A LOW-LEVEL CONTROL INTERFACE FOR ROBOT MANIPULATORS

M.L. Fitzgerald and Anthony J. Barbera
National Bureau of Standards
Gaithersburg, Maryland

INTRODUCTION

This paper will discuss a possible low-level control interface for a robot manipulator. The first section will present background information describing a proposed system modularization and the capabilities and limitations afforded by the use of interfaces. The next section presents three possible low-level robot control interfaces within this system. These will be elaborated on including a specification of the interface information and its use, timing considerations, and potential limitations. The paper concludes with a summary discussion and recommendation.

I. BACKGROUND

This section provides the background discussions on the low-level interfaces into a robot controller. The first section describes a proposed architecture for a real-time control system that defines generic task decomposition modules and the interfaces between them. The second section defines the different types of information that a well specified interface must provide. The next section explains the function of an interface as a tool for abstracting information into a simpler format so that it may be used to construct more complex groupings of information, and provide the capability of plug compatible systems.

I.1 A Proposed System Modularization

In order to specify interfaces, the system modularzation should be defined. An architecture for a real-time control system has been proposed by the National Bureau of Standards (1-4). It consists of a number of component modules which are generic control levels. These generic control levels can be stacked intoa multiple level structure that provides for the hierarchical decomposition of a task. The more complex the task, the more levels are required. Depending on their location in the overall architecture, some of the levels will not only decompose a task into simpler parts, but also coordinate activities of several lower control levels (Figure 1).

The interfaces are defined by the data that communicate the command and status information between these control levels. As shown in Figure 1, there is a robot task controller that receives commands from the workstation controller and sends back status data relevant to these commands. This task decomposition controller of the robot also interfaces by way of a request and a feedback buffer with a sensory processing and world model system that can be pictured horizontally along side of it. This robot task controller decomposes the high level task commands into low-level control commands for the robot.

Below the task decomposition controller is the robot joint controller. It is responsible for developing the drive signals to a particular robot, instant by instant, required to carry out the commanded tasks, while coordinating all of the joint motions and not exceeding any joint limits. It is this interface between the task decomposition controller and the robot joint controller that is to be discussed in this paper.

Each of the controller modules shown in Figure 1 is itself composed of one or more generic control levels. The generic control level is used as the fundamental building block in this architecture. Figure 2 provides a more detailed look at a generic control level. Each level interfaces to four other components in the system. The first is an interface to a control level above that performs a higher level decomposition of the task. This interface is composed of two buffers - a command bufferfrom, and a status buffer to the higher level. In like manner, there is a similar interface to a control level below that consists of command and status buffers. There is an interface to a sensory processing/world model system which provides feedback information describing the present status of the environment. This interface consists of a request buffer to this system and the feedback response buffer.

In order for the system to display the type of flexibility required, it is important that the data, as much as possible, be separated from the actual processing within each level. For example, a transfer task - to move an object from one point to another - is independent of the type of object to be moved and of the particular locations. The particular numeric representation that defines the object and the locations in space can be separated from the programs and supplied at execution time through the fourth interface into a knowledge base that carries all the task specific data required for the controller. This separation of task and data - the creation of a data driven system - provides a system where it is possible to think of programming off-line the task specific data, perhaps through a CAD system, without having to reprogram the generic task decomposition algorithms within the control levels to handle the new task to be accomplished. This adds a great deal of flexibility to the system (5).

This interface to a specific task knowledge base allows the command-status interfaces to represent levels of abstraction in a description of the task. For example, the command from the Workstation Controller to the Robot Task Controller might be

TRANSFER object FROM source TO destination

where the object, source and destination fields contain the symbolic names of these items. To properly carry out this task, there is task specific data - grip points on the object, grip force parameters, approach and departure paths, intermediate trajectory points, acceleration and deceleration profiles, etc. - which are required to fully describe the task. This type of information should be provided through the interface with the task knowledge base.

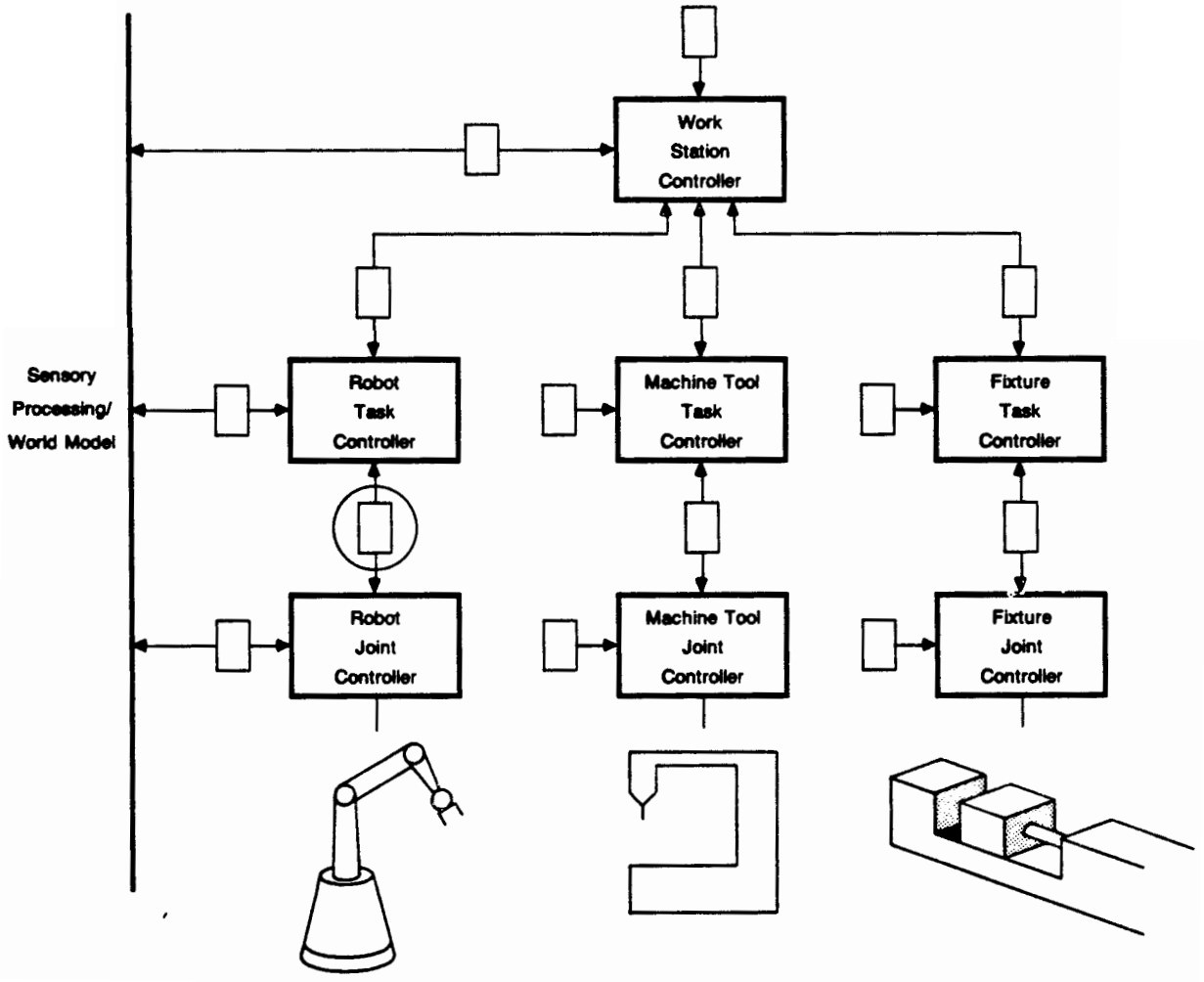


Figure 1: A view of the major controller within a workstation. The interfaces are represented by the small rectangular boxes with arrows. The interface that the paper addresses is between the Robot Task Controller and the Robot Joint Controller. It is circled in the figure.

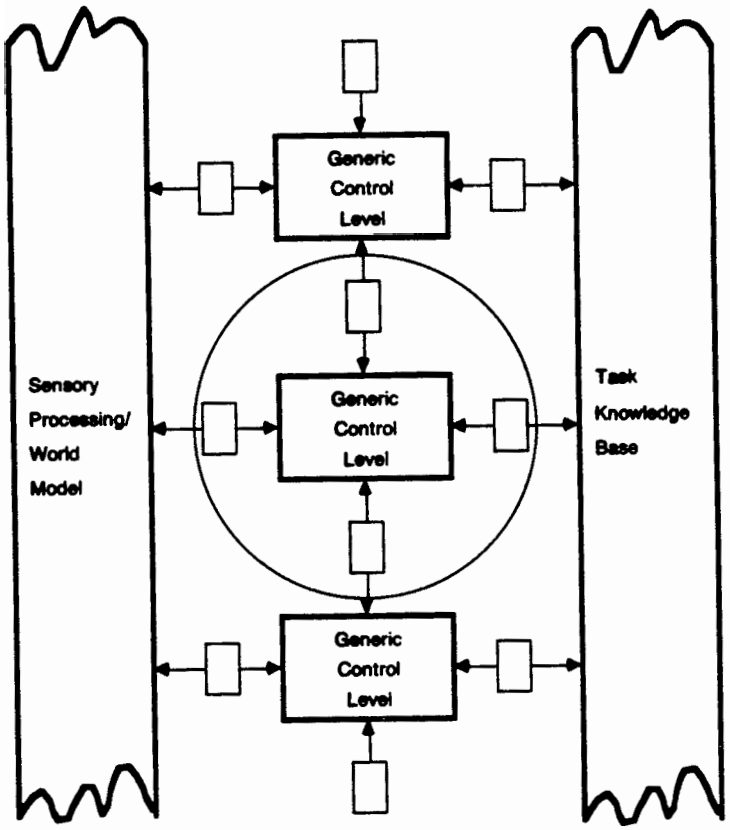


Figure 2: Shown here are the interfaces surrounding each generic control level. The generic control level is the fundamental building block of the system. It has a command-status interface to the level above and to the level below. It interfaces to the Sensory Processing/World Model for real-time status on the state of the environment. It interfaces to the Task Knowledge Base for all of the data, such as grip forces, approach/departure paths, task decision programs, etc. relevant to the particular task being executed.

In the system implemented at NBS that uses this architecture, the robot control system below the workstation controller which consists of the Robot Task Controller and the Robot Joint Controller is made up of five different levels of task decomposition (Figure 3). The interface to any of the three lower levels could be specified as the potential candidate for the interface to the robot joint controller. It is the intent of this paper to examine these three and to recommend one of them as the most appropriate at this time for a model of a low-level robot control interface to be supplied as an option on commercial robot controllers.

The lowest level interface shown is to the servo-control level. At this level joint position, velocity, or torque information defines the robot motion command during the next time increment. This level uses that command together with joint feedback data to calculate the next drive signal values to be sent to the actuators.

The level above this, identified as the coordinated joint level, receives commands in a robot independent format that specifies the position and orientation of the tool-mounting plate at the end of the robot. This might also be a velocity or force command, but specified in a convenient world-based coordinate frame of reference. It is through the coordinated joint level that this information will be transformed into the joint representation of the robot to cause the tool-mounting plate and therefore the attached tool to behave as commanded.

The level above this - the primitive level - receives commands in the form of goal points or trajectory segments. It generates all the intermediate positions in space of the tool-mounting plate that define the control path to the specified goal point.

Interfaces above these levels will not be addressed in this paper. These three interfaces then, that represent the inputs to these three levels of control, the primitive, the coordinated joint and the servo, then become the target interfaces for a low-level robot control interface.

I.2 Complex Interface Specification

Interfaces are the complementary result of the modularzation of a system into functionally separate components. The interfaces are the connections between these components. A system is modularized to partition out simpler components or modules to make it easier to understand, design and implement that system. For example, a robot controller might be modularized into a number of components such as the coordinate transformation routines, trajectory routines, pallet-offset routines, servo routines, etc., or whatever set of modules that the designer finds appropriate to meet the functional requirements. These modules (sets of routines) have interfaces between them. These interfaces can be defined by the data that passes between the modules as they are called to carry out their portion of the overall function.

However, the interfaces are also implicitly defined by the timing and the functional use of this data during execution. That is, the data by itself does not completely specify the interface between component sets of programs. When the trajectory module calculates the next intermediate point for the robot and passes it to the servo module, it is implicitly assumed that the servo module will execute on it at that time rather than at some later time. It is also implicitly assumed that the servo module will do whatever is required to cause the robot to reach that commanded position. In reality, the servo algorithm might cause the robot to go to a different position, offset from the commanded position because of a gravity loading force creating a steady state error that the servo algorithm is not set up to overcome. In this situaion, the programmer might provide status back to the calling routine that includes information about the robot's actual position. Alternatively, a new servo algorithm might be written with additional calculations used to servo out steady state errors. In any case, it is important to note that the programmer has passed data between the modules with implicit assumptions about the timing and functional use of that data due to his detailed knowledge of how each part of the entire system was executing.

Thus, the data specification constitutes a valid interface between two modules only if the modules execute on it in an expected time interval and in an expected manner. The interface, such as the one being described in this paper, will connect two modules that will not have been writen by the same person, or even by the same company. Therefore it is important to explicitly determine and specify all of the information, including what is implicitly assummed.

All interfaces to be discussed later in the paper will be defined by:

1. The data specification that includes the format and legitimate values within the buffer that represents the information passed between the connecting modules. This includes the information passing in both directions between the two modules.

2. The timing specification that includes the maximum time delay in communicating a data set from one module to the other, the expected time interval before the receiving module begins execution on the data, the expected time interval before a resultant change in output will occur, and the expected time before status will be received back from the module.

3. The functionality specification that includes what behaviour is expected from each module as a result of each piece of information passed to it: not only what a module should do, but what it should not do.

The data specification (the explicit information) and both the timing and functionality of connected modules (the implicit information) are all needed in the definition of the interfaces between modules. When a system is designed and built as a stand alone, bundled system, these problems of understanding and specifying this additional timing and functionality of the interface data usually do not occur because the programmer resolves these issues with implicit knowledge about the detailed operation of all parts of the overall system. If, however, a system is to have interfaces from external systems into it at different levels of its capabilities, or is it itself to be a component of a yet larger system, then these issues of the interface specification must be addressed.

A totally separate issue from the above description of an interface is the specification of the communication mechanism that will be used to transfer the interface data. Whether an RS-232 serial link, a parallel link or a network coaxial base-band or broad-band system with its associated protocol, is used is part an independent issue and will not be covered in this paper.

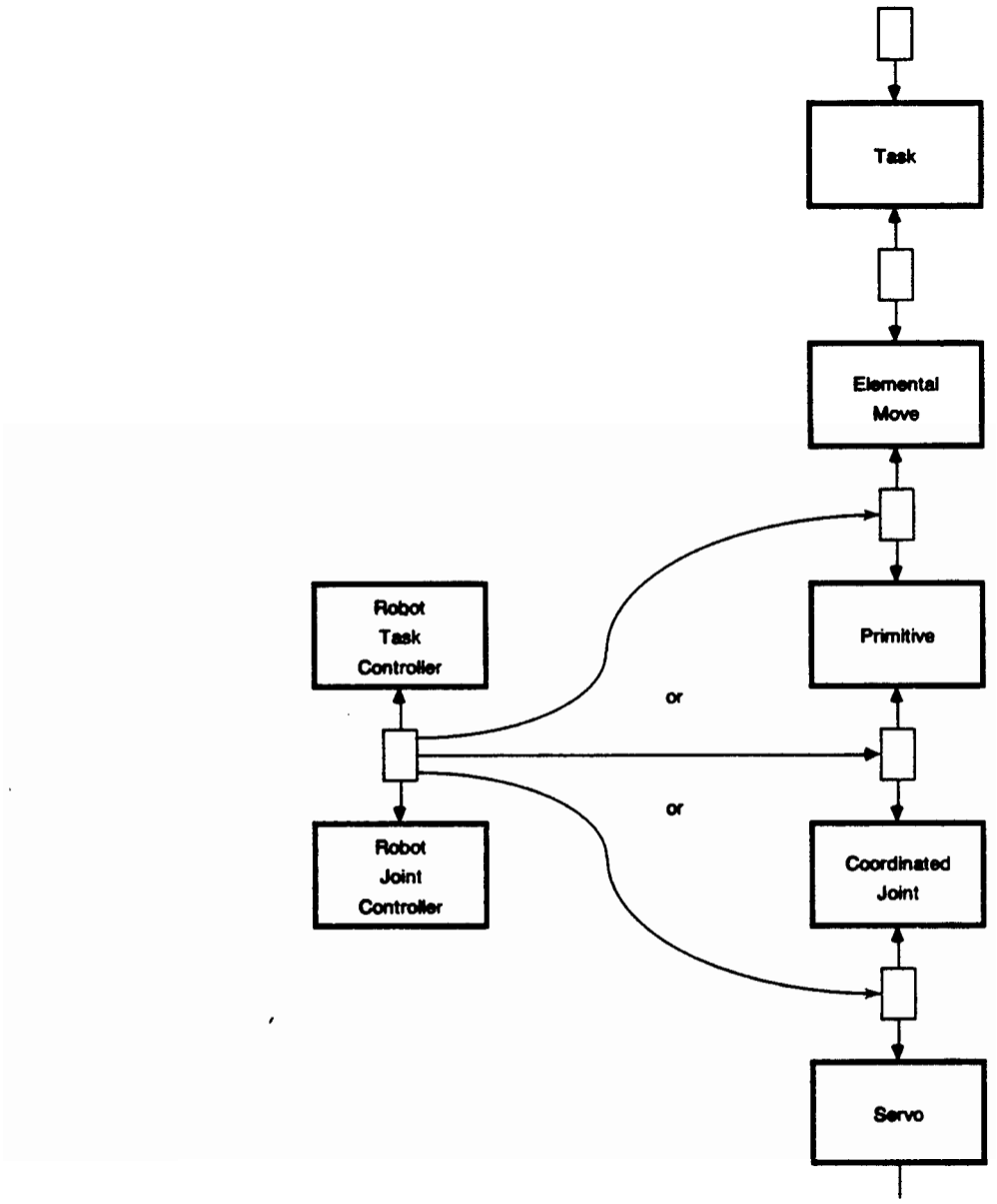


Figure 3: The Robot Task Controller and Robot Joint Controller are composed of five generic control levels. There are three possible interfaces between the lower control levels that might be used to define this interface between the two pictured controllers.

I.3 Interfaces and Levels of Abstraction

Interfaces provide levels of abstraction within a system. Within the control architecture described in section I.1, the interfaces represent different levels of task description or abstraction. The interface data to the Coordinated Joint Level includes task commands which might be the specification of the next pose of the tool-mounting plate in space. A group of these poses represents an approach path. Therefore, it is possible to assume that the interface to the next higher level (Primitive) could contain commands such as "GO-THRU Intermediate-Point-I1" which is really an abstraction by way of "chunking" or grouping together a set of poses through space.

Figure 4 illustrates an application task decomposed through the five levels in the controller architecture. The interface data reflects the different levels of abstraction for this task as implemented. The highest level shown depicts information at the level of a command to TRANSFER an object. The next lower level interface contains information in the form of simpler commands like MOVE-TO a source location, PICKUP an object, etc. The function of each module becomes the manipulation of data between a higher and lower level of abstraction: the breakdown or decomposition, of a broader scope representation of the information into its component, more detailed subcomponents.

An analogy to control interfaces in a robotics system can be made with programming languages used as interfaces to computers. At the lowest level, computer hardware is controlled by patterns of bits that represent various gate levels in the control logic. This machine code programming can be abstracted to a higher level interface known as assembly language, where single mnemonics represent a pattern of 8, 16, 32 or more bits in the machine code. This can be abstracted to a high level language where single words or operators can represent 1, 5, 10 or more assembler mnemonics.

The control interfaces, the language constructs, are defined for computers for the same reasons as for robots. First, they allow plug compatibility, i.e. programs written in FORTRAN can run on different computer hardware systems. They also identify levels of abstraction so the task or program can be specified at a high level of abstraction without getting involved in the lower level details of the system.

This analogy also points out some of the limitations that occur. The high level languages may not be able to utilize specialized hardware capabilities for a specific system. Thus, the higher the level of abstraction, the less detailed information about the low-level system can be represented. There is the dilemma that if all of the low-level detailed capability can be represented in the high-level interface, then the value of the interface is lost since there has been no abstraction or apparent reduction in the amount and complexity of the information at the different levels.

I.4 Interfaces and Large System Integration

Command-status control interfaces offer a number of advantages for large system integrations. Interfaces define a plug compatible data set that allows modules to be changed to upgrade their functional capabilities with minimum ripple effects on the rest of the system. For example, a trajectory generation module could be upgraded to provide a smoothly varying acceleration, instead of a step function acceleration. The input data is still the goal point. The output data of the module is still the sequence of intermediate positions. The function of the module has been replaced with an improved version.

A larger view of the plug compatible feature is that interfaces provide the mechanism for assembling components into larger systems. For example, well-defined interfaces might allow the integration of any of several different robots to a generic robot task controller module, so that the application programs do not have to be changed or rewritten to carry out the same task, using different robots.

The low-level control interface within a real-time system is complex. The robot interfaces are bi-directional: control information is sent to the next lower module and status information is received back. In addition, the control of the robot joints may be commanded by joint position, velocity, acceleration or torque or some combination of these.

The design of a system to create a low-level control interface to gain plug compatibility and information abstraction, could potentially limit the efficiency with which individual robot manipulators can be used in realizing their unique capabilities. However, it would provide the ability to be able to integrate many different robots, with different capabilities, into a system to perform different tasks without having to change the upper levels of control.

The use of interfaces to define plug compatiblity and levels of abstraction also encourages the development of data driven systems. These include developing structures, languages and programming styles to specify application tasks in a data independent form where only the generic task decomposition is described. The data that uniquely specifies particular workpieces in particular locations and orientations are tagged to variables during execution through a separate interface to the task specific knowledge base as described above. For example, a transfer application task would involve going to a source location, picking up an object, going to a destination location, setting down the object and going to a final location. This sequence of operations is generic across all transfer tasks. The numerical identification of relevant grips, approach and departure paths for a particular object as well as the coordinates that define the locations in the workspace could be obtained from data structures during execution by associating particular objects with locations and retrieving all of the associated data records.

The abstraction in the command structure of the control interfaces tends to preclude the specification of all of this data with the command and therefore, if this data is not to be buried in the internal programs, a separate interface to an external representation of this detailed task data is suggested.

Thus, control interfaces would contain the generic task commands with only those data arguments that are appropriate to that level of command abstraction (for example, only the symbolic names of particular objects or locations.) To deal with new parts or a change in the work environment, would be a change in the data, not in the programs. This same application program would accomplish the task using a different robot or in a different workstation without reprogramming.

In addition, the task description is modularized by the control interfaces into different levels of abstraction which enhance the user's understanding of the system and help identify appropriate places in the system for the addition of new features, sensors, etc.

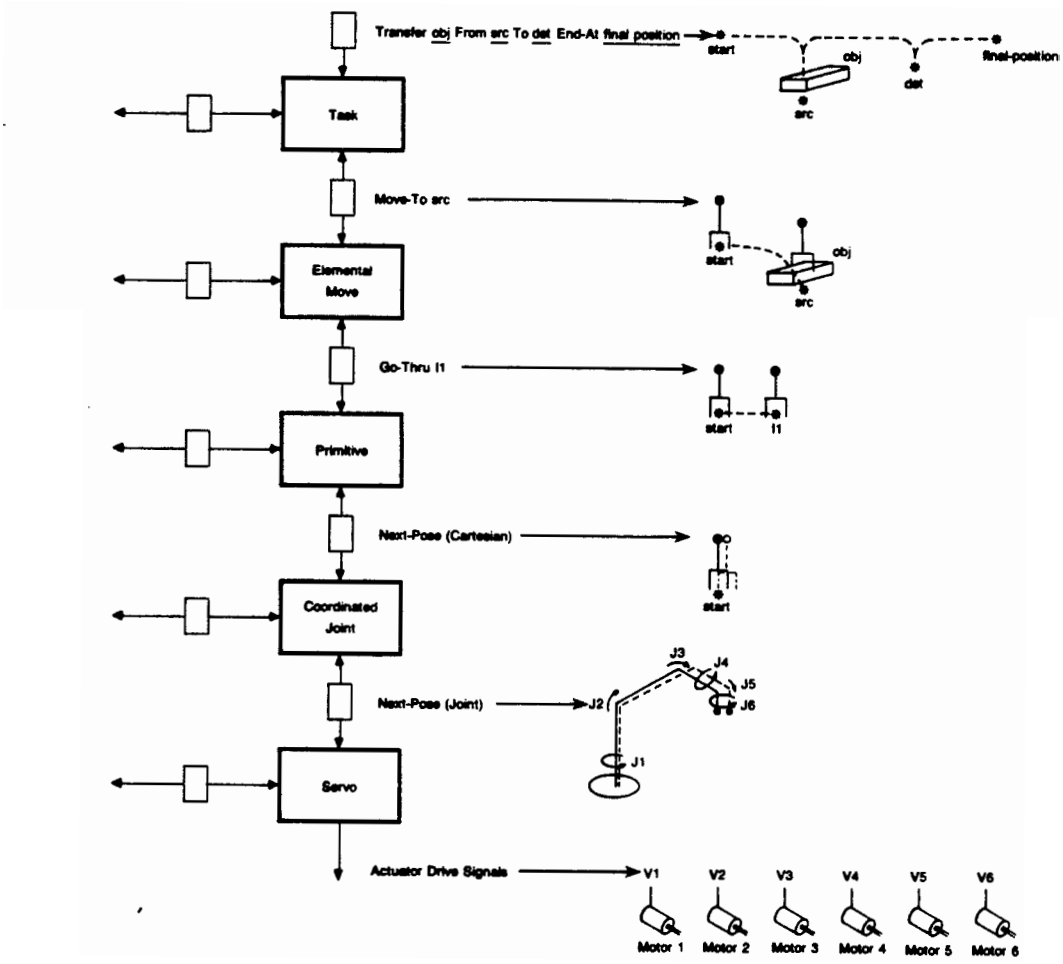


Figure 4: The function of each of the five task decomposition control levels is illustrated here. The Task Level receives commands through its input interface. An example is the command to TRANSFER an object from a source to a destination location and move the robot to a final position. The Task Level decomposes the task into a sequence of simpler commands, such as MOVE to the source location. Each lower level decomposes a simpler and simpler command until the lowest level is commanded the drive signals to the actuators. At each level, there is illustrated the overall effect of the input command at that level with respect to the complete task.

These advantages come at a cost of more effort in the design and implementation of controllers to partition and provide these interfaces and a probable limitation in the efficiency or optimized use of a particular robot manipulator's features.

II. LOW-LEVEL ROBOT CONTROL INTERFACE

The above sections were used to lay the background for the discussion of a low-level robot control interface. To have an interface, a system must first be partitioned into component modules. The suggested architecture described above identifies an interface between the Robot Task Controller and the Robot Joint Controller. A finer partitioning of the control system represented by these two controllers leads to three potential interfaces at different levels in the task decomposition structure. Any one of these three might be made to represent the interface between the higher level task controller and the lower level joint controller.

The possible advantages of a control interface of this type will be reviewed before discussing these three specific interfaces in detail. These advantages can then be used as a checklist against which to evaluate each of the individual interfaces to determine a figure of merit of its usefulness.

II.1 Externally Generated Control Capabilities

The purpose of a low-level control interface is to provide a means of giving the robot additional capabilities over what might be provided by the vendor's controller. The interface provides a mechanism whereby a user can control the robot, taking advantage of the robot's manipulation capabilities while integrating it into user specific tasks too diverse to be easily accomodated by the vendor's controller.

The following is a brief description of a number of more sophisticated control capablilties that are not available with many vendor's controllers. Without the vendor providing the ability to do these types of tasks, the user's only option is to supply the computing system and algorithms necessary to calculate his specific control requirements, decompose the task execution into the low-level robot control information necessary to command the robot, and pass this data through the proposed vendor supplied interface.

Desired control capabilities are:

1. To control the path of the robot using trajectory capabilities such as velocity and acceleration control, or to define arbitrary paths through space to follow (a particular mathematical surface or curve), or to change the path or trajectory parameters at any instant before the original goal was reached.

2. To allow interaction with any sensors, simple or complex, to furnish data for the real-time control of robot motion based on these data (i.e. continuous real-time path modification based on sensory feedback.) This includes the ability to interact with safety systems and interlock signals between equipment components.

3. To control detailed fine motion of the robot as a closely coupled operation with another device. This device, such as a glue gun, a welder or paint sprayer, may be attached to the robot and the feedrate of this device might vary, be sensed and used to control the path rate of the robot. The device may be another robot, locked in a tightly coupled task requiring this type of high speed fine motion control.

4. To make the actions of the robot manipulator part of an integrated task of a much larger scope. For example, the robot might be one component of a totally automated workstation involving a machine tool, robot, automatic fixturing, materials transport, etc. where its actions have to be commanded and coordinated with all of the other activities in order to accomplish the overall task.

5. To make use of information in external data bases that might represent points in space, trajectory paths, etc. where accessing this information would save much effort over teaching the robot or entering this data into its programs in the vendor provided manner. As an example, a data base might contain the location of 500 holes to be drilled in an airplane wing. Clearly, retrieving this already existing data is preferable to teaching all of these points to the robot.

6. To create generic task application programs so that the same higher level program can be used with a different robot of either the same type, o ultimately a different type that still meets the manipulative requirements of the task. For example, an application program could be written to spot weld a fender, and if the particular robot that normally carried out the task was to become inoperative, another robot on the line could be controlled to carry out the same task by sending the same control information through the interface.

This ability would also be valuable in a system of a number of different vendor robots where the programming of one vendor's controller does not match that required for another vendor's system. Therefore, a user might connect through a low-level interface into a number of different robots and provide his own single generic controller to do the task decomposition, and thereby make all of the robots appear identical to both the programming and control system. This would greatly reduce the programming support burden and simplify the task description in large integrated systems.

By providing a well-specified low-level control interface, the above capabilities could be achieved. All increased capabilities, however, come at some cost and these limitations will be mentioned with respect to each of the particular interfaces. In general, the cost of interfacing in loss of efficiency or decrease in optimization of use or functionality. That is, an interface is a greatly reduced information set, used to designate a command or action. Because so few bits of information pass through an interface, there is a limited amount of unique specialized control that can be communicated. This will be elaborated on in the following three sections which will describe the three possible control interfaces into the robot joint controller.

II.2 The Servo Level Control Interface

The first interface, that of the interface into the servo control level, requires information unique to a particular robot. At this level, the interface commands are identify particular control actions for each of the individual joints of the robot. This level interface is robot dependent. That is, the information that passes is specific to an individual robot and could be very different from one robot to another.

This interface contains data of several types (Figure 5). There has to be command information passing from the level above into this servo level to provide position, velocity, acceleration or torque data for each of the individual joints. It is also desirable to be able to modify this information at any time during execution. That is, in one command, request a position servo, and in the next command request a torque servo or velocity servo, etc. This requires the capability of not only passing a set of data that represents the values for each of the joints of the robot, but also some sort of flag information to indicate that these fields represent torques, positions, velocities or accelerations.

It might be desirable to be able to include parameters that would change different gain factors or coefficients that are used in the servo equation. For example, at some time it might be desirable to have the robot behave as if it were a very high gain system for high precision work, and at other times to have it behave as if it were a low gain system for smooth trajectory motions. Either these or some other parameter could be used to identify changing load situations.

The interface returns information that is status. This status information represents whether or not the servo level was able to carry out the last command given. For example, did the servo level successfully move the joints to the commanded positions. If it hasn't, then the status parameter would indicate this with either a qualitative or quantitive measure of success, such as the percent of the move accomplished, or perhaps some timing information that indicates when that command can be expected to be realized.

Part of this information being returned to the level above is status of the last command that was given to the system. In order for there to be no confusion about what the status relates to, the interface will have to include information to define which command the status is referring to. Depending on the communications mechanism used to transfer the data, it is entirely possible that status being received may be for the command issued before the last command that was sent (that is, two commands before the current command.) Therefore, to eliminate this confusion, there should be additional information in the interface that uniquely identifies each command, and the status should reflect this identification.

As mentioned above, there must be specified a rate as to how often commands are to be accepted and acted upon, and how often the status data is to be returned. Further, these command and status transmissions should be parallel activities, both occuring essentially simultaneously, that is with the same effective rate.

In addition to the command and status information, there is the expected timing of the interactions. Real-time control systems imply timing constraints. When a command is passed to the servo level, there is an expectation that that command will be acted upon by the servo level within a certain limited time period, and that it will be acted upon even if the previous command is not yet finished. The timing specification in this interface, should include the maximum time allowed after a command is given before the level below must begin executing on it, as well as a maximum time before the expected completion of that command. This would require that the level below not pipeline or buffer input commands. Nor should the level wait until it has actually finished a previous command before it begins working on the next one. It is expected that a command passed through the interface will be acted upon and executed during the next control cycle of the lower level. Therefore, the maximum times allowed to ensure this happening should be specified in the interface.

The communication rate through this interface into the servo level might be such that new commands will be provided at a rate less often than the servo rate. New joint positions might be commanded at a rate of 50/sec, for instance, whereas the servo update rate for the actuators might be on the order of 1000/sec. In this case, an interpolation would be required within the servo calculations to provide new set points at the servo rate. Additional parameters could be passed through the interface to help define the expected velocity or acceleratin profile so interpolation routines other than linear could be used to smooth the motion.

Further, certain information about the robot that may be returned in the status might require some manner of time stamp. Part of the status information might include the measured position of the robot at each instant in time. This becomes very important for use with sensory data that comes from sensors mounted on the robot itself. When the robot is moving and sensory readings are taken, those sensory readings must be correlated with the exact position of the robot at the time the readings were taken. Therefore, there must be a mechanism to define exactly when the robot positional information is read as opposed to when it is reported through the interface. If the robot is only to be used in tasks where it is moved to a location point, allowed to come to rest and the sensors sampled, then this timing of when the robot's position sensors are read is not critical, as long as it happens during the period when the robot is stationary. But for more sophisticated sensory interaction, where sensors are sampled during robot motion, it is critical to be able to identify at what time the data was sampled in order to correlate these readings with the absolute position of the robot.

Four possible ways of dealing with this issue are described. First the low-level controller samples the positional values at a fixed time in its control cycle and passes these values back through the interface with a fixed known time delay. The second requires external common hardware connections between the modules that provide either the signals of an externally generated clock, or that allows separate transmission of unique codes at the time the position sensors are read by the low-level controller. This code is appended to the sampled joint position data and passed with it to the higher level. The higher level correlates this data with the sensory readings marked with the same code.

A third method is to provide another interface, separate from the control-status interface, that allows the higher control level to treat the robot position sensors like any other sensor and sample them at an appropriate time with respect to the other sensors. Both the second and third methods require an additional interface into the low-level controller for either a hardware signal or to request that the position sensors to be sampled.

The fourth technique is to use a totally separate positional measurement system from the joint position sensors on the robot and to sample this external system as to the location of the robot at appropriate times relative to the sensor readings.

This particular aspect of the low-level control interface has been dealt with here in greater detail because of the importance of knowing the exact position of the robot for sensory interactive control.

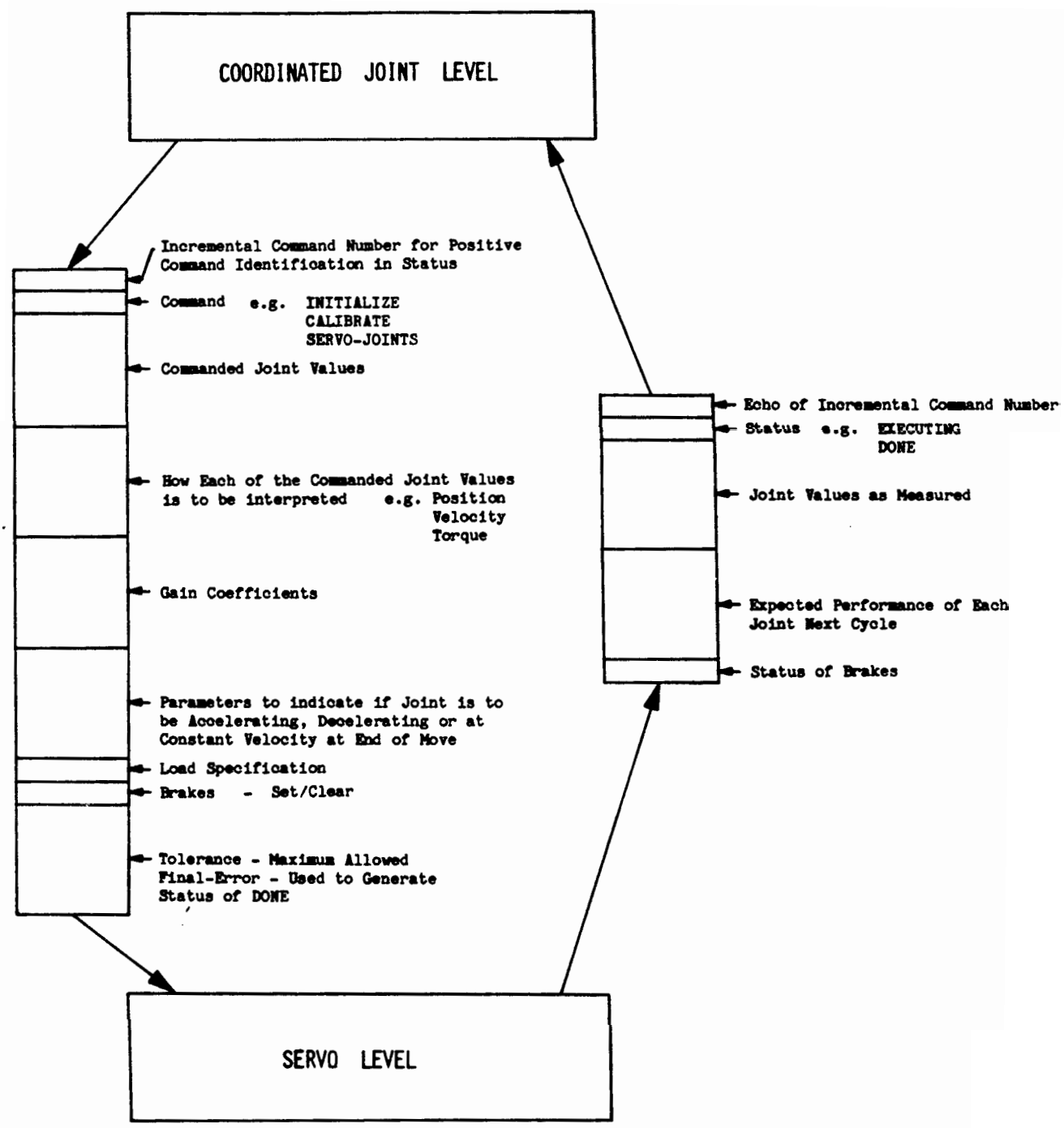


Figure 5: Sample interface specification of information to send as command and status between the Coordinated Joint Level and the Servo Level.

The major disadvantage of this level interface is that the command data is not robot independent. The burden would fall upon the user to perform the transformations from the generic world space coordinate system generally used in the the task decomposition into the particular joint representation necessary for this particular interface. The advantage of this interface is the very detailed control of the robot in both space and time that it provides for the user.

II.3 The Coordinated Joint Level Control Interface

The second interface is the one that occurs between the previously defined Primitive and Coordinated Joint levels. At this interface, the information takes on a much more robot-independent form. The interface at this level is meant to be a description of the position and orientation of the tool-mounting plate of the robot in a world space coordinate frame of reference. This could also represent a robot-independent specification of a velocity, acceleration or force vector.

This is the next higher level of interface and as such represents a further abstraction of the robot control information. It is this abstraction that has freed the interface from specifying unique joint information and allowed the high-level tool-mounting plate pose specification. From this information, the responding low-level controller would transform these values to obtain the particular set of joint commands to the servo level to accomplish this command for a particular robot.

It is this very abstraction, however, that will make it more difficult to specify fine motion control of the robot through this interface. For example, there may be several joint configurations (as is the case with most six axis robots) that will cause the robot to position its tool-mounting plate in the same position and orientation. If one joint configuration is more desirable than another for reasons of joint member collision with items in the workspace, how is this to be specified? This type of specification is unique to particular joint configurations and it is not clear how the interface can handle this information in a general way consistent with the intended generality of the robot-independent specification of the pose description. This problem is greatly increased for robots with more than six axes where very large sets of multiple solutions exist.

Additional effects of an interface at this level is the much increased functionality the vendor would have to supply below this interface to give the large range of control needed to carry out the previously mentioned sophisticated capabilities. The vendor would have to provide methods to deal with singularity problems in the mechanical system. That is, as the robot passes through certain configurations, very large joint velocities and accelerations are required if the pose of the tool-mounting plate is to be maintained. For some tasks, the tolerance on maintaining this pose is more critical than the speeds of the motion, while for other tasks, the speed is much more important than the pose. There would have to be ways of specifying these tolerances through this interface and that functionality would have to exist in the vendor supplied controller to deal with them appropriately.

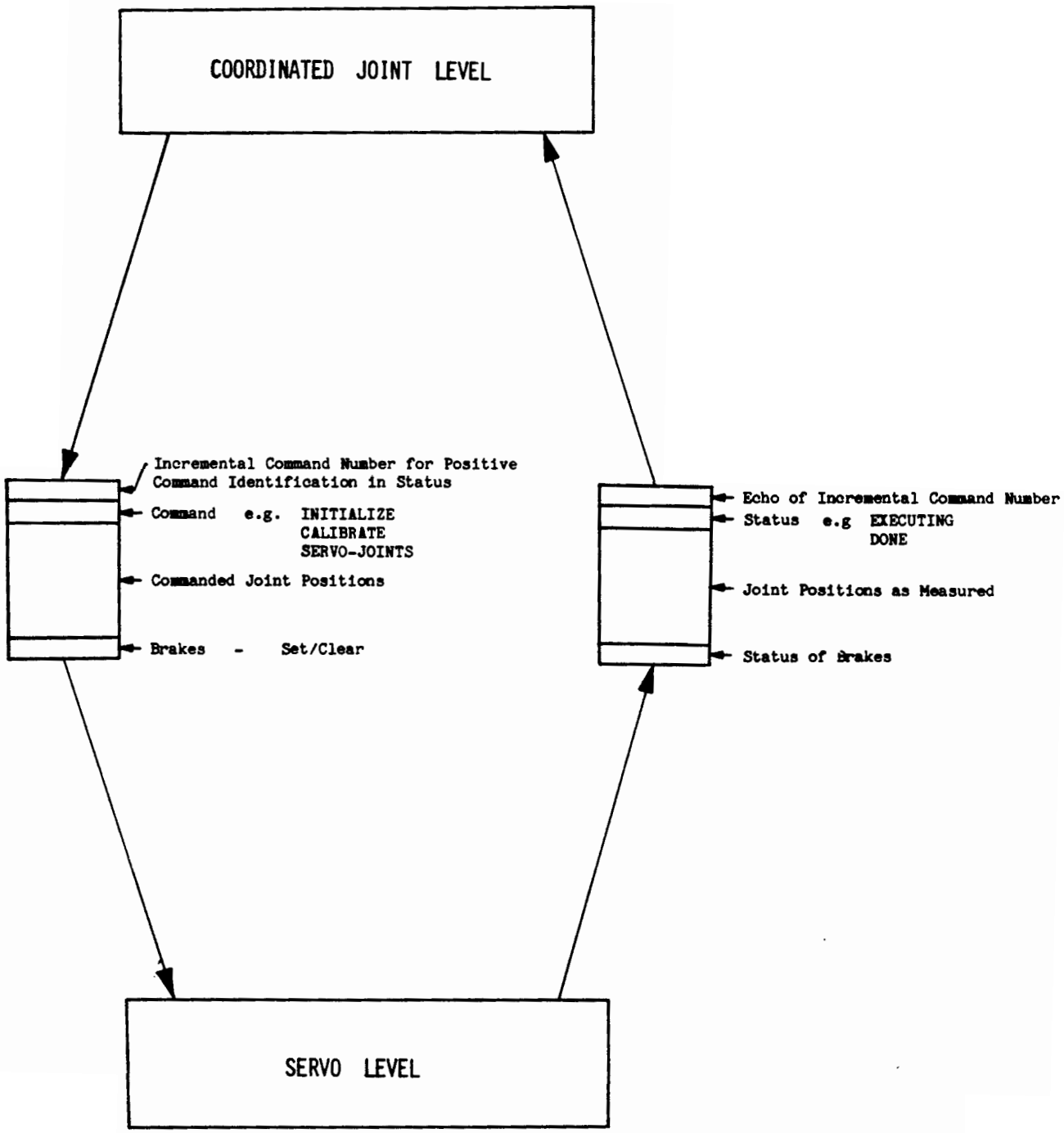
Another difficult area concerns limitations in joint travel. Particular commanded poses in world space coordinates might result in a joint limit being exceeded. However, the robot might have a joint configuration solution to the command, but it will require the joints to flip to the new configuration. In this case, the pose will probably not be maintained during the joint flip to the new configuration. Further, the robot might be sufficiently close to other pieces of equipment to collide with them during the flip. How is this to be controlled through this interface? There are certain times when it would be better to leave the joint commanded up against its limit, other times when the flip is appropriate and still other times when the robot could back out of a congested area, flip the joints to a better configuration and return. All of this capability would have to be provided in the vendor's controller and a mechanism set up through the interface to allow the higher-level controller to decide on the best action for a particular task.

Another situation exists where the most convenient trajectory for the robot to make would be in joint space. The upper level controller might provide a goal in the robot-independent format through the interface. The transformation of this pose into the joint space representation would define a set of joint values that are far enough from the present positions to take a number of control cycles to reach. These values could be passed directly into the servo level allowing the joints to move at whatever speed they can to the final position.

Conversely, after doing the transformation and calculating the final joint configuration, another vendor supplied algorithm could test for joint velocity and accelerations being exceeded if it tries to get to this position in the next control cycle. If it is too large a distance for any of the joints to move in one cycle, it scales back all of the joint values so that all joints will receive a legal value and passes this scaled set to the servos. This would be repeated for however many control cycles are required until the final joint configuration is reached. In this way, a trajectory can be constructed through joint space in a manner to coordinate all of the joint motions so that they all arrive at the final position together. It would also be desirable to control the velocity of this joint space trajectory. Again, this means that the vendor supplied controller would have to be able to perform all these functions and additional parameters passed in the interface would be required to specify the choices of which function to do when.

With respect to the status back through this interface, there exists the same requirements as discussed in the low-level servo interface concerning timing interactions between the levels and marking of the status relative to the exact command it is refering to. There is also another layer of complexity concerning the real-position of the robot as determined from the position sensor readings. There still exists the difficulties in the determination of exactly when the robot position sensors are read relative to other sensors that will be sampled by the high level controller. In addition, if the position of the robot is reported as part of the status through this interface, it would be as a description of the pose of the tool-mounting plate in robot-independent coordinates to coincide with the level of abstraction of the command data at this interface. In this format it is not known which multiple solution of the joint configuration is being used.

For example, the robot might have an elbow joint in the down position just as easily as the up position. Nothing in the specification of the position and orientation of the tool-mounting plate in world space coordinates would indicate this. It is unclear what sort of functionality each vendor should provide and how the higher level controller could use additional information about this configuration without defeating



TIMING SPECIFICATION

* Command and status exchanged in parallel at a minimum of 30/sec
* Execution on command begun within 1/30 sec after receiving command regardless if last command finished
* Status on measured joint values returned at fixed time interval from actual measurement time

Figure 6: Recommended command-status low-level control interface along with timing requirements.

the robot-independent abstraction intended by this level interface.

A possible method of dealing with these problem areas is in the use of the interface to the task knowledge base described in the section on the system modularization. This might allow the command-status interface to maintain the abstraction of the robot-independent form and for, particular tasks and robots, to have the lower-level controller access the task knowledge data base to retrieve the unique parameters required. No specific recommendations or comments can be made here about this because investigations into this technique are still in beginning stages at NBS.

As can be seen by the above discussion, the interface into the Coordinated Joint Level, which on first glance appeared to offer a robot-independent interface, requires considerable additional control information and a large set of control capabilities to be supplied by the vendor. Without these, the fine motion control and real-time path modification required for sensory interactive control would not be possible.

II.4 The Primitive Level Control Interface

As defined in the NBS architecture, the interface at this level has abstracted the task commands to the form of goal points in space, specified in a robot-independent format (a tool-mounting plate pose.) The primitive level is to calculate the intermediate poses required for trajectory and path control to this goal point and to either decelerate and stop, or fly past it according to some path tolerance, velocity and path smoothing algorithms. It is also expected that various sensory data will be used by this level to carry out real-time path modification. The user will have the capability to add whatever sensors are required and to modify the path calculation algorithm accordingly. Further, the commands at this interface encapsulate not only control of the robot manipulator, but also control of its end-effector in as much as its actions are to be coordinated with the robot motions.

All of these functions, including the ability to modify the path generation algorithms and interface to sensors would have to be provided by the vendor to support this level of interface. Because this level of interface makes so many implicit assumptions on so much of the design of the control system, and requires so many capabilities to be installed in the vendor supplied portion, it seems inappropriate at this time to consider it as a general purpose low-level controller interface.

III. RECOMMENDATIONS

The interface into the Primitive Control Level is not considered a reasonable interface at this time because of the reasons made in the discussion above. The interface into the Coordinated Joint Level, while offering the benefit of a potentially robot-independent specification, requires such a high level of vendor support and so much robot-dependent parameter specification that it is also probably an inappropriate level at this time.

The interface to the Servo Control Level, while placing the burden of the coordinate transformations on the user, has the advantage of allowing a high degree of motion and robot configuration control in real-time. Before a more abstract higher level interface is used, it is reasonable to attempt this low-level interface capability with several vendors so that a number of users and researchers can experiment with, and gain experience by programming a sufficiently large set of tasks with this interface to better understand the real-time control problem, and, by so doing, better understand how to modularize and specify interfaces.

Since even the interface to the Servo Control Level as described above is very complex, a subset of this interface is recommended. Because most vendors calculate a position serve algorithm, this could be considered the minimum joint command data. If others could also offer velocity or torque control, the interface could easily accomodate them. A minimum set of actual commands should be provided such as INITIALIZE, CALIBRATE, TURN-OFF-SERVOS, and SERVO-JOINTS. In like manner, a minimum set of status values should be provided such as EXECUTING and DONE.

Fields in the data specification should be provided for the commanded joint positions, the values to set or reset individual brakes, and some argument to indicate the command number or similar mechanism for the status to echo for positive identification of the particular command that the status refers to.

Additional status fields could contain the actual brake status and the measured joint positions. Timing specifications could be of the form as shown in Figure 6.

An interface of this level provided by all of the robot vendor's would greatly enhance a number of user's capabilities in studying task deconposition, sensory interactive control and large integrated system issues.

ACKNOWLEDGEMENTS

This work is partially supported by funding from the Navy Manufacturing Technology Program.

This article was prepared by United States Govennment employees as part of their official duties and is therefore a work of the U. S. Government and not subject to copyright.

REFERENCES

1. J.S. Albus, A.J. Barbera, and R.N. Nagel, "Theory and Practice of Hierarchical Control", 23rd IEEE Computer Society International Conference, Sept. 1981.

2. J.A. Simpson, R.J. Hocken, and J.S. Albus, " The Automated Manufacturing Research Facility of the National Bureau of Standards", Journal of Manufacturing Systems, Vol.1(1):17-32, 1982.

3. J.S. Albus, C.R. McLean, A.J. Barbera, and M.L. Fitzgerald, "Hierarchical Control for Robots in an Automated Factory", 13th ISIR/Robots 7 Symposium on System Theory, Chicago, IL, April 1983.

4. A.J. Barbera, M.L. Fitzgerald, and J.S. Albus, "Concepts for a Real-time Sensory-Interactive Control System Architecture", Proceedings of the 14th Southeastern Symposium on System Theory April 1982.

5. M.L. Fitzgerald, A.J. Barbera, and J.S. Albus, "Real-Time Control Systems for Robots", to be presented at the Plastics Exposition Conference, McCormick Place, Chicago, June 1985.

6. T.E. Wheatley, J.S. Albus, and R.N. Nagel (editors), Proceedings of the NBS/ICAM Workshop on Robot Interfaces, June 1980.