Control Architecture for the Robonaut Space Humanoid

Hal Aldridge[1], William Bluethmann[2], Robert Ambrose[3], and Myron Diftler[4]

[1] NASA Johnson Space Center, Robotic Systems Technology Branch,
Mail Code ER4, Houston, TX 77058
hal.a.aldridge@jsc.nasa.gov

[2] Hernandez Engineering Inc.
16055 Space Center Blvd, Suite 725, Houston, TX 77062
bluethmann@jsc.nasa.gov

[3] Metrica, Inc
1012 Hercules Blvd., Houston, TX 77058
robert.o.ambrose@jsc.nasa.gov

[4] Lockheed Martin Space Mission Systems and Services
2400 NASA Road 1, C35, Houston, TX 77058
diftler@jsc.nasa.gov

Abstract. The Robonaut project at the NASA Johnson Space Center is building a humanoid robot for use in space. This robot has a control architecture designed to support teleoperation and development of advanced intelligent control to automate complex tasks. This architecture is influenced by the architecture of human brains that embeds sequencing, safety, and control at a low level. The agent based methodology that allows for a peer to peer interaction between independent subelements is also used in this system. This architecture specifies elements called subautonomies that group together sequencing, safety, and control functions while allowing the elements to be networked similar to agents. This architecture provides a robust and safe environment for advanced humanoid intelligence research by providing low level functionality with system safety implicit in the design. The architecture has been implemented on the Robonaut using experience gained from another humanlike NASA robot project, the Dexterous Anthropomorphic Robotic Testbed. The Robonaut shows the capability of supporting complex orbital, planetary, and medical tasks.

1 Introduction

Control system development for humanoid robots faces several significant technical challenges. These challenges relate to the complexity of the system and its required tasks. Humanoid robots by design are very high degree of freedom (DOF) systems. A dual arm system with dexterous hands has approximately 30 DOF and a full

humanoid can exceed 60 DOF. These motions must be coordinated and controlled safely and effectively. To accomplish complex tasks in real environments, the control system must be flexible, safe, and have some level of intelligence. This intelligence must plan and sequence autonomous tasks. It must learn new tasks or adapt existing capabilities to meet new requirements. For non-autonomous tasks, the intelligence must assist the human operator in controlling the complex humanoid.

Different architectures have been discussed for robot control and the Robonaut architecture is influenced by several of these architectures. To maintain reasonable computational complexity, most architectures separate the control system into layers [1,2]. Layers are usually groups of components designed for similar functionality and computational requirements. Each level builds on the data provided by lower levels.

The human brain also uses a layered control system. Although not fully understood, the basic functionality that the cerebral cortex requires information from the other sections is known [3]. The cortex expects low level control, primitive sequencing, basic sensor conditioning, involuntary system management, and reactive safety systems to be handled by other sections of the brain so it can concentrate on higher level task control and learning.

Another architectural approach is an agent based system. The agent based approach to artificial intelligence distributes intelligence into subsystems which work together to solve complex problems [4]. This architecture does not necessarily require layering. The agents are organized as peer elements that exchange information as necessary over a shared communication link.

The design and implementation of an architecture depends on the application. The NASA Johnson Space Center has significant experience developing control systems for teleoperated humanlike robots. The Dexterous Anthropomorphic Robotic Testbed (DART), shown in Figure 1, was constructed to determine the feasibility of telepresence based control of humanoid robotics [5]. It has successfully shown the ability of human operators to work with a semi-autonomous control system to perform complex tasks in an intuitive manner.

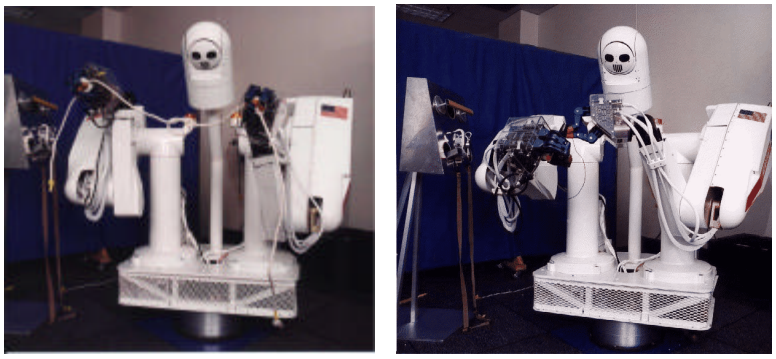

Fig. 1. DART tying a knot and cutting wire

The Robonaut project is using the experience gained from DART to build a humanoid robot capable of working outside the laboratory in the space environment. The goal of the Robonaut project is to provide a humanoid robot, shown in Figure 2, with the dexterity of a suited astronaut to assist astronauts in complex space construction, repair, and maintenance tasks. It contains more degrees of freedom in its arms and hands than DART, enabling more complex tasks. Its mechanical and electrical systems are designed for the harsh space environment. To perform its required tasks, Robonaut will need to incorporate more autonomy than DART to augment and replace teleoperated functions.

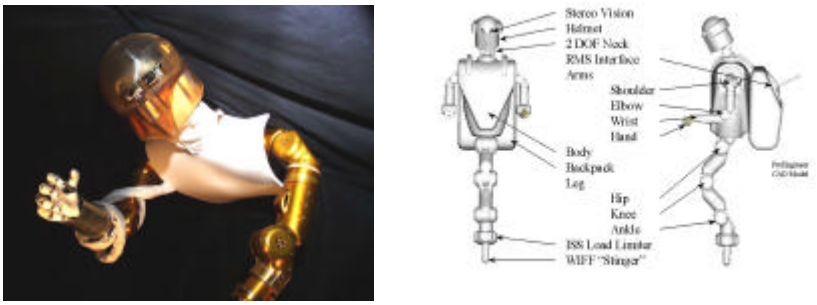

Fig. 2. Robonaut system anatomy

The control architecture for Robonaut is influenced by the human brain, the layered architectures, agent based architectures, and the experience gained with DART. The Robonaut architecture distributes low level control, primitive sequencing, and reactive safety systems in a peer based network. This distribution results in a robust, object oriented control design which will support the development of artificial intelligence, automated learning, and other high level intelligent control functions. The following details the design elements called subautonomies that form the core building blocks of the architecture and gives specifics on the tools and techniques used in the implementation of the controller.

2 Robonaut Architecture

The control architecture for the Robonaut humanoid is being developed around the concept of subautonomies. Subautonomies are independent elements that combine controllers, safety systems, low-level intelligence, and sequencing. The subautonomies work with each other as peers similar to agents.

2.1 Architectural Influences

The method by which brain elements such as the thalamus, cerebellum, and brain stem work with the cortex is a significant part of the brain's architecture [3]. The cerebral cortex interacts with the other elements of the brain by supervising tasks that

are carried out by the other elements. Although it is involved in the original learning stages of a task, as the task is repeated the cognitive part of the cortex is freed to concentrate on higher level tasks such as planning.

While motion control system for a robot can be a very simple system of controllers that follow commands and provide raw information feedback, the brain has evolved a significantly different mechanism. The brain embeds functions such as primitive gaits, muscle monitoring, and other tasks at a low level [6]. Some of these functions are embedded even deeper in the spinal cord and the nerves themselves. The cortex has the ability to actively control or suppress some of these responses but only with significant effort. The training that programs the actions into the proper brain elements allows for fluid and precise control without direct intervention by the cortex.

A brain influenced control design should attempt to emulate this interaction for a humanoid robot. The idea is not to attempt to replicate brain mechanisms but to be influenced by the brain anatomy's breakdown of tasks. Just as robot arm design can be influenced by arm anatomy without building muscles, the control design can be influenced by brain anatomy without building neurons. Neural network or other brain inspired control approaches can be a part of the overall system but they are not necessary to the architecture.

This embedding of functionality into independent subsystems is a design element the Robonaut architecture seeks to emulate. This breakdown has several advantages. It encapsulates functions complete with internal safety and intelligence that can be used by other functions. No single safety system is responsible for all system safety, leading to a more conservative, reliable system.

This distributed organization is similar to an agent based architecture used in artificial intelligence [4]. In an agent based architecture, multiple routines run concurrently, each attempting to perform a function such as optimizing a particular piece of the system. Data is passed between agents as needed, usually across a common communication link. Systems built around agents have been successfully used for robot and humanoid control [7,8]. Distributing the intelligence around the system can enable complex actions by allowing for interaction of proven subsystems that understand their individual parts of the task.

The strength of the Robonaut architecture is its specification of agent characteristics. It takes from the brain the embedding of sequencing, control, and safety at multiple levels. The distribution of the intelligence among elements is related to the agent based systems. The structure and functionality of the individual agents is more strictly defined in the subautonomy model described in the next section.

2.2 Subautonomy Description

System subautonomies can be task sequences, Cartesian control, vision processing, teleoperator interfaces, joint controllers, and grasping control, among others. Subautonomies make decisions as to what services they require from other sub-autonomies to perform the required tasks. Each subautonomy handles its own internal safety and decision making. If a failure occurs, a subautonomy can request a

shutdown or reconfiguration from other subautonomies in addition to performing its own internal safety related functions.

The subautonomies for sensor feedback and motor control in a humanoid robot perform functions similar to the brain's thalamus, cerebellum, and stem. These brain elements take commands from and process data for the cortex. For the robot controller, they form a safe, flexible, and reliable foundation for higher level cognition. These subautonomies can work for software systems of different intelligence levels or directly under human teleoperated control. In the teleoperated mode, the intelligence embedded in the subautonomies forms a shared control system with the operator allowing for safe and effective operation.

Making the sensory and motor systems more independent and less reliant on external coordination allows the high level controller to concentrate on task level goals. The data provided by these systems is preprocessed to keep the possible system states tractable for the intelligent system. This is essential for a learning system that must separate the necessary parts of a task from the unnecessary. Lowering the number for states also reduces the computational complexity of the sequencing or other cortex related functions. A generic subautonomy is shown in Figure 3.

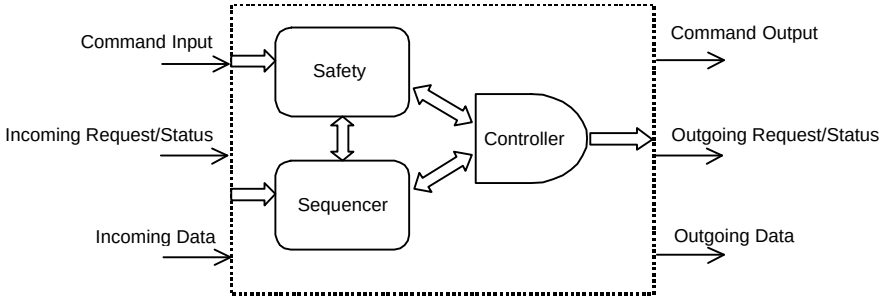

Fig. 3. An example of a generic subautonomy

Within each subautonomy, sequencing, safety, and controller functions work together to form a reliable, independent unit. Safety and sequencing form the basis of the low-level intelligence that configures the controller, protects it from spurious commands, and monitors the controller's states. The triad of safety, sequencing and control allows the subautonomy to operate without reliance upon its peers.

To communicate with its peers, each subautonomy has the ability to send and receive commands and requests/status reports. A command is a synchronous signal while a request/status report is an asynchronous signal. In an arm control system, the output command of a Cartesian control subautonomy would be the input command of a joint control subautonomy. Upon reaching joint control subautonomy, the safety and sequencing aspects would review the incoming command and modify or reject it if necessary. Subautonomies also communicate through the use of data. Data is

synchronous information, but differs from commands because it is used internally by a subautonomy to make decisions, plans, and to execute the control laws.

A request made by a subautonomy is a direct message from a subautonomy to one or more peers. For example, a request comes from a task sequencer subautonomy to a Cartesian control subautonomy asking to transition from an idle state to an active state, permitting the system to enter a Cartesian control mode. As with any message coming into a subautonomy, the safety and sequencing functions review the request and act upon it based on their internal state.

A status report differs from a request in that it is broadcast to all subautonomies in the system. It may be in response to an unexpected event or an announcement of a change in subautonomy's mode. Often a peer will ignore a status report; for example, the sequencer with a teleoperation subautonomy determines that the status report of the completion of the first step of a vision driven grasp of a tool may be ignored. Requests and status reports are grouped together as the primary methods for asynchronous interaction between peer subautonomies.

2.3 Subautonomy Elements

The sequencer function configures the subautonomy for the commanded mode and executes the primitive actions. As required, the sequencer will communicate with other subautonomy sequencers to request mode changes to support the required actions. A hierarchy among subautonomies exist which determines which can request a mode change from others. The system design must make conflicts in requests for services either impossible or allow for arbitration by system level autonomies. This is usually not a problem unless the system is required to satisfy competing goals. For example, the force control subautonomy should not make a torque mode request to the joint controller subautonomy while the trajectory subautonomy is making a position mode request.

The controller function of the subautonomy is designed to meet performance and stability requirements using the appropriate control theory. Humanoid robots must perform a wide variety of tasks. As a result, one gain set and/or controller implementation may not be adequate for all regimes. The controller design must be able to transition between configurations as required by the sequencer.

The safety system is an integral part of the subautonomy. The sequencer sets the safety limits when it configures the subautonomy. The safety system monitors the controller's actions and determines when an action is outside of the operational range. At this point, the safety system informs the sequencer and the sequencer takes appropriate action. This action could range from a warning status message, to a new command limit, to a shutdown request. Although the safety system will act without consent from other systems, it is essential for the subautonomy to inform other subautonomies through status messages of the actions it took. This status information allows other subautonomies to reconfigure as required and helps a learning system understand what it can and cannot do.

Embedding the safety systems in a redundant fashion at the lowest possible level makes system safety independent of the commands. An example of this function in humans is the burn reflex that reacts to prevent harm before informing the cortex.

This functionality enables one of the most powerful methods in learning, the ability to make mistakes with limited damage. Although the redundant safety systems can conflict, causing unnecessary actions, this interaction serves to make the overall system safety more conservative.

The command, data, status, and request variables which are passed between the subautonomies are acted upon as required to perform the functions. The system is organized such that each subautonomy receives the information it needs to make its own internal decisions. Safety related actions are carried out locally in subautonomies with direct access to the appropriate variables or requests are sent to the controlling subautonomy to perform the required action.

The grouping of elements into subautonomies leads to an object oriented design. A subautonomy is a self-contained unit that can be tested individually for functionality and performance. Subautonomies can start off with only basic functionality and evolve at differing rates in the overall system.

2.4 System of Subautonomies

The organization of the subautonomies in a system is similar to an agent based approach [4]. Through data, command, request, and status variables the subautonomies can interact as required. The layering inherent to some architectures is not strictly enforced. Although layering takes place as in many classical systems, the layers are more flexible. Elements that require mode changes of numerous other subautonomies are “higher” task level subautonomies while subautonomies that provide data to or perform actions for numerous subautonomies without requiring many mode changes can be considered “lower” functional level subautonomies. Depending on the situation, the “lower” level systems can overrule the “higher” level systems. This is possible due to the embedding of system specific intelligence into the “lower” levels.

Figure 4 shows the subautonomy system implementation for a single Robonaut arm (without the hand) with a teleoperator interface, a simple task planner, input from a console operator, and impedance force control.

The following example shows the interaction of several subautonomies during a force controlled insertion task.

1. To perform an insertion task, the task sequencing subautonomy sends a mode request to the force control subautonomy to configure force control for an insertion along the Z axis of the manipulator.
2. The force control subautonomy sequencer sets the controller and safety systems to the required states and requests the Cartesian subautonomy accept Cartesian command deltas from the force control subautonomy.
3. The Cartesian subautonomy was not active. The request from the force control subautonomy causes the Cartesian sequencer to enable its systems and send a request for the status of the joint control subautonomy.
4. The joint control subautonomy is active in position control mode and reports its status to the Cartesian subautonomy.

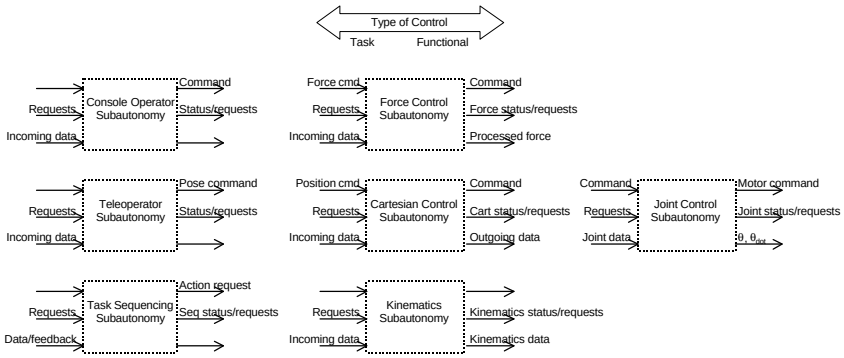

Fig. 4 Robonaut arm subautonomy layout

5. The Cartesian subautonomy accepts the joint control status and completes its initialization. It begins sending joint position commands to the joint controller. It sends out a status message that it is ready and is accepting Cartesian command deltas from the force control subautonomy.
6. With the Cartesian status message, the force control subautonomy completes its initialization and reports its status as compliant in the Z axis.
7. The task sequencer accepts the force control status and continues to the next step.
8. During that step, the manipulator makes contact with the environment and the Cartesian subautonomy reports that the servo error along the Y axis is exceeding tolerance but does not yet exceed the safety limit.
9. The force control subautonomy notes this status and checks the force level on the Y axis. It is high, confirming an unwanted tip contact along that axis. It reconfigures the controller to allow compliance in the Y axis in addition to the Z axis. It reports unwanted contact in the Y axis and its status as compliant in the Y and Z axes.
10. The task sequencing subautonomy notes the force control status and decides that something is wrong with the task. It starts a task shutdown sequence that moves the manipulator away from the contact area.
11. The task shutdown sequence finishes properly. The task sequencing subautonomy sends a request to the force control subautonomy to configure for Z axis compliance only to set up for the next attempt.
12. The force control subautonomy receives the request and checks the force in the Y direction. It is very low so the force control sequencer accepts the request and reconfigures its controller and safety system. It reports its status as compliant along the Z axis.

This example points out some of the features of the architecture. The task sequencing subautonomy only knew that it needed compliance along the Z axis for an insertion. It informed the force control subautonomy what it needed and allowed the force control subautonomy to send the proper requests to configure the system. These requests were acted upon and these actions generated new requests to other

subautonomies not directly involved with the force control subautonomy. The status messages confirming proper initialization were received, concluding with the force controller status that the Z axis is compliant. When the force control subautonomy concluded it had excessive contact in the Y axis through its own data and status of other subautonomies, it acted to correct the situation unilaterally and reported what it did to the system. The subautonomies worked together to satisfy the task sequencing requirements.

2.5 Intelligence

The intelligence embedded in a subautonomy is not restricted to simple sequencing. Any intelligence specific to the subautonomy can be included at this level. For example, the dexterous hand grasping subautonomy could modify its baseline grasps to adapt to new objects. This level of intelligent learning is similar to the cerebellum learning capability [6].

Depending on the level and types of intelligence embedded in the subautonomies, interesting emergent behaviors should be possible. The behaviors will result from the peer to peer interaction between elements as in agent based theory. These abilities may not need to learn or evolve to play a significant role in the overall system intelligence. The actions of a force control subautonomy selectively making axes less rigid while accepting commands from a computer vision based controller could allow for robust manipulation of complex objects without significant artificial intelligence.

The Robonaut architecture is designed to provide support for teleoperation and advanced automation development. It has the capability to build in intelligence at several levels. However, it is recognized that there are other techniques for intelligent control that should be evaluated for use on Robonaut. These techniques do not necessarily need to follow the described architecture.

The Robonaut control system provides data and command paths to other control software through an application programmer's interface (API). The embedded control system built around the described architecture provides intelligent functionality and system safety for the external controller. This breakdown will allow the external intelligence, software or human, to concentrate on task level functions. The Robonaut control system protects itself as required from improper commands while providing intelligent functionality to the external system.

3 Implementation

The Robonaut project presents one of the most interesting humanoid control challenges available today. Robonaut must work safely around multi-billion dollar equipment and humans wearing space suits in a hostile environment. It must perform its tasks reliably to maintain critical systems. These complex tasks require high bandwidth system performance. These tasks also require varying levels of control from fully teleoperated to fully autonomous.

To accomplish these tasks, the control system must provide safe, reliable control for 47+ degrees of freedom. It must maintain performance in a harsh thermal

environment. It must execute at the required rate on reasonable computing hardware. These challenges cannot be met by using only classical robot control methods. Advanced control theory in the areas of grasping, force control, intelligent control, and shared control must be developed to the point where the control is suitable for critical applications to fully realize the capability of Robonaut.

Robonaut is required to perform diverse tasks. Robonaut must use the same tools that astronauts use, in order to reduce the launch weight and development effort required for robot specific tooling. The manipulation and use of these tools is the key to the ability of Robonaut to accomplish the tasks for which it is designed. Figure 5 shows the basic capability of Robonaut to perform tool handling tasks under teleoperation. Robonaut has the capability to handle orbital, planetary, and medical tool types among others. Some of these tasks will become more automated as more advanced control techniques are implemented.

The subautonomy based architecture described here is the basis for the control design. The next sections cover some of the implementation details, design techniques, describe experiences from the DART project that influenced Robonaut, and other issues involved in the Robonaut control design.

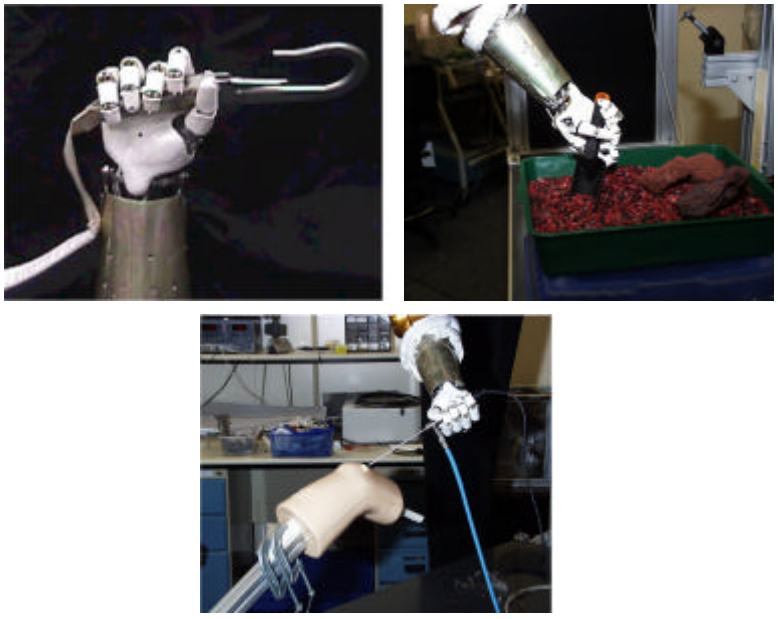

Fig. 5. Robonaut performing space, planetary, and medical tasks.

3.1 Robonaut Computing environment

The computing environment chosen for the Robonaut project includes several state-of-the-art technologies. The PowerPC processor was chosen as the real-time

computing platform for its performance and its continued development for space applications. The computers and their required I/O are connected via a VME backplane. The processors run the VxWorks™ real-time operating system. This combination of flexible computing hardware and operating system supports varied development activities.

The software for Robonaut is written in C and C++. ControlShell™, a software development environment for object oriented, real-time software development, is used extensively to aid in the development process. ControlShell provides a graphical development environment that enhances the understanding of the system and code reusability.

Due to the requirements of the space mission, Robonaut can only carry a limited amount of computing capability. As a result, the controller designs chosen for implementation must be tractable with reasonable computing resources in real-time. This is one of the reasons behind the teleoperation used in current development. The amount of computation realistically carried using current computers limits system development to subautonomies that will enhance sensor feedback and motor control. In the near future, these functions will be ported to faster computers that can be successfully embedded in the Robonaut system. Initial proof of concept development for advanced intelligent control systems will be done utilizing external computing resources and the API.

3.2 DART Experience

The DART system with the Full Immersion Telepresence Testbed (FITT), shown in Figure 6, provided the starting point for the telepresence aspects of the control architecture currently used by Robonaut. DART and FITT use a distributed architecture with all subsystems receiving and sending commands via a router. The subsystems are distributed over a number of CPUs all connected via Ethernet. These subsystems are an earlier version of the subautonomies noted above. They contain the basic features of a subautonomy but are not object oriented in design.

This router based DART/FITT system works well for low bandwidth teleoperator commands such as position control and simple mode changes. Higher bandwidth responses such as impedance control are performed locally on individual processors using high speed I/O. In a general sense, Robonaut adheres to this same philosophy, but eliminates the router based system in favor of a VME based shared memory supplemented with Ethernet based communication. Several important lessons learned from DART/FITT [5] are incorporated in the subautonomies used by Robonaut.

This router based DART/FITT system works well for low bandwidth teleoperator commands such as position control and simple mode changes. Higher bandwidth responses such as impedance control are performed locally on individual processors using high speed I/O. In a general sense, Robonaut adheres to this same philosophy, but eliminates the router based system in favor of a VME based shared memory supplemented with Ethernet based communication. Several important lessons learned from DART/FITT [5] are incorporated in the subautonomies used by Robonaut.

Fig. 6. DART/FITT system

The DART arm subsystem can receive position commands from either a teleoperator based client or an automated client. One of the early enhancements to this subsystem came out of initial teleoperator testing which revealed the need for relative motion control for several reasons. While DART is anthropomorphic, its arms are longer than a typical operator's arm and it has greater than human travel in all joints. In addition, the operator needs the ability to have the robot work at full extension, while keeping his own arms in a relatively comfortable pose. To take advantage of the robots capabilities and accommodate the operator, the arm subsystem provides, on request, current position information to client processes. Teleoperator commands are easily combined with this data, allowing the operator to re-index the relative motion at any point in time.

Additional arm features that are useful building blocks when developing high level controllers include: coordinated dual arm motion, compliance control, and kinematic solution selection capability. In dual arm mode, the arm subsystem accepts position commands for a point of resolution (POR) centered between the two arms and then resolves them back into commands at the individual arm PORs. Compliance control utilizes two force/torque sensors and is available with all other arm operating modes. Given the mounting of the PUMA arms shown in Figure 6, four solutions are available for any kinematic pose and orientation of each arm. Flipping the elbow yields two solution and flipping the wrist yields two more. The arm subsystem accepts commands to move between these four solutions in a controlled manner for obstacle avoidance or to enhance operator viewing.

The DART end effectors are Stanford/JPL hands, and while dexterous, these hands are not anthropomorphic. Each finger has three joints, and the thumb directly opposes the other two fingers that are kinematically dissimilar to a human finger. This makes simple joint or Cartesian teleoperator control of the Stanford/JPL hand difficult. If the human operator is trying to perform highly dexterous tasks, his intentions may not be mapped properly to the robot. The DART/FITT solution to this problem is to map not only hand position, but hand functionality as well.

Venkataraman and Iberall [9] identify a partial taxonomy of grasps used by machinists when working with metal parts and hand tools. From this partial taxonomy, a useful set of voice-invoked grasp primitives are made available for control of the DART robotic hands. These grasp primitives consist of pinch grasp, key grasp, hook grasp, spherical grasp, and cylindrical grasp. The spatial configuration of the fingers is modulated by the human operator and mapped into one of the primitive grasp geometries available within the hand subsystem. This primitive approach to shared control provides for the mapping of finger positions as well as mapping the functional intention of the human operator. With this method of control, the DART/FITT system is able to perform a larger variety of tasks more efficiently and productively.

Health monitoring is an important part of a subautonomy. The DART subsystems include self monitoring that prevents damage and also sends out messages to other subsystems when limits are being approached. The arms track limits and singularities and when either is approached, a message is sent to the voice subsystem that provides an audio command alerting the teleoperator to the situation. Similarly the fingers on the Stanford/JPL hand can use the friction in their cable drive train to their advantage and actually resist more force than they can actively apply. In certain instances this is useful, but the overall cable tension still must be limited. The hand subsystem monitors the tension and initiates similar commands to the voice subsystem when then tension approaches excessive levels. At sufficiently high tension levels the hand will shut itself down to prevent damage.

3.3 Control System Prototyping

The Robonaut program also uses the Cooperative Manipulation Testbed (CMT) facility shown in Figure 7 to develop and test software and control strategies. The CMT is made up of three manipulators and their tooling. All three manipulators are seven DOF devices. Two manipulators are identical while the third is a larger, scaled version of the others. This similar/dissimilar arrangement allows for testing of homogenous and heterogeneous tasks. The smaller manipulators have three fingered hands for tooling. This flexible tooling allows the manipulators to handle a wide variety of tasks. The larger manipulator has a quick-change mechanism allowing it to autonomously change special purpose end-effectors. All manipulators have six axis end-effector force/torque sensors and joint torque sensors for high bandwidth force control. The computing and development environment for CMT is identical to the Robonaut system for rapid software transfer.

The use of CMT to augment software development for Robonaut has been successful. Subautonomies such as Cartesian control and force control have been prototyped and tested using CMT and quickly ported to Robonaut. Although the mechanical hardware is dissimilar, the physical capabilities, with the exception of grasping, are similar. The identical computing environment and the object oriented design of the architecture allows rapid software exchange between the two systems. The capability to develop software using a system that is more available for test than Robonaut and incorporates future features of Robonaut that are still in development reduces the overall software development cycle.

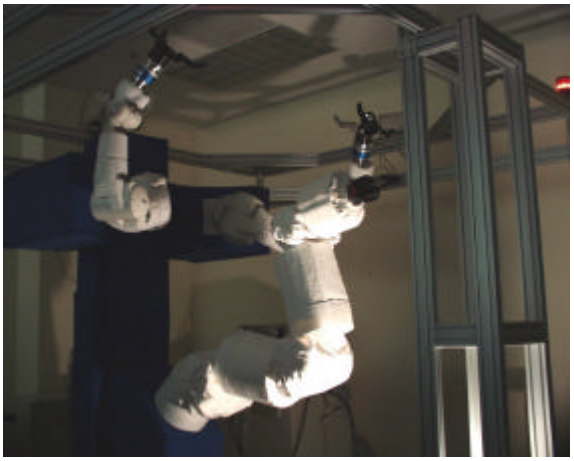

Fig. 7. Cooperative Manipulation Testbed (CMT)

3.4 Primitive Based Automated Grasping

The initial development of primitives is required for teleoperator assistance. These primitives use both force and position data as required by the task they are automating. When using primitives, the operator is not required to directly control all the hand axes. The primitives interpret the operator's glove commands and map them to multiple hand axes making the required decisions based on hand sensor data.

The first finger primitives being tested are similar to the ones implemented with DART. On Robonaut the impetus for the primitives is a little different. The Robonaut hand is a more anthropomorphic design than the Stanford/JPL hands on DART. This design makes operator to humanoid finger mapping less of an issue. However, the operator will not be holding the same object as the robot. In this case ease of use and workload become issues. If Robonaut needs to spread its fingers to grasp a spherical object, the human will very quickly become uncomfortable palming the virtual object. A spherical primitive will allow the operator to maintain a comfortable finger separation while Robonaut maintains the required spread. Similarly, when only two fingers are required to grasp, for example tweezers, a primitive that automatically moves all other fingers out of the way is very useful.

Primitives are also useful in repetitive tasks and fine motion operations. A good example of a repetitive task is manual bolt tightening or dial spinning. Robonaut has a primitive that commands 6 degrees of freedom in the hands using only two joint inputs from the operator. The operator lines up the Robonaut hand with the bolt and then simple steps through the primitive using relatively coarse inputs. The Robonaut fingers reposition themselves precisely throughout the cycle and the operator's work load is significantly decreased. Primitives can also be used to readjust the gain between the human and the robot. When precision motion is required, 50 degrees of

human finger motion can be converted into 5 degrees of robot finger motion. Robonaut has the capability to exceed nominal anthropomorphic mapping in many instances.

The use of primitives is the first step leading to an automated grasping subautonomy for Robonaut. The general grasping problem for dexterous hands using enveloping grasps is currently too computationally complex for the Robonaut control system. Instead of solving the general problem, discrete grasp primitives will be defined and studied. Metrics used to evaluate the progress of the primitives in accomplishing a task will be tested experimentally. These primitives and metrics can be sequenced to perform complex operations. The safety system that determines when a grasp is about to fail, or when fingers are colliding among other things, will be embedded at the subautonomy level.

4 Conclusions

The Robonaut control architecture has been designed to build a robust and safe foundation that supports teleoperation and will enable development of intelligent control. The subautonomy based architecture embeds safety, sequencing, and control at all levels. The distribution of intelligence and safety through the system enhances safety and improves functionality. The self-contained design of the subautonomy leads to an object oriented system whose elements can be tested independently. The Robonaut embedded system supports advanced development in humanoid intelligence by providing system safety and intelligent functionality to other types of intelligent control systems. The architecture has shown benefits in teleoperated control that should translate into enabling capabilities in advanced automation.

References

1. Albus, J., McMcain, H., and Lumia, R.: NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NBS TechNote 1235, National Bureau of Standards, Gaithersburg, Maryland, (1987)
2. Bonasso, P., Firby, R., Gat, E., Kortenkamp D., Miller, D., and Slack, M.: Experiences with an Architecture for Intelligent, Reactive Agents. Journal of Experimental and Theoretical Artificial Intelligence, vol 9, no 2 (1997)
3. Molavi, D.: Neuroscience Tutorial. The University of Washington School of Medicine, (1997)
4. Wooldridge, M., and Jennings, N.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review, vol 2, no. 2 (1995)
5. Li, L., Cox, B., Diftler, M., Shelton, S., Rogers, B.: Development of a Telepresence Controlled Ambidextrous Robot for Space Applications. Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN (1996) 58-63
6. Albus, J.: A Theory of Cerebellar Function. Mathematical Biosciences, 10 (1971) 25-61
7. Mori, A., Naya, F., Osato, N., and Kawaoka, T.: Multiagent-based Distributed Manipulator Control. IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (1996)

8. Peters, R., D. M. Wilkes, D. M. Gaines, and K. Kawamura: A Software Agent Based Control System for Human-Robot Interfaction. Second International Symposium on Humanoid Robots, Waseda University, Tokyo, Japan (1999)
9. Venkataraman, S.T. and Iberall, T.: Dexterous Robotic Hands. Springer-Verlag, New York, (1990)