A Task-Oriented Architecture for Dextrous Manipulation [1]

S. T. Venkataraman[2]
Tele/Autonomous Systems Group
MS 198-330, Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

D.M. Lyons
Manufacturing Systems Research Dept.
Philips Labs
345 Scarborough Road
Briarcliff Manor, NY 10589

Abstract

Much of the previous work on dextrous manipulation has concerned itself primarily with hand construction, sensor development, servo-control or grasp planning. Most efforts towards the development of an overall manipulation system have either been restricted to a particular set of implementations, or far been too general to be practical. In this paper, we take the position that a dextrous hand can be studied most efficiently when embedded within the context of an *application domain*. The main focus of this work is to develop a controller architecture for a flexible assembly cell equipped with dextrous hands.

The central idea is to use constraints from the description of the task to be executed to select an appropriate abstract model and physical robot hand-arm system with which to execute the task plan. With this idea as target, we describe how allocation can be performed on the basis of a *task criterion* for a task plan. We also introduce a set of control models that can be allocated and interfaced to real hand-arm systems for the actual execution of task plans.

1 Introduction

In this paper, we develop a controller architecture for a flexible assembly cell equipped with dextrous hands. This controller accepts a high-level assembly operation as input and decomposes it into more primitive operations taking into account the robot resource capabilities and availabilities in the cell and executes these operations. A unique aspect of this paper is that it describes a first step at integrating dextrous hands into industrial assembly. The main thrust of this paper is to bring out the constraints that result during resource allocation. The applicability of this approach is demonstrated through the integration of control models for the execution of task plans within the architecture.

A flexible assembly cell consists of a number of robot systems, along with the tools used by these systems during assembly. Each system is composed of an arm, some number of end-effectors, and associated sensory and actuation equipment. In particular, some of the end-effectors may be *dexterous hands*, commonly defined as devices that are capable of imparting arbitrary small motions and forces upon the object that they hold. These assembly cells are typically required to handle small and mixed batch assembly. To make this technology viable, and make 'one of a kind'

[1] *This paper was presented at the IEEE Workshop on Dextrous Hands, Philadelphia, PA, April 25th, 1988.*

[2] *Some of this work was done while the author was at University of Massachusetts under NSF Grant No. DMC – 8511959 and CER Grant No. DCR – 8500332.*

assembly, [3], a reality, it must function with little or no setup/change-over time. It must also be robust. Note that a set of robot systems, each of which may be computationally or structurally different, can be used to handle unexpected events or error conditions that might cause problems for a single robot or piece of fixed automation.

Dextrous hand research, thus far, has concerned itself primarily with hand construction and servo-control (a survey of related articles may be found in [19]), sensor development (an overview of related articles may be found in [?]), grasp planning[2, 9, 17], or with manipulation[22]. We believe that dextrous hand research is at a point where giant strides can be made only after the problems associated with its effective use have been resolved. This paper attempts to integrate dextrous hand use into an automated assembly system, and describes in detail *how* a hand may be effectively used.

The design and construction of a controller for a flexible assembly cell is a complex problem involving issues in task decomposition, resource allocation, control, and sensing. Our work in this paper focuses on modelling and using dextrous hands, and on the development of a framework for allocating and utilizing robot resources (such as dextrous hands) in assembly.

The input to the controller is an object-level assembly operation. This input could come from a human programmer or from an automatic task planning program[20]. Our view of an assembly operation is based on our previous work in [21, 13]; the main extension to that view in this work will be the concept of a *task criterion* associated with a task plan (i.e., assembly operation). Each task criterion specifies demands made on robot resources by the task plan to enable successful task execution.

The controller decomposes a high-level assembly operation into more primitive actions which can be carried out on appropriate robot arms and end-effectors. The user need have no knowledge of the arms and tools available to the cell; this mapping is dynamically evaluated by the controller to suit the task and the state of the workcell. This is what we mean when we call our architecture *task-oriented*: The choice of a physical manipulator as well as the dynamic and control models of that manipulator are determined by the requirements of the task to be executed. The controller also deals with the issue of reallocating resources when the current allocation is no longer appropriate (e.g., if an error occurs). We find that some problems, such as regrasping an object, can be naturally expressed by the concept of *reallocating resources* for a task plan.

Once a robot resource has been allocated, an interface procedure is called up to translate commands coming from the task plan into a form on which the robot can act. We will describe in detail dynamic and control models for using a dextrous hand in the workcell. A key step in this will be the introduction of a vocabulary of *task primitives* to program *actions* performed by each finger, and constructing hand task primitives, using finger task primitives. Note that a wide selection of grasps, such as those in [9], can be built using these hand primitives. Note also that manipulation tasks involving a dexterous hand (small manipulations), or involving multiple arms (gross manipulations) can be programmed using these primitives.

The remainder of the paper is laid out as follows: In section 2 we describe the design process for the architecture. Section 3 introduces an assembly operation example that reappears throughout the more detailled sections of the paper. The architecture is described in detail in section 4. In section 5 we discuss a useful format for the task criterion, and describe the mechanism of robot resource allocation. In section 6, we describe a set of appropriate control models for arms and dextrous hands and discuss the mechanism for interfacing to robot resources. We conclude in section 7 by outlining the current state of implementation of this system.

2 Design

In this section, we motivate the design of the assembly cell architecture. The central design tenet is to base the architecture on the structure of assembly operations.

Assembly Operation Structure. An *assembly operation* may be considered to consist of four phases:

1. *Reaching* for the component.
2. *Acquiring* the component.
3. *Transporting* the component to the vicinity of the appropriate sub-assembly, and finally,
4. *Parts-mating* the component with the sub-assembly.

Although distinct, these phases are strongly linked. For example, a component should be grasped to enable the subsequent parts-mating operation operation [14, 2, 9] and to facilitate finding a path to the vicinity of the parts-mating operation [7]. The transportation phase must bring the component into a suitable position from which to start the parts-mating operation. However, different strategies for the parts-mating operation may generate different sets of feasible grasps and transportation paths. Thus, although the phases of an assembly operation are distinct, they can not be decoupled.

Robot Information. Two types of information about the robots in the workcell will be made use of information about their capabilities (*allocation* information), and information about how to send commands to them (*interface* information). An important extension is that in addition to allocation and interface information for physical arms and end-effectors, we also include this information for control models, such as a grasp configuration for a dextrous hand, or a compliant control model for an arm. We will assume that allocating a model entails allocating a physical robot on which to implement the model[3]. This combination of a model and a physical robot will be called a *virtual robot*; the reason why this concept is necessary and what form it should have is discussed in detail in section 5.

Candidate Architectures. We build our architecture based upon the structure of assembly operations. By this we mean that if a number of robot systems are available to the controller, then,

[3]We assume this for technical convenience: Allocating the model and then choosing a robot to match the needs of that model could be phrased within framework we present – it's a case of 'more of the same.'

broadly, the choice of an arm is made for the reaching/transportation phase, while choice of an end-effector is made for the acquisition/manipulation phase. In figure 1, the three major candidate architectures are presented. In each case, the shaded boxes represent allocation functions (programs to allocate a task plan to an appropriate robot system based on given task criteria), and the non-shaded boxes, the interface function (programs to send commands to the allocated robot). In option A, allocation and interfacing of the end-effector and arm are performed *together*; in B, the arm allocation precedes the end-effector's; in C, the end-effector allocation precedes the arm's.

Evaluation of Candidate Architectures. To evaluate these candidates, we need to introduce some terminology. Let A be the set of all available arms in the cell, and H the set of all end-effectors in the cell. The library functions $LA(a), a \in A$ and $LH(h), h \in H$ describe the characteristics of the arms and hands in the cell. The functions $IA(a), a \in A$ and $IH(h), h \in H$ describe the way in which robots in the cell can be interfaced. (The construction and use of LA, LH and IA, IH will be discussed in detail in sections 5 and 6 respectively).

We will assume that we can construct $H_a \subset H, a \in A$ the subset of end-effectors on, or available to, arm a. For example, for an arm with a gripper exchange mechanism, H_a would list all the end-effectors that could be used on the arm. For an arm with a dextrous hand, H_a will include all the *different grasps* that can be done with the hand (since we have included allocation and interface information for control models as well as for physical manipulators). Similarly, $A_h \subset A, h \in H$ describes what arms can be used with any one end-effector. When A (or H) is subscripted with a set, we mean the union of all A_h (H_a respectively) for h in that set.

Let χ, the *task criterion* for a task plan, be the statement of what robot resources are necessary for the successful completion of that task plan. (We will expand χ and describe the information contained within it presently.) χ will be used in conjunction with the library functions LA and LH to allocate the execution of task plans to available robot systems. We will assume that we can break χ down into χ_a and χ_h the criteria for selection of arm and hand respectively.

We evaluate these candidate architectures based on how well they model the structure of the assembly operation, how well they facilitate allocation, interfacing and reallocating, and how well they suit the assembly cell scenario we postulate.

1. **Assembly operation structure:** In scheme B, the reach phase is scheduled first. In this, the scheme reflects the relationship between allocation and the assembly operation structure: Unless the object can be acquired, the assembly operation cannot even be started. Scheme C, on the other hand, places allocation for the manipulation phase at highest priority, while A does not represent the above relationships at all.

2. **Allocation, interfacing and reallocation:** The following table outlines allocation and interfacing for the three candidates:

Figure 1: The Three Candidate Hierarchical Architectures

	Allocation	Interface
(A)	ARM: scan all $a \in A$ for χ_a. HAND: scan all $h \in H$ for χ_h.	Arm and hand commands must be split by task plan.
(B)	ARM: scan all $a \in A$ for χ_a and produce $\hat{A}$ and $H_{\hat{A}}$. HAND: scan all h in $H_{\hat{A}}$ for χ_h.	ARM: Filter out gross motions. HAND: Manipulative motions only.
(C)	HAND: scan all $h \in H$ for χ_h and produce $\hat{H}$ and $A_{\hat{H}}$. ARM: scan all a in $A_{\hat{H}}$ for χ_a.	HAND: Filter out manipulative motions. ARM: Gross motions only.

Scheme A is the most computationally complex scheme. The other two schemes differ only in the order in which they allocate resources types. Both of these schemes facilitate reallocation: the allocation scheme is hierarchical, and thus lower-level choices can be replaced without affecting the higher levels. Computationally, the complexity will depend on the types of arms and hands in the workcell: if there are many complex arms, the set $H_{\hat{A}}$ will probably be large and thus the complexity in scheme B will approach that of scheme A. In similar fashion, if there are many complex hands, then $A_{\hat{H}}$ will be large and the complexity of scheme C will approach that of scheme A.

3. **Assembly cell scenario:** In reallocation, B allows choice of hand to be switched while using the same arm (e.g. a new grasp chosen, or a replacement end-effector), while C allows choice of arm to be changed while maintaining the same end-effector (a difficult feat to implement). Both schemes allow reallocation of arm and hand (forcing either a hand-off situation, or temporary placement of the object). Our workcell description emphasizes dextrous hands: In general, we expect the hands in the workcell to be more complex than the arms. Thus, in our assembly cell scenario, B is the most useful alternative.

Our criteria support architecture B as the best choice for our workcell controller.

3 Running Example: Compliant Placement

We introduce an example assembly operation, compliant placement, which we will use to illustrate the more detailled parts of this paper. The actual choice of assembly operation is not that important, since the majority of our architectural components are similar no matter what operation is involved. Our example involves a place operation for which a high level assembly task planner (such as the one described in [20]) has prescribed a compliant control strategy. The example task may be formally defined as follows (see [21] for similar examples):

Figure 2: Compliant Place Strategy

- Goal, G_p: Object surface D against surface W.
- Strategy, S_p: Move coarsely into position over the destination surface W (Figure 2). Move towards W complying to torques on x and y (in object coordinates) until some threshold force f_z is reached.
- Invariant, N_p: There should be no intervening objects between D and W.

We will introduce more and more detail into this example as our exposition proceeds.

4 The Architecture

The Command Input and Task Decomposition. The input to the controller is an object-level assembly command, e.g. insert A into B, place C to the left of D, etc . The command must directly reference a task plan that carrys out the operation. The task-plan contains information about *goal states*, *sensory pre- & post-conditions*, *control strategies* and a *task invariant condition*, [21]. The hierarchical process of expanding a command into its components is called *task decomposition.* The details of task decomposition are not covered in this paper, for this and more complex planning techniques see [20, 13, 21].

The control strategy describes one *particular* way to achieve the goal. For example, for a peg-in-hole-insertion assembly, at the object-level, a control strategy would be to be to comply to reaction forces from the environment along all but the insertion axis. Note that a poor choice of control strategy may result in its failure to achieve its goal; in the compliant place example, if there are objects between the end-effector and the target surface, then the strategy will fail.

Figure 3: The Task-Oriented Architecture

Allocation. Allocation will be implemented hierarchically according to scheme B. At each level, a robot that exactly meets the task criterion is called the *ideal* robot. In general, such a robot will not physically exist, and a *real* robot plus an interface procedure will be used to implement the ideal robot. Note that decomposition of commands and allocation of resources can occur in parallel; in fact, to allow reallocation of resources during the execution of an assembly operation, they *must* occur in parallel.

The Hierarchical Levels. Figure 3 describes the three levels of the controller architecture. The top-most level (denoted *ideal task level*) is the third level in our architecture. The input command is an object-level assembly instruction. The control strategy, goal condition and the sensory conditions are *operated upon* to derive a set of *actuation signals* that will nominally cause task execution to terminate only when the object reaches its goal state. Output from this level (denoted motor output) represents commands to the ideal robot.

At level 2 (denoted *ideal arm level*), we begin the allocation process. In keeping with the philosophy of allocation scheme B, we allocate the reach phase first. If this cannot be scheduled, then the task cannot be executed. If all robot systems having the required functionalities are not available (scheduled on other tasks), then this task must wait in a queue until one becomes available. Since dexterous hands have been chosen for end-effectors, the likelihood of an end-effector on the chosen arm performing the acquisition phase is high and so, it may be more important to perform the allocation for transportation immediately after reach. If this allocation fails, hand-off will be necessary. If successful, then one or more arms become potential candidates for the task.

Allocation for the parts-mating phase follows at level level 1 (denoted the *ideal hand level*). After complete allocation, control strategies, goal conditions and sensory conditions are used to generate task plans to directly command the arm or hand. The interfaces between the boxes denoted *real arm* at level 2 and *real hand* at level 1 and the hardware, as well as the details of the control models that can be allocated along with physical manipulators, are discussed in detailed in the section 6. The following is a summary of what *control actions* imply at each level of the architecture:

- Ideal Task Level:

  The object-level assembly command is used to select a task-plan at this level. The task plan is decomposed into its components. The components, such as the control strategy, output commands to the next level. These commands assume that the next level can provide an ideal robot that can change the state of the object directly.

- Ideal Arm Level:

  At this level, the ideal task robot is decomposed into an ideal arm and an ideal hand, and the task is decomposed into a reach/transportation for the ideal arm and an acquisition/parts-mating for the hand. Upon successful allocation of the ideal arm tasks to a real arm and/or control model, an interface algorithm is chosen so that ideal arm commands can be transformed into real arm commands. This algorithm may need to add extra constraints to the incoming commands in order to make the translation possible. Ideal hand commands are passed to the next level. Global path planning information (e.g. from cameras or a built-in world model) is used to decompose and execute arm commands at this level.

- Ideal Hand Level:

  At this level, an real hand and/or control model (e.g. a grasp) is chosen to match the ideal hand. Upon successful allocation, an interface algorithm is chosen so that ideal hand commands can be transformed to real hand commands. A model of the grasped object (e.g., a geometric model which may be updated by sensory information) is used to decompose and execute hand commands at this level.

  Real hand commands describe changes in the states of the hand, and specify constraints induced by measures such as grasp stability or manipulability, [6] and [22], and the inequality constraints on contact forces. The actuation signals command the hand directly and have the ability to control constraint parameters.

Reallocation. Reallocation is performed subject to certain *consistency* constraints. For example, when hand-offs occur in scheme B, consistency demands either that the object be placed down for regrasping by the more appropriate arm-hand system, or the exchange happen through some dual-arm-hand coordination scheme. Allocation (and reallocation) and command decomposition occur in parallel in this architecture. The boxes labelled *global pathplanning information* at level 2 and *component models* at level 3 provide the necessary sensory information for command decomposition and execution (figure 3).

5 Task Criteria and Robot Allocation

We now explain the notion of task criteria and describe how it helps in allocating a robot system for the execution of task plans.

Task Space. We define *task space* $\mathcal{C}_T$ to be the space in which a task plan may be completely defined. Let $\mathcal{C}_G$, and $\mathcal{C}_S$ represent the spaces in which goal conditions, and sensing conditions and task invariants are defined. Let $\mathcal{C}_C$ be the space in which the control strategy and actuation signals are defined. Then, for a task plan representation as discussed in [21], we have:

$$\mathcal{C}_T = \mathcal{C}_G \cup \mathcal{C}_C \cup \mathcal{C}_S \tag{1}$$

Note that in contrast with [21], actuation signals are explicit in the task plan in this paper. We place no constraints on the structure of the task space, but for many assembly operations it will typically be changes in the kinematic and force states expressed in terms of object-centered coordinates.

For example, for the compliant placement operation, an element of the task space $(R \cup \#)^6$ is $(p_x, p_y, p_z, \theta_x, \theta_y, f_z, \tau_x, \tau_y)$ where the τs are torques in object coordinates (see Figure 2), f_z is the z component of force and the ps (θs) are position (orientation) components, again in object coordinates. The # element will denote we don't care what happens on that degree of freedom; allowing us to specify commands in a hybrid position force paradigm.

Task Criterion. A *task criterion* is a specification of what robot resources a task plan needs in order to execute successfully. Thus, the task criterion is strongly connected to the description of the task space. For example, Nakamura et al. [16] deal with the issue of simultaneously executing several prioritized task plans on a single redundant robot. Only the configuration subspace, R^{m_i} that needed to be controlled for some task plan i, was considered as a definition of the task criterion. Li and Sastry [14], on the other hand, describe a more complex task criterion for grasping. It consists of two (grasped) object-centered task spaces, one with *wrench* and one with *twist* coordinates. The task criterion in each space is an ellipsoid inequality called the *task ellipsoid*. The hand-object configuration which maximizes the task-ellipsoid is the most appropriate one with which to execute the task plan. This method extends the task criterion to include *ranges* on the axes. A limitation is that this method is strongly linked to the twist and wrench space descriptions, and so, generality (i.e., arbitrary nature) of task space is lost.

On approach to extend the power of the task criterion is to continue to 'tack' components on as necessary. For example, defining the task criterion as including a task ellipsoid constraint, a payload constraint, etc. However, the danger is that the task criterion would soon become unwieldy. Lyons [9] represents the task criterion for a dextrous hand using a *grasp-based feature space*. The feature space axes represent the amount of precision movement necessary and the amount of grasp security necessary to complete the task, and a set of coarse object properties. The space is segmented into three regions, each corresponding to a *grasp*, a stereotype hand-object configuration that includes an appropriate simplified hand model and object *acquisition* and *manipulation* information. The advantage of using a *set of models* of the robot (rather than a single 'hardware' model) is that it simplifies translations from a task criterion to a dextrous hand control model. Thus, the task criterion can remain simpler and more elegant. A disadvantage is that it forces task criteria to be expressed in terms of the stereotypical grasps (in the dextrous hand case) rather than for the task. This disadvantage can be overcome by choosing the set of models so that they are task-oriented; that is, that the models are chosen with some task domain in mind.

In summary, from this short survey of literature, it is clear that a task criterion should consist of at least a task space definition, but is made more versatile by considering aspects of the quality of control in the task space. The inherent complexity of the task criterion can be reduced by linking it with appropriate task-oriented models of the robot system. This is the reason that we extend the allocation and interface information in our architecture to include control models as well as physical manipulators.

5.1 Construction of Task Criterion

We now discuss the pieces of information that should be in the task criterion. Let the task space for task plan i be written as R^{m_i} and let its task criterion be χ_i. Let $R = A \cup H$ be the set of all available robot systems.

We postulate a function f_i which for any task i maps from the task space to a single 'common' space C. This function will be a necessary part of any task specification. We assume that a sufficiently complex C is chosen to support an interesting set of tasks[4]. For each robot $r \in R$, we postulate a mapping function f_r that maps from C to the configuration space of that robot.

[4] One example C might contain Cartesian positions, forces and torques.

Clearly, many f_r will not be surjective. The essential criterion for determining if a robot r supports a task i is to see if the range of f_i is a subset of the domain of f_r: $ran(f_i) \subset dom(f_r)$. Given this, the interface from task commands to robot commands can then be implemented by composing the functions, $f_r \circ f_i$. Not all robots will support all tasks. For example, if the task space contains a force axis, then $f_r \circ f_i$ is undefined for a robot whose dynamic behavior is described using a standard Lagrangian, [19]. Another example is when the task space may contain the three Cartesian position axes, $f_r \circ f_i$ is not defined on a 2-D robot. Clearly then, f_i should be part of χ_i, and f_r should be part of the library functions LH, LA that were discussed earlier.

Given that the task space can be mapped onto the configuration space of some robot, the next question is one of the necessary quality of control in the task space. We propose the following additional components for χ_i:

1. *Range.* The necessary controllable range on each axis of the task space is given by:

$$\rho : Z \longrightarrow R \times R \tag{2}$$

   where $\rho(n) = [p, m] \quad n \in Z$ where $Z = 1, ..., m_i$ and $p, m \in R$. In words, this means that the map ρ takes an integer for each task axis and gives the interval on that axis that must be controllable. This allows us to set unilateral conditions (e.g. for pushing an object rather than grasping it and moving it) through either $p = 0$ or $m = 0$.

2. *Accuracy.* The accuracy required on each task space axis is given by:

$$\alpha : Z \longrightarrow R \tag{3}$$

   A command of $r \in R^{m_i}$ to the robot, will result in the robot carrying out $r = (r_1 \pm \alpha(1), r_2 \pm \alpha(2), \ldots, r_m \pm \alpha_i(m))$.

It is interesting to note that these components can provide information to decide if it's necessary to grasp the object (for some unilateral ρ requests, acquisition may not be necessary). Note that the outputs of a grasp planner like the one described in [17] are usually desired wrenches and twists. In the task space, accuracy measures may be combined with the task ellipsoid concept through specification of error ellipsoids in the wrench and twist spaces rather than ellipsoids of the actual wrench and twist values.

The task criterion also needs to have information on what ranges and payloads will be necessary for the operation. The following three quantities capture this:

1. The position of the component to be acquired, specified as a homogeneous transformation matrix, T_c, with respect to some base coordinate frame B.

2. The location of the subassembly or parts-mating site, T_s, as a homogeneous transformation matrix relative to B.

3. The loads that the robot will experience while acquiring the part, W.

These pieces of information will be used to determine the minimum workspace envelope necessary for an arm to carry out the reach, acquisition, transportation and parts-mating phases. Note that allocation of a robot implies that information contained in the task criterion successfully matches the characteristics of the chosen robot. The library functions LA, LH discussed earlier should contain enough information to determine if a task criterion can be met. A useful solution is to have LA, LH store exactly the same types of information as χ but for a specific robot rather than for a task. So, for example, if χ_i is the task criterion for task i and $LA(r)$ is the library function for robot r, then

$$\begin{aligned} \chi_i &= (f_i, \rho_i, \alpha_i, T_{ci}, T_{si}, W_i) \\ LA(r) &= (f_r, \rho_r, \alpha_r, W_i) \end{aligned} \tag{4}$$

Based on the task description, we can outline the task criterion for the compliant place operation, χ_p, as follows:

- α_p: The strategy hinges on good force and torque control, but the position accuracy can be coarse.
- ρ_p: All our position ranges will be bidirectional, and of the order of the component size. The torque and force ranges should be small, and the z force range could be unidirectional.
- T_c, W and T_s are given by the particular situation.

5.2 Task Criteria in the Architecture

At the top level (ideal task), an input assembly command is expanded into its components (as described in section 3), one of which is a task criterion in the format discussed in the previous section.

$$\chi = (f, \rho, \alpha, T_c, T_s, W) \tag{5}$$

where, ρ specifies the range required in the task space from the ideal robot, α specifies the accuracy required and, T_c and T_s specify the transformation matrices for the object centered coordinate system and the sub-assembly centered coordinate system respectively. W describes the payload imposed by the task.

At the next level (ideal arm), this criterion is broken up as follows: The workspace envelope for an appropriate robot arm is constructed as a function of the range ($RANGE(\cdot,\cdot)$) from the current position of the robot to the component, and from there to the sub-assembly. The purpose of this level is to carry out the acquisition transportation phases of the assembly operation; to 'set the scene' for the parts-mating strategy.

$$\rho_a = Func(\ RANGE(T_r, T_c), RANGE(T_c, T_s)) \tag{6}$$

where T_r is the current position of the robot r. The first range is crucial; you could ignore the second if you're willing to choose a new arm once the component has been acquired. These concerns can be represented in an implementation of $Func$.

Figure 4: Allocation and Interfacing of Robot Model

At the next level (ideal hand), the choice of a particular hand model is subject to $\rho_h = \rho$ and $\alpha_h = \alpha$ being satisfied. Since we will consider a specific grasp on a dextrous hand as an example of a hand model, this allocation step *subsumes* grasp selection. Once a hand model has been chosen, commands issued to the ideal hand level are translated into the hand specific commands (Section 4) for a particular physical (possibly dextrous) hand; this mapping is $f_h \circ f$.

The basic building block in implementing this architecture is shown in Figure 4. At each level, the motor actions are issued to what might be called a *virtual robot* (by analogy with [1]); that is, a combination of a physical robot plus some interface and/or control model.

For the three levels of abstraction in our task-oriented architecture, we have:

Level 3: Ideal Task Level

- Allocation: $\chi = (\rho, \alpha, T_c, T_s, W)$
- Virtual Robot: Ideal Task Robot

Level 2: Ideal Arm Level

- Allocation: $\chi_a = (f_a, \rho_a, \alpha_a, T_{c_a}, T_{s_a}, W_a)$
- Interface: $f_r \circ f_a$ for some arm $r \in A$. (Reach-To, Transport-To in figure 3).
- Virtual Robot: Real Arm, Ideal Hand

Level 1: Ideal Hand Level

- Allocation: $\chi_h = (f_h, \rho_h, \alpha_h, T_{c_h}, T_{s_h}, W_h)$
- Interface: $f_r \circ f_h$ for some hand $r \in H$. (Grasping, Manipulation in figure 3).
- Virtual Robot: Real Hand

Allocation Constraints. We apply the following two constraints to the allocation process:

1. In the allocation of resources, if more than one robot meets the task requirements, then we should choose one that *minimally* meets the task requirements.
2. We should allow *dynamic reallocation* of resources to account for the possibility of the present allocation becoming *non-optimal*. Such re-allocations must be subject to *consistency* constraints of the form:
   - Ideal Robot: Object must be placed down in workspace of new robot, which must now meet the full χ specification again.
   - Ideal Hand: Object may need to be placed down, but only χ_h needs to be satisfied again.

Note that the interface and allocation modules (figure 4 are contained within the *Reach-to, Transport-to* box at level 2, and the *Grasping, Manipulation* box at level 1 (figure 3).

6 Task Execution

The sections that follow describe how commands generated by the box denoted *Real Hand* in figure 3 are executed. Similar arguments may be provided for the execution of commands generated in the box denoted *Real Arm* in figure 3. The commands generated in these boxes represents low-level atomic tasks. Execution occurs on the particular arm/hand hardware that the task has been allocated upon.

What do Low-Level Tasks Represent?

Low-level atomic tasks represent actions that we expect the arm (hand) to perform in the respective hardware's *task space*. For example, free motions of the endpoint of a non-redundant 6 *dof* arm would be represented by the location of a motion coordinate system at the endpoint of the arm, and its rigid body motions with respect to some other fixed coordinate system. For motions under contact, the task space may be located at the mechanical contact itself, with its coordinates representing forces and motions allowable by the contact. Henceforward, low-level atomic task units will be referred to as *task primitives*.

We will develop a vocabulary of task primitives in subsequent sections for dexterous hand usage. We will also perform the decomposition of task primitives input into the architecture in figure 5 into their respective actuator commands.

Subsequent sections are organized as follows: We begin with a description of the overall execution architecture in the next section, and develop a vocabulary of task primitives for the execution of ideal hand commands on an allocated dexterous hand and associated hardware. In the following section, we describe the process of task decomposition across levels of the hierarchy represented in figure 5. Thereafter, we provide with a mathematical representation for the task primitives, their implications on control.

The actual dynamic modelling and control methodologies for dexterous hand usage has been omitted from this work. Pertinent details are available in chapters 4, 5 and 6 of [19]. Hand control architectures, as well controller design methodologies are described in chapter 6 of [19]. Chapter 6 of [19] also analyzes task control of JPL/Stanford Hand, [15], using linearized hand dynamic models, approximated interaction force constraint models, and pole placement methods using multivariable feedback control techniques, [**?**].

6.1 Description of Execution Architecture

Figure 5 shows the task execution architecture for a dexterous hand. At level 2 in figure 3, the *reach-to* and *transport-to* phases are allocated on a robot arm. A corresponding set of *ideal hand commands* are generated at this level for the *acquisition* and *parts-mating* operations. Such ideal hand commands are considered input into the execution architecture shown in figure 5. We will refer to commands input at the ideal hand level as *object task primitives.* As shown in figure 5, object task primitives are used by *Grasp* or *Manipulation* planning algorithms to generate *Hand Task Primitives*, which are input into the level at which we consider a real hand to consist of a set of *ideal* fingers. Ideal fingers have also been referred to in literature as *virtual fingers*, [1]. I have used 'ideal robot' as the 'desired' robot description, and 'virtual robot' as the machine that implements this. The difference is subtle but important for selection of 'appropriate' virtual robots, something that thea and michael assume is 'built in' to a grasp description. At the next level, *finger task primitives* represent commands input into the fingers of the dexterous hand. The idea behind this scheme of considering the virtual fingers first, and then, the real fingers, is to subdivide the problem of allocating the grasping or manipulation task into two steps: (i) allocate the task onto a hand that consists of a minimum number of fingers of variable dimensions, and (ii) allocate the virtual fingered hand onto a real hand. Thereafter, each finger primitive, is decomposed into *Joint Task Primitives*, and further, into *Actuator Task Primitives.* At the actuator level in figure 5, actuator task primitives are used to generate the actuation signals that actually drive the hand's actuators.

Note that structure of the execution architecture bears a strong resemblance to the overall task architecture. Note also that the execution architecture for a single arm will accept commands at the same abstraction level as real finger task primitives and execute them.

<u>The Premise of Task Analysis</u>

Our task analysis is based on the assumption that a robot remains in contact with its *environment* at all times. Thus, for example, a robot arm moving in free space is in contact with an environment that exhibits *null* dynamic resistance, while, a robot arm whose endpoint has been

Figure 5: Architecture for Execution on a Dexterous Hand

firmly bolted down onto the floor is in contact with an environment that offers *infinite* dynamic resistance. Pushing a block on a friction table, moving in a viscous medium or against a spring, and working the piston of a pneumatic cylinder are all examples where the environment offers *finite* dynamic resistance. This perpetual mechanical coupling between a robot and its environment will be used as a fundamental invariant in our task analysis.

6.2 Task Primitive Vocabulary

In this section, we generate, from first principles, a command vocabulary of task primitives for each level in the architecture shown in figure 5.

<u>Ideal Hand Task Primitive Vocabulary</u>

In general, the actions performed by a component during the acquisition or parts mating phases can be classified into six modes depending on the component's interactions with *its* environment. These modes are free motion, guarded motion, fine motion, free force application, guarded force application and fine force application, [19]. The meanings of these action modes are elucidated below:

- Free Motion occurs when a body moves about in an unconstrained fashion and there is neither any obstacle in its path, nor surfaces against which it establishes contact. Robotic free motions will not pass through structural, kinematic or algorithmic songularities.
- Guarded Motion occurs when a body has to *guard* itself from establishing mechanical contact while moving about. During obstacle avoidance it guards itself from establishing contact with the obstacle. For singularity avoidance while using a robot, it treats singularity *manifolds* themselves as obstacles to avoid. I think you introduce this too blithely, have there been any papers in the literature about this, if not, then you need to introduce it better.
- Fine Motion occurs when a body has the ability to move about while remaining in stable contact with its environment. Sliding motion of an arm while tracing a friction surface is an example of fine motion, and grinding and milling operations in manufacturing are examples of fine motion.
- Free Force Application occurs when a body applies forces upon its environment in an unconstrained manner. A robot arm bolted to the ground is an example of this free force application. Note that most practical situations involving forces may never fall under the category of free force application, and its introduction at this level is for completeness of the task primitive vocabulary.
- Guarded Force Application occurs when a body has to *guard* itself from losing contact while applying forces upon its environment. A robot arm maintaining contact with a moving object is an example of guarded force application.

- Fine Force Application occurs when an arm applies forces upon the environment that it is in contact with. A robot arm pushing a block on a friction surface is an example of free force application.

At the real hand level in figure 5, the vocabulary of hand task primtives necessary for the execution of the six object task primitives can be summarized as given below:

<u>Real Hand Task Primitive Vocabulary</u>

Each hand task primitive describes the motions and/or forces imparted upon its immediate environment. During flexible assembly, this implies consideration of the interactions between the hand and a component that needs to be mated with a sub-assembly. We develop a vocabulary of task primitives using which interactions that occur between the hand and the component during acquisition and parts-mating phases may be described.

- Preshape: During the acquisition phase, the fingers of a hand establish contact with the component to be acquired, and stably overcome its gravitational effects. Prior to the establishment of mechanical contact, the hand must *preshape* to a configuration that will enable its fingers to come in contact with the component at prespecified points on the object. The importance of preshaping, as well as the existence of a strong relationship between preshape, and the geometric and physical characteristics of the component, have been discussed in literature ([**?**] presents a survey of work in this area). So, preshape is chosen as our first hand task primitive.

- Grasping and *Acquisition*: Once the hand is preshaped, its fingers are required to close in towards the object until stable mechanical contacts have been established with the object. We desire, in addition, that all fingers establish contact with the component at the same instant, since this results in atmost interaction stability . Thereafter, in a *coordinated* fashion, the fingers are required to overcome the gravitational effects of the component, so that the state of the component may be modified, as desired, thereafter. The aforementioned two atomic stages that complete the acquisition phase are sometimes coupled together and referred to as *grasping*, [22], [14], [9], while in [15] and [17], the actual imparting of forces upon the object, once mechanical contacts have been established, is referred to as grasping.

  In this paper, grasping implies closing the fingers in towards the component until mechanical contacts have been established, while, the term *acquisition* is used for actually imparting the finger contact forces required to overcome the component's gravitational effects. The main reason for this is the clear distinction between the control architectures required to implement the primitives, as we will show in subsequent sections.

- Manipulation: Once the component has been acquired, a dexterous hand is required effect changes in its state, subject to interactions between the component and *its* environment. Actions performed by the object may be described using free, guarded and fine motion task primitives, and, free, guarded and fine force application task primitives. We refer to this whole suite of actions performed by a hand as *manipulation*. More specifically, we denote actions of the hand that result in free, guarded, or fine motions of the component as *motion* *manipulation*, and those that result in free, guarded and fine force applications of the component as *force application manipulation*. Note that in both cases, the fingers have to perform in a *coordinated* fashion and satisfy object level performance requirements.

During the execution of the grasping, acquisition and manipulation task primitives, the actions performed by the fingers of a dexterous hand are not decoupled from each other. This was indicated in the discussion above with words such as *coordinated.* This aspect will be dealt with in greater detail in the following section on task decomposition.

Finger, Joint and Actuator Task Primitive Vocabulary

We do not discuss the actual decomposition between virtual and real fingers in this work. The task primitive vocabulary that we will develop here for real fingers is directly applicable to virtual fingers also.

Finger Task Primitives

Each real finger, when completely decoupled from the others, is a manipulator. In general, it is a serial kinematic chain, and when used for *precision grasps*, [9], has only one point of contact with the object. *Lateral* and *power* grasps may have upto one contact for each link on the finger. For *precision* dexterous hand usage, the interactions between each finger and its environment may be sufficiently described using the six task primitives. Note that for *lateral* and *power* applications, the same vocabulary can be used. However, the information contained in each of the task primitives will include inter-phalange constraints in addition to the description of the actual contacts. Therefore, the interactions between a single finger and *its* environment will be described using free, guarded, and fine finger motions, and, free, guarded, and fine finger force applications.

Joint and Actuator Task Primitives

It is the responsibility of every joint, and in turn, every actuator, to actually execute the finger task primitives. In general, a vocabulary of free, guarded and fine joint motions, and, free, guarded and fine joint force applications would provide a sufficiently rich vocabulary at the joint-level. Likewise, for the actuator, free, guarded and fine actuator motions, and, free, guarded and fine actuator force applications would do splendidly. But a key point is that as we decompose tasks to more primitive levels, they must become more specific to the hardware in question. We now illustrate this aspect with a few example hardware configurations.

Consider free motions of a robot finger that is a serial kinematic chain, having rotational joints, each with a single degree of freedom. Let, in addition, the joint be directly driven by an electric motor. For this configuration, joint and actuator levels in the architecture shown in 5 are almost identical. Further, only torque control is required at the joints (torque computed from one of the large spectrum of control algorithms), and only current control for the actuators. This implies that the joint and actuator task vocabulary simply consists of the free force application task primitive, and nothing else.

Now consider a variation of the illustration provided above. Let the joint be remotely driven using elastic tendons, [15], [**?**]. Since tendons can be used only to impart *pulling* effort, the minimum number of tendons required to drive an n jointed finger is $n+1$, [15]. This redundancy in the actuator to joint mapping is regularized with one or more constraint equations that impose the condition that all tendon tensions must be non-negative at all times. For such a scheme, at the actuation-level, a fine actuator motion task primitive will be necessary, although at the joint-level, free joint force application task primitive would suffice. Note that at the finger-level, free force application task primitive is almost unusable under most practical situations.

The lesson to be learnt is that the six task primitive vocabulary may be sufficient for a large spectrum of tasks, but not necessary. A direct implication of this is that when the architecture is actually implemented, a *pruning* stage will precede the actual development stage.

6.3 Task Decomposition

We now describe the decomposition of ideal hand task primitives that are input into the architecture in figure 5 into actuator commands. In [21], we described this process as a production rule, and concentrated on the language aspects. We also presented the notion of a *task invarient* that would turn true when sensory conditions indicates an error in task execution. Here, we describe the actual decomposition process.

Reach Phase

The reach phase of assembly is executed in two parts. During the first, that we denote *gross reaching* here, the robot arm is commanded to execute a free motion (or guarded motion if the assembly environment is ridden with obstacles), and the dexterous hand mounted on it is commanded to execute a preshape task primitive *concurrently*. Upon completion, the hand is preshaped for acquiring the component and the arm is in its proximity. At this point, local sensing systems (proximity sensors and local cameras) may be used to gather more accurate information about the component. The second part of the reach phase, that we denote *fine reaching*, involves *concurrently* moving the arm guardedly to bring the component within the workspace of the hand, and fingers of the hand to correct, if necessary, the preshape posture.

Acquisition Phase

The acquisition phase involves active usage of only the dexterous hand. The robot arm acts as a compliant *seat* on which the dexterous hand is placed. Acquisition phase involves the execution of a grasping task primitive, followed by an acquisition with the dexterous hand.

Transportation Phase

During the transportation phase, just as in the reaching phase, the arm first transports the component to the proximity of the parts-mating site, during *gross transportation*, and then guardedly aligns the component for the parts-mating operation, during *fine transportation*. This involves a free arm motion task primitive, followed by a guarded arm motion task primitive. During this the hand continually executes an acquisition task primitive to retain the component stably within itself.

Parts-Mating Phase

Part-mating involves, first, a guarded move by the arm to bring the component in contact with the sub-assembly. We denote this with the term *pre-mating*. Several sequences, each involving the concurrent execution of a manipulation task primitive by the hand, and a guarded force application by the arm follow. The actual number of such sequences is derived from the plan that describes the parts-mating operation. We refer to these as the *mating sequences*.

The above decomposition describes the nominal feedforward decomposition. At any point during execution, if sensory information flags a task execution error, replanning may require the execution of an alternate task primitive. For example, during the first part of the parts-mating phase, if the component makes unexpected contact with its environment due to geometric modeling errors, the acquisition hand task primitive would be modified to fine force application during replanning. We are presently addressing this issue in a formal manner, [**?**].

Concurrency Constraints during Decomposition

In the previous sections where we developed task primitive vocabularies, we employed terminologies such as *coordinated* finger actions to describe concurrency constraints that must be satisfied during task primitive execution. In addition, we also imposed similar constraints during the decomposition of arm and hand tasks for the four phases of assembly. We now describe two types of concurrency constraints that will be accomodated in the task architecture, and explain how they may be realized during task execution.

Concurrency constraints that arise during assembly can be categorized into two types. The first involved state relationships, and the second, time relationships. We denote the former using the terminology *Coordination*, and the latter, *Synchronocity*, and define the two as follows.

- Coordination: Actions performed by two bodies are coordinated when a prescribed relationship in the states of the two bodies is maintained throughout. Consider for example, an acquisition hand task primitive. Fingers of the dexterous hand are required to apply forces upon the component (being acquired) in a coordinated fashion to overcome object level gravitational forces.
- Synchronocity: Actions performed by two bodies are synchronous if they begin and terminate executing the action at the same instant. For example, during grasping, when fingers establish contact with the component, maximum stability results only if all fingers establish contact at the same instant. This can be achieved by considering grasping to be made of synchronous finger guarded motions.

Task Decomposition

We will now describe the decomposition of ideal hand commands into the respective actuator commands. Similarities may be drawn for the decomposition of arm commands. specifically, we will use a hand similar in kinematic structure to the JPL/Stanford hand, [15] as a testbed to illustrate our ideas. This implies that virtual and real finger levels of the architecture in figure 5 merge into one. The difference is that we will consider the joints to be directly driven by DC servo motors. This will merge the joint and actuator levels in figure 5, in addition.

- Preshape consists of free finger motions, which in turn may be executed using free joint force application task primitive.
- Grasping consists of synchronous guarded finger motions, which may be executed using free joint force applications task primitive.
- Acquisition consists of coordinated guarded finger force applications, which may be executed using free joint force applications task primitive.
- Motion manipulation implies that free, guarded or fine object motions are required. This is realized using fine finger force applications, which in turn implies that free joint force applications are required.
- Finally, force application manipulation implies that free, guarded or fine object force applications are required. This may be realized using fine finger force applications, which in turn implies that free joint force applications are required.

6.4 Task Primitive Representation

A task primitive must capture the essence of the interactions between a robot and its environment, and provide information pertinent to the control of these interactions. In addition, it must allow the specification of states that describe the assembly in question throughout the period during which the task is executed. Consider, for example, free motions of a robot arm. Due to the *null* resistance offered by the environment, only robot motions are necessary to completely describe interactions that occur in-between. In addition, a specification of the desired manipulator trajectories during task execution completes the description of the free motion task primitives. When mechanical contacts are present, the *frame* in which task information is provided usually coincides with the location of contact itself. The choice of coordinates (position *vs* force along a direction, etc.) reflect the interactions between robot and its environment, and the desired motion and force trajectories (or constraints) specify state changes (or constraints to be satisfied) during task executions.

Every task primitive command is, therefore, parametrized with desired trajectories and/or constraint equations necessary to completely describe task execution. The *Representation* of a task primitive translates this into a language more apt for the purposes of task control. In the succeeding sections, we describe the information with which each task primitive is parametrized, and develop appropriate task primitive representations.

Task Variable

Task information pertinent to control is stored in a variable called the *task variable*. A task variable represents that quantity that needs to be controlled for successful execution of the task primitive. For example, for free motions, motion trajectory following is required. Many approaches have been taken to deal with guarded motion. The approach that lends itself most naturally to integration into control is the use of generalized *compliance* (related to the work by Mason in [15]), or generalized *admittance* [5]. On the other hand for guarded force application, the use of generalized *stiffness* (related to the work by Salisbury in [15]) or generalized *impedance* [5] is most natural.

Most of the work in fine motion and fine force application has pertained itself to planning. Very little emphasis has been placed thus far on the execution of these task primitives. Since in general fine motion control will require that the actuation signals be generated after monitoring both motions and forces. For motions, motions errors will be operated upon by a controller function. For forces, the actual forces will be used to investigate the satisfaction of constraints and no contribution will be provided from the force part of control. On the other hand, if constraint equations are not satisfied, from a predetermined set of control stratgies, one will be chosen depending on the violation. Depending upon the control strategy, force related actuation signals will be generated, added to the motion related actuation signals and used to drive the system. Although the actual details are beyond the scope of discussion at this point, it is clear that the task variable should contain both motion trajectories as well as the constraint equations of actual forces. Using similar arguments, it may be said that the task variable for fine force application should contain force trajectories, and motion inequality equations.

Hand task primitive representations will simply be a collection of those of individual fingers. The reason for this is our decision to implement synchronocity or coordination between the fingers through control.

Ideal Hand Task Representation

Let the task variable at the ideal hand level be denoted by Y. The task representations are as given below: Explain M, M_t, F and P in the paragraph underneat these equations. In the case of P its okay to say you will explain it in the next paragraph, but *say* that, don't just ignore it.

$$
\begin{array}{lcll}
Y & = & [M]^T & \text{for Free Motion} \\
Y & = & [M \, + \, C \, (F_t, \, F)] & \text{for Guarded Motion} \\
Y & = & [M, \, \eta \, (F, \, P)] & \text{for Fine Motion} \\
Y & = & [F] & \text{for Free Force Application} \\
Y & = & [K \, (M_t, \, M) \, + \, F] & \text{for Guarded Force Application} \\
Y & = & [\eta \, (M, \, P) \, , \, F] & \text{for Fine Force Application}
\end{array}
$$

where, $C\,(.)$ and $K\,(.)$ are the compliance and stiffness operators that operate upon the actual and threshold force and motion values respectively. $\eta\,(.)$ is the funtion that describes the corresponding inequality constraint. Note that in [15], the compliance and stiffness operators are linear and operate upon the difference between threshold and actual values. We use this model in this work. Therefore,

$$
\begin{array}{lcll}
Y & = & [M \, + \, C \, \Delta F] & \text{for Guarded Motion} \\
Y & = & [K \, \Delta M \, + \, F] & \text{for Guarded Force Application}
\end{array}
$$

where, ΔM and δF are $(M_t - M)$ and $(F_t - F)$ respectively.

The function $\eta\,(.)$ is used to evaluate if inequality constraints are satisfied, and output a signal the *extent* to which there is an error in satisfying the constraint. Note that in this case, the function itself (instead of a trajectory) is the planner specification. P in the parameter list of η describes a list of constants that may be used in the function. η is an important *mapping function*. In a completely automated system, the planner must generate these functions automatically. The functions must provide enough information to design controllers with, but at the same time have a sufficiently general, yet formal structure to enable execution monitoring by the planner. This is one of the points in the hierarchical structure where integration of planning and control needs to be carefully addressed.

Real Hand Task Primitive Representation

Finger task primitives will be identical to the ones described above for the task primitives at the ideal hand level. For dexterous hand usage, let the motions, forces, and the task variable of the i^{th} finger be represented by M_i, F_i and Y_i respectively. With this:

$$
\begin{aligned}
Y_i &= [M_i]^T && \text{for Free Finger Motion} \\
Y_i &= [M_i + C_i\ (F_{t_i},\ F_i)] && \text{for Guarded Finger Motion} \\
Y_i &= [M_i,\ \eta_i\,(F_i)] && \text{for Fine Finger Motion} \\
Y_i &= [F_i] && \text{for Free Finger Force Application} \\
Y_i &= [K_i\ (M_{t_i},\ M_i)\ ,\ F_i] && \text{for Guarded Finger Force Application} \\
Y_i &= [\eta_i\,(M_i)\ ,\ F_i] && \text{for Fine Finger Force Application}
\end{aligned}
$$

Accumulating over all the fingers, we have:

$$
\begin{aligned}
Y &= [M]^T && \text{for Preshape} \\
Y &= [M + C\ (F_t,\ F)] && \text{for Grasp} \\
Y &= [K\ (M_t,\ M)] && \text{for Acquisition} \\
Y &= [\eta\,(M)\ ,\ F] && \text{for Manipulation}
\end{aligned}
$$

where,

$$
\begin{aligned}
Y &= [Y_1,\ Y_2,\ .....,\ Y_i,\ .....,\ Y_{n-1},\ Y_n]^T \\
M &= [M_1,\ M_2,\ .....,\ M_i,\ .....,\ M_{n-1},\ M_n]^T \\
F &= [F_1,\ F_2,\ .....,\ F_i,\ .....,\ F_{n-1},\ F_n]^T \\
\eta &= [\eta_1,\ \eta_2,\ .....,\ \eta_i,\ .....,\ \eta_{n-1},\ \eta_n]^T
\end{aligned}
$$

$$C = \begin{bmatrix} C_1 & & & & & & & \\ & C_2 & & & & & & \\ & & \cdot & & & & & \\ & & & \cdot & & & & \\ & & & & C_i & & & \\ & & & & & \cdot & & \\ & & & & & & \cdot & \\ & & & & & & C_{n-1} & \\ & & & & & & & C_n \end{bmatrix}$$

$$K = \begin{bmatrix} K_1 & & & & & & & \\ & K_2 & & & & & & \\ & & \cdot & & & & & \\ & & & \cdot & & & & \\ & & & & K_i & & & \\ & & & & & \cdot & & \\ & & & & & & \cdot & \\ & & & & & & K_{n-1} & \\ & & & & & & & K_n \end{bmatrix}$$

Note that we have not assumed at this point that the off-diagonal terms of the C or K matrices are zero. In actual implementation, these off-diagonal reflect inter-finger dependencies, [19].

7 Conclusion

In this paper, we have developed a controller architecture for a flexible assembly cell equipped with dextrous manipulation capabilities. The central idea was to use constraints from the description of the task to be executed to select an appropriate abstract model and physical robot hand-arm system with which to execute the task plan. We this idea as target, we described how allocation can be done on the basis of a task criterion for a task plan, and we introduced a set of control models for an arm and dextrous hand that can be allocation and interfaced to real hand-arm systems.

Implementation has been proceeding in both of the main components of this paper: the execution of task plans with task criterion, and the construction of useful hand models containing task-specific robot and environmental dynamics. The task plan representation work has been carried out using the $\mathcal{RS}$ model [10] as both a formal tool [11] and a multiprocessor implementation language [12]. The computational complexities of the architecture we have described are very high, especially the issue of remapping robot models. $\mathcal{RS}$ allows us to structure our plan representation as a network of concurrent cooperating computing agents (small grain processes). An $\mathcal{RS}$ formulation of this architecture is described in [13].

We have chosen the JPL/Stanford hand [15] as representative of dextrous hands. All implementations done so far are strongly bottom-up. Simplified preshape primitives have already been executed. The simplification is that fingers were commanded at the joint level, and inter-finger motion dependencies were neglected. The main reason for these simplifications was the scarcity of computational resource. So far, all implementations have been performed on a single MicroVax CPU executing control programs written in C, and using the VAXELN real-time environment. In addition, simple grasps have also been executed without fingertip force feedback to demonstrate the structural compliance of the hand. Grasp stability was experimentally demonstrated through robustness to force perturbations on the object. In terms of analysis, we have applied the theoretical framework developed in this paper as well as chapters 4 and 5 of [19] to the JPL/Stanford hand turning a bolt. Chapter 6 of [19] gives a detailed discussion of this, as well as all the implementations described above.

We see this paper as a natural extension of our work on formal ways to represent the notion of *task context* (the use of information related to a task at hand). We argue that our view of some parts of the task planning problem as an allocation/interfacing of set of task-oriented models is a novel and useful way to use task information in dealing with complex robots, and dextrous hands in particular. We see this work as a natural extension of our work on formal ways to represent the notion of *task context* (the use of information related to a task at hand). We argue that our view of the task planning problem as an allocation/interfacing of set of task-oriented models is a novel and useful way to use task information in dealing with complex robots, and dextrous hands in particular. TV, I thought you said there were some point you wanted to discuss in here.

References

[1] Arbib, M.A., Iberall, A., and Lyons, D., "Coordinated Control Programs for Movements of the Hand," *Exp. Brain Res. Suppl.*, 10, 1985, pp.111-129.

[2] Ahmad, S., Feddema, J.T., "Static Grip Selection for Robot-Based Automated Assembly Systems" *Journal of Robotics Systems* 4 (6), 1987, pp.687–717.

[3] Bourne, D., "An Overview of the Intelligent Machining Workstation," to appear in *Proceedings NASA Conference on Space Telerobotics* Jan. 31 – Feb. 2, 1989, Pasadena Center, Pasadena, CA 91101.

[4] Djaferis, T. E., Franklin, J. A., Murah, B., *Development of Multivariable Feedback Methods for Surface Tracing*, Technical Report No. CCS-100-86, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

[5] Hogan, N., "Impedance Control: An Approach to Manipulation: Part I - Theory, Part II - Implementation, Part III - Applications", *Journal of Dynamic Systems, Measurement and Control*, 1985, pp. 1-24.

[6] Li, Z., Sastry, S., "Dexterous Robot Hands: Several Important Issues," *Proceedings Workshop on Dexterous Robot Hands at the 1988 IEEE International Conference of Robotics and Automation,* April 24, Franklin Plaza, Philadelphia, PA, pp 68 – 108.

[7] Lozano-Perez, T., and Brooks, R., "An Approach to Automatic Robot Programming," *AI Memo 842*, MIT, Cambridge, MA, Apr., 1985.

[8] Khatib, O., "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation", *IEEE Journal of Robotics and Automation*, vol RA-3, no. 1, pp.43 – 53, 1987.

[9] Lyons, D.M., "A Simple Set of Grasps for a Dextrous Hand," *Proceedings of the 1985 International Conference on Robotics and Automation*, St. Louis, MO, Mar. 25–28, 1985, pp.588–593. Also, "A Generalization of: A Simple Set of Grasps for A Dextrous Hand" *COINS Tech. Rep. 085-37* University of Massachusetts at Amherst, 1986.

[10] Lyons, D.M., and Arbib, M.A., "A Formal Model of Computation for Sensory-Based Robotics" *IEEE Trans. on Robotics & Automation*, Vol. 5, No.3, June 1989.

[11] Lyons, D.M., "$\mathcal{RS}$: A Formal Model of Distributed Computation for Sensory-Based Robot Control" *Ph.D. Dissertation* and *COINS Technical Report* # 86-43, University of Massachusetts at Amherst, Amherst, MA 01003, 1986.

[12] Lyons, D.M. "Implementing a Distributed Programming Environment for Task-Oriented Robot Control" *Philips Technical Note* TN-87-054, April 1987.

[13] Lyons, D.M., "On-line allocation of robot resources to task plans" *1988 SPIE Symposium on Advances in Intelligent Robotics Systems* 6-11 Nov., Cambridge, MA, 1988.

[14] Li, Z., Hsu, P., Sastry, S., "On Grasping and Dynamic Coordination of MultiFingered Robot Hands" ERL Memo UCB/ERL M87/63, College of Engineering, University of California, Berkeley, CA 94720, September, 1987.

[15] Mason, M.T., Salisbury, J.K., *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, 1985.

[16] Nakamura, Y., Hanafusa, H., Yoshikawa, T., "Task-Priority Based Redundancy Control of Robot Manipulators" *IJRR 6,2* Summer 1987, pp.3–15

[17] Nguyen, V., 'Constructing Force Closure Grasps', *Proceedings, IEEE Conference on Robotics and Automation*, San Francisco, CA, 1986, pp. 1368-1373.

[18] Paul. R. P., *Robot Manipulators – Mathematics, Programming and Control*, MIT Press, Cambridge, MA, 1981.

[19] Venkataraman, S.T., "Task Dependent Dextrous Hand Control" *Ph.D. Dissertation*, Department of Electrical & Computer Engineering, University of Massachusetts, Amherst, MA 010033, May 1988.

[20] Vijaykumar, R., and Arbib, M.A., "Problem Decomposition for Assembly Planning" *Proceedings, 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, Mar.31-Apr.3rd, 1987, pp.1361-1366.

[21] Vijaykumar, R., Venkataraman, S.T., Dakin, G., and Lyons, D., "A Task-Grammer Approach to the Structure and Analysis of Robot Programs" *IEEE Workshop on Languages for Automation* Aug. 24-27th, Vienna, Austria, 1987.

[22] Yoshikawa, T., Nagai, K., "Analysis of Grasping and Manipulation by Multifingered Robot Hands," *Proceedings Workshop on Dexterous Robot Hands at the 1988 IEEE International Conference of Robotics and Automation,* April 24, Franklin Plaza, Philadelphia, PA, pp.149-171.