https://ntrs.nasa.gov/search.jsp?R=19910002119 2020-03-19T19:58:37+00:00Z

MCR-89-559
Contract NAS8-36431

Phase II
Final Report

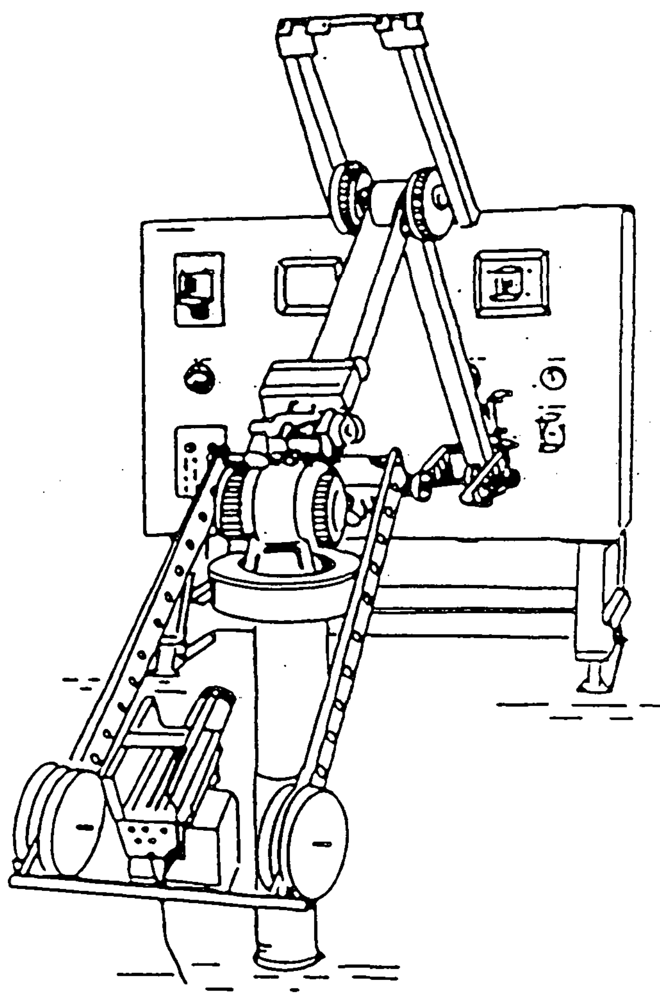
May 1990

Intelligent Robotic Systems Study (IRSS)

(NASA-CR-183963) INTELLIGENT ROBOTIC SYSTEMS STUDY (IRSS), PHASE 2 Final Report (Martin Marietta Corp.) 44 p CSCL 09B

N91-11432

Unclas
G3/63 0281603



MARTIN MARIETTA

MCR-89-559
Contract NAS8-36431

Phase II
Final Report

May 1990

INTELLIGENT ROBOTIC SYSTEMS STUDY (IRSS)

MARTIN MARIETTA
ASTRONAUTICS GROUP
P.O. Box 179
Denver, Colorado 80201

Contents

1 Introduction 1

2 Software Design 3

2.1 Robotic Systems Design Philosophy 3

2.1.1 Robotics Systems Qualifications 3

2.1.2 IRSS System Goals/Philosophy 4

2.2 IRSS System Architecture Philosophy 7

2.2.1 Layered Architecture 7

2.2.2 World Model 11

2.2.3 Communications 11

2.2.4 Decision Tables 12

2.2.5 Task Distribution 13

2.2.6 Control Flow 14

2.2.7 General Data Flow 15

2.2.8 User Interface 16

2.3 Implementation Details 19

2.3.1 Hardware 19

2.3.2 Layered Architecture 20

2.3.3 Tasking Models 22

2.3.4 VRTX Operating System 23

2.3.5 Communications 23

2.3.6 World Model 24

2.3.7 General Data Flow 26

2.4 Software Management 27

2.4.1 User Development Folder Distribution 27

2.4.2 Procedure Traceability Numbering 27

2.4.3 Global Variables 28

2.4.4 System Simulator 28

3 IRSS Control Electronics Hardware Description 30

3.1 Introduction 30

3.2 System Components 30

3.3 IRSS Controller Rack, Fans, and AC Power Distribution 31

3.4 Digital Computer System 31
3.4.1 Heurikon User Interface and Computational System 31
3.4.2 Data Acquisition I/O Chassis 33
3.4.3 Bit3 VMEbus to VMEbus Repeater Bridge 37
3.5 Peripheral Equipment 38
3.5.1 Force/Torque Sensor 38
3.5.2 Hand controllers 38
3.5.3 Matrix Video Switcher 38
3.5.4 User I/F Box 39
3.6 Servo Control Electronics 39
3.6.1 PWM and Rate Servo Cards 39
3.6.2 Brake Release Cards 40
3.6.3 Tach Filter Cards 40

1 Introduction

Under the Intelligent Robotics System Study (IRSS) contract, a generalized robotic control architecture has been developed for use with the ProtoFlight Manipulator Arm (PFMA) which resides at Marshall Space Flight Center (MSFC) in Huntsville, Alabama. Based upon the NASREM system design concept, the controller built for the PFMA provides localized position based force control, teleoperation and advanced path recording and playback capabilities. Various hand controllers can be used with the system in conjunction with a synthetic time delay capability to provide a realistic test bed for typical satellite servicing tasks. Figure 1 shows the configuration of the IRSS system.

The PFMA has six computer controllable degrees of freedom (DOF) plus a seventh manually indexable DOF, making the manipulator a pseudo 7 DOF mechanism. Because the PFMA was not developed to operate in a gravity field, but rather in space, it is counter balanced at the shoulder, elbow and wrist and a spring counterbalance has been added near the wrist to provide additional support. Built with long slender intra-joint linkages, the PFMA has a workspace nearly 2 meters deep and possesses sufficient dexterity to perform numerous satellite servicing tasks. The manipulator is arranged in a shoulder-yaw, pitch, elbow-pitch, and wrist-pitch, yaw, roll configuration, with an indexable shoulder roll joint.

Joints on the PFMA are driven via 7 pulse width modulated amplifiers(6 DOF + end effector). Resolvers and tachometers are used to measure relative joint positions and velocities. Motor drive currents are controlled via analog inputs to joint amplifier cards, and analog outputs are provided that feedback actual motor currents to the PFMA controller. Currently, there are two hand controllers for use with the PFMA: a compact rate (CAE) hand controller and a larger hand controller developed by Seargent Laboratories which has force feedback capabilities. Under the IRSS contract, a Lord Force/Torque Sensor has been added to the manipulator near its end effector to be used to implement various force-based control schemes and, at a later date, force reflection.

Digital control of the PFMA is implemented using a variety of single board computers developed by Heurikon Corporation and other manufacturers. The digital hardware architecture is comprised of four Heurikon V2F processor cards (68020 based) in a single 32 bit VME chassis connected to a second 24 bit VME chassis via a shared memory card. The first chassis (the computational chassis) performs all controls computations and operates the user interface. The second chassis contains all the electronics that interface directly to the PFMA. This chassis uses a Motorola MVME 104 processor card (68010 based) to perform all system data acquisition. Because of its function, this second chassis is called the Input/Output (I/O) chassis.

The IRSS controller is designed to be a multi-rate, multi-tasking system. Independent joint servos run at a 134 Hz rate and position based impedance control functions at 67 Hz. Autonomous path generation and hand controller inputs are

processed at a 33 Hz rate. System error response time is less than 20 ms. A real time operating system kernel called VRTX (Versatile Real Time Executive) is used to perform multi-tasking and handle real time interrupts.

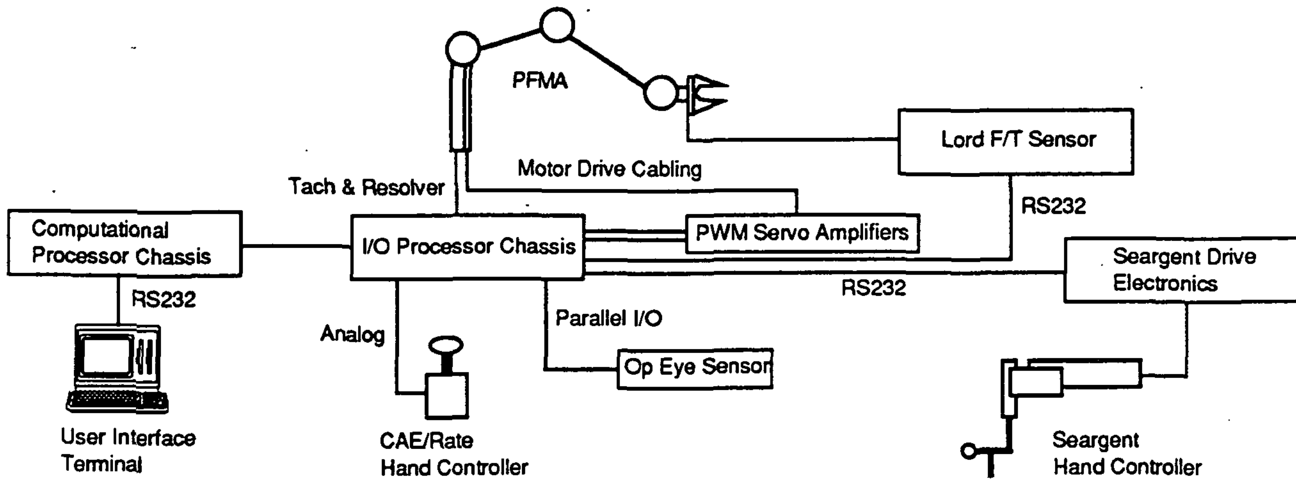


Figure 1. IRSS System Configuration

2 Software Design

2.1 Robotic Systems Design Philosophy

Within this section, abstract robotics system design goals and concepts are identified and their pertinence to and impact upon the IRSS system is evaluated.

2.1.1 Robotics Systems Qualifications

The IRSS system architecture is based loosely on the NASREM architecture developed by the National Bureau of Standards (NBS) for the Flight Telerobotic Servicer (FTS). Although not explicitly expressed, the NASREM architecture has as its primary aim the development of systems which exhibit the following features:

- Utility — Because of the sophistication and variety of tasks to be performed using robotics systems like the FTS, the implemented system must be capable. A utilitarian system should not obstruct its users ability to perform designated tasks or operations. Ideally, a system should actually augment its operators natural ability to perform tasks. Examples of how this goal might be realized are perhaps high servo rates resulting in "tight" joint servocontrol, sophisticated data analysis and prediction capabilities, and friendly system interfaces.
- Reliability — If a robotic manipulator is ever to be used to perform delicate tasks on expense satellites or space stations, it must be reliable. Inherently, the design of any system should encourage programming and system upgrade techniques that produce a reliable system. Reliable systems are repeatable systems that have controlled state transition and determinant state marking.
- Maintainability — With the majority of software costs not accruing during the development, but during the maintenance phase of a project,a satisfactory system must be organized so as to be maintainable. A system must be put together such that hardware and software modifications and updates have a controllable amount of overall impact and can be implemented quickly and reliably. This goal is achieved by well structured systems, organized in the methodical ways. Well prepared and accurate documentation also help increase system maintainability.

- Expandability — With the constant emergence of new technologies,algorithms, and requirements, a system designed in the 1980's needs to be flexible enough to accommodate growth well past the year 2010. Large monolithic systems that don't adapt well to change have again and again shown themselves to be expensive, cumbersome and often obsolete before they are even completed. An expandable system will have the "hooks" built into it from the start that make future expansion (both hardware and software) not only possible, but relatively easy and straight forward to implement. Modular hardware and software generally help enhance expandability of systems.
- Manageability — Manageability has to do with how easy a system is to create, update, modify and in general, work with. Manageable systems are those that are, for example, easy to debug, and have documentation procedures which are straight forward and natural to maintain. In a manageable system, whether a person is a servo controls expert or a novice implementing rudimentary path planning algorithms, information is readily accessible that allows him to segment and study the aspects of the system that interest him without having to mull through and understand every implementation detail of the entire system. Software components with loose, yet structured coupling tend to be much more manageable than tightly coupled systems where modules have vaguely defined interfaces.
- Understandability — Because it is humans that create, maintain and use systems like the FTS, they must be understandable. Understandability is not just the ability to conceive of a system but also to do so in a realistic manner. Because most systems must be used by individuals of divergent backgrounds, adequate systems must be structured so as to be clear and understandable to as many as people as possible.
- Divisibility — Useful systems are those that can be broken down multiple ways and into multiple parts (segmentable). Divisibility aids understandability, manageability and clarity. When data and control flows can be studied at many levels and in various ways, insight can be gained into the nature of all systems. Divisibility makes high level understanding easier and modification manageable.
- Affordability — Not only do all of the previously mentioned requirements have to be met, but they must be met in an affordable manner. Affordable systems will normally hold macro-efficiency as their primary goal as opposed to micro-efficiency.

2.1.2 IRSS System Goals/Philosophy

The IRSS system architecture has as its goals the same as those discussed previously. To achieve these ends, a variety of foundational principles have been

and were established early in IRSS system design. These rudimentary principles have been used to steer the main thrust of the software/hardware development effort. These principles were derived from past experience with intermediate size robotics hardware and software efforts and the examination of a variety of other system architectures which have been developed and/or implemented. The IRSS architecture is a blending of what is believed to be the best of all the competing designs. In the following sections, the foundation upon which the entire IRSS system concept is based is presented.

Object Oriented Design

Object oriented design is relatively old in concept, but new in technique having only become realizable with modern software design methodologies and sophisticated operating system tools. Fundamental concepts of object oriented design advocate structured programming techniques, data hiding and data abstraction. Object oriented systems can be described diagrammatically with data flows linking subsystems which perform well define operations on abstract data items. Control and data flow are generally isolated within these systems making them relatively easy to understand and debug. The IRSS system makes full use of object oriented design techniques.

Standardized Programming Techniques

Because of the importance of maintainability, reliability and expandability of the IRSS system, various general programming practices have been identified and used throughout the system software. It is believed that general adherence to these guidelines will produce the most efficient software system possible. These specific guidelines are described below:

- Descriptive Procedure/Variable Names — In general, procedure and variable names are as descriptive as necessary to illustrate their function and input/output characteristics. This reduces documentation costs (because code is readable) and makes maintenance easier.
- Clearly Defined Input/Output Characteristics For All Procedures -Tools and other procedures are defined by their input/output characteristics and general function rather than by their internal workings. This is a fundamental principle of object oriented design and programming. This encourages reusability and makes software easy to work with and understand.
- Separation of Control and Data Flow — Because control and data flow are clearly separated and all procedures have clearly defined interfaces, data flow is easily traced and problems which appear at a high level can quickly be traced to their low level source. Separation of control and data flow result in easy to understand control procedures which are series of if's, while's and subroutine calls and clean data manipulation procedures, each of which can be highly optimized by existing compilers.

- Short Single Function Procedures With Standard Constructs — Solid structured programming techniques are used throughout the system, and independent functions are distinctly separated to aid human understanding.
- Common Function Tools and Tool Boxes — Widely used functions are broken out as tools. This reduces code duplication, enhances reliability, and makes add on development much less costly and time consuming.
- Liberal Use of Literal Replacement Strings — Wherever possible,literals are used to enhance code readability and make logic expressions more understandable.
- Subroutine Grouping by Function and Layer — Similar procedures are all grouped in common files, sharing only modules level variables as opposed to global variables.
- Limited Global Variables — Global variables are used as little as possible, and those that are are normally abstract data types used in only explicitly defined ways.
- Functional Error Checking and Handling — As often as is reasonable,error checks are performed on input and output variables to procedures and errors are quickly identified and brought to the users attention.

Validity and Visibility

One of the greatest problems found in many computing systems is the lack of visibility into the internal working of the hardware and software. A system without this visibility is almost always difficult to validate and evaluate. Transient errors are virtually impossible to track and correct and new software capabilities are difficult to add. This lack of visibility occurs for a number of reasons: lack of computational power, unobservable intermediate hardware/software states, etc. Most often, however, this occurs because systems are not design to provide internal visibility.

The IRSS user interface has been designed to offer users a maximum amount of visibility for system monitoring and study. These capabilities are available for use not only during non-real time operation, but also when it is often most critical, while the system is powered up and running. These tools include user selectable and configurable menus that display important system variables virtually real time and advanced data recording capabilities. With halt, high and low level debug, and system stepping capabilities built directly into the system (not just added on) problems can quickly be isolated, repaired and validated.

Growth Capability

For many systems, it is acceptable to produce single use software and hardware components that are never modified after primary integration. In such systems growth capacity is not a major issue. The IRSS system was designed and built to be

prepared for growth. Because of this unique design, revision and expansion of the IRSS system can occur in a controlled manner.

Because of the uniform nature of the system control architecture found in IRSS, additional IRSS layers can easily be created and integrated into the existing system. To experiment with new ideas, new system functions merely need to be written as tasks interfacing to data that already exists within what is known as the world model. Task execution rates are user selectable (provided adequate processing power exists) and the movement of layers and tasks between processors is a relatively simple task. Ignoring physical bus bandwidth limitations, there is no limit as to how many or how few processors can be used within a system. This flexibility is made possible because of the data communication implementation used and the looseness of coupling between independent layers and independent tasks. In the matter of a few days, a user can distribute his entire system across the multi-processor environment completely changing task execution rates, task distributions, and data flow patterns.

2.2 IRSS System Architecture Philosophy

To achieve the sometimes lofty goals for a system as described in previous sections, standardized methods of dealing with system level problems must be established. Software and hardware concepts and components must be recognized and evaluated. The concepts and components foundational to IRSS are presented in the following sections. These are the parts that make up the IRSS system: A layered architecture, a world model conception, communications protocols, decision tables, task distribution protocols, control and data flow organization, the user interface, and multi-rate tasking.

2.2.1 Layered Architecture

A primary concept of NASREM is that of layering to achieve isolation. This idea is based upon the principle that the more a system can be cleanly subdivided and the subdivisions defined, the easier it is to understand. This principle is not unlike the idea upon which a dictionary or encyclopedia is based. Words are not randomly placed into a dictionary but are ordered according to certain rules making them easy to find once the rules of organization are understood (e.g. the alphabet). Similarly,it is believed that all functions of an advanced robotic controller can be sorted according to some defined rule base. Theoretically, any task, operation, or function that can be developed can be sorted according to the rule basis resulting in an extremely well organized system, no matter how large or complex.

As in the NASREM architecture, a layering structure exists within the IRSS system. The difference between the IRSS and NASREM systems lies in the granularity of the layering. In its original form, what NASREM calls its lowest two and a half layers, IRSS has subdivided into five. In the current implementation of the IRSS system, there are 7 layers total. Each of these layers have certain basic

characteristics that they share with all other layers. Every layer contains a decision table. A decision table examines the layers current status, the command coming to it from the layer above, and the status reported from the layer below and generates a command for the layer below. This can be thought of simply as a decision matrix which controls system operation and handles system state marking. Command input to the highest layer comes directly from the user via the user interface and command output and status input to the lowest layer are non-existent. Tasks associated with each layer are scheduled according the current status (or state) of the given layer. The frequency with which a task executes is determined within a scheduling table indexed by the frame number. The relative phase of all tasks is maintained using a frame numbering scheme.

The seven layers in the IRSS architecture (listed from highest to lowest level) are:

- System Up/Down Layer — The primary function of this layer is to provide a master control layer from which all commands are generated and to which all responses return. This layer is charged with seeing that the system is powered up and powered down in a repeatable and reliable manner. Data recording and safety are included in this layer.
- Multi-Segment Path Planner Layer — This layer manages tasks that handle multi-segment path manipulation (recording and execution)and generates intermediate level commands for the Trajectory Path Planner Layer which exists below it. More advanced AI path planners and/or vision systems would be integrated in through this layer. Tasks at this layer execute at relatively low frequencies (20 Hz) relative to tasks in other parts of the system.
- Trajectory Path Planner Layer — The trajectory planner has as its main function the transformation of an intermediate trajectory plan generated by the multi-segment path planner into a series of Cartesian poses which represent a distinct Cartesian motion. This layer can execute only one trajectory at a time after which it requests its new trajectory plan from the multi-segment path planner. The trajectory planner operates at twice the rate of the multi-segment path planner (40 Hz).
- Cartesian Engine Layer — The primary function of the Cartesian engine layer is to combine all pertinent Cartesian engine inputs to form a composite Cartesian commanded position. This composite will be made up of hand controller, force/torque and op-eye offsets as well as Cartesian trajectories coming from the Trajectory Path Planner Layer. Some components of this layer will operate at 40 Hz while others operate at 80 Hz. Note that all Cartesian components use quaternions to represent orientations.

- Joint Planner layer — The Cartesian to joint and joint to Cartesian transformations (forward and inverse kinematics) are found at this layer. This layer will produce commanded joint positions and current Cartesian positions and orientations at an 80 Hz rate.
- Servo Control layer — This layer combines the current commanded joint positions with the current joint angles (and tachometer data)to produce motor control voltages. During Phase II of IRSS first order compensators are being used to perform independent joint control.
- Data Acquisition layer — This is the layer through which all I/O data (other than user keyboard input) comes. Data provided by this layer are hand controller joint angles (via ADC and RS-232),resolved joint angles, motor velocities (tachometer -> ADC), force data, op eye inputs and parallel I/O. This layer also controls the DACs that command joint motions and control motor current limits.

Within each of these layers exist three major components: Decision tables,schedulers and tasks. Decision tables are associated with the system control flow within the overall architecture. As discussed earlier, they handle state marking and order state transition for the system as a whole. The primary purpose of schedulers are to see that tasks are run in proper sequence and at the correct relative frequency. Tasks, on the other hand, are in many ways very different from decision tables and schedulers in that they are the components that actually perform the data manipulations that make IRSS specifically a robotics controller.

Like the layered approach discussed for the overall system, each task is also built using a layering approach that aids understandability and clarity. This task layering concept is presented below (from the highest to lowest layers):

- Task Input/Output Layer — This layer is assigned the job of acquiring all task inputs from the world model, passing these as arguments to the Task State Transition Layer or Task Principle Control Layer, posting all task outputs to the world model, and providing all interactions with the VRTX multi-tasking operating system.
- Task State Transition Layer (Optional) — This layer (when necessary) handles high level task state transitions. An example of a decision at this level might be to transition between a zeroing of the force/torque sensor to *impedance control or the determination of whether the CRL or rate hand* controller task is to be run. This layer provides control at such a high level that it can determine which task inputs are to be brought in and which will eventually be output. This layer is normally found within the task but can sometimes be found within the layer's scheduler.

- Task Principle Control Layer — The Principle Control Layer has as its function the execution flow that implements groupings and execution orders of high level algorithms to perform specific functions.
- Task Secondary Control Layer (Optional) — Because certain problems naturally fall into multiple or high and low control layers, a second level of control is also made available. Note that this layer will be much "closer" to the data then the Task Principle Control Layer.
- Task Fundamental Algorithm Layer — This is the layer where data items are first manipulated to perform a required function. At this layer, algorithms tend to be relatively high level operations operating on relatively abstract data types.
- Task Primitive Algorithm Layer — This layer performs the low level, element by element algorithmic data operations. Procedures found in this layer normally come from tools or tool boxes.

Note that at one time or another, any of these layers could be non-existent, and that each layer may not be found in a single procedure. In some cases all of these layers could be in a single procedure or one layer could be spread across multiple procedures. This more a functional breakout meant to guide task design as opposed to a procedural break out that must be rigidly followed. Below is an example of how a typical task (a hand controller) might be broken down using this functional task layering approach.

- Task Input/Output Layer — Get hand controller gains and mode/status information.
- Task State Transition Layer (Optional) — Determine which of multiple hand controllers is in use and schedule task outputs to go to appropriate locations in the world model.
- Task Principle Control Layer — If deadman switch is depressed connecting the hand controller to the robot, then execute robot connected algorithms, else suspend the task.
- Task Secondary Control Layer (Optional) — Depending upon the specific hand controller mode (i.e. rate, position, etc.) execute high level algorithm grouping that modify hand controller commands to activate certain robot behaviors.
- Task Fundamental Algorithm Layer — Sequentially execute low level algorithms groups. Work on groupings of data to produce mode effects. All function calls come from hand controller toolbox.
- Task Primitive Algorithm Layer — Execute low level algorithms that operate on data element by element to produce desired mode effects. All functions come from quaternion and math tools.

2.2.2 World Model

Another concept primary to the NASREM ideology is that of a world model. A world model is an effective method of increasing visibility into the inner workings of a sophisticated system, with the intent of "opening it up for all to see". The world model concept is that of a large "blackboard" (found in memory or on some mass media device) which contains all system level state and data information which any task or process within a system can read. Like a blackboard, when fresh data is available old data is erased from the blackboard and replaced. Replacement of data is periodic and occurs according to a set of communications framing rules. The IRSS architecture embraces the world model concept and uses it effectively to enhance internal system visibility. Within the system implemented, the world model is distributed across multiple slave computational processor cards with ram buffers existing between separate chassis. A few simple rules govern access to the world model and determine its distribution across the processor group:

1. A task or process running on an individual processor card can only write its outputs to the section of the world model that exists within local RAM.
2. A task or process running on an individual processor card can read inputs from sections of the world model existing on any other processor card or itself.
3. Any world model variable will be modified by one and only one task or process, but can be read by infinitely many.
4. Any world model variable must always, after initialization,contain a valid data value (in type, scope and magnitude).

2.2.3 Communications

Within the IRSS system, communications are handled in much the same manner as outlined in the preliminary NASREM document, however, at a slightly slower rate (330 Hz as opposed to 1000 Hz). Periodically, all processors are simultaneously interrupted. This signals that a minor frame has begun and the communications portion of that minor frame is to be enacted. During the communications segment of the minor frame, all processors will simultaneously update the section of the world model which exists in their local memory space (Note that since each processor can only write to the section of the world model that exist within local memory there is no VME bus activity and thus no bus contention problems). It is important to be aware that not all global memory on a given processor will be updated during each communications frame, only those for which fresh data exists. During the update period, the world model is "locked out" (using software arbitration) and any attempted accesses of the world model will pend until the end of the communications frame. When each processor has completed updating the world model during the communications frame, it is free to return to the task that it was executing when the

start communications frame interrupt occurred. At the end of the communications segment of the minor frame, all processors are again simultaneously interrupt to signal the end of the communications portion of the minor frame. In general, the communications frame will need to be roughly 15% of the entire minor frame time.

2.2.4 Decision Tables

Within IRSS, one decision table is found for each of the hierarchical system layers. These decision tables are used to provide and insure orderly state transition within each layer of the IRSS system. Modular in design and implementation, these decision tables control all task creation and deletion (schedulers handle task scheduling). The scope of decision tables assigned to each layer are limited. A decision table can only respond to the the command coming from the layer above itself (its superior layer) and the status of the layer below (its subordinate layer). Decision tables will never use statuses coming from more than one layer below or commands coming from more than one layer above themselves. This causes an effective skipping of intermediate layers which makes determinate state transition practically impossible. With a system built as IRSS is, it becomes easy to segment and isolate state transition problems that are often difficult to find in less structured environments. In addition, systems segmented in this manner are easy to update and manage. Expansion comes by adding new system layers which have well defined inputs and outputs and whose domain of control is limited.

Examining decision tables in the IRSS system, one notices that each is broken up into four component parts. The first part is the data acquisition section. This section is comprised of procedure calls that read various state variables (such as superior layer commands and subordinate layer statuses) from the world model. The second component of all decision tables is error checking of the subordinate layer. Error states for any layer can be achieved if an inappropriate command coming to the layer is initiated or if a subordinate layer is found to be in a fault state. Using decision tables to control each layer, errors are quickly propagated upward and depending upon each layers state, various shut down procedures are executed. If no errors are found in the subordinate layer, the third section of the decision table is encountered which modifies the existing layers current status as well as the command going from it to the layer below. This section of the decision table is normally a large case statement which uses the command coming to the decision table as its selection argument. The final task performed by a decision table is the output of new command and status information to the world model.

Normally, the command generated by a given decision table will not change each time a decision table is evaluated. On the contrary, once a system is powered-up, localized task control will perform most of the robot control function while system level control acts as a safety net catching and recovering from unexpected problems. Most decision tables also have a relatively limited number of achievable states. Generally, most decision tables within the IRSS system will have only an initialize,

start-up, shut-down and an error reset state. This make decision tables smaller and more efficient.

Because the decision table for each layer is executed immediately after the communications segment of each minor frame, they are optimized for high speed. This is accomplished through hand optimization, maximized compiler optimization and use of specialized world model interface tools. Decision tables are run at such a high rate to retain robustness and insure rapid system response to changes in internal and external system states. Operating at such a high frequency, command and status propagation through the layered tree can happen at an extremely high rate. For example, when an error occurs in the data acquisition layer (the lowest layer) of the IRSS system, the system up/down layer (the highest layer) will be notified of the problem within 6 minor frames. With a minor frame time of 3 ms, the entire system will be alerted to any problem within a maximum of 18 ms.

With the structured system design under the IRSS contract, decision tables can be isolated from the system layered architecture and studied independently. It is normally quit easy to extract an IRSS layer from the system by removing its decision table, scheduler and tasks. This is due to the loose coupling between layers. Usually, a few simple drivers can be written that appear to the extracted layer as its superior and subordinate layers and it can be functionally examined. This can be done in real time on a slave computational card (to perform timing perhaps) or in non-real time using Unix (to debug tasks).

2.2.5 Task Distribution

The IRSS system can conceptually be viewed as five separate system level components: The user interface, the architectural skeleton,data processing, the world model, and the I/O processing subsystem. The architectural skeleton is what holds the system together and gives it form. Upon this skeleton, data processing is added (as tasks) to perform computations necessary to operate the system and drive the PFMA. The world model interconnects the various data processing elements and the I/O processing subsystem brings in external information for the system. The intelligence and command system is found in the user interface as it controls the methods and mode of data processing throughout the architecture.

In the IRSS system, the user interface function has been relegated to a single processor, this being the Unix processor. This was done not only because the Unix processor has established tools for user interface, but also because Unix cannot be used to reliably execute time critical software. The I/O processing function is also performed by but a single processor, this residing in the I/O chassis. This is the only function of the I/O processor: acquire hardware data. The architectural skeleton is distributed across all the computational processors and is formed by decision tables which handle overall state transition for the system. The tasks that make up the data processing components of the system are also distributed across the computational processors and constitute the bulk of the software. These tasks execute at various

rates and perform diverse functions depending upon the IRSS layer with which they are associated.

Within the IRSS system, the frequency at which a task will execute is determined by the layer of which it is part. In general, the lower in the layer hierarchy a task lies, the more frequently it will execute. Tasks and layers were allocated to processors so as to achieve an optimal load balance. To gain this load balance, relatively high rate tasks are distributed to the same processors as relatively low rate tasks. This distribution tends to produce optimal processor utilization.

2.2.6 Control Flow

Control flow within the IRSS system is found at two levels: System control flow and task control flow. System control flow deals with how the multi-processor, multi-tasking components work to function together as a whole. This control level is given the task of handling system power up and power down, and handling catastrophic system failures. System control flow is constituted by the group of 7 IRSS layers each of which responds and reacts to commands and statuses of the layers around them.

A brief description of how the IRSS system is powered up will help to clarify how system control flow operates. (Note that phrases which are capitalized represent actual layer states and commands found in software). To power the IRSS system up, the user begins by commanding a system power up via the user interface. This command is processed by the system up/down layer which will command the multi-segment path planner layer to initialize for start-up. Upon receiving this command, the multi-segment path planner layer will then check the trajectory planner layer status for errors, and if no error is found, will schedule an initialization task for the multi-segment layer. A transition occurs within the multi-segment path planner layer to an INITIALIZATION IN PROGRESS state until the initialization task is complete. During this time the system up/down layer simply waits for an INITIALIZATION COMPLETE state to be reached by the multi-segment planner layer. Until it is, no new command will be generate from the system up/down layer. (Not even a SHUT DOWN SYSTEM command can be issued until the system reaches an INITIALIZATION COMPLETE state. This keeps IRSS system layers from entering into impermissible/non-determinate states. Deterministic, repeatable, state transition is the key to achieving true system reliability).

With initialization complete, the multi-segment planner layer will then command the trajectory planner layer to initialize for start up and transition to a WAITING FOR SUBORDinate TO INIT STATE. The same sequence of status checking, initialization task scheduling and execution, and then command generation will occur at the trajectory planner layer much as it did at the multi-segment planner layer. This sequence of events will continue to occur at each layer as the INIT TO START command is propagated downward through the layered tree. This propagation will end when the bottom layer is reached.

With its initialization complete, the bottom layer will then transition to a INITIALIZATION COMPLETE state which will then be passed up through successive IRSS layers until the up/down command layer is reached. At this point a power down command from the user could be handled and the system could return to a SHUT DOWN state if desired. System shut down will occur in much the same manner as the initialization did however each layer will directly pass a shut down command to the layer below itself before it ever reaches a shut down state. These two examples illustrate how the IRSS architecture can easily adapt itself to sequential system level state transitions (as where each layer completes its own initialization before the next layer begins) or to faster (and often necessary) parallel state transitions (where the layers are all shut down virtually simultaneously). In this particular case it is important to note that power down is not complete until a completely shut down state is achieved by each and every layer within the IRSS system.

Power up is completed by the up/down layer issuing a START UP command. This command is immediately propagated down until it reaches the bottom layer. When this command reaches the bottom layer, a transition to start up task is scheduled, the layer waits for its completion, and then marks a START UP COMPLETE state. As the start up complete response is passed upward, each layer in turn schedules its own start up task and once the task is finished identifies that its start up is complete. System power up is complete when all layers have reach a START UP COMPLETE state.

Task control flow is a much more localized type of control and is concerned with program flow within a single task in a single processor. An example of this type of control might be a transition from on to off of an integration operation in a joint controller or a transition from a world to end effector reference frame in impedance control. As can be seen, this is a much more localized form of control than that described as system level control. Task level control is easily identified because it doesn't extend outside of the task layer in which it is found (other than by modification of some output variable going to the world model).

2.2.7 General Data Flow

For the IRSS system, data flow is a relatively easy to monitor because of the manner in which it moves through the world model. For all tasks within the IRSS system, all inputs will come from the world model and all outputs will be posted to it. In addition, all state variables reside within the world model. With the layering approach which is again subdivided into task components, inputs to and outputs from moderate size primitive blocks can been viewed as data flows through the world model. Because only one task or process modifies each particular world model variable, system errors can quickly be traced directly to their source.

2.2.8 User Interface

The user interface to the IRSS robot controller was designed to maximize utility while maintaining a structured and concise layered architectural approach. Keep in mind that the user interface discussed here is primarily via the system terminal, although a small camera control box does exist that is used to perform the pan, tilt and select functions for several cameras.

There are generally five types of displays seen while using IRSS. The first is the primary real time operations display. This is the display that the user sees when the system is first powered up. As can be clearly seen when viewing the terminal, this display is divided into five distinct areas. The first two areas are in the top half of the terminal and are called the status display areas. The local status display area is on the left and the world status display area is on the right. The local status display area constantly displays information pertinent to individual IRSS system layers. There is one display for each layer, and these displays are manipulated using the left and right arrow keys. The world status display area displays system information just as the local status display area does, however, parameters in this area area mixture of important global states variables. The format of this display window cannot be changed because it contains significant system wide information pertinent to all system layers.

Within the primary real time operations display there are also three other screen areas. The third is the message display window. The message display window lies below the local status display area and is used to display the three most recently received ASCII message strings coming from any processor in the system (including even the user interface). Because just three displayable messages is not many, within the debug options menu there is an option to display in full the last 100 messages received. Designed this way, the message display window on the primary display can alert the user that messages have been received, and if messages are received at a low rate, can actually display them while still maintaining the integrity of other system displays. The final two areas found on the operations display are the command windows. Much like the display windows (and existing directly below them), there is a local command window and a global command window. The local command window contains those user control options that are specific to a selected IRSS layer. This window of options is moved up and down the IRSS layer tree by using the up and down arrows found on the terminal keyboard. The global command window contains those menu options that are independent of IRSS layering and that must be available to the user independent of system layer.

The other types of displays found in IRSS are much less sophisticated. The debug display window is just a selection menu that allows the user to choose various sections of the world model for examination. From this menu the entire message buffer can be dumped and manual interrupts can be asserted. Also, IRSS BUG (a low level monitor type debugger) can be entered via this menu. When examining

data in the world model, data display windows are used. Within these, data can be displayed but, without proper access keys, cannot be modified. This is a safety feature that can keep unauthorized users from inadvertently corrupting the world model. The final type of displays are the parameter modification displays. These displays encourage and permit the user to quickly modify various system parameters. These displays are entered through selection of various options from the primary real time operations display. In addition to these types of displays, there is also a small menu tree within the multi-segment path planner used to edit various path buffers.

A small group of powerful user interface tools are used by IRSS to provide sophisticated data display and debug capabilities. With a structured, object oriented approach to system tool design, tools can be made easy to understand, compact and can use other available tools to share many common functions and thus reduce overall system size and complexity. The user interface tools are designed to compliment one another and are often used together to perform complex functions. There are three categories of user interface tools available: World model interface, real-time/system simulation and data recording.

The world model interface capabilities are the most often used user interface tools. The first and most prominent of these is the user menu capability. Using this tool, a programmer can quickly design and build variable size screen menus that permit data display and modification in a user friendly format. The data displayed and its arrangement is completely at the control of the designer, and helpful diagnostics tools are provided to aid debug. A second capability available is similar to the first, but is used to monitor system variables during real time operation. This function allows display menus to be created that will constantly be refreshed and updated. These are also totally user definable and easy to create and debug. Finally, a low level monitor type debugger is found in the IRSS system. Called IRSSBUG this tool allows users to literally look anywhere in the world model, displaying and modifying virtually any type of data during real time or non-real time operation. In addition, there are also two and pseudo three-dimensional graphics displays of items such as forces and torques for use during real time operation.

The real-time/system simulation tools help to provide rapid debug and validation capabilities. Using these tools, real time operation can be instantaneously halted and the world model interface tools used to browse the world model examining various system states. In addition,the capability to initiate single system wide frame interrupts is also available. Using this, relative timing and data flow between individual tasks, and world model updating procedures can easily be examined. and validated. This "single step" function is also useful in debugging and verifying system control flow.

Data recording capabilities provide utilities for analysis of high level system functions such as control laws and task motion compensation functions. With user definable data buffer sizes, various recording modes and the ability to record virtually

any data type, sophisticated research can quickly be implemented and documented with actual hardware data.

Power-up and power-down of the system is accomplished via the user interface using two primary functions. The first function is the halt function. This function controls the generation of real time interrupts. When the system is in a halted state, no minor frame interrupts are generated. With no interrupts generated the world model will not be updated periodically and decision tables will not be run making it impossible for the system to reach a powered up state. The second thing that must occur in addition to real time interrupts being operational, is that a command must be issue to the system up/down layer for power-up or power down. With the decision tables functioning properly a complete power-up sequence should complete in a few hundred milliseconds. To debug the decision tables directly from the user interface, a start-up command can be sent to the power-up/power-down task and manual interrupts asserted individually using the manual interrupt option in the debug window. This allows the user to view the power-up or power-down sequence in a frame by frame manner spotting decision table errors quickly.

Communication of user messages with the slave processors is accomplished via individual message queues which exist on each processor card. These queues can hold up to 50 messages per processor and are effectively a circular buffer managed by a leading/trailing pointer system. The user interface flushes each of the message queues every time it refreshes the display console. Within the user interface, up to 100 messages are buffered and time/processor correlated to provide maximum synchronization information.

Because the IRSS user interface runs in Unix on a non-real time processor, it is asynchronous to real time operations. This aspect of the system can cause minor user interface problems and dictates that certain procedures must be adhered to within the user interface software as it works together with the world model and real time system. These protocols and oddities between the user interface and the real time system are listed below:

- The user interface cannot modify certain variables used as inputs to some real-time tasks while the robot is powered up. An example would be joint control law parameters. The user cannot modify joint compensators while the robot is under computer control. The robot must first be powered down before control law modifications can occur.

- System level control commands coming from the user interface are handled via a request/response protocol. Using this scheme, when the system is to be, for example, powered up, the user interface will issue a request to power-up to a real time task running in the system up/down layer. This command is issued asynchronously. Upon recognizing the request to start the robot, the power-up task will execute a power-up sequence synchronously relative to the real time system. Upon proper power-up, the task in the system up/down layer which handled the power-up request will then issue a response to the user interface indicating that power-up is complete. This request/response method is effectively a method of synchronizing two asynchronous systems.
- Since the user interface processor does not acknowledge real time interrupts and observe communications frames, it can read data from the world model at virtually any time. This provides the user interface with the potential for having invalid data for display while real time operation is in effect (because a communications frame could be interrupted). This is a random problem and happens infrequently, and is virtually unobservable. Note that although data displayed in real time can possibly be wrong, any data directly modifiable by the user through the user interface will never be displayed or updated incorrectly. This due to isolation between the real time and non-real time system.

2.3 Implementation Details

In the following sections are found more specific details describing the manner in which the IRSS system has actually be implemented and used with the PFMA. The discussion of these details is not intended to be exhaustive, but does include all of the most important system level concepts that must be understood before a complete grasp of the IRSS system can be achieved.

2.3.1 Hardware

Often when the idealistic systems level concepts meet the real world hardware that significant compromises have to be made with regards to how the conceived system is actually realized, and IRSS is no exception. To implement the proposed design certain compromises did have to be made, however most of these compromises do not represent a significant deviance from the system as it was originally conceived. What will follow is a description of the major hardware components existing within the IRSS system and any special implementation details that should be noted.

The IRSS system has three computational processor cards (Heurikon V2F) which are memory mapped onto the Heurikon VME bus at the following locations:

- Board 1: 0xE0000000 — 0xE0100000

- Board 2: 0xE0400000 — 0xE0500000
- Board 3: 0xE0800000 — 0xE0900000

Each board has 1 Meg of local RAM, a 68881 co-processor, and a 68020 MPU using a 20 MHz clock. None of the computational processors have memory management units. The majority of onboard I/O functions are handled by a 68901 Multi-Function Peripheral (MFP) chip found on each board. The MFP controls the serial port, four timers, the processor board interrupts and baud rate generation.

All hardware data acquisition is performed in the I/O chassis. Using a Motorola 104 processor card (with a 68010 at 16 MHz) this card refreshes resolver and ADC data at a 160 Hz rate, and force/torque data at a 80 Hz rate. DAC outputs are refreshed at 320 Hz.

The I/O and computational chassis are interconnected using a bus extender card called the Bit3™. Initially the computational chassis was set up as a master that would read data directly out of the I/O processor card's memory, however, spurious bus errors occurred and a bus buffer card had to be added between the two chassis to insure system integrity. The Bit3 card is mapped into the computational chassis VME bus at addresses 0x01ED0000 — 0x01EE0000. It is important to note that because of the configuration of the Bit3 card, access of the I/O chassis common memory buffer by a computational processor while that processor is in a supervisory state will cause the Unix kernel to panic and crash. All accesses of data via the Bit3 must occur while the processor is in its user state.

Currently it is not believed that the V2F computational processor cards can access the Unix card's memory using the VME bus. This has facilitated the need to place the Unix portion of the world model on one of the slave computational processor cards (specifically on card 2). This violates the canon that each processor card will only write to local memory but was necessary to realize the actual system. In practice, no problems have yet been traced to this deviance from the standard architecture and a net system speed up may occur. Card 2 was selected to hold the world model because of the moderate rate tasks running on it and the lack of path buffering and storage areas already existent in memory.

Using the Green Hills C optimizing compiler found on the IRSS system a significant speedup is seen in comparison to the Unisoft CC5.0 compiler. The difference is due to two primary reasons. First, the green hills compiler is designed to be run specifically on the 68020 processor using its floating point co-processor. CC5.0, on the other hand, does not use the full 68020 instruction set and does not use the co-processor. Secondly, the Green Hills compiler has much more sophisticated code optimizing than does the CC5.0 compiler.

2.3.2 Layered Architecture

The layered architecture is realized as decision tables for each layer schedule and create tasks that cause data to be manipulated so as to produce desired robot

motions. Evidence of a clean layered architecture is the loose coupling between layers, repeatable performance of independent layers.

Inter-layer Communications

To help maintain system integrity each layer within the IRSS system should be thought of as having a limited world model scope (or impact). This means that each system layer will and should operate upon only a segmented, identifiable section of the world model. In practice, this means that certain state and data variables used within a particular layer are not shared with other layers but are maintained solely for the use of that layer. In addition, all other variables may be shared by two adjacent layers but by no others. In general, there should be no intermixing of variables accessible by a given layer with variables accessible by layers more than one layer away (other than in the case of certain hardware data). A system built this way will generally be much easier to maintain and understand because the conglomeration and intermixing of high level functions and low level functions is eliminated. An example of a poorly designed system that violates this canon would be a Cartesian path planner that would need to know what control law parameters were being used to determine what types of segments it can execute. A user would now have to understand how the servo control layer works and how and when state transition occurs for it to understand the Cartesian path planner. As can be seen in this example, high and low level functions are interwoven and thus produce a system that is difficult to understand (let alone test and debug). Use of such unorthodox programming practices are strongly discouraged by the IRSS system architecture.

Data Hiding

All tasks have two general types of data that they manipulate: Local and Input/Output. Tools and tool boxes have four data types that they can manipulate: Local, Internal, Tokens and Input/Output. The difference between tools and tool boxes are that tools are generally single function algorithms which are not inherently linked to other procedures each of which work together to provide a general functionality. All components of a tool box must be used to together or none are useful. Tool boxes can be described as component subsystems and tools as algorithmic procedures. A relatively easy way of determining whether a function belongs as part of a procedure or a tool is that tools don't normally deal with tokens, or need to be configured or returned after use.

A description of the types of data found within procedures would be useful. Local data is data declare at the procedure level, allocated on the stack that disappears once a procedure completes execution. Input/Output data is data coming into or going out of a task to the world model (it doesn't necessarily have to be Hardware I/O data). In the case of a tool, Input/Output data is simply data that is passed in or out through the procedure header. Tokens are identifiers used in tool boxes as reference keys for allocations of descriptive functions (i.e. upon configuration, the filter tool box passes back a token for a filter with user defined characteristics).

Finally, internal data is comprised of type "static" data that is resident all during program execution.

Data hiding occurs in a number of ways throughout the IRSS system. In their design, tasks are first examined in light of their world model input/output behavior, identifying the necessary world model data needed to perform their delegated function. All of this data is defined as the tasks input/output data. All other data used by a task is defined as local and, in general, never finds its way to the world model and is never available for users to see. This restricts the size and scope of the world model keeping it manageable from an intelligibility point of view.

Tools and tool boxes are also designed in terms of their interface characteristics and generic functions. Provided that a tool is properly debug and used, those utilizing a tool should need to know nothing about its internal workings. Tool boxes in general have four distinct components through which a user interfaces with them. First, the user must usually initialize the tool box. This sometimes means allocation of work areas for the tool (like a recording buffer for example) and entry of system level characteristics into the tool (i.e. sampling rate). Second, functions are generally allocated from the toolbox and a token is used to access that function (i.e. to record a group of variables at a given rate and in a given manner, a recording entry must be made). During the execution phase of tool box use, the token returned during the configuration phase is used to perform a specific user defined function on user selected data. Finally, once a particular function is no longer needed or needs to be reset (as in the case of previous states for a filter) the token is then used to reallocate resources used by a function (i.e. a portion of memory used for data recording is returned to the memory heap). Tools on the other hand are just generic functions that don't need to be initialized, configured, or relinquished (i.e. a 3x3 matrix multiply operation in the math tools).

2.3.3 Tasking Models

The tasking model for IRSS can be thought of as a four layered system. The first part is the interrupt control and synchronization layer. Without the interrupt control layer generating minor frames, the IRSS system is dead,not being able to change states or schedule tasks. Below the interrupt control layer resides the system control layer. Within this layer all system level state transition occurs. System states attained within this layer directly influence the system scheduling layer which lies just below. The system scheduling layer determines task execution rates and relative task phasing. Within the main tasking model, neither the system control layer nor the system scheduling layer should be considered as tasks, but rather offspring of the interrupt control layer. The final layer is the tasking layer which manipulates world model data to accomplish robotic functions. Although they appear to be independent all components of the task layer work together to accomplish a specific operation. Any tasks that do not work toward this common goal are extraneous and represent flaws within the system.

2.3.4 VRTX Operating System

VRTX is a compact, operating system kernel that was design for use in high speed systems like IRSS. VRTX promises interrupt servicing latency of less than 50 micro seconds and tasking context switches in less than 60 micro seconds. The IRSS system takes advantage of just a few of the capabilities offered by VRTX, namely, multi-tasking and interrupt servicing. VRTX is used to set up and control tasks on each board.

2.3.5 Communications

Communications is assigned the responsibility of seeing that data needed by various processes is available and valid.

Synchronization

For reliable real time operation, it is important that there exist some type of synchronization between all processors, tasks, and events within a system. In IRSS this is achieved by using VME bus interrupt 4 to notify all processors and chassis when a minor frame is beginning and when a communications frame is ending. These two interrupts are generated periodically by V2F computational board 2. Board 2 determines the time between minor frames by using a hardware timer found on its MFP. A normal minor frame start is initiated when this timer produces a local on board interrupt to the 68020 processor. In the timer interrupt service routine, the processor board uses the MFP to generate and then resend the VME interrupt 4. All processors in the system (except for the user interface Unix processor, and the board generating the interrupt)will respond to this interrupt almost simultaneously, providing system wide synchronization. After processor card 2 has asserted the VME minor frame start interrupt it then will set-up a timer that is used to time the communications frame and will then begin its communications frame. When the communications frame timer count reaches zero, the MFP will then again generate a local interrupt on board 2. This time the service routine will again generate a VME bus interrupt notifying all processors that the communications frame has ended. Upon receiving this second interrupt all processors will re-enable the world model for use and will begin executing their decision tables and schedulers.

Framing

Within the IRSS system there are what are called minor frames and major frames. Minor frames are short periodic segments of time which are made up of several component parts. For the IRSS system, minor frames are normally about 3 ms in duration. Major frames are made up of multiple minor frames grouped together. What makes major frames and minor frames different is not only their size, but also aspects of their periodicity. Within a group of minor frames, tasks, as they are scheduled in the multiprocessor environment will be staggered relative to one another to achieve optimal data flow. For example, when a system is started up, in the first minor frame force data may be read and filtered. In the second, the force

data could be used in a position compensation algorithm, and on the in the fifth minor frame the output of the force control algorithm might be used as a commanded Cartesian offset. As is obvious, there must be a phase relationship between these tasks for them to function properly. Minor frames make task phasing possible. Staggering tasks works well within the minor frame concept but there comes a time when the relative phasing of tasks needs to be repeated so that each basic operation can be done again and again. This periodicity of repeating the same relative phasing between tasks gives rise to the major frame. Within each major frame the same number and relative phasing of various task is always equivalent. Major frames are repeated over and over again to produce desired robotic functions. Within the IRSS system 16 minor frames make up one major frame.

Access Tools

The types of tools used to access the world model are dependent upon whether the access is in a synchronous fashion or an asynchronous fashion. To access the world model synchronously, four separate procedures are available: two for reading and two for writing. The two input routines are get-task-inputs and get-state-variable. Both of these procedures observe communications frame lockouts and thus will always return with valid data. Post-task-outputs and post-state-variable are the two output routines used to move data through a temporary queue to the world model. Both get-task-inputs and post-task-outputs are used to move variable size blocks of data. Optimized for high speed use in decision tables, get-state-variable and post-state-variable are used to access and update only type "short" system state variables.

To access the world model in an asynchronous manner two methods are used. First, when the user interface wants to pull a buffer of data from the world model to modify it, or wishes to write a buffer to the world model it will use get-world-model-data and post-to-world-model. Since neither of these routines check whether a communications frame is effect they cannot be used to access data that the user interface does not have exclusive modify rights to. These routines appear to function just as get-task-inputs and post-task-outputs but differ only in the fact that they do not observe communications framing protocols. A second and very simple way of accessing the world model asynchronously is by using the procedure assign-pointer. This procedure will assign the world model address of a user selected variable to a pointer which can then be used to access that variable. All of the display menus use this routine to assign pointer so that they can extract information directly from the world model. Special care must be taken when using assign pointer to observe sound programming practices, seeing that code remains understandable and that world model pointers are not corrupted.

2.3.6 World Model

Implementation details of the world model are presented below.

World Model Distribution

With each processor card containing a section of the world model,organizational rules are established to maintain clarity and graceful utility. For each of the computational processor cards the world model is placed in the upper most 128 K of ram. Because each card has one megabyte of memory, the world model exists in the 14th and 15th pages of memory. Since local ram on each card begins at megabyte 32 (0x2000000) the onboard blackboard addresses for the world model are from 0x20E0000 to 0x20FFFFF. This 128 K block of memory is divided into 8 separate areas each of which contains a specific type of data.

A brief explanation of the data types found in each area is in order. All data that flows between tasks can be found in the task input/output areas. To complement this area, a task internal status information area has been created. Within this area individual tasks are permitted to write internal task information that does not constitute direct task input and output data. An example of the type of data found in this area would be control law internal state information made available for recording and analysis, but not used as input to other control blocks within the robotic system. All layer status and command information is found in the command input and status output areas. These areas also contain requests and responses coming to a layer on a given processor from the user interface. The control flag area holds all control variables for use in task level control and the response output area is a general purpose area for reporting system response information. Error condition reporting areas have not been widely used yet, but are available to identify error conditions encountered during processing and state transition.

Assurance of World Model Access Validity

Within the IRSS system, a special queueing method is used to insure that the world model is only updated during the communications portions of each minor frame (the communication frame). As discussed earlier, tasks use special tools to access the world model. One of these tools, called post-task-outputs, is used to output data to the world model from a task. So that tasks upon completion don't have to wait for a communications frame to occur to output their data to the world model, this procedure can be called at any time during a minor frame. Although it appears to the user that task data is immediately moved to the world model, output data is actually just being moved to a temporary queue where it is held until the next communications frame. Each time a communications frame occurs this buffer is flushed to the world model. This queueing method works well although there are several problems which must be addressed before it will function reliably.

To assure that the beginning of a communications frame interrupt does not disturb the movement of data to a temporary queue entry, a special logic arbitration scheme is used. Two flags are examined while data is being moved to the world model temporary queue: COMM-INTERRUPT and IN-COMM-FRAME. Each of these flags is a Boolean. The first, COMM-INTERRUPT, identifies that a communications

interrupt has occurred. The second, IN-COMM-FRAME, states that the communications portion of a minor frame is in effect and that the world model is locked out.

This logic protects the user from two possible errors that can occur while writing to the temporary storage queue. The first error would occur if a communications frame interrupt occurred while data was being moved to the temporary storage area. This would produce a bad queue entry (one where invalid queue data would be present). The second problem is that if a communications interrupt occurs after all data has been move to the temporary storage area but before that data is identified as valid. In this case the communications interrupt can fail to dump fresh data to the world model, when in fact it was ready for output. The logic scheme described above will eliminate both error possibilities. The software locking scheme used for reading from the world model is similar, although somewhat easier to follow.

The only error that can occur while data is being read from the world model is that a read can be interrupted causing data that is input to be partially from one minor frame and partially from another. In both cases note that output and input are not permitted while a communications frame lockout is in effect.

2.3.7 General Data Flow

IRSS User Interface Layering

As the user scrolls through the layer dependent menus in the lower left hand corner of the primary user interface menu he will move up and down through the IRSS layer architecture. The user interface was designed to be modular and to handle input errors at local input layers, reducing the amount of error checking that has to go on at high levels thus making the code more readable and increasing reliability. The user interface can be thought of as a 6 layered system. The upper most layer is the executive which is passed through each time a layer dependent command window is changed. This layer gives cohesion between the independent layers and determines their hierarchical order. The second user interface layer is the IRSS layer dependent menu layer. It is at this point where all commands specific to each layer are handled. Below this layer lies the menu handler. This layer will handle all layer independent commands and pass all layer dependent commands to the layer above. Next comes the screen handler layer. This layer handles the console display, checks message queues, accepts keyboard inputs and performs command input checking. When valid user commands are found, they are propagated upward until one of the top three layers handles them. The next layer below the screen handler is the global debugger and below that is the low level debugger, IRSS BUG. These two lowest layers are in some sense functions of the screen handler but are unique enough to be broken out explicitly.

2.4 Software Management

This section covers special aspects of the IRSS system which have not been discussed under the major components sections found previously, but which have a great deal of impact on the management of the IRSS system. Documentation organization and system simulation are two topics found in this section.

2.4.1 User Development Folder Distribution

The IRSS system is described using three Unit Development Folders (UDFs): Controls, user interface and tools. Each of these UDFs contain system design and documentation as well as algorithms, cross referencing information and software. The controls UDF contains all the information that has to do with system wide control and tasks used in the real time system. The tools UDF contains all the information about all real time tools and tool boxes within the IRSS system, and the user interface contains all software used to interact with IRSS users. With regards to overall size, the tools UDF has the fewest number of lines of software absorbing about 20 % of the total. Both the controls and user interface each constitute about 40 % of the system software. The IRSS system is estimated to be roughly about 16,000 line of 'C' code.

2.4.2 Procedure Traceability Numbering

To help organize the software work environment and provide traceability for procedures within IRSS a procedure numbering system has been implemented. All procedures used within the system have a specific identifier associated with them. These identifiers are made up of several characters followed by a series of digits. There are three possible characters that will prefix a procedure identifier number: *ctl* — for controls, *tl* — for tools, and *usr* — for user interface. Each of these identifying strings indicates which of the Unit Development Folders a particular procedure is a part of.

In the case of the user interface and controls procedures, the digits following the first characters identify the layer and subsystem in which each procedure is found and used. All user interface and controls procedures have a four digit number following their leading characters. The first digit indicates whether a procedure is a controls (digit 3) or a user interface (digit 2) procedure. The second digit identifies the layer in the current architecture with which the procedure should be associated (1 = Up/Down Layer, 7 = I/O layer). The third digit identifies the grouping or series of a given procedure (i.e. 50's series could be all impedance control procedures). The last digit identifies the specific procedure within the series that is being examined.

The identifier numbers used with tools are similar however they sometimes have five digits. The first digit (1) identifies that a procedure is a tool. The last two digits identify the specific tool number and the second and sometimes third digits identify the particular tool collection or tool box from which a procedure comes.

2.4.3 Global Variables

The use of global variables is limited within the IRSS system. In general, the use of global variables is unnecessary and merely makes software more more difficult rather than easier to work with. There are, however, cases when global variables are acceptable. Such cases are listed below:

Variables Accessed by Interrupt Service Routines — The variables used by both foreground and background tasks must be made public. Knowing this to be the case, as few variables as possible are modified in interrupt service routines. Two important variables to note are COMM-INTERRUPT and IN-COMM-FRAME and task mail boxes.

Task Mail Boxes — Task mail boxes are made public because of the consistent way in which they are used (and because they must be because they are used in interrupt service routines). Task mail boxes are always written to by schedulers and operated upon by tasks, or written to by initialization or shut down tasks and operated upon in decision tables. There are no other ways that mail boxes are used.

System Wide State Variables That Don't Change During Execution — In most programming languages these would be called DATA statements. These are static variables that are assigned a value at compile time or once and only once during code execution.

Abstract Data Types Used By Tool Boxes — These are data types,defined in tool boxes, that are not manipulatable by software other than by the tool boxes that created them (i.e. tokens). No arithmetic operations can be performed on these data types,and they cannot be used as arguments in expressions. Only comparisons can be performed on these variable types. The only way that these data types are used are as calling parameters passed to tool box subroutines.

(NOTE: The global variables for each processor are declared together in a single file call init-procN.c (where N is the processor number).

Although in each of these cases, global variables are permissible,if at all possible, an attempt to avoid them should be made. Global variables can quickly contribute to the downfall of any system. With good software layout and data structure design, the need for global variables can almost always be eliminated.

2.4.4 System Simulator

To help debug IRSS, a system simulator has been developed that can be used to emulate the multi-processor, multi-tasking and multi-rate environment encountered when IRSS is downloaded to slave processors. Although the simulation is non-real time, the fact that it is all done within the Unix operating system permits it to be very flexible yet powerful. Using the simulator, task execution can be performed one line at a time using standard Unix tools. With this capability,programmers updating the IRSS system can ring out virtually all of the non time related bugs of a system well before a robot is ever turned on. In addition, the IRSS debugger is sophisticated

enough that joint compensator design, modeling, and data recording can be done within the simulator (provide an adequate plant model is available). This can eliminate time wasted translating and validating code as when real time operation and simulation happen on separate machines.

The power and simplicity of the IRSS system simulator is due to the decoupling that exists between the various system control layers, tasks and user interface. Since all user interface requests and responses come through the world model, the user interface and real time software can be run sequentially rather than in parallel as is done on the real time system. With all task data being passed through the world model, task execution can also be performed sequentially without any special accommodations having to be made. An example of the flexibility of the simulator is the fact that most procedures normally don't even need to be recompiled before they are moved from the system simulator to the real time processor cards.

When using the IRSS system simulator a complete world model is maintained and used by the system. Because the world model is not mapped across the VME bus and several processor cards but rather in Unix,a special module of world model interface tools and VRTX tools are all that is needed to convert the real time system to the simulator system.

3 IRSS Control Electronics Hardware Description

3.1 Introduction

This section provides a description of the electronics associated with the IRSS. Only basic characteristics of the PFMA are included in the System Components paragraph. The remainder of the section concerns components integrated into a two bay 19 inch rack. This rack and a few peripheral items are referred to as the IRSS Controller.

Detailed configuration information is provided for reference. If greater detail concerning system operation or configuration is required, it can be found in the vendor and/or Martin Marietta documents cited.

3.2 System Components

The PFMA has six computer controllable DOF plus a seventh manually indexable DOF, making the manipulator a pseudo seven DOF mechanism. Because the PFMA was not developed to operate in a gravity field, but rather in space, it is counter balanced at the shoulder, elbow and wrist and a spring counterbalance has been added to help counter gravity. Built with long slender intra-joint linkages, the PFMA has a workspace nearly two meters deep and possesses sufficient dexterity to perform numerous satellite servicing operations. The manipulator is arranged in a shoulder-yaw, pitch, elbow-pitch, and wrist-pitch, yaw, roll configuration, with an indexable shoulder roll joint. Dual-path spur gear drive trains combined with DC motors are used for joint actuation. Resolvers and tachometers are used to measure relative joint positions and velocities.

Motor power, tachometer and resolver signals are routed to the Servo Electronics Module through cabling and breakout boxes. Each joint has a Pulse Width Modulation (PWM) amplifier located in the Servo Electronics Module. Internal to each PWM amplifier is a limiting analog rate loop servo using the joint tachometer.

The position servo loop for each joint is closed by a digital computer system. Feedback and feedforward signals are monitored by the digital computer through an Analog to Digital conversion (A/D). The computer generates command voltages for the motor amplifiers through a Digital to Analog Conversion (DAC).

Currently, there are two hand controllers for use with the PFMA: a compact rate (CAE) hand controller and a larger, more dexterous hand controller developed by Seargent Laboratories (CRL) which has force feedback capabilities. Under the IRSS contract, a Lord Corp. Force/Torque Sensor has been added to the manipulator near its end effector to be used to implement various force based control schemes.

The IRSS Controller has capabilities and hardware that will be utilized in future implementations. This section will primarily document the IRSS hardware integrated at the close of phase two.

3.3 IRSS Controller Rack, Fans, and AC Power Distribution

Each bay of the two bay rack will hold 52 vertical inches of standard EIA 19 inch rack mountable chassis. Considerable room exists for future expansion, both in the front and rear of the rack. Two Optima blowers with 350 CFM of air flow each have been installed in the rack to provide cooling airflow. Most OEM components have internal fan assemblies to meet specific cooling needs.

Primary power for the system is facility 115 VAC. All components requiring AC power have standard AC power cords going to a single terminal strip attached to a rear vertical rail of the rack.

3.4 Digital Computer System

The digital computer system is comprised of three primary components:

- The Heurikon User Interface and Computational System.
- Data Acquisition System (I/O chassis).
- Bit3™ VMEbus to VMEbus repeater bridge.

3.4.1 Heurikon User Interface and Computational System

For the scope of this section, the Heurikon system is an ``open system" VMEbus based system. Martin Marietta Drawing 849IRS10002 shows a good functional view of the Heurikon system. Vendor documentation of the Heurikon hardware includes:

- Heurikon HSE/17 Enclosure Hardware Manual
- Heurikon 120 Schematic Diagram
- Heurikon HK68/V20 Users Manual
- Scientific Micro Systems OMT15000 SCSI Reference Manual
- Mitsubishi Electric Corp. 5.25 inch Flexible Disk Drive, MF504B-3 Manual
- Archive Corp. Scorpion 1/4 inch Streaming Tape Drive Reference Manual

The Heurikon chassis requires 17.5 vertical inches of 19 inch rack space. Fans, power supply, card cage, Mass Storage devices, and I/O controllers were Integrated into the System by Heurikon. Four Single Board Computers (SBC) have been configured and added to the system to meet the requirements of the digital computer. The power supply is a 500 watt unit, operating at approximately 50 percent duty. The card cage contains twelve standard VMEbus card slots numbered zero thru eleven.

Slot zero is at the bottom of the cage, and has the highest priority for VMEbus arbitration. The Heurikon system supports single level (level 3) bus arbitration, and all potential bus masters and so configured. Refer to the HK68/V20 User's Manual section 10.2 for details. The Heurikon VMEbus backplane supports 16 bit (short), 24 bit (standard), and 32 bit (extended) addressing schemes. Greater detail concerning these addressing schemes is available in sections 2.22 and 2.3.5 of the VMEbus Specification Rev C.1.

Heurikon system mass storage devices include a 86 megabyte hard disk drive, a 1 megabyte floppy drive, and a 1/4 inch streaming tape drive for back-ups. These mass storage devices are controlled by a peripheral control board using the SCSI standard. This OEM controller is manufactured by Plessy Corp., and is installed into slot two of the card cage.

Installed in slot six is a XYCOM 400. This board provides four additional RS-232 ports for the system. Internal cabling connects the XYCOM 400 to four connectors (female DB-25) labeled TTY0 through TTY3 on the back of the Heurikon chassis. Devices connecting to these RS-232 ports are:

- TTY0 — Not used
- TTY1 — Not Used
- TTY2 — Download to the I/O Chassis
- TTY3 — PRINTER

Unix User Interface Processor

The User Interface processor (HK680/V20) is installed in slot zero of the card cage. Components of the processor extend beyond the normal card height, thus slot one is unusable. This 20 MHz 68020 based SBC has four megabytes of Dynamic Random Access Memory (DRAM), and support chips including:

- 68881 Floating Point Coprocessor
- 68851 Memory Management Unit
- 68901 Multi-Function Peripheral (MFP) — containing a serial port and four timers

Non-volatile static Random Access Memory (RAM) contains configuration information in 256 four bit words. Reference the Heurikon HK68/V20 User's Manual Section 9.7 for details.

The User Interface processor has HBUGS V20 firmware in a single vendor supplied Read Only Memory (ROM) chip. This bootstrap/monitor firmware loads Unix from the hard disk. The user may then initiate downloads to the other Heurikon processors via the shared VMEbus; or to the I/O chassis processor via an RS-232 link. On board ROM capacity of 128K bytes is provided by a second socket also configured to accept the 64K byte ROMs.

Heurikon chassis internal cabling connects the MFP serial RS-232 port to a connector (female DB-25) labeled CONSOLE adjacent to the TTY0-3 RS-232 ports on the back of the Heurikon chassis. This port is the main Unix user interface for a terminal.

Computational Processors

Each of the three computational processors (HK68C/V2F) is a 20 MHz 68020 based SBC with one megabytes of DRAM. Support Chips and nonvolatile RAM are identical to the user interface processor, except these processor do not have, the 68851 Memory Management Unit. For these processors, all onboard RAM is dual ported to the VME extended address space. The Computational Processors will also accommodate 128K bytes of ROM firmware. These processors however have the real-time operating system VRTX by Ready Systems Corp. installed as firmware in a single 64K byte ROM site. The MFP Serial Port for each Computational Processor was only used during debug, and is presently not connected to a device.

Computational Processors are installed in the following slots:

- Board number two - Slot five
- Board number one - Slot four
- Board number zero - Slot three

Computational Processor board zero alone has been configured to provide the System Bus Clock Signal.

3.4.2 Data Acquisition I/O Chassis

This Motorola (MVME 940-1) VMEbus system requires 10.5 vertical inches of the rack space. The power supply is a 200 watt (MVME 940-3) unit. The card cage contains eight standard double width VME slots, and ten single width I/O card slots. Martin Marietta drawing 849IRS20000 details the I/O chassis assembly, with card location information and general appearance. More detailed information about the card cage or processor card may be found in vendor documents:

- MVME 940-1 Chassis/card Cage User's Manual
- MVME 10X Series of SBC User's Manual

Motorola Microcomputer

The I/O Processor (MVME 104) initializes and controls data acquisition boards in the I/O chassis. This SBC is based on the 68010 and has 512K bytes of DRAM. One RS-232 port connector (female DB-25) is present on the front panel of the card. This serial port is used for downloads from the Heurikon system. A Centronics printer port is also available on the front panel. A serial RS-485 port is available on the P2 connector of the card. An 68230 programmable timer is used to implement a local bus time out of 115 microseconds maximum, and a watchdog timer of 15 milliseconds. The factory jumper configuration is mostly unchanged. Refer to the

MVME 10X series of SBC User's Manual Section 2.3.1 for jumper information. Martin Marietta drawings 849IRS20200 and 849IRS20202 provide configuration information, and a functional block diagram.

The I/O Processor will except four 64K byte ROMS for a total of 256K bytes. Presently a single bootstrap/monitor ROM (MVME 105 Bug 2.0) is installed. This firmware initializes the processor and waits for a download from the Heurikon System.

Resolver to Digital Conversion Boards

Two VME Microsystems (VMIVME-4940-04-06) Resolver to Digital (RTD) boards have four 14 bit resolver to digital converters each. One board has short I/O address 0x0100. The other board has short I/O address 0x0200. The I/O Processor will physically address 0xFF0100 and 0xFF0200 to obtain the VMEbus short address defined above. These boards convert the resolver angles to digital values for the following joints:

- Board 1 (0100)
  1. Elbow Pitch
  2. Wrist Yaw
  3. Wrist Pitch
  4. Wrist Roll
- Board 2 (0200)
  1. Shoulder Yaw
  2. Shoulder Pitch
  3. Not used
  4. Not used

More detailed information can be obtained from the VMIVME-4940 Instruction Manual.

Digital to Analog Conversion Board

A DAC board (VMIVME-4100) converts digital commands into 16 independent analog voltage outputs. The board is accessed in VMEbus short I/O address space 0xD000. The I/O processor will physically address 0xFFD000 to control this board. Detailed programming information is found in the VME Microsystems VIMVME-4100 Instruction Manual.

The 16 DAC outputs are assigned as follows:

- Channel 0 — Shoulder Pitch Command
- Channel 1 — Shoulder Yaw Command
- Channel 2 — Elbow Pitch Command

- Channel 3 — Wrist Yaw Command
- Channel 4 — Wrist Pitch Command
- Channel 5 — Wrist Roll Command
- Channel 6 — Gripper Command
- Channel 7 — Shoulder Pitch Current Limit
- Channel 8 — Shoulder Yaw Current Limit
- Channel 9 — Elbow Pitch Current Limit
- Channel 10 — Wrist Yaw Current Limit
- Channel 11 — Wrist Pitch Current Limit
- Channel 12 — Wrist Roll Current Limit
- Channel 13 — Gripper Current Limit
- Channel 14 — Spare
- Channel 15 — Spare

Analog to Digital Conversion Board

An A/D conversion board (Datel part DVME-611B) multiplexes 32 single-ended analog signals to a fast 12 bit Successive Approximation Register A/D converter. A single conversion takes four microseconds to complete. The board is accessed at VMEbus short I/O address space 0xC000 to 0xC0FF. The I/O processor will physically address 0xFFC000 to 0xFFC0FF when addressing this board.

Detailed programming information is found in the Datel document number VME-DVME-611/612.

The 32 A/D inputs are assigned as follows:

- Channel 0 — Shoulder Yaw Motor Current
- Channel 1 — Shoulder Pitch Motor Current
- Channel 2 — Shoulder Roll Motor Current
- Channel 3 — Elbow Pitch Motor Current
- Channel 4 — Wrist Pitch Motor Current
- Channel 5 — Wrist Yaw Motor Current
- Channel 6 — Wrist Roll Motor Current
- Channel 7 — End Effector Motor Current
- Channel 8 — Shoulder Yaw Rate
- Channel 9 — Shoulder Pitch Rate

- Channel 10 — Shoulder Roll Rate
- Channel 11 — Elbow Pitch Rate
- Channel 12 — Wrist Pitch Rate
- Channel 13 — Wrist Yaw Rate
- Channel 14 — Wrist Roll Rate
- Channel 15 — End Effector Rate
- Channel 16 — CAE Up/Down
- Channel 17 — CAE Forward/Backward
- Channel 18 — CAE Right/Left
- Channel 19 — CAE Yaw
- Channel 20 — CAE Pitch
- Channel 21 — CAE Roll
- Channel 22 — CAE Discrete 1
- Channel 23 — CAE Discrete 2
- Channel 24 — CAE Discrete 3
- Channels 25-31 — Spares

Anti-aliasing Filter Board

This board only receives DC power from the VMEbus, and is not address by the I/O Processor. Sixteen single ended four pole low pass active filters and thirteen differential input four pole low pass filters are used to prevent possible aliasing of tachometer, motor current, and CAE hand controller signals. These filters are situated between the signal source and the A/D conversion board. Refer to Martin Marietta drawings 849IRS20600, 849IRS20601 and 849IRS20604 for schematics and other detailed information.

Resolver Electronics Module

Resolves on the PFMA joints, and the RTD board require a precision 400 Hz sinusoidal reference frequency. This card generates this source. The card is a half-height VME form factor card that plugs into the short I/O Channel backplane. All signals come in on Front Panel connectors and the Back Plane only provide DC power. The module is wider than a single card and the front panel take two slots. See the 849IRS20000 drawing for card placement in the I/O chassis.

Drawing 849IRS20701 is the schematic of this card. Briefly a Burr Brown 4423 precision oscillator is configured for 400 Hz operation.

The PFMA has two different types of resolvers. The sine and cosine output voltages from the Shoulder Yaw and pitch resolvers must be reduced to be

compatible with the RTD board. Voltage dividing resistors are on this board to meet this need.

Motorola Parallel and Serial I/O Cards

A Serial I/O Module (MVME 400) and two Parallel I/O Modules (MVME 410) complete the list of components used in the I/O chassis. Placement of these I/O modules within the I/O chassis is shown in drawing 849IRS20000.

The serial I/O module contains two RS-232 ports (female DB-25) on the front panel of the card. One port allows communication with the Lord Force/Torque Sensor. The band rate is under software control, but is set at 19200 baud. A second port is intended to communicate with the CRL Hand controller. The I/O Channel Address for this card is 0x0000 — 0x000F. The I/O Processor will physically address 0xFF0000 — 0xFF000F. Details of the Serial I/O Module jumper configuration are listed in 849IRS20800. The vendor document MVME400/02 should also be referenced.

Each Parallel I/O Module contains four eight bit ports with handshake control lines. These ports can be configured as inputs or outputs. The left most Parallel I/O module provides an input interface to the user I/F Box. The Optomux interface for future video equipment control is not yet implemented. The short I/O Channel address for this card is 0040. The I/O Processor will address FF0040. Part of the right most Parallel I/O module provides an output interface for the Matrix Switcher. The OP-EYE interface was deleted. The short I/O Channel Address for this card is 0x0020. The I/O Processor will address 0xFF0020. Details for the Parallel I/O modules jumper configuration are listed in 849IRS20800. The vendor document MVME410/D2 should also be referenced.

3.4.3 Bit3 VMEbus to VMEbus Repeater Bridge

This component consists of 3 parts: two Model 411 VME Bus Repeaters, and a Bit3 repeater cable between the 411 boards. Both 411 boards are configured for level 3 bus arbitration since the respective VME buses so require. Neither 411 provides the Bus Clock Signal, and Interrupt 4 is configured to pass between the 411 boards.

One 411 is installed in the Heurikon Chassis at slot seven. This card is configured as a slave. This slave 411 has a model 400-210 daughter board, with 32K bytes of dual port static RAM, installed on board. This RAM forms the shared buffer between 411 repeaters. Since the slave 411 in the Heurikon Chassis has the RAM buffer locally access to it is possible. This RAM buffer appears at 0x01ED0000 thru 0x01ED7FFF to the Heurikon SBCs.

The second 411 is configured as a master in the I/O Chassis. This master will obtain access to the 32K buffer via the Bit3 repeater circuitry and the connecting cable. The shared RAM appears at Locations 800000 thru 807FFF to the I/O Chassis Processor.

More detail can be found in the Bit3 VME-VME Adapter Model 411 User's Manual, and drawing 849IRS00100.

3.5 Peripheral Equipment

Peripheral input devices, and control components for the IRSS Control Electronics will be described here.

3.5.1 Force/Torque Sensor

The Force/Torque sensor is a six element sensor, which will measure forces and torques applied to it in the X, Y and Z axes. Voltages proportional to the applies forces are converted to digital information in a small preprocessor box. The digital information is then formatted by a processor unit, and communicated to the I/O Chassis via an RS-232 link. More detail can be found in the Installation and Operations Manual for the F/T Series Sensing System document published by Lord Corp. Drawings 849IRS00002 and 849IRS00004 provide system level location information. Drawings 849IRS30000 and 849IRS30004 detail the Force/Torque Sensor Top Drawing and I/O cable.

3.5.2 Hand controllers

The CAE hand controller translates manual actions into related voltage signals. An operator moves a ball shaped knob in the up/down, forward/backward, right/left, yaw, pitch, and roll directions. These directions have related analog signals proportional to the hand controllers position. These signals are converted to digital information by the A/D board. A pushbutton switch informs the computer when hand controller movements are valid. All of this information is used by the system to direct the PFMA's position.

The CRL hand controller internally generated digital information concerning position and communicates this to the digital computer through short haul modems to a RS-232 port of the I/O Chassis. The short haul modems are used because the distance between the CRL hand controller and the digital computer may be more that 50 feet. This will support teleoperation work in the future.

3.5.3 Matrix Video Switcher

The Matrix Video Switcher (model 8824) contains special relays capable of switching video signals. The switcher will direct video signals from three active camera to two monitors. Discrete control signals from a parallel port of the I/O chassis controls this switching.

Refer to drawing 849IRS00002 and 849IRS00004 for connecting details. Drawing 849IRS40000 and 849IRS40004 concern the Matrix Camera Switcher drawing and I/O Cabling.

3.5.4 User I/F Box

This box generates discrete signals which are monitored by a I/O chassis parallel port. The User I/F box switches control the Camera Matrix Switcher, and the Pan, tilt and Zoom of the cameras in the lab. The camera matrix switcher has been used, but the pan, tilt, and zoom for camera mounts, has not been implemented.

3.6 Servo Control Electronics

The Servo Control Electronics Chassis and the associated +28 VDC high current power supply are NASA supplied equipment. The supply provides power for:

- +/- 15 volt DC to DC Converter which powers the analog electronics
- PWM amplifier "H" output drive stage to the DC Brush Motors
- Brake release coils on the PFMA joints

Control switches on the servo control electronics modules' front panel include:

1. (left side) On/Off for the seven PWM output stages driving PFMA motors (joints listed below and gripper)
2. (central) +28 VDC On/Off, and On/Off for +/- 15V DC to DC converter.
3. (right side) switch +28 VDC to brake release circuits for each of the joints listed below:
   a) Shoulder Yaw
   b) Shoulder Pitch
   c) Elbow Pitch
   d) Wrist Pitch
   e) Wrist Yaw
   f) Wrist Roll

3.6.1 PWM and Rate Servo Cards

Seven PWM and analog rate servo cards are present in the left side of the chassis. Each card receives a conditioned Tach signal, which is input to the rate servo and added to a rate reference input. A gain adjustment switch is provided for joints listed above. A current limiting signal from the computer will clamp the input which get to the PWM voltage to current stage. Motor current is monitored and output from each respective card.

3.6.2 Brake Release Cards

When front panel switches are enabled, and the current limit threshold from the computer is raised, these cards will apply 28 VDC to the respective release brake coils.

3.6.3 Tach Filter Cards

These simple cards provide a one pole resistor/capacitor filter, with a voltage follower buffer, between the joint tachometer and the analog rate for each rate servo described above. The break frequency of this low pass filter is 10 Hz.

CN22 (0) (2)
TURNER J/PUBLICATION
MARSHALL SPACE FLIGHT CENTER
HUNTSVILLE AL.

RETURN ADDRESS CN22D