DISTRIBUTED CONTROL AND SCHEDULING FOR ROBOTS

Krithi Ramamritham and Michael Arbib

COINS Technical Report 88-90

DISTRIBUTED CONTROL AND SCHEDULING FOR ROBOTS

A Progress Report on work conducted under NSF Grant DMC-8511959

Krithi Ramamritham
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01002
(413) 545-0196

Michael Arbib
Department of Computer Science
University of Southern California
Los Angeles CA 90089
(213) 743-6452

ABSTRACT

One of the most challenging problems today is the building of robots which have some measure of the versatility of a human. As generally defined, a *robot system* consists of a *manipulator*, a set of *sensors* and a *controller*. Complex controllers use *sensory input* and some *internal task description* to make the response of the manipulator appropriate to the external environment of the robot. This report discusses ongoing work that addresses the conceptual and practical problems underlying the design and implementation of such controllers. In particular, it focuses on three facets of our work, namely the development of a formal distributed model of robot computations, the development of scheduling algorithms and architectures appropriate for implementing such a model, and the development of control-theoretic solutions for task-oriented robot control.

1 Introduction to the Problem Domain

One of the most challenging problems today is the building of robots orewhich have some measure of the versatility of a human. As generally defined, a *robot system* consists of a *manipulator*, a set of *sensors* and a *controller*. The manipulator is that part of the system (normally mechanical) which is used to effect changes in the robot's external environment. Each sensor 'measures' some property of the external world. The controller is that element of the robot system which determines the behavior of the manipulator.

In the most common class of robot system, the *Industrial Robot*, the manipulator is usually a single "arm", a series of rigid links connected by controllable joints. Controllers in such simple robots just store a sequence of joint position values which they transmit to the joint actuators in some fixed sequence. More complex controllers use *sensory input* and some *internal task description* to make the response of the manipulator appropriate to the external environment of the robot.

Our work addresses the conceptual and practical problems underlying the design and implementation of such complex controllers. In almost all cases, this controller is a general-purpose computer. The programs which are executed on this computer are not, however, general-purpose programs -- they can be placed in one of the following classes:

1. Servo-Control: These programs usually implement a control-theoretic approach to ensuring that parameters of the manipulation structure, such as joint positions, velocities, torques, impedances, etc., can be reliably maintained.

2. Sensory Processing: Programs which refine or combine the raw data produced by the robot's sensors. These may include programs for object recognition -- classifying objects in the robot's (immediate) environment into parameterized instances of some set of object classes.

3. Tactical Control/Planning: Programs which break down the description of a single atomic operation to be carried out on an object (one of a fixed class of such operations), into a series of commands to the servo-control and sensory processing programs. Operations which are referenced against objects, as opposed to parts of the manipulator, are called *task-level* actions.

4. Strategic Planning: Programs which take a high-level, goal-oriented description of some complex task and determine a sequence of atomic task-level actions which will result in the robot carrying out the task. A *Dynamic Planner* is a strategic planner which can determine from sensory information if some part of its planned sequence of operations has failed, and how to resolve this failure.

Whereas, at the Laboratory of Perceptual Robotics, University of Massachusetts, all of the above four topics are being investigated, through the current grant from the National Science Foundation, we are focussing on topics one and three as well as their synergism.

The vast majority of robots today are *Industrial Robots*; these occupy the lowest rung of the robot hierarchy. Usually, they have only primitive position sensors, a rudimentary servo-control system, and not much else. Increasingly, the trend in robot construction is to build robots with multiple, possibly redundant, degrees of freedom, and which have many sensor systems. We shall refer to such a robot as a *complex robot system*. These robots exist currently in research environments only — not because they are specially difficult to construct, *but because they are difficult to control.*

To provide focus for our investigations, we have been working in the context of one particular complex robot domain. As we have mentioned, there currently exist a number of complex robot systems. We choose as our example domain, that of grasping and manipulation with a *dextrous robot hand*. The control of a dextrous hand involves many of the problems central to robotics: coordinated, coherent control of multiple and redundant degrees of freedom, the use and fusion of complex sensory information, and the development of an appropriate task-level view. Potentially, the dextrous

robot hand offers an improvement in the performance of robot assembly systems since it is more versatile than standard two-fingered, parallel-jawed robot grippers. Such a hand can grasp a wider range of objects, and manipulate held objects. However, deriving the principles behind the use of the mechanism involves understanding both the physical interface between hand and object, and also the task context constraints on the hand.

One of the problems in the control of a complex robot lies is the fact that robots are currently being programmed using general-purpose programming languages, or dialects of such languages, in which the robot is treated simply as a peripheral device. It is difficult to program complex robot systems to exhibit their full potential for versatile and adaptable behavior with the tools developed for general-purpose programming. Our work is based on the premise that the computation carried out by a robot controller is a *special class of computation*. In order to be able to program complex robots well, it is necessary to construct a *model of computation*, a way to describe how computation occurs, which is specifically tuned to the robot domain. The first part of our work involves the specification, construction and analysis of just such a model of computation. Details of our work in this area are provided in Section 2.

The second part of our investigations concern the implementation aspects of robot computations. Here, we are studying the issue of mapping the components of the distributed model of computation developed in the first part onto one or more processing elements that form the computational engine supporting the functioning of the complex robot. Specifically, we are experimenting with different *scheduling algorithms* and different *parallel architectures* appropriate for our domain. Section 3 outlines our efforts in this area.

Among the many shortcomings that we have to overcome in realising the full potential of a complex robot such as a dexterous hand is the issue of accurate, reliable and robust control - after all, the power of any end effector is measured only in terms of how *effective* it is in performing tasks. The need, then, to systematically study the control of such end effectors

performing various tasks is now more acute. As discussed in Section 1, the third part of our work is directed towards addressing some of the issues involved in the *task dependent* control-theoretic aspects of complex end effectors. As it turns out, the computations involved in such control can potentially be expressed via the model discussed in Section 2.

2 A Formal Model for Robot Computation

We began our work in this area by investigating what was involved in programming a particular complex robot system, a dextrous hand. In this regard, we made a set of four observations about the computational characteristics of the robot domain which were used as the fundamental principles upon which the computational model was constructed. These four observations concerned:

1. The inherent parallelism in robots and robot programming.
2. The special role of perception and action in robotics, and the connection between the two.
3. The usefulness of prototypical action plans and object models.
4. The need for formal semantics and verification methods.

With these basic specifications, we constructed the *RS* (for *Robot Schema*) model of computation. *RS* is a model of distributed computation to suit the parallelism inherent in the robot domain (Observation One).

To permit specification of both the parallelism and the interdependence between parallel activities in the robot domain, we use the concept of *distributed computation* as our most basic structure. Computation is performed in a distributed model by the *interaction* of a number of *concurrent* computing agents. A computing agent subsumes the concept of a process,

but can also refer to hardware; it is simply a locus of computation. The concurrency could be virtual, as in a time-sharing computer system, or real, as occurs in parallel and distributed systems. In section 3 we show how the actual parallelism can be realized by mapping these computing agents to physical computing entities, i.e., processors.

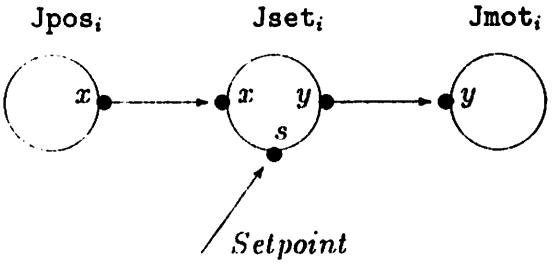
A *schema* is a generic specification of a computing agent; each schema describes a *class* of computing agents. A computing agent is created from a schema by the *instantiation operation*. The term *schema instantiation*, or SI, and the term computing agent are synonymous in *RS*.

To allow communication between concurrent SIs, we define each SI to have a set of communication objects called *ports*. Each SI has a set of *input ports* and a set of *output ports*. Data is sent from an SI by writing to an output port; data is received into an SI by reading an input port. The instantiation operator takes, as parameters, the *connections* for the ports of the new SI. With each port we associate a *type*. This allows us to form rules about what may or may not be connected to a particular class of computing agent. This communication mechanism is a form of *message passing*.

For example, let `Jset` describe a simple position servo process; taking current position through some port x, a position setpoint through some port s, and producing appropriate motor output on some port y. The instantiation operation allows us to build arbitrary computing agents fashioned after this schema, and depending on exactly what is connected to the input and output ports, the servo process can be applied to any combination of sensors and actuators.

A key parameter in such a servo process might be some gain k. Notice that this parameter is *static* in comparison with the input and output port connections. The gain can be initialized when the SI is created, and need not be changed again. The instantiation operation can specify *initial parameter values*, such as an initial value for k in `Jset` for example, in addition to dictating what connections are applied.

We predefine a set of *primitive schemas* to interface the *RS* model with the robot and with the world. Instantiations of these primitive schemas are computing agents which represent the sensory input and motor output 'channels'. Computing agents representing sensory data are updated *dynamically*, and computing agents representing motor or actuator output control their hardware *dynamically*. For example, Jpos is a primitive sensory schema, which reports a joint position through its output port x. The *ith* instantiation of Jpos is a computing agent which reports on the current joint position of the *ith* joint of the robot system (under some prearranged numbering of the robot mechanism). In similar fashion Jmot is a schema which accepts a motor input through its port y. The *ith* instantiation of Jmot is a computing agent which controls the *ith* actuator (the actuator of the *ith* joint) of the robot system. If we have a position servo process such as an instantiation of Jset, then we *establish the following convention*; to use the *ith* instantiation of Jpos and Jmot create the *ith* instantiation of Jset, such that:

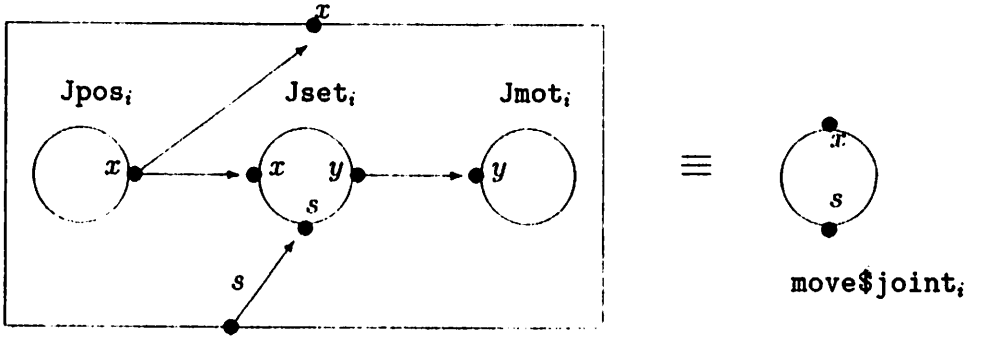


The above mechanism based on the use of schemas as building blocks was chosen to allow for prototypical action plans and object models (Observation Three). We have included a number of structuring concepts to construct robot computations out of these building blocks.

An *Assemblage SI* is a computing agent whose behavior is defined by the interaction of a network of communicating SIs. This aggregate SI can be considered the instantiation of a single *schema*, an *assemblage schema*,

which must contain information on how the individual SIs are created and connected, and how the ports of the component SIs appear as the ports of the assemblage SI. The assemblage construct was introduced to structure both object models and plans (Observation Three), to allow local groupings of control SIs (Observation One), and also to form the basis of the task-unit (Observation Two).

For example, the network of `Jpos`, `Jmot` and `Jset` previously discussed can be described in generic fashion using an arbitrary instantiation number i. We describe this network as an assemblage schema `move$joint`; the *ith* instantiation of which *internally* will be the network of *ith* instantiation of `Jpos`, `Jmot` and `Jset` connected as described, but *externally* will simply be a position servo process for the *ith* joint. It will have one input port s through which a desired position set point comes and one output port x through which the current position is read, i.e.:



The *task-unit* is a special assemblage with designated sensory and motor components. The task-unit represents a task; its perceptual SIs are the object model(s) for the task, and the motor SIs the basic motor actions of the task. This reflects the parallelism between perception and action, and also the special connection between the two.

Following this, we developed the formal semantics for the *RS* model based on the port automaton model of Steenstrup, Arbib and Manes. Our first concern was to build a version of a port automaton which suited our schema definition. Having satisfied ourselves that we had not lost any of the power or versatility of the original port automaton model, we began to construct the semantics of *RS*. The important steps here were to be able to characterize well the behavior of SIs, the nature of communication between them, and the operations for creating, deleting and aggregating SIs.

We have also developed a temporal logic methodology to write schema specifications, and to verify if a schema, or a network of SIs, satisfied that assertion. The trace model of Nguyen, Gries, and Owicki (from Cornell University) provided the basic structure of the temporal logic model. The definition of the *input-output trace* of a port automaton was then used to adapt this model to *RS*. Extensions to the basic model included dynamic process creation and deletion and world-descriptive predicates (reflecting the structure of the world external to the robot).

A number of examples have been implemented in the model, including one of a simplified grasping and manipulation system, in order to demonstrate its representative power and the validity of the assumption that developing a special model of computation for robots allows the specification of complex robot behavior.

What we have accomplished so far is simply the start. Based on the experience derived from implementing the schema model in a distributed environment, we have enhanced the model, in particular to facilitate programming sophisticated forms of robot control. Some of the enhancements are described in Section 3.

Some formal topics also need further attention. Currently, one of the weaker parts of *RS* is the verification and design methodology. In concept, it is correct; however, the actual mechanisms for expressing schema assertions and for verifying them are too complex. Part of this complexity is inherited from the behavior definition of the Nguyen et al. trace model. A likely

approach here would be to redefine behavior more in terms of the *input-output trace* of the port automaton. As well as simplifying the definition of behavior, this would have the added benefit of a more uniform approach to both schema verification and assemblage verification.

Our use of world-descriptive predicates to axiomatize the behavior of the real world is a very useful tool. The primitive schemas of *RS* then provide an excellent way to 'involve' these predicates into the description of process behavior. Another profitable avenue of research is to determine a useful set of world-predicates for dealing with real equipment. So far, we have adopted a heavily motor-oriented view of complex robot systems; we have neglected many sensory processing issues. This is one major area in which *RS* needs development.

3 Environment for Robot Control (ERC)

The schema model poses several interesting problems when one attempts to implement it on an actual computer system. The first problem arises from the fact that a task in a robot domain has real time requirements which have to be met by the physical realization of the schemas that express the task. Secondly, the number of schemas that may be created in the process of executing a task may be much larger than the number of processors in the computing environment through which the robot is control. Hence an algorithm needs to be devised to map the schemas to processors and to order the execution of schemas assigned to each processor. The third problem is that both the schema environment as well as the robot workplace environment are dynamic. Thus we are unable to tell a priori which schemas will be activated and when they will be activated.

What we need is a dynamic scheduling scheme that will function in a parallel or distributed environment. Even though several scheduling policies have been proposed in the literature, none of these is suitable for the schema environment. The problem with most of the proposed policies is

that they are static (in terms of when processes will be created and destroyed) in nature and are designed for simple process structures – ones in which processes are independent or form a chain. A question of practical significance that arises in the context of implementing the schema model is the determination of the best computer architecture appropriate for the schema model. Currently several different architectures have been proposed and developed that are used to control robotic systems, such as the UTAH-MIT hand. These systems were all designed so as to provide the necessary processing power to control the robot that they are connected to. However, no study has been made to understand which architecture and scheduling algorithm is most suitable for controlling complex robots. Following from this, one of our major goals in this research is to understand how different architectures and scheduling algorithms affect the performance of controlling a robot.

The necessary understanding could be gained by experimenting with actual computer systems at what would probably be an enormous expense. Hence we have chosen to develop a simulation engine that will enable us to simulate competing architectures as well as different scheduling algorithms. From the results of the simulation studies, we hope to be able to determine the most suitable architecture and operating system *suitable for the schema model of computation* given a certain set of contraints, e.g. robot domain constraints, real-time requirements, fault tolerance, etc.

Our main tool in observing the performance of the different architectures and scheduling algorithms is a graphics system. The output derived from the simulation system is used to drive a graphical representation of a robotic workspace. From this graphical display, we can view the performance achieved using the different architectures and scheduling algorithms. However, due to the complexity of simulating dynamics, the engine will only simulate the kinematics of the robot. Thus, in addition to the purely graphical information, the simulation system will also output other statistical information that will aid in the evaluation of the performances.

The design and development of the simulation system, known as the

ERC, has been broken down into three phases. The first phase involves giving a concrete shape to the schema language that enhances its programmability and provides information necessary for the scheduling algorithm. The second phase is the development of the computer architecture and scheduling algorithm simulator. The third phase is the development of the robot workspace simulator. These three phases are discussed in more detail below. The first phase has been completed. The two simulators have been designed and implemented and experimentation is in progress.

The basic schema model described in the previous section has been given a programmable form using a subset of C. To provide the necessary information to the scheduling algorithm for making processor assignment and ordering decisions, several enhancements have been made. These are intented to determine the computational load that results from an SI. For a simple SI, one containing no loops or branches, the load is defined as the sum of the load caused by each machine instruction to be executed by the SI. In complex SIs, this is no longer true. Hence a means of *unfolding* SIs was developed: In conjunction with a loop, the programmer must specify the average and maximum number of times the loop is to be executed. Once these values are defined, the loop can then be *unfolded* by multiplying the load of a single pass through the loop by the specified values.

For a branch, the programmer specifies the average and worst-case probabilities of taking a particular branch. The branch structure can then be *unfolded* by multiplying the load of the branch by the probability factor. For example, with the **IF** structure, the user specifies the probability that the if-condition is true. Since in the unfolding process, two load values are produced, every schema has two load values associated with it, an average load and a maximum load. These values are used by the scheduling algorithm in balancing the load of a network of processors.

In addition to these changes in the schema language, a change in the basic understanding of the schema was necessary. In the original model, once activated, an SI repeatedly, i.e, cyclically, executed its body until it either halted itself or was killed by another SI. We have extended this

model by introducing five new process types. These are the

1. the assemblage process,
2. the aperidoc process,
3. the cyclic process,
4. the time-delay periodic process, and
5. the time-interval periodic process.

Associate with each of these processes is a deadline that is used to indicate the time at which the process must complete execution. In addition, we have associated a period and a bumping window with the time-delay and time-interval periodic processes. For the time-interval periodic process, the period is the amount of time between successive invocations of the process whereas for the time-delay periodic process, the period is the amount of time between the completion of one invocation and the start of the next. Finally, a bumping window is associated with each periodic process. This window is used to determine whether an instance of a periodic process can afford to miss its deadlines.

Once the design of the new version of the schema language was completed, a new compiler was constructed to compile the source code into a form that would be accepted by the ERC simulation engine. Now we move to the problem of mapping SIs to processors in the robot's computing environment. Our initial investigations revealed that a parallel processing environment provides the best support for executing such sensory and control processes since it will allow us to reconfigure processing power as demands change (provided the scheduling algorithm is dynamic). Thus most of our effort has been on the development of a dynamic, scheduling algorithm that will support the execution of the SIs in a truly concurrent environment.

The simulation engine has been designed to compare the performance achieved by a variety of parallel computer architectures in combination

with different scheduling algorithms. On a system that contains several processors, a local scheduler needs to be provided that will assign SIs to different processors as the SIs are activated and the processors become free. The basic implementation of this local scheduler has been completed. It consists of four queues that hold the SIs as they move between ready, blocked, sleep, and running states. It also consists of the functions necessary to move the SIs from one queue to another and to start and terminate the execution of an SI on a processor. A major problem with this basic scheduling scheme is that as the number of processors increase, so does the contention for the system queues. This contention can become so great that it is the central bottleneck of the system. To handle this problem, we are now exploring mechanisms whereby several sets of system queues are used.

In addition to the scheduler, primary operating system functions needed to be designed and implemented. One such function is message passing between SIs. Another is the Instantiate and Deinstantiate functions, that is, the schema creation and destruction functions.

In order to observe and evaluate the performance of the different architectures and scheduling algorithms, the executing SIs must be connected to a simulated robot workspace. For this purpose, a simulator has been created. This simulator performs two primary functions. The first is that it simulates the kinematics necessary to determine how a robot has moved over a certain time period given a set of velocities. This information can then be accessed by the executing SIs by reading simulated motor counters. In addition, the simulator also writes necessary information out to a file that can later be read by a graphics display program. Thus the user can watch the actions taken by the simulated robot and evaluate the performance achieved using the different architectures and scheduling algorithms. As stated above, the graphical display will not be able to display the actual dynamics found in the workspace. Thus additional statistics will be reported that will aid us in evaluating the performances.

The second function of this simulator is to simulate both tactile and

visual sensing. The tactile sensing has been implemented and consists of polygon intersection routines. Contact information is held in simulated tactile sensors that can then be accessed by the executing SIs. In addition, contact information is written out to a file to be later used by the graphics display program. The visual sensor basically returns information concerning the location and type of all objects within view.

In order to evaluate the performance of the different architectures and scheduling algorithms, we have currently chosen two metrics. The first metric is concerned directly with the performance of the operating system. One of the major goals of this system is the scheduling of processes in such a manner that these processes meet their constraints, i.e., deadlines. Thus, as the system is executing, the number and type of processes that miss their deadlines are recorded. The second metric is based on a much higher level concern, namely the response time of a robot. If the response time is too long, a robot can seriously damage itself or other objects in its workspace. To measure response time, ERC reports when contact is made between a robot and an object. This is accomplished by reporting the time that a tactile sensor changes from a non-contact to a contact state. This will be followed by the time the sensor is actually read by an SI, i.e., the time the control system becomes aware that it is in contact with an object. Finally the time when the joint having the tactile sensor is stoped will be reported. From these three values, it is possible to get a measure of the response time. The smaller the differences between these times, better the system's performance. In addition, we will also be able to tell where most of the response lag is, in the sensory processing or in the motor control processing.

In summary, in the area of implementing the schema model, we have solved most of the conceptual and design problems. Once we complete the ongoing experimentation, we will be able to report on the efficacy of different architectures and competing scheduling strategies.

4 Task Dependent Dextrous Hand Control

The true test of success for our approach to solving some of the issues in controlling complex robots will be in an actual implementation. There will be a substantial increase in the computational load while considering the dynamics of the robot, as well as its control algorithms. In addition, the operating system has to meet with the requirement of a fixed servo rate. ¿From the mathematical standpoint, we now need a formal structure to integrate the *task level* with the *robot level*. We use a *task representation* to enable us to formally specify tasks, and integrate them within the context of robot control. We use *task dependent dynamic modelling* to specify the dynamics of robot(s) and environment within the context of the task, and finally, use *task dependent control architectures* to actually execute the tasks. We refer to the whole process of low-level task planning and execution as *task dependent robot control*, and in the context of this report, *task dependent dexterous hand control*. We have focussed on three pertinent issues. These are:

- *Interaction behaviour*: Since, inherently present in most complex robot tasks are robot to environment dynamic interactions, we first systematically study this.
- *Task Representation*: Since any command that the control system recieves is a task level command, we need a representation of tasks in the control structure, within the context of interactions. As, shown later, this results in a small but powerful vocabulary of task primitives that can be (partially) sequenced to succinctly represent most prototypical robotic tasks in the control structure.
- *Task Control Strategies*: And, finally, we present a control architectures for task dependent control of dexterous hands. For simplicity, in this work, we have used *Multivariable Feedback Control* strategies for the design of controllers.

Presently, representation of robotic tasks are either very specific to robotic control, or to task planning. For example, representation of the task of robot tracing a surface is done either (control specifically) as combined position control and force regulation, with the appropriate (robotic) compliance measures or, (planning specifically) as unconstrained and constrained degrees of freedom in the configuration-space, along with appropriate task compliance measures. Such representations describe only partially the interaction behaviour between the robot in contact and the environment. In this work, we study interactions based on the *theory of contacts* and then develop a representation for tasks within the context of end effector to environment interactions.

The philosophy used here is that in most robotic tasks interactions between the end effector of the robot and the environment is essential, and performing a task entails controlling the interactions in a task specific manner. If the combined dynamic behaviour of the robot and the environment (together) is encoded in the interaction space, control of the robot performing the task in question corresponds to controlling the interaction state variables that affect the task. The collection of such state variables form the task space, and mapping function from the interaction to the task space is the representation of the task.

For example, consider a robot pushing against a wall (at any location). Let the robot be in contact with the wall at some point. The interactions are between the robot and the wall, the interaction space consists of three cartesian force coordinates and three rotational motion coordinates. The task space consists only of the the motion and normal force coordinates of the interaction space. And, the task representation is a *compliant* mapping of the interaction state variables into the task space.

Study of interaction is based on the theory of contacts. That is, if two bodies come in contact with one another, depending on the *type* of contact, we can represent mathematically the constrained (unconstrained) degrees of freedom at the contact. An interaction state space can be defined at the point of contact (interaction port) between the two bodies. Such a state

space will consist of the *wrench* and *screw* coordinates defined by the type of contact. The interaction behaviour is characterised in the interaction state space by the combined dynamic description of the two bodies (robot and the environment) in contact.

The dynamic description of the bodies in contact must necessarily be expressed in the combined form, because each body forms a part of a kinematic chain. In addition, its state space must consist of coordinates (and rates of its change) of the interaction space. This way, task control can be achieved through following prespecified trajectories of the coordinates of the interaction space. All dynamic descriptions in this work are derived using the *variational principle*, at the interaction port, in terms of the interaction state variables.

The *Task Space* at the interaction port consists of interaction coordinates needed to describe the task completely. Control of these variables implies successful completion of the task. For example, in inserting a peg into a hole, axial motions of the peg must be unconstrained, and so must be represented as a task screw coordinate. A bad choice of *task coordinates* therefore results in poor execution regardless of the merits of the control architectures or algorithms. We denote this mapping between the interaction space and the task space as the *task representation*. The representation also includes specifications of compliance directions.

Such a unified representation allows us to not only integrate low-level task specification and execution, but also, permits us to investigate if such a task representation lends itself to higher levels (in particular the *RS* environment). If such is the case, then we can use *task* as the basis of integration of the low level with the tactical, and further the strategic level of control. In addition, such an approach also helps us to investigate the dependence of computational architectures on robotic tasks. That is, the computational architectures may be dependent on the task primitives chosen. Finally, the coherence in representation of task (through the choice of task representation as a basis for integration) allows us to address the issue of feedback across levels of hierarchy - a feature lacking in most existing robotic control

architectures, but a feature much needed in most intelligent systems.

We have performed the analysis on representation of tasks and have the following task primitive definitions. The vocabulary is small, yet powerful. In addition, it can be easily built into a control structure. For dexterous hand usage, the task primitives, with the corresponding control strategies are:

- Preshape – Pure Position Control of Fingers.
- Grasping – Compliant Position Control of Fingers.
- Acquisition – Compliant Force Control of Fingers.
- Manipulation – Combined Position and Force Control of Fingers.

We reduce the combined dynamic description of the bodies in contact to the linear form in this work. This is done so that we may use well developed tools in linear systems theory and analyse the system, as well as design appropriate controllers for successful completion of the task. The task space represents the interaction states that we want to control. Now, control is possible *iff* the task states are controllable. By this we mean that when we develop the *Grammian* for the linearised combined dynamic description of the bodies in contact, the task states *must* lie in the non-singular subspace of the grammian. If so, then the task can be successfully accomplished by the robot in question. We may then use standard *Multivariable Feedback Control* techniques, design compensators and guarantee local stability for the task at the interaction port.

The control architecture that we propose for Task Dependent Hand Control ($TDHC$) on the Salisbury Hand. The i^{th} command input ($\mathcal{T}_{pi}$) into $TDHC$ is the corresponding task primitive. It is parametrized with *Identifiers* to classify the robot hand ($\mathcal{H}_i$), the acquired part, if any, ($\mathcal{O}_i$) and the environment ($\mathcal{E}_i$). From a database, for the scenario in question, the appropriate dynamic descriptions of the entities are acquired. The

task primitive also provides the task representation mapping function. The control architecture for *TDHC* presents a method to design controllers for the actuators that drive the hand, in a task dependent manner.

The control problem may be broken into three (sub)levels under the following assumption. If the Jacobians and other Kinematic transformations are full rank throughout the task, then we may assume the existence of *pseudo* control signals in the task space. We therefore, design the required controllers in the object interaction space first. We then transform the information to the points of contact between the fingers of the hand and the (manipulated object)environment. We denote this as *Transformation from the Task to Hand Interaction Space.* This forms the second (sub)level in the control architecture. Finally, once the controller designs are transformed to the hand interaction space, we may then easily transform them to the *Actuator Interaction Space.* We explain each of these stages in greater detail below.

The design of controllers in the object interaction space is done using design techniques from multivariable feedback control theory. The poles of the system can be appropriately placed in the task space, and this not only guarantees local stability in interactions between the robot system and environment, but also enables us to reasonably modify the interaction behaviour.

Transformation from object to hand interaction port is done under the assumptions that:

- Contact between hand and object is not lost; and
- Information about internal stiffness of the hand may be derived from the definition of the task.

The desired motions of the object held in the hand (if any) can be transformed from the object interaction space to the hand interaction space using

standard homogeneous kinematic transformations. The desired object to environment forces (if any) can be transformed using jacobians, and so can the object interaction controller. Resolution of redundancy between object and finger wrenches for transformation of forces is done through internal forces and for transformation of the controller through *internal controllers.* Controllers at the hand interaction port have to not only satisfy the task requirements, but also have to guarantee desirable behaviour between the fingers and the object (such as maintaining contacts). This is done through the errors in the relative positions of the fingers. Thus the design of controllers at the hand interaction space *must* guarantee (i) execution of the task in the task space, while (ii) guaranteeing contacts between the fingers and the object.

Transformations between finger interaction ports and the actuator interaction ports is performed through the standard finger kinematic transformations and jacobians. The issue of non-uniqueness in kinematic transformation between the joint and the motors is resolved through the kinematic constraint equation, and the joints to actuators jacobians are made square by introducing a fourth equation that sets the appropriate tendon to its minimum value. Each finger interaction port controller transformes into the corresponding actuator controllers. The actuator controllers control both the motor positions and the tendon tensions.

We have implemented a simple position controller for joint control on the Salisbury Hand. This controller approximates the combined joint and motor dynamics under the assumption that motor dynamics are negligible. We have also performed simple grasping experiments (available on videotape upon request) using position control for the fingers to illustrate the hands robustness due the the stryctural compliance, and its ability to conform to a wide variety of objects.

5 Conclusions

Under the current NSF grant we have embarked on an integrated project to study the problem of task-level and robot-level control of complex robots. Even though, for concreteness, we have adopted the Salisbury hand as our test example, the three facets of our work, namely

- the development of a formal schema model of distributed computations,
- the development of scheduling algorithms and architectures appropriate for implementing such a model, and
- the development of control-theoretic solutions for task-oriented robot control

are applicable to any complex robot.

6 Publications

- Lyons, Damian, A Simple Set of Grasps for a Dextrous Hand, *1985 IEEE International Conference on Robotics and Automation*, March 25-28, 1985, St. Louis, Missouri, pgs 588-593.
- Lyons, Damian and Arbib, Michael A Task-Level Model of Distributed Computation for Sensory-Based Control of Complex Robot Systems, *Symposium on Robot Control '85* November 6-8, 1985, Barcelona, Spain, pgs 407-412.
- Lyons, Damian Tagged Potential Fields: An Approach to Specification of Complex Manipulator Configurations, *IEEE International Conference on Robotics and Automation*, April 7-10, 1986, San Francisco, California, pgs. 1749-1754.

- Lyons, Damian, A Formal Model of Distributed Computation for Sensory-Based Robot Control, *Ph.D Thesis*, COINS, UMASS, Amherst, 1986.

- Lyons, Damian and Arbib, M. A. A Model of Computation for Sensory-Based Robot Control, (to appear in), *IEEE Journal of Robotics and Automation*, 1989.

- Pocock, G., The Design and Analysis of a Computer System for Sensory-based Robots, *Ph.D. Thesis*, (forthcoming) COINS, UMASS, Amherst, 1988.

- Pocock, G. and Lyons, D., Robot Schemas: A Language for Controlling Sensory-based Robots, (submitted to) *IEEE Conference on Robotics and Automation*, Oct 1988.

- Pocock, G. and Ramamritham, K., System Support for Controlling Robots in Dynamic Environments, Technical Report, University of Massachusetts, Nov 1988.

- Pocock, G., Computational Characteristics of Processing Involved in Controlling Sensory-based Robots, (in preparation) Nov 1988.

- Ramamritham, K., Pocock, G., Lyons, D., and Arbib, M., Towards Distributed Robot Control Systems *Symposium on Robot Control '85* November 6-8, 1985, Barcelona, Spain, pgs 107-113.

- Ramamritham, K., "Real-Time System Support for Robotics", 1988 IEEE Workshop on Special Computer Architectures for Robotics and Automation", April, 1988.

- Venkataraman, S.T., and Djaferis, T.E., Multivariable Feedback Control of JPL/STANFORD Hand, *Proceedings, I.E.E.E Conference on Robotics and Automation*, Raleigh, North Carolina, April 1987.

- Venkataraman, S.T., Dexterous Hand Control through Impedance, *Ph.D Thesis Dissertation Proposal*, Department of Electrical and Computer Engineering, UMASS, Amherst, Massachusetts (February, 1987).

- Venkataraman, S.T., Task Representation in Robot Control: Part I, *Proceedings Conference Systems, Man & Cybernetics* Alexandria, VA, Oct 1987.

- Venkataraman, S.T. and Lyons, D.L., A Task Oriented Architecture for Dexterous Manipulation, *IEEE Workshop on Dexterous Robot Hands*, Philadelphia, PA, 1988.

- Venkataraman, S.T., Task Dependent Dexterous Hand Control, *Ph.D Thesis Dissertation*, Department of Electrical and Computer Engineering, UMASS, Amherst, Massachusetts, May 1988.

- Venkataraman, S.T. and Iberall, T., *Dexterous Robot Hands*, (to appear) Springer Verlag, 1989.

- Vijaykumar, R., Venkataraman, S.T., Dakin, G., Lyons, D.L., A Task Grammar Approach to the Analysis of Robot Programs, *IEEE Workshop on Languages for Automation*, Austria, 1987.