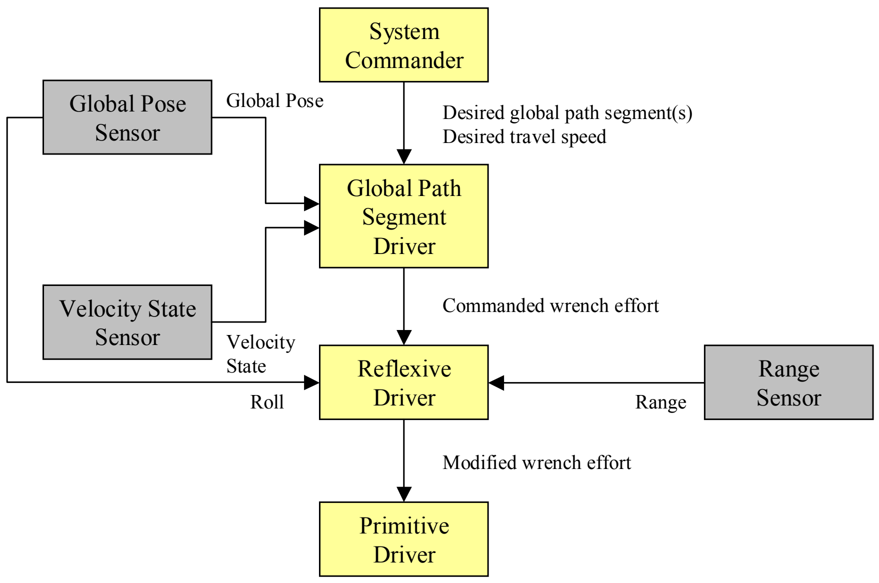


Figure 5-16 Path Segment Driving with Reflexive Driving Block Diagram

5.3.11 Local Path Segment Driver (ID 48)

Function:

The function of the Local Path Segment Driver is to perform closed loop control of position and velocity along a path where the path is defined in a generic manner.

Description:

A three-dimensional path segment will be defined by specifying the coordinates of three points, $\mathbf{P}_0$, $\mathbf{P}_1$, and $\mathbf{P}_2$ together with one scalar weighting value w_1. The equation of the path is defined as

$$\mathbf{p}(u) = \frac{(1-u)^2\,\mathbf{P}_0 + 2u(1-u)\,w_1\,\mathbf{P}_1 + u^2\,\mathbf{P}_2}{(1-u)^2 + 2u(1-u)\,w_1 + u^2},$$

where **p** represents some point on the path segment as u varies from 0 to 1. A more complete discussion of this path segment representation is presented in the description of the Global Path Segment Driver component (See Section 5.3.10).

The two examples shown in **Figure 5-17** and **Figure 5-18** are scenarios where the operator, acting as the System Commander, sets the desired path segments and travel speed without and with reflexive driving, respectively.

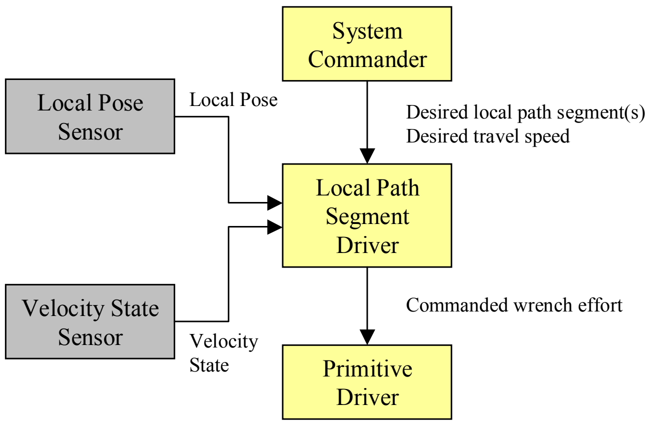


Figure 5-17 Local Path Segment Driving Block Diagram

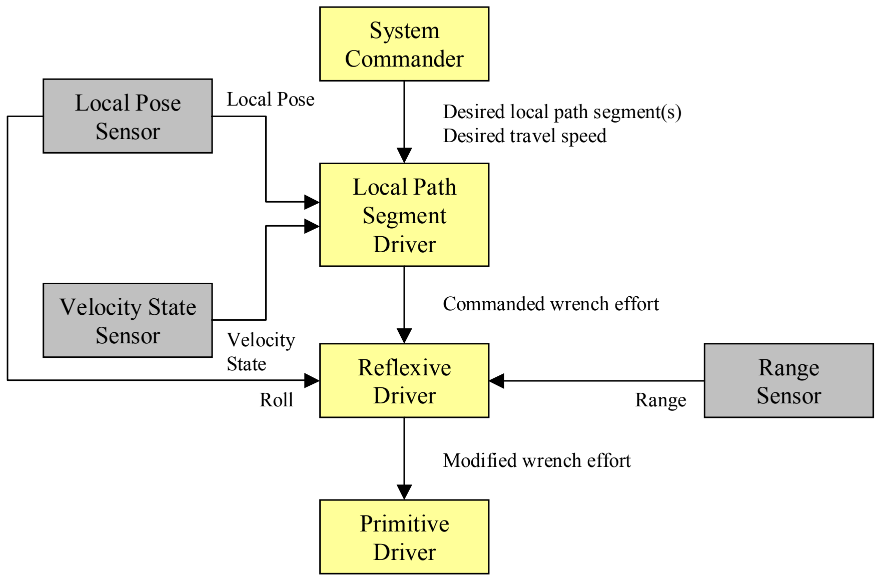


Figure 5-18 Local Path Segment Driving with Reflexive Driving Block Diagram

5.4 Manipulator Components

This section provides a brief explanation of several components that can be used for command and control a manipulator arm. The components focus on a serial manipulator comprised of any number of prismatic and revolute joints. The components are grouped according to function into the following categories:

Low Level Manipulator Control Components – The one component in this category allows for low-level command of the manipulator joint actuation efforts. This is an open-loop command that could be used in a simple tele-operation scenario. The component in this category is listed as follows:

- Primitive Manipulator Component

Manipulator Sensor Components – These components, when queried, return instantaneous sensor data. Three components are defined that return respectively joint positions, joint velocities, and joint torques or forces. The components in this category are listed as follows:

- Manipulator Joint Position Sensor Component

- Manipulator Joint Velocity Sensor Component

- Manipulator Joint Force/Torque Sensor Component

Low Level Position and Velocity Driver Components – These components take as inputs the desired joint positions, the desired joint velocities, the desired end-effector pose, or the desired end-effector velocity state. Closed-loop control is implied. No path information is specified. The components in this category are listed as follows:

- Manipulator Joint Positions Driver Component
- Manipulator End-Effector Pose Driver Component
- Manipulator Joint Velocities Driver Component
- Manipulator End-Effector Velocity State Driver Component

Mid Level Position and Velocity Driver Components – Two components are grouped here under this heading. The first takes as input the goal values for each joint parameter at several time values together with motion constraints, i.e. maximum joint velocity, maximum acceleration, and maximum deceleration. The second takes as input a series of end-effector poses at specified time values. Closed-loop control is implied. The components in this category are listed as follows:

- Manipulator Joint Move Driver Component
- Manipulator End-Effector Discrete Pose Driver Component

5.4.1 Definition of Coordinate Systems

Global Coordinate System

Points that are defined in this coordinate system are defined in units of longitude, latitude, and elevation. The X-axis is defined as pointing North and the Z axis points downward.

Vehicle Coordinate System

This coordinate system is defined as attached to the vehicle frame. The X-axis points in the forward direction and the Z-axis points downward. The Y-axis is defined so as to have a right-handed coordinate system, i.e. $\mathbf{i} \times \mathbf{j} = \mathbf{k}$ where **j, k,** and **i** are unit vectors along the X, Y, and Z coordinate axes.

Manipulator Base Coordinate System

In most cases, this coordinate system is attached to the vehicle base just as the vehicle coordinate system is. The origin is located at the intersection of the line along the first joint axis, $\mathbf{S}_1$, and the line along the first link, $\mathbf{a}_{12}$. The Z axis is along $\mathbf{S}_1$ and the direction of the X-axis is user defined, but fixed with respect to the vehicle frame. Since the manipulator base coordinate system and the vehicle coordinate system are both attached to the vehicle base, the transformation matrix that describes the relative position and orientation of these two coordinate systems will be constant (in some cases a manipulator may be attached to the end of another manipulator and this would not be the case).

End-Effector Coordinate System

This coordinate system is attached to the last link of the serial manipulator. The origin is located at the point that is a distance S_n (S_6 for a six axis manipulator) along the last joint axis vector ($\mathbf{S}_6$ for a six axis manipulator) from the intersection of the lines along the last joint axis and the preceding link axis ($\mathbf{S}_6$ and $\mathbf{a}_{56}$ for a six axis manipulator). The Z-axis is along the $\mathbf{S}_n$ vector. The direction of the X-axis is user defined, but of course must be perpendicular to the Z-axis. The Y-axis is defined by the right-hand rule.

Note: The distance S_n is returned by the Primitive Manipulator Component via the Report Manipulator Specifications Message.

5.4.2 Primitive Manipulator (ID 49)

A modification from JAUS Reference Architecture release 3.1 to the Report Manipulator Specifications message has been made to account for the necessity to define the relative position and orientation of the manipulator base coordinate system with respect to the vehicle coordinate system.

Function:

This component is concerned only with the remote operation (open-loop control) of a single manipulator system. The manipulator may or may not have joint measurement sensors that would provide joint position and joint velocity information. Without joint measurement sensors, the operator can only send joint motion efforts as commands to the manipulator.

This component does not use any joint angle or velocity feedback from the manipulator. It only receives the desired percentage of maximum joint effort for each joint as an input.

Associated Messages:

- Set Joint Effort
- Query Manipulator Specifications
- Query Joint Efforts
- Report Manipulator Specifications
- Report Joint Efforts

Description:

This component is the low level interface to a manipulator arm and is in many respects similar to the Primitive Driver component for mobility of the platform. When queried, the component will reply with a description of the manipulator's specification parameters, axes range of motion, and axes velocity limits. The notations used to describe these data are documented in many popular text books on robotics and were previously presented in Part 2, Section 2.5. The mechanism specification parameters as reported by the Report Manipulator Specifications Message consist of the number of joints, the type of each joint (either revolute or prismatic), the link description parameters for each link (link length and twist angle as shown in Part 2, Figure 2.2), the constant joint parameter value (offset for a revolute joint (see Part 2, Figure 2.3), and joint angle for a prismatic joint (see Part 2, Figure 2.4)). The minimum and maximum allowable value for each joint and the maximum velocity for each joint follow this information. Motion of the arm is accomplished via the Set Joint Effort message (See **Figure 5-19**). In this message, each actuator is commanded to move with a percentage of maximum effort.

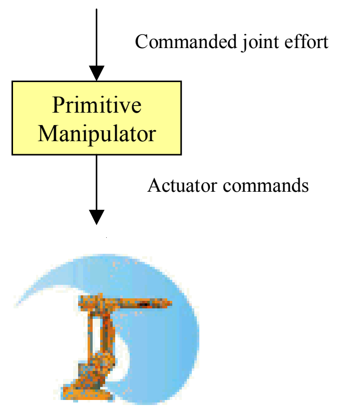


Figure 5-19 Joint Effort Provides Basic Manipulator Mobility

5.4.3 Manipulator Joint Position Sensor Component (ID 51)

Function:

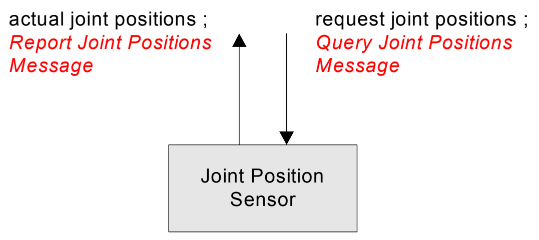
The Function of the Joint Position Sensor Component is to report the values of manipulator joint parameters when queried.

Associated Messages:

- Query Joint Positions
- Report Joint Positions.

Description:

The Report Joint Positions message provides the instantaneous joint positions. The positions are given in degrees for revolute and in meters for prismatic joints. The component is depicted in Figure 5-20.



actual joint positions ;
Report Joint Positions Message
request joint positions ;
Query Joint Positions Message
Joint Position Sensor

Figure 5-20 Joint Position Sensor Component

5.4.4 Manipulator Joint Velocity Sensor Component (ID 52)

Function:

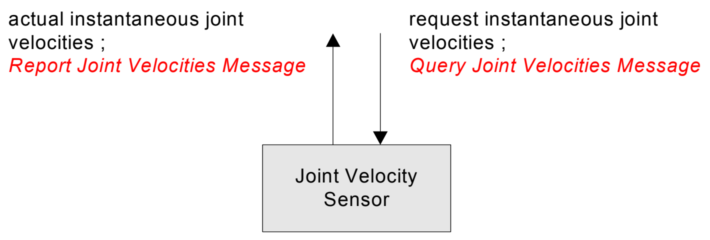
The Function of the Joint Velocity Sensor is to report the values of instantaneous joint velocities when queried.

Associated Messages:

- Query Joint Velocities
- Report Joint Velocities.

Description:

The Report Joint Velocities message provides the instantaneous joint velocities. The velocities are given in radians/sec for revolute and in meters/sec for prismatic joints. The component is depicted in Figure 5-21.



actual instantaneous joint velocities ;
Report Joint Velocities Message
request instantaneous joint velocities ;
Query Joint Velocities Message
Joint Velocity Sensor

Figure 5-21 Joint Velocity Sensor Component

5.4.5 Manipulator Joint Force/Torque Sensor Component (ID 53)

Function:

The Function of the Joint Force/Torque Sensor is to report the values of instantaneous torques (for revolute joints) and forces (for prismatic joints) that are applied at the individual joints of the manipulator kinematic model when queried.

Associated Messages:

- Query Joint Force/Torques
- Report Joint Force/Torques

Description:

The Joint Force/Torque Sensor component provides the instantaneous joint forces or torques that are acting on each joint of the manipulator kinematic model. Forces are returned for prismatic joints in units of Newton's. Torques is returned for revolute joints in units of Newton-meters.

5.4.6 Manipulator Joint Positions Driver Component (ID 54)

Function:

The Function of the Joint Positions Driver is to perform closed-loop joint position control.

Associated Messages:

- Set Joint Positions
- Report Manipulator Specifications
- Report Joint Efforts
- Report Joint Positions
- Set Joint Effort

Description:

The inputs are the desired joint values, current joint angles, and the manipulator specifications report. The output is the joint effort level that is sent to the Primitive Manipulator component. It should be noted that in most implementations that this component and the Primitive Manipulator component would be embedded in the same node that will facilitate the control process. Figure 5-22 depicts the Manipulator Joint Positions Driver component.

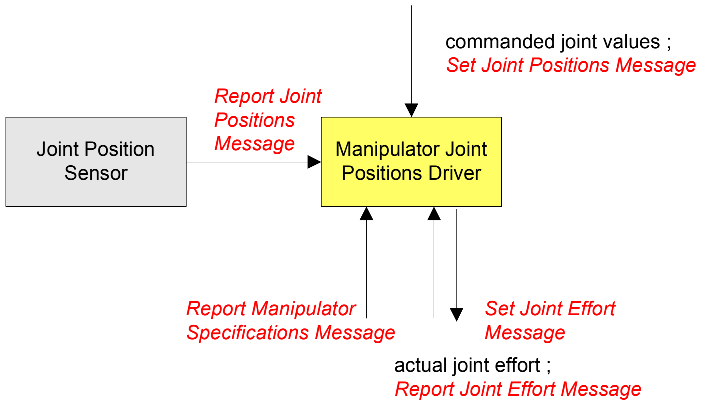

commanded joint values ;
Set Joint Positions Message
Report Joint Positions Message
Joint Position Sensor
Manipulator Joint Positions Driver
Report Manipulator Specifications Message
Set Joint Effort Message
actual joint effort ;
Report Joint Effort Message

Figure 5-22 Manipulator Joint Positions Driver Component

5.4.7 Manipulator End-Effector Pose Driver Component (ID 55)

Function:

The Function of the Manipulator End-Effector Pose Driver is to perform closed-loop position and orientation control of the end-effector.

Associated Messages:

- Set Tool Point
- Set End-Effector Pose
- Query Tool Point
- Report Manipulator Specifications
- Report Joint Efforts
- Report Joint Positions
- Set Joint Effort
- Report Tool Point

Description:

This component performs closed-loop position and orientation control of the end-effector. The input is the desired position and orientation of the end-effector specified in the vehicle coordinate system, the current joint angles, and the data from the manipulator specification report. The output is the joint effort level that is sent to the Primitive Manipulator component. It should be noted that in most implementations that this component and the Primitive Manipulator component would be embedded in the same node that will facilitate the control process. The component is depicted in Figure 5-23.

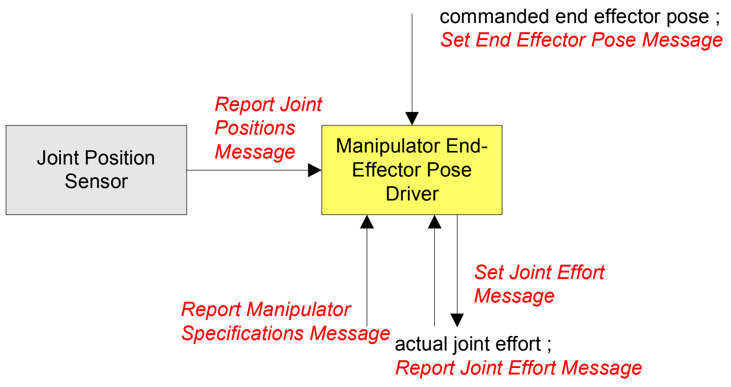

commanded end effector pose ;
Set End Effector Pose Message
Report Joint Positions Message
Joint Position Sensor
Manipulator End-Effector Pose Driver
Set Joint Effort Message
Report Manipulator Specifications Message
actual joint effort ;
Report Joint Effort Message

Figure 5-23 Manipulator End-Effector Pose Driver

5.4.8 Manipulator Joint Velocities Driver Component (ID 56)

Function:

The Function of the Joint Velocities Driver is to perform closed-loop joint velocity control.

Associated Messages:

- Set Joint Velocities
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Velocities
- Set Joint Effort

Description:

The input consists of the desired instantaneous joint velocities, the current joint velocities, and the data from the manipulator specification report. The output is the joint effort level that is sent to the Primitive Manipulator component. It should be noted that in most implementations that this component and the Primitive Manipulator component will be embedded in the same node which will facilitate the control process. The component is depicted in Figure 5-24.

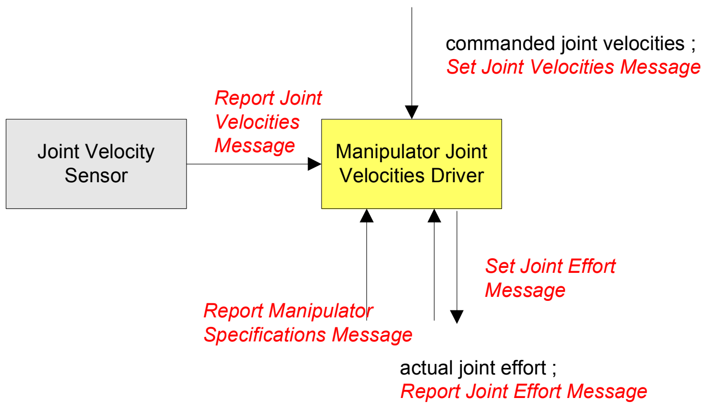

commanded joint velocities ;
Set Joint Velocities Message
Report Joint Velocities Message
Joint Velocity Sensor
Manipulator Joint Velocities Driver
Set Joint Effort Message
Report Manipulator Specifications Message
actual joint effort ;
Report Joint Effort Message

Figure 5-24 Manipulator Joint Velocities Driver

5.4.9 Manipulator End-Effector Velocity State Driver Component (ID 57)

Function:

The Function of the Manipulator End-Effector Velocity State Driver is to perform closed-loop velocity control of the end effector.

Associated Messages:

- Set End-Effector Velocity State
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Positions
- Report Joint Velocities
- Set Joint Effort

Description:

The input is the desired end-effector velocity state, specified in the vehicle coordinate system or the end-effector coordinate system, the current joint positions and joint velocities, and the data from the manipulator specifications report. The output is the joint effort level that is sent to the Primitive Manipulator component. It should be noted that in most implementations that this component and the Primitive Manipulator component would be embedded in the same node that will facilitate the control process. Figure 5-25 depicts this component.

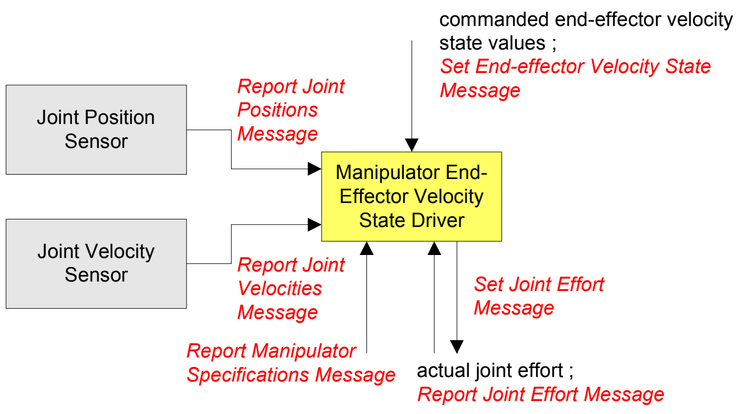

commanded end-effector velocity state values ;
Set End-effector Velocity State Message
Report Joint Positions Message
Joint Position Sensor
Manipulator End-Effector Velocity State Driver
Joint Velocity Sensor
Report Joint Velocities Message
Set Joint Effort Message
Report Manipulator Specifications Message
actual joint effort ;
Report Joint Effort Message

Figure 5-25 Manipulator End-Effector Velocity State Driver Component

5.4.10 Manipulator Joint Move Driver Component (ID 58)

Function:

The function of the Manipulator Joint Move Driver is to perform closed-loop joint level control of the manipulator where motion parameters for each joint are specified. The specified motion parameters are the desired values, maximum velocity, maximum acceleration, and maximum deceleration for each joint.

Associated Messages:

- Set Joint Motion
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Positions
- Report Joint Velocities
- Set Joint Effort

Description:

The inputs are the desired joint values at specified time values together with data to define a trapezoidal velocity profile, i.e. the maximum joint velocity, maximum joint acceleration, and maximum joint deceleration. No explicit path is defined, only the values of the joint angles at distinct times. The time values are measured in units of seconds and are relative to the time that the movement to the first joint angle set is started.

Additional inputs are the current joint values, joint velocities, and the data from the Report Manipulator Specifications message. The output is the joint effort level that is sent to the Primitive Manipulator component. It should be noted that in most implementations that this component and the Primitive Manipulator component will be embedded in the same node which will facilitate the control process. The component is depicted in Figure 5-26.

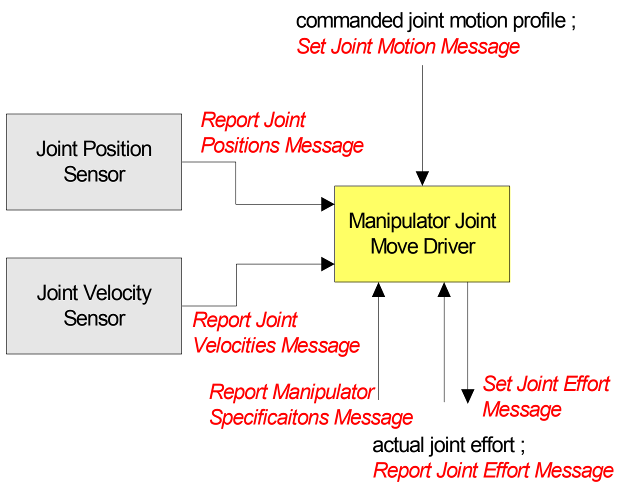

commanded joint motion profile ;
Set Joint Motion Message
Joint Position Sensor
Report Joint Positions Message
Manipulator Joint Move Driver
Joint Velocity Sensor
Report Joint Velocities Message
Report Manipulator Specificaitons Message
Set Joint Effort Message
actual joint effort ;
Report Joint Effort Message

Figure 5-26 Manipulator Joint Move Driver Component

5.4.11 Manipulator End-Effector Discrete Pose Driver Component (ID 59)

Function:

The function of the Manipulator End-Effector Discrete Pose Driver is to perform closed-loop control of the end-effector pose through a series of specified positions and orientations.

Associated Messages:

- Set End-Effector Path Motion
- Report Manipulator Specifications
- Report Joint Effort
- Report Joint Positions
- Report Joint Velocities
- Set Joint Effort

Description:

This component performs closed-loop control of the end-effector pose as measured with respect to the vehicle coordinate system. The inputs are a path motion description (discrete end-effector position and orientation at time t measured in the vehicle coordinate system which is defined by a point and a quaternion at time t), the current joint values, the current joint velocities, and the data from the Report Manipulator Specifications message. The output is the joint effort level that is sent to the Primitive Manipulator component. It should be noted that in most implementations that this component and the Primitive Manipulator component would be embedded in the same node that will facilitate the control process. The component is depicted in Figure 5-27 in the vehicle coordinate system.

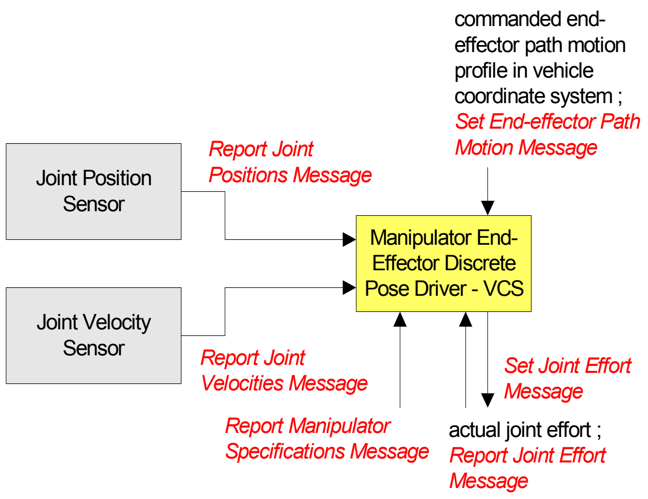

commanded end-effector path motion profile in vehicle coordinate system ;
Set End-effector Path Motion Message
Report Joint Positions Message
Joint Position Sensor
Manipulator End-Effector Discrete Pose Driver - VCS
Joint Velocity Sensor
Report Joint Velocities Message
Set Joint Effort Message
Report Manipulator Specifications Message
actual joint effort ;
Report Joint Effort Message

Figure 5-27 Manipulator End-Effector Discrete Pose Driver Component

5.5 Environment Sensor Components

5.5.1 Visual Sensor (ID 37)

Function:

The Visual Sensor component has the responsibility of controlling the camera(s) of a subsystem.

Description:

Each camera has an associated coordinate frame. The Z-axis of the coordinate frame is positive along the centerline of the camera lens, pointing toward the field of view. From an observer looking through the camera, the X-axis is positive to the right and is in the horizontal plane of the image sensor. The Y-axis completes the right-handed orthogonal coordinate frame.

The visual component has the capability to control the positioning and orientation of a fully articulated camera mounting. The visual component also provides the mechanisms to determine and set the imagery format, camera settings, and audio (if supported).

5.5.2 Range Sensor (ID 50)

Function:

This component reports range data to requesting components. The range data is measured relative to the platform coordinate system at the time reported.

Description:

The Range Sensor component provides information from proximity sensors for the purpose of object detection. The Range Sensor shall output the locations of detected objects with a certain measure of accuracy. The Range Sensor may consist of a single range sensor or multiple sensors. With this most primitive sensor, raw data from the sensors are processed and formatted into standard, semi-raw data and returned when requested.

5.5.3 World Model Vector Knowledge Store (ID 61)

Function:

The function of the World Model Vector Knowledge Store (WMVKS) component is to provide a central repository for system-, subsystem-, node-, and/or component-level vector-formatted geospatial data.

Associated Messages:

- Create Vector Knowledge Store Objects

- Set Vector Knowledge Store Feature Class Metadata
- Delete Vector Knowledge Store Objects
- Query Vector Knowledge Store Objects
- Query Vector Knowledge Store Feature Class Metadata
- Query Vector Knowledge Store Bounds
- Terminate Vector Knowledge Store Data Transfer
- Report Vector Knowledge Store Object(s) Creation
- Report Vector Knowledge Store Feature Class Metadata
- Report Vector Knowledge Store Objects
- Report Vector Knowledge Store Bounds
- Report Vector Knowledge Store Data Transfer Termination

Description:

Storing and sharing of vector-formatted geospatial data is supported by the World Model Vector Knowledge Store. The primary benefit of this world modeling method is that vector data typically require significantly less bandwidth to transmit as compared to raster data.

For the vector knowledge store, objects are represented as points, lines and polylines, and polygons. Figure 5-28 shows the format of these vector objects. Polylines and polygons may consist of up to 65535 vertices. Rather than assigning the points that make up these objects Cartesian coordinates with respect to an arbitrarily chosen datum, all points are expressed as points of latitude and longitude (WGS84).

The vector objects on the right side of Figure 5.28 have a region buffer parameter. The region buffer is defined as an offset distance in meters that 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 buffer feature allows a region to be established in proximity to vector objects. For example, United States Geological Survey (USGS) digital line graph (DLG) road data is presented in vector form representing the center-line of such roads. It may be useful to search for objects within an area along a particular route defined in the 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 DLG data and assign a region buffer to each line segment.

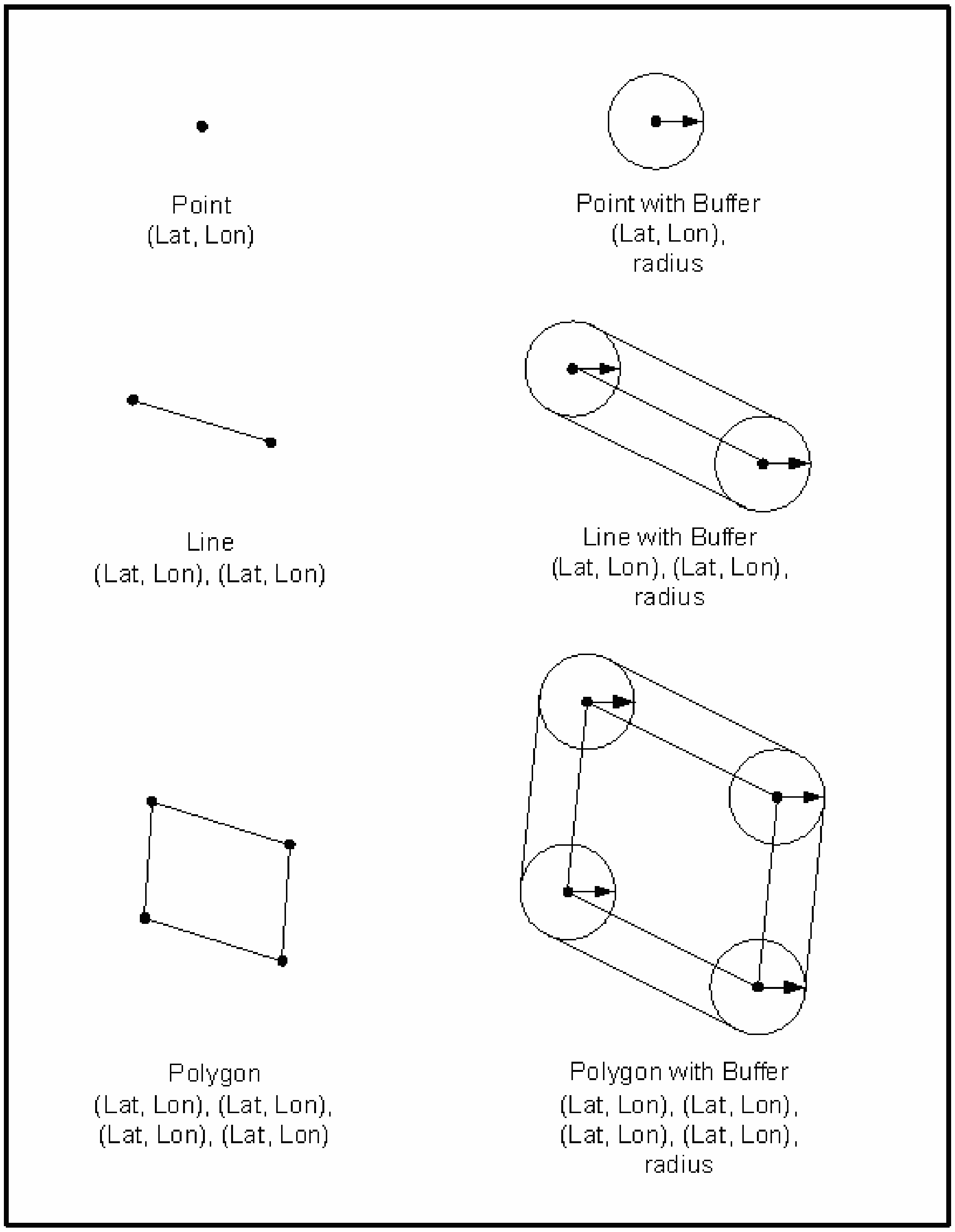


Figure 5-28 Vector Objects

5.6 Planning Components

This section provides a brief explanation of components that can be used for mission planning, coordination, and execution. The components and their message sets provide detailed knowledge of the machine's capabilities, the mission goals and objectives, and a mechanism for conveying the plan to the primitives within the machine. The Mission Planning, Coordination and Execution process for JAUS systems is graphically depicted below where the process, or pieces of the process, can be located and/or duplicated anywhere in the system. The process allows for the generation and execution of complex mission plans made up of mobility, sensing, and payload commands.

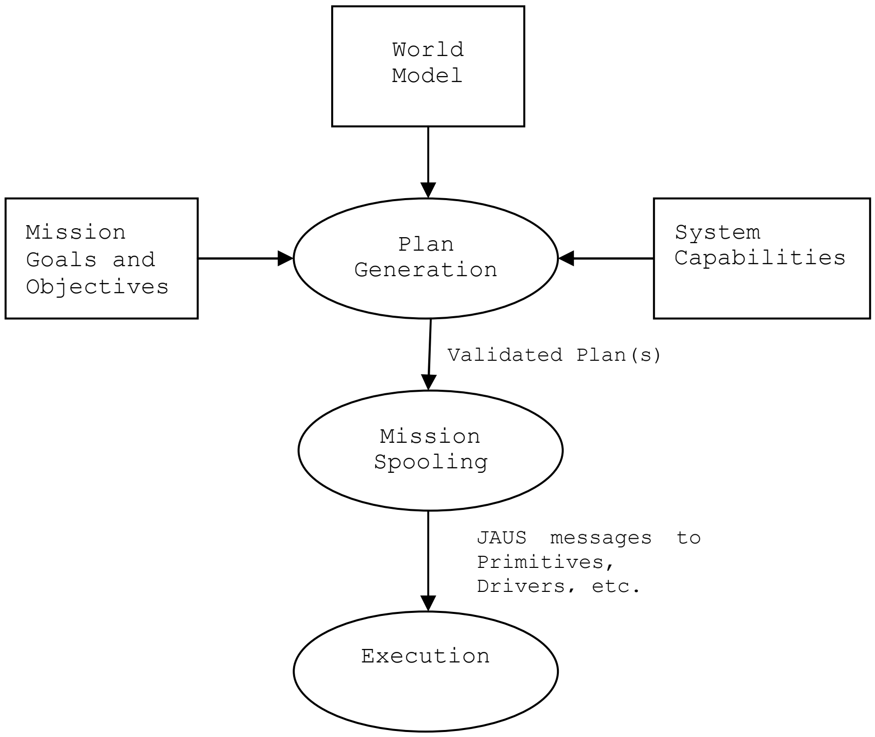


Figure 5-29 Mission Planning, Coordination, Execution Process

5.6.1 Mission Planner

The Mission Planner is being defined. This section will be completed when the Mission Planner is complete.

5.6.2 Mission Spooler (36)

Function:

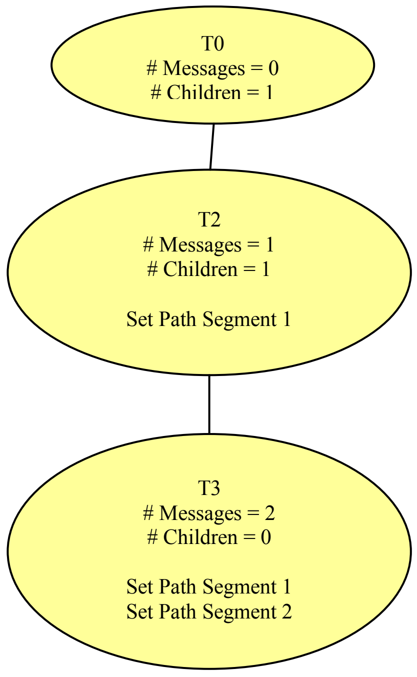
The Mission Spooler provides a central location for mission plans during execution. It is responsible for parceling out elements of the mission plan for execution by machine primitives.

__**Associated Messages**__:

- Spool Mission
- Run Mission
- Abort Mission
- Pause Mission
- Resume Mission
- Remove Messages
- Replace Messages
- Query Spooling Preferences
- Report Spooling Preferences
- Query Mission Status
- Report Mission Status

__**Description**__:

The Mission Spooler is responsible for spooling mission plans. A **mission** is a set of JAUS commands to be performed by one or more components on board one or more unmanned subsystems. The mission *structure* is an N-ary tree, which allows for parallel, sequential, iterative, conditional, and coordinated mission plans and mission plan tasks. Each mission has a unique ID allowing for multiple mission plans. A mission plan is made up of **tasks,** which contain JAUS messages, and/or children tasks. An example mission plan and corresponding mission structure follows:



Mission Plan 0:
  Waypoint 1
    Path Segment 1*
  Waypoint 2
    Path Segment 1*
    Path Segment 2*

*A waypoint, in this example, is broken down into path segments by a higher level component (i.e. planner)

A JAUS message within a mission plan can be **blocking**. The Mission Spooler shall not spool messages beyond a blocking JAUS message until the unmanned system has completed the action associated with the blocking JAUS message and a Mission Status message with the Message Finished status is received for that message. Payload commands are a good example of where blocking messages may be used. Some payloads can only perform their functions (e.g. soil sampling, video image) when the robot is stationary while other payloads can (or must) perform their functions (e.g. start mine flail) while in motion. The blocking lag ensures that no other messages are spooled until the blocking message is complete.

Example configurations of the Mission Spooler follow:

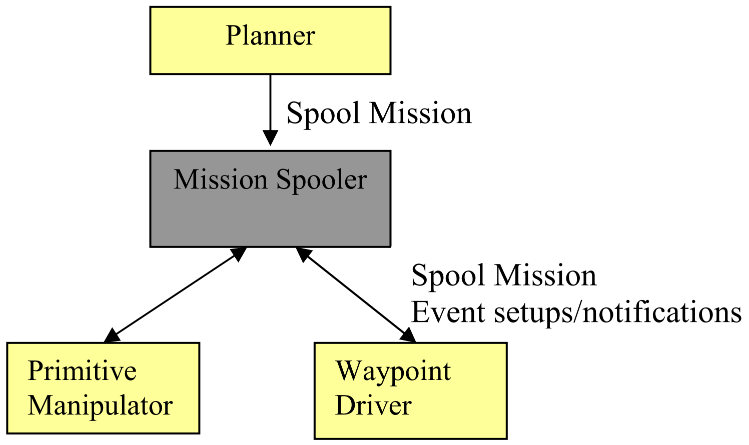


Figure 5-30 Mission Spooler Configuration Example 1

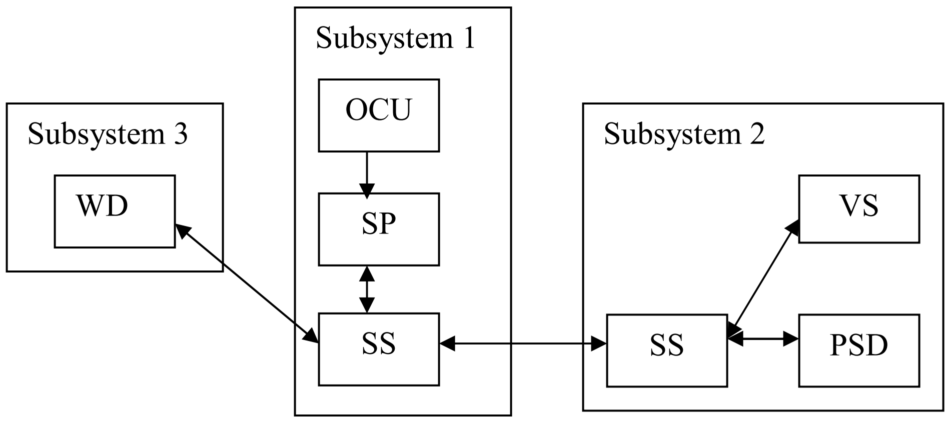


Figure 5-31 Mission Spooler Configuration Example 2

Subsystem 1: Operator Control Unit (OCU)

Mission Planner (MP)

Mission Spooler (MS)

Subsystem 2: Mission Spooler (MS)

Visual Sensor (VS)

Path Segment Driver (PSD)

Subsystem 3: Waypoint Driver (WD)

The Mission Spooler on Subsystem 1 is coordinating multiple mission plans for two unmanned systems. In the case of Subsystem 2, it sends the mission plan on to the Mission Spooler located on that subsystem which then parcels the plan elements out to the correct component. Subsystem 3 does not have a Mission Spooler available, so Subsystem 1's Mission Spooler parcels the plan elements out to the correct components on Subsystem 3.

Messaging Requirements:

The following requirements shall be followed:

- The Mission Spooler shall not spool messages beyond a blocking message until a Report Mission Status message for Message Finished is received for the blocking message.

Requirements for unsupported Mission Spooler messages:

- If a component does not respond to the Query Spooling Preferences message, the Mission Spooler shall default to sending a Spool Mission message with 1 message at a time to the component.

If a component does not support the Spool Mission message, the Mission Spooler shall default to sending the raw JAUS message to the component, 1 message at a time.

6 LIST OF ACRONYMS

ASCII	American Standard Code for Information Interchange
DCP	Document Control Plan
DGPS	Differential Global Positioning System
DM	Domain Model
JAUS	Joint Architecture for Unmanned Systems
JTA	Joint Technical Architecture
LADAR	Laser Detection & Ranging
MP	Mission Planner
MS	Mission Spooler
NIST	National Institute of Standards and Technology
OCU	Operator Control Unit
PSD	Path Segment Driver
RA	Reference Architecture
SC(s)	Service Connection(s)
SI	The International System of Units
UGV(s)	Unmanned Ground Vehicle(s)
UAV(s)	Unmanned Air Vehicle(s)
USV(s)	Unmanned Surface Vehicle(s)
UUV(s)	Unmanned Underwater Vehicle(s)
VS	Visual Sensor
WD	Waypoint Driver