A Sensor Based Strategy for Automatic Robotic Grasping

by

David Lawrence Brock

B.S.M.E., Massachusetts Institute of Technology
(1984)

M.S.M.E., Massachusetts Institute of Technology
(1987)

B.S. Mathematics, Massachusetts Institute of Technology
(1989)

Submitted in Partial Fulfillment
of the Requirements for the
Degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

January 1993

Signature of Author Signature Redacted [illegible]

Department of Mechanical Engineering
January 1993

Certified by Signature Redacted

Dr. J. Kenneth Salisbury, Jr.
Thesis Supervisor

Accepted by Signature Redacted

Professor A.A. Sonin
Chairman, Department Graduate Committee

A Sensor Based Strategy for Automatic Robotic Grasping

by

David Lawrence Brock

Submitted to the Department of Mechanical Engineering
on 1 January 1993 in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Abstract

To operate effectively in many domains, the robot must react to unexpected events, as well as generate persistent action to achieve a task. This thesis describes a sensor based strategy, termed a *task-level control system*, which allows a robot to respond appropriately to perceived changes in the environment while retaining the ability to define sequences of preplanned motion. The technique has been applied to robotic grasping, allowing the theoretical development to be tested in a practical implementation. This research develops a framework for the task-level control system, including methods for state estimation based on previous sensor history and environmental models, as well as proofs of stability and convergence. This thesis also describes analytic tools to facilitate the design and analysis of task-level control systems, as well as implementations in both simulation and on the robot hand/arm system.

Thesis Committee:
Dr. Kenneth Salisbury, Chairman, NE43-837
Prof. Tomás Lozano-Pérez, NE43-836A
Prof. Mark Raibert, NE43-765
Prof. Haruhiko Asada, 3-350

ACKNOWLEDGMENTS

This thesis report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by Sandia National Laboratories under contract 23-2299, in part by the Systems Development Foundation, in part by the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685, in part by the National Aeronautics and Space Administration under NASA Grant NAG9-319, and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124.

Contents

Chapter 1

Introduction

1.1 Introduction

A robot is a machine designed to autonomously perform a task. In order to achieve a given task, the designer must consider not only the mechanics of the robot, but also of the environment in which it operates. In a static, structured and known environment, preplanned motion and feedforward control may successfully perform many tasks; however in a dynamic, chaotic and unpredictable environment, successful execution is more difficult. Between these two extremes lay environments which may be considered *semi-structured*, being neither perfectly regular nor completely random. This type of domain, characteristic of many "real-world" situations, models reality in that many orderly and uniform structures are embedded in chaotic and unpredictable surroundings. To operate effectively in such an environment, the robot must take full advantage of all uniform aspects, by generating persistent and rational plans, yet simultaneously compensate for all unpredictable features, through the immediate generation of corrective action.

The main result of this thesis is a control architecture designed to operate effectively in semi-structured environments. The control system combines preplanned sequences of action with the ability to react immediately to perceived environmental changes. The architecture follows the functional decomposition common in reactive or behavioral systems, yet combines geometric and kinematic models from traditional planning. Because of the incorporation of explicit models, we are able to develop criteria for stability and convergence, and the avoid chattering instabilities and limit cycles common in reactive control systems. The models also provide a method for the design of robot action given the limited information gathered from the sensory systems. The system also facilitates the incorporation of disparate sensory systems, feature abstraction algorithms, modeling systems, and controller functions. Finally, the system also provides a hierarchical task description, allowing complex operations composed of functional primitives.

The control system is composed of a collection of independent modules designed to accomplish a specific task. Associated with each module is a set of feature abstraction algorithms which resolve only those features necessary to accomplish the given task — thus minimizing the complexity of the object recognition algorithms and providing an opportunity for real-time control. The features abstracted from the current sensor data are used to construct a partial model which is then compared to one which uses the complete sensor history. Discrepancies in the models are resolved by assuming the current sensor information reflects some unexpected change in the environment, and the internal representation is updated accordingly. Finally, for each model generated, there is a unique stereotypical action, defined as either a sequence of tasks or a specific robot motion.

Geometric and kinematic models are used to design the stereotypical actions, as well as to guarantee successful execution of the specified task. In this way, it is possible to use explicit models in the generation of the real-time control algorithms without the associated complexity of embedding these models in the control algorithm itself. In order to demonstrate these concepts and to provide a practical application, this thesis will focus on robotic grasping, which allows both a physical implementation on a robot hand/arm system and thorough analysis using geometric models.

1.2 Background

Servo control systems which allow robotic manipulators to follow predetermined trajectories have been well characterized. The difficulty lies at the next higher level of control, in which robot is required to perform a specific task within a particular domain. At this level it is necessary to include a description of both the robot *and the environment* into the design and analysis of the control system. Two approaches have emerged to address this issue of "task-level" control. These are geometric planning and reactive control.

1.2.1 Planning

Planning systems are characterized by a linear sequence of computational modules, which interpret sensor data, generate a world model, reason within this description, and produce an appropriate trajectory, as illustrated in Figure 1-1. Raw data acquired from physical sensors, such as passive vision (Horn 1983), laser scanning (Jones 1988), tactile sensing (Grimson 1984) or range sonar (Drumheller 1985), is interpreted by a perception algorithm, which abstracts relevant feature data, such as surface points and normals, edges and corners. Using tree search techniques, these abstracted features are compared to the features from library of possible geometric shapes in a variety of possible positions (Gaston 84; Grimson 84, 85). By limiting the search of feasible interpretations through geometric constraints and empirical pruning, the algorithm eventually produces a set of consistent objects and poses. The result is a model of the environment consisting of a set of polyhedral objects together with coordinate transformations describing their position relative to some reference frame. At this point a planning or reasoning system is used to derive a sequence of trajectories necessary to achieve a desired task. Such tasks which have been extensively studied, include collision free motion, fine motion planning, automatic grasping, sliding and assembly.

Collision free robot motion has been an area of intense research. Collisions can be predicted by intersecting polyhedral models of the robot and environment (Ambler 1975, Lozano-Perez 1976) or by finding polyhedral contact conditions (Canny 1986; Kawabe 1987). To find collision free motion, polyhedral obstacles can be mapped into the configuration space of a moving polyhedron and paths are found using general conic freeways (Brooks 82a, 82b, 83), rectanguloids bounding slice projections of the obstacles (Arimoto 1987; Lozano-Perez 1979, 80, 81), or more generally as in (Donald 1985; Canny 1987, 88). While free space pathways through polyhedral obstacles proved sufficient for global motion, fine motion control was found necessary near grasp locations (Lozano-Perez 1984). These grasp locations were generated on polyhedral objects assuming the necessary conditions of kinematic feasibility and force closure (Abel 1985; Brost 1986; Cutkowsky 1987b; Jameson 1985; Laugier 1981; Li 1987; Nguyen 1986a, 86b, 86c, 87a, 87b; Pertin-Troccaz 1987; Pollard 1989). With the object securely in a grasp, internal forces may be optimized and major slipping modes resolved (Brock 1987, 88; Fearing 1984, 86; Markenscoff 1989; Mason 1982). Finally, the robot may impose arbitrary position, force or stiffness trajectories (Chiu 1985; Salisbury 1985), or perform some operation, such as insertion (Caine 1985; Whitney 1985).

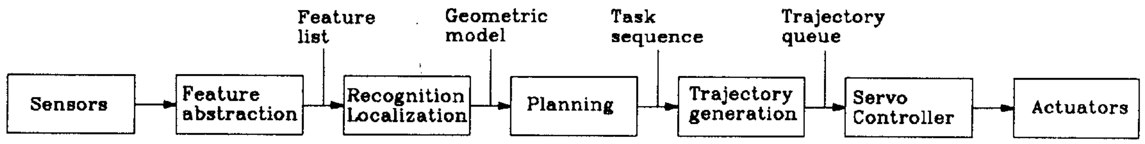


Figure 1-1. Planning systems consist of a linear sequence of computation modules. Raw sensor data is interpreted by a perception algorithm which abstract relevant feature data. This features is then compared, using search techniques, to a library of shapes to produce an explicit internal model. A planning or reasoning system then analyzes this model to determine motions necessary solve the particular task. Finally, a trajectory list is generated which is issued to the servo control system to be executed on the physical robot.

The planning methodology has a number of distinct advantages. Since explicit mathematical representations are used of the environment, operations such as collision free motion, force closure grasps, and object insertion can be guaranteed to succeed within the limits of the model. Furthermore, object recognition and localization, path planning, and task execution may all be incrementally improved by increasing the accuracy of the models or through more efficient tree search techniques.

There are, however, a some disadvantages with traditional planning. First, since tree search techniques are used to fit various models into acquired feature data, increasing the number and complexity of the objects often greatly increases the computational complexity of the recognition and planning systems. Second, the internal representation generally relies on accurate models of the robot and objects. Thus inherent errors due to sensor noise and robot calibration, may produce aberrant results, or at least contribute to the overall complexity. Third, the acquired sensor data may be inadequate to resolve a unique description, and in situations where robot action is either demanded or advantageous, motion would not be possible since the system would not have a complete motion with which to operate. Fourth, some assumptions in traditional planning may be overly restrictive and could, in some circumstances, be eased. For example, guaranteed collision avoidance could be replaced in some non-critical domains by local contact force minimization together with compliant robotic surfaces. Fifth, the models used in current planners may be inappropriate for some environments. For example, many common objects display a great deal of symmetry and simplicity. These simple objects would be obscured by a more general polyhedral model; while other objects, particular non-rigid ones such as paper, cloth and wire, cannot be modeled by rigid polyhedron at all. Finally, the complexity of current planning systems generally precludes their use in rapidly changing environments. Small changes in position of the robot or objects in the environment would generally require complete path replanning. Whereas a small change of the current plan — or no change at all — may have been sufficient.

1.2.2 Reactive control

An alternative approach to model based planning is reactive control. The reactive control strategy consists of a collection of independent modules which relate preprocessed sensor input to either actuator output or stereotypical motor responses, as shown in Figure 1-2. These modules are generally of uniform computational structure and are organized into hierarchical layers or amorphous networks. The individual modules essentially embed the functions of feature abstraction, model construction, and trajectory planning into a single intrinsic function, thus allowing minimum computation and real-time response.

The concept of reactive control developed from a number of sources, including the need for dynamic task-level control, reduction of computational complexity, increased robustness, and improved fault tolerance, as well as observations from the biological sciences, including ethology (the study of animal behavior under natural conditions), neurobiology, and psychophysics. The need for real-time, task-level control became apparent when sensorless ballistic trajectories generated from planning systems cause aberrant or catastrophic motion when sudden environmental changes or calibration errors produced discrepancies between the internal model and the physical world. The solution proposed by reactive systems was to directly link sensors to actuators at every level of control, thus improving response in dynamic situations, as well as minimizing error propagation from sequential computation. To further reduce complexity in reactive control systems, the definition of the tasks associated with the individual modules was greatly reduced. In other words, instead of a general purpose algorithm, such as obstacle avoidance among polyhedral objects, the modules solved simple tasks for specific situations. In addition, the designers of reactive control systems endeavored to produce competent strategies in common situations rather than optimized behavior in abstract domains.

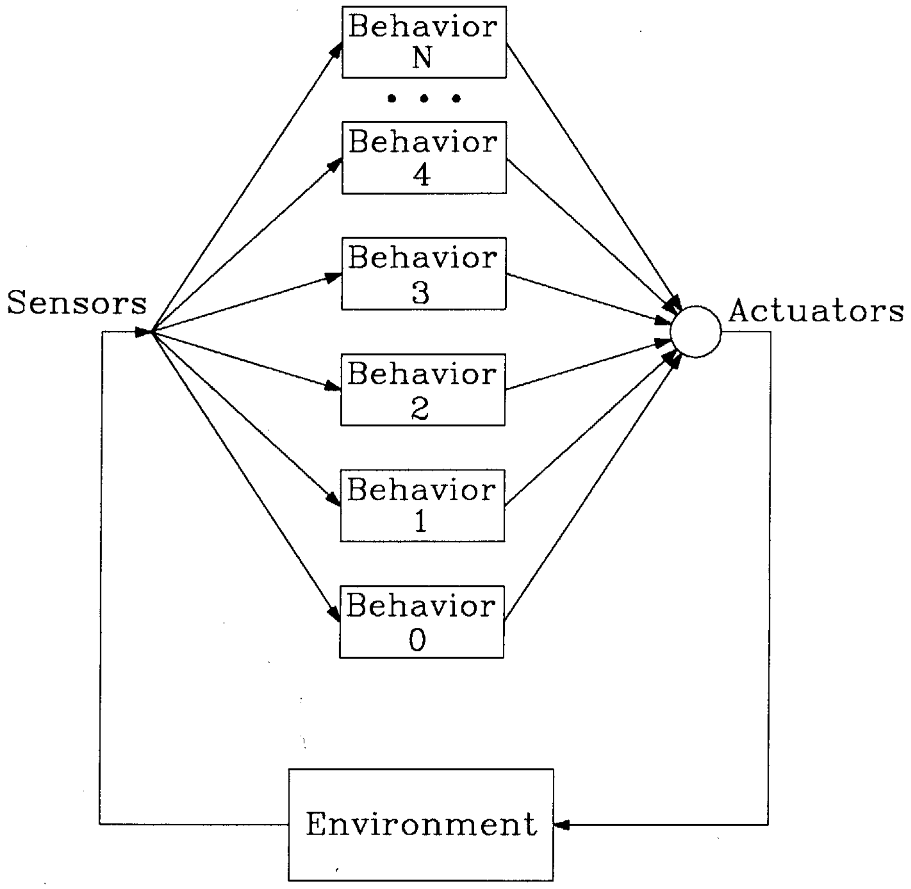


Figure 1-2. The reactive control system consists of a collection of independent modules arranged in either a hierarchy or an amorphous network which directly relate sensor input to actuator output. The individual modules essentially embed the functions of feature abstraction, model construction, and trajectory planning into a single intrinsic function, thus allowing minimum computation and real-time response.

Part of the motivation for reactive control strategies came from observations of biological systems, whose robust and efficient behavior inspire serious consideration. Studies of animal behavior in their natural environment show that many animals possessed motor programs (coordinated sequences of motor response) evoked by releasers (particular patterns of sensory input) (Gould 1983, 87). Thus, by analogy, the individual modules of a reactive control system could be considered motor programs, while their specific function depends on releasers. Neurobiological research into human and primate cerebral function seems to indicate that the brain is composed of a loosely connected set of independent computational structures specialized for particular functions. Studies of the visual pathway (Zeki 1988; Livingstone 1988, 89), sensory motor, and auditory, show that, in general, raw sensor data is analyzed by a hierarchy of increasingly more numerous, abstract, and specialized computational modules. Finally, psychophysical research into human manipulation and perception demonstrates that, in many cases, stereotypical movement is generated by different individuals in similar situations (Gesell 1952). Thus it could be argued that the general pattern for biological systems is represented by reactive control systems (and neural networks, by analogy, at a finer level). While it would be difficult and perhaps inadvisable to copy biological systems exactly, their study may provide concepts whose independent and thorough analysis may yield practical and efficient systems.

The reactive control strategy was applied to robotics in a number of different ways, including the emulation of human tactile exploration (Bajcsy 1984, 87), stereotypical manipulation primitives (Chammas 1989, 90; Greiner 1990; Grupen 1988; Iberall 1987; Stansfield 1988, 89, 90; Tomovic 1987), qualitative control strategies (Jacobsen 1987), and the subsumption architecture (Brooks 1985, 86a, 86b, 86c, 86d, 87, 88, 89a, 89b; Connell 1987, 88, 89; Flynn 88, Mataric 1989). The subsumption architecture consists of a hierarchical arrangement of small independent functions, in which superior units subsume, when appropriate, control over less competent modules.

The reactive control strategy has a number of advantages, some of which have been outlined above. Since only simple computational elements are used with a minimum of sequential processing, the system can respond instantly to perceived environmental changes. Furthermore, the parallel computational architecture minimizes error propagated between sequential elements, as well as increases redundancy and resistance to catastrophic failure along critical pathways. In addition, homogeneous computational elements allow uniform algorithmic definitions and compiler designs. Finally, the segregation of control into multiple independent elements provides flexibility and diagnostic ability, in that individual modules may be independently enabled.

There are, however, a number of disadvantages with reactive control systems. First, there are few theoretical tools for the design and analysis of such systems. The computational modules which compose the control system are usually designed through heuristic trial and error, while their resulting behavior is proven through simulation or physical prototypes. Second, it is not possible to prove, in general, the success of the control strategy, or even the stability, when applied to a given environment. Third, it is difficult to determine the rate or the time necessary for these systems to converge, that is to accomplish a given task. Fourth, there are no methods or guidelines to provide incremental improvement to the system once it has been designed. Fifth, while it may be possible to verify the performance of an individual module, it is difficult to determine the interaction between modules in the complete system. It is this interdependence and collective behavior which makes these systems particularly difficult to analyze.

1.2.3 Summary

The planning and reactive control strategies evolved independently to meet the needs of fundamentally different environments and thus employ separate sets underlying assumptions. Planning systems initially developed from the need to perform autonomous assembly in manufacturing environments. In this domain, objects were assumed to be static rigid solids, easily represented as complex polyhedron. It was also assumed these objects could be in arbitrary orientations and in the presence of other objects which could occlude or constrain them. Reactive control systems, on the other hand, were applied primarily to the control mobile vehicles in dynamic unstructured domains, many of which were populated by humans. These mobile vehicles were required to perform such fundamental tasks such as collision avoidance, terrain contouring, and simple navigation. Objects were generally assumed to be obstructions or free-space pathways which may change at any time. Thus the environment was considered to be unpredictable and without structure or regularity. There are, however, many environments which can be considered regular, but retain their ability to undergo unpredictable change. These are the domains to which the task-level control strategy will be applied.

1.3 Task-level Control

The task-level control scheme describe in this thesis attempts to combine the advantages of both reactive control and geometric planning into a single system. From reactive control, the task-level control system provides dynamic response, reduced complexity algorithms, independent computational modules, parallel architecture, and redundant structure; and from the planning approach, feature abstraction algorithms, explicit world models, analytic design and evaluation tools, and proofs of stability and convergence. The basic architecture of the task-level control system is illustrated in Figure 1-3. Control of the robot is decomposed into a collection of discrete *tasks*. Each task achieves a particular objective given a specific arrangement of the environment as perceived by the robot. In other words, rather than solving a complex problem for a broadly defined environment, we solve a number of specific problems in narrowly defined situations. By linking these specific solutions together in an appropriate way, we create a seamless task-level control system, able to solve more general situations the robot may encounter.

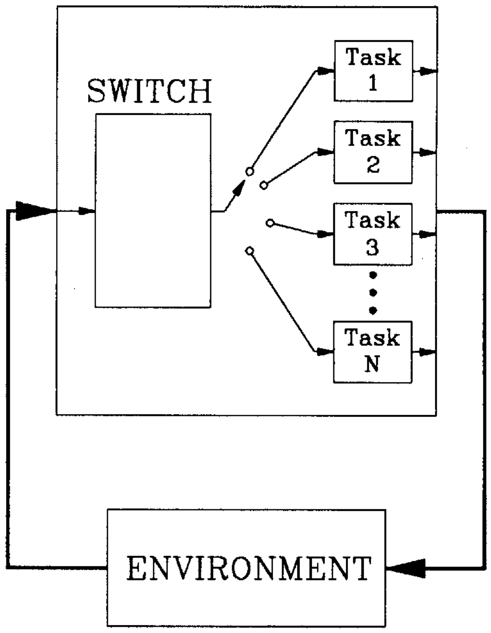


Figure 1-3. Control of the robot is decomposed into a collection of discrete *tasks*, where each task solves a particular problem for a specific arrangement of the environment. By linking together these specific solutions, we create a seamless control system, able to more accommodate general situations encountered by the robot.

Each task in the system, as shown in Figure 1-4, is comprised of a *context*, that is an arrangement of the environment as it is known to the robot; a *goal*, that is an objective to be achieved within the context; and an *action*, which is either a further collection of tasks or a stereotypical robot motion designed to achieve the immediate objective. Since the action may be defined as a sequence of tasks, this produces a recursive structure of tasks and subtasks which operate on increasingly specific arrangements of the environment, as shown in Figure 1-5. Since the environment is monitored at every level, small changes in the world may only require minor revisions in the execution of the tasks at lower levels without affecting the overall execution at higher levels. The advantage is that overall plans need not be thwarted by minor variations in the perception of the environment.

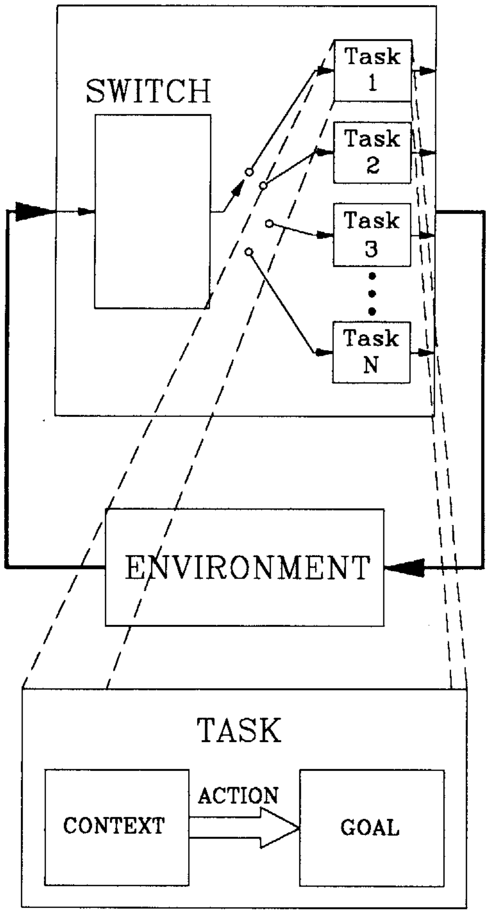


Figure 1-4. Control of the robot is decomposed into a collection of discrete *tasks*. Each task is described by a *context*, that is a particular environment and arrangement of the environment as it is known to the robot; a *goal*, that is an objective to be achieved within the context; and an *action*, which is either a further collection of tasks or a stereotypical robot motion designed to achieve the immediate objective.

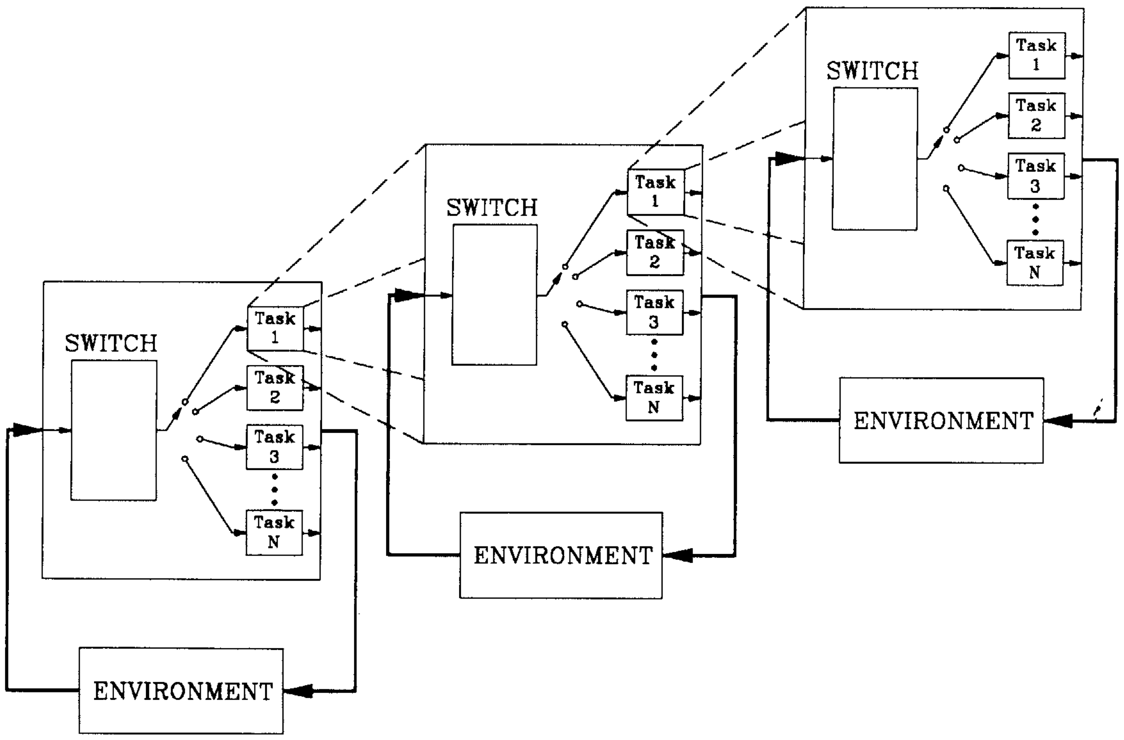


Figure 1-5. The action associated with a task may be defined as a sequence of tasks, thus producing a recursive structure of tasks and subtasks which operate on increasingly specific arrangements of the environment.

The context is defined as a specific arrangement of the environment as it is known to the robot. To determine whether a particular task should execute, the task-level control system must determine whether the environment, as currently perceived, satisfies the conditions described by the context. Therefore each context has an associated set of computational modules which perform the functions outline in Figure 1-6. Features abstracted from low-level sensory data are used to construct a partial model of the environment. This model is then compared to one based on the previous sensor history. Discrepancies arising between the current and the previous models are then assumed as result of some sudden and unexpected change in the environment. Given this situation, the system incorporates only the current knowledge, and the previous model is disregarded. If, on the other hand, there exist common states between the previous and current models, the new model will represent their union. Since the set of computational modules shown in Figure 1-6, is defined for each task, the algorithms are greatly simplified since they need only determine those environmental features relevant to a particular task. Furthermore, because of the limited scope of these algorithms, real-time computation is possible — thus providing an accurate representation of the current environment. Also combining Figure 1-6 with 1-5, we see that the task-level control system provides a model of the environment of increasing resolution and one specific to the needs of the particular tasks.

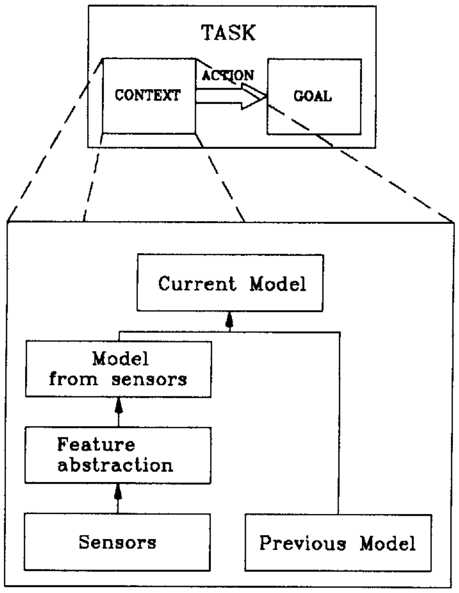


Figure 1-6. The context associated with each task is resolved using a hierarchy of perceptual elements. These elements detect features from lower level sensory data to construct a current model of the environment specific to the associated task.

The task-level control system has a number of advantages. First, since recognition is coupled to specific tasks, the associated modeling algorithms may be greatly simplified, thus allowing the possibility of real-time response. Second, since the task-level control system provides a hierarchical representation of increasing complexity, changes in the execution of the lower level tasks may have no effect on the course of higher-level task execution. Third, incremental improvement is allowed by replacing only some tasks or perceptual algorithms without affecting the rest of the system. In addition, new tasks and contexts may be added to increase the scope and the competence of the system without affecting the existing algorithms Fourth, the modular structure provides an opportunity for parallel execution, since the computational dependencies are explicitly delineated. Fifth, the control system employs explicit world models which allow traditional geometric analysis and planning, as well as proofs of convergence and performance criteria. Sixth, the system provides a link between lower level trajectory generation and higher level task description.

1.4 Assumptions

The task-level control scheme is designed to operate in a *semi-structured* environment, that is an environment composed of regular objects in various configurations, which may, at any time, undergo random displacements as a result of an unexpected disturbance or an unanticipated consequence of robot motion. The idea of a semi-structured environment was derived to contrast with *structured* and *unstructured* environments

— terms used previously to describe the robot domain. Here we describe a structured domain as one composed of objects whose geometry and position are orderly and predictable. This description essentially represents manufacturing environments used in the development of robotic control system for automated assembly and computer control machinery. Unstructured environments, on the other hand, refer to potentially chaotic surroundings associated with undersea, extraterrestrial, and hazardous environments. Objects in this domain are assumed to be of arbitrary form and location, and whose positions may change dynamically. The description of a semi-structured environment essentially lies between these two extremes; that is the objects in this domain are of uniform geometry and their position may only vary within certain constraints. This description attempts to model many man-made and natural environments, in which the objects and their components contain symmetric and regular structures and that their position, while usually stationary, may change unexpectedly.

In this thesis, we will concentrate on the geometric and kinematic aspects of grasping rather than on the mechanical dynamics and contact force interaction. The objective is to exploit the available sensor information to build a partial geometric representation, which will guide addition exploration or partial task completion. Although we use simple geometric solids in the examples, some of the techniques and analyses are applicable to more general geometries.

1.5 Example

Consider the problem of grasping a small rectangular object, such as a blackboard eraser or small box, from a planar surface, such as a table or shelf, as shown in Figure 1-7. In this example, we assume that the robot or human is without vision and must grasp the object using only tactile and kinesthetic sensing. We make this assumption for the purposes of illustration, and to demonstrate that robust grasping behavior is possible without complex vision hardware. Also the application of strategies based on subsets of the available sensors provides a basis for fault tolerant intelligent control systems. Suppose the position and the exact size of the object are not know, except that it lies on the planar surface and is approximately rectangular.

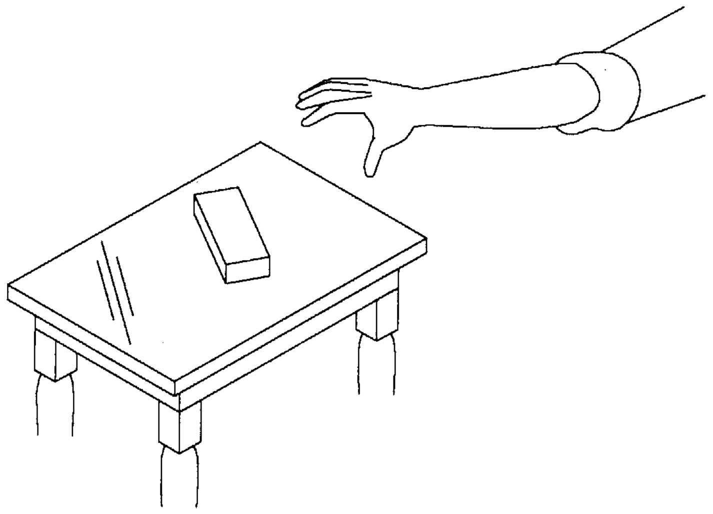


Figure 1-7. The objective is to grasp a small rectangular solid from a planar surface without the use of vision. We suppose the exact size and position of the object are unknown, but that it is approximately rectangular and lies on the planar surface.

Thus the *task* in this example is to grasp the object, the *context* is a rectangular object on a planar surface, the *goal* is a force closure grasp, and the *action* is either a single trajectory or a sequence of subtasks will achieve this goal. There are a number of actions that one could image to solve this task. For example, the hand could move through a fixed trajectory over the surface of the table, and without using sensing, simply sweep the object into the grasp. The action in this case would be a single trajectory which moves the hand over every portion of the surface. However, this would be less efficient than otherwise possible, since it does not take advantage of the available contact sensing during the course of the motion.

Suppose we initially probe the surface and acquire a single contact on the object. The position of the object is now partially constrained to lie within a disk centered at the contact point, as illustrated in Figure 1-8. Now we must decide on a subsequent action. One strategy would be to continue to probe in order to acquire a additional contacts to uniquely resolve the position of the object. Alternatively we could generate a more efficient action, that is one which would bring us more quickly toward the goal of a force closure grasp. Suppose we move the center of the palm directly through the acquired contact point. Not only would this guarantee a second contact, but would also move the hand in the most likely position to grasp the object, as illustrated in Figure 1-9. With a second contact point resolved, it is possible to move the fingers into opposing regions about the longitudinal axis of the rectangular solid, as shown in Figure 1-10. By squeezing the fingers together, we are assured a force closure grasp, since it is not possible for the object to rotate to such an extent as to miss the converging fingertips.

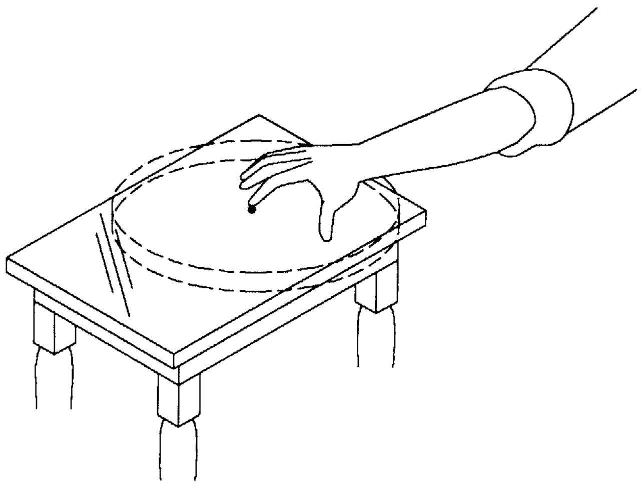


Figure 1-8. A single contact constrains the position of the rectangular solid to lie within a disk centered at the contact location.

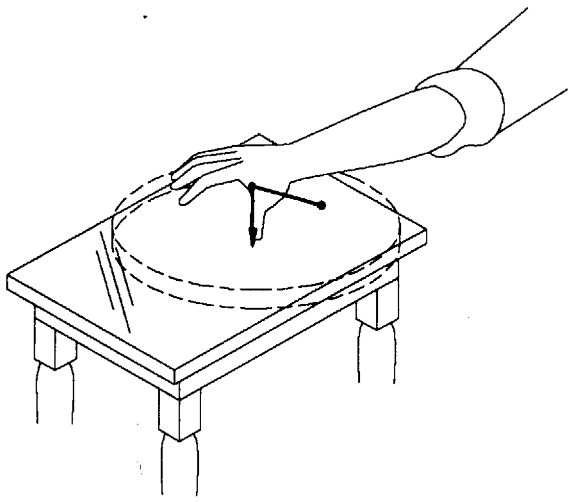


Figure 1-9. By moving the center of the palm through the acquired point, we are guaranteed an additional contact.

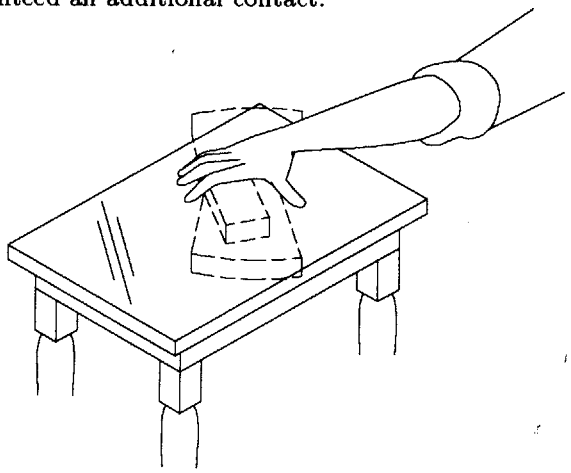


Figure 1-10. By moving the fingers about the longitudinal axis and squeezing them together we are assured a force closure grasp, since the object cannot rotate to such an extent as to miss the converging fingertips.

The task-level control system in this case could be described by four tasks, as illustrated in Figures 1-11 and 1-12. Each task has an associated context, that is a partial object model; a goal, which is a desired configuration between the robot and the object; and an action, which, in this case, is either a simple stereotypical trajectory or a task sequence.

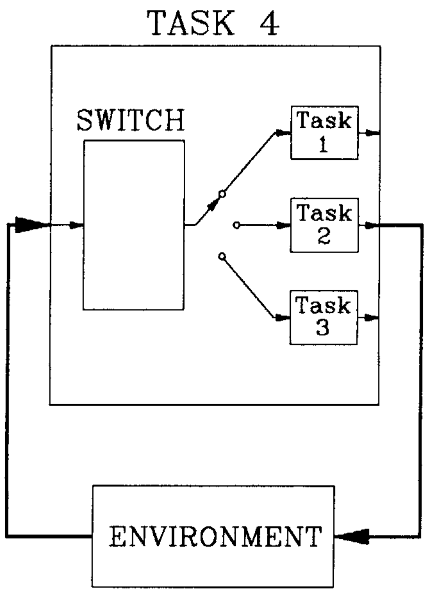


Figure 1-11. The task-level control system for this example is composed of four tasks, the higher-level task to grasp the block, and three subtasks: probe, center, and grasp.

Task 1:	
Context:	Rectangular object on a planar surface
Goal:	Single point constraint
Action:	Patterned search

Task 4:	
Context:	Rectangular object on a planar surface
Goal:	Force closure grasp
Action:	Tasks: 1, 2, 3

Task 2:	
Context:	Single point constraint
Goal:	Two point constraint
Action:	Palm through point

Task 3:	
Context:	Two point constraint
Goal:	Force closure grasp
Action:	Rotate about axis close fingers

Figure 1-12. Each task has an associated context, that is a partial object model constructed from the initial assumptions and information gained from the contact and position sensors; a goal, that is a desired configuration; and an action, which in this case is a simple stereotypical trajectory.

Again, the environment is continually monitored to determine the appropriate context and to execute the associated task. Suppose that during the execution of the final task in the sequence, the object was suddenly displaced, as shown in Figure 1-13. The discrepancy would be recognized as soon as an inconsistent contact was achieved or an expected contact did not occur. Since the new contact is inconsistent with the previous model, the task-level control system ignores the past information and accepts the new contact point. Given this scenario, task 2 is again executed, and the hand attempts center the palm over the contact point. If, as in the case in Figure 1-14, no contact was acquired where one was expected, the system is forced to reexecute the entire sequence.

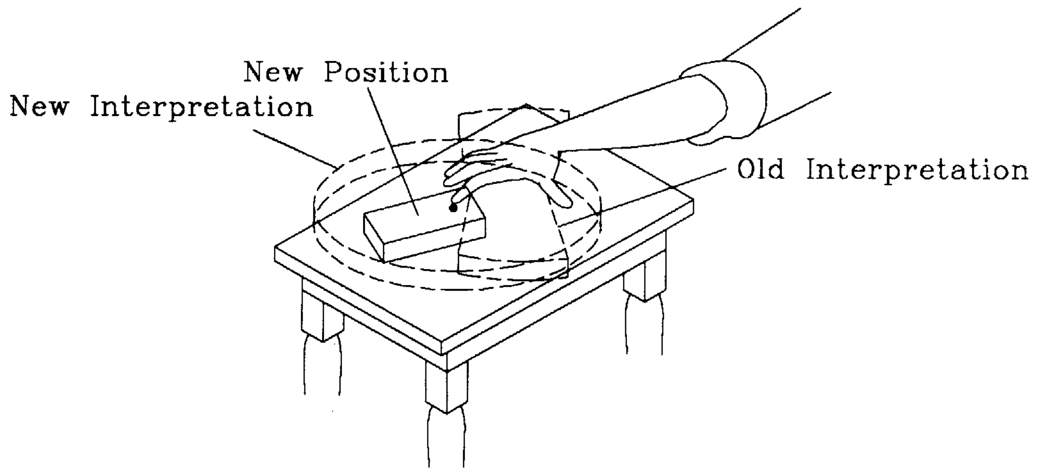


Figure 1-13. The object was unexpectedly moved such that a new inconsistent contact was generated.

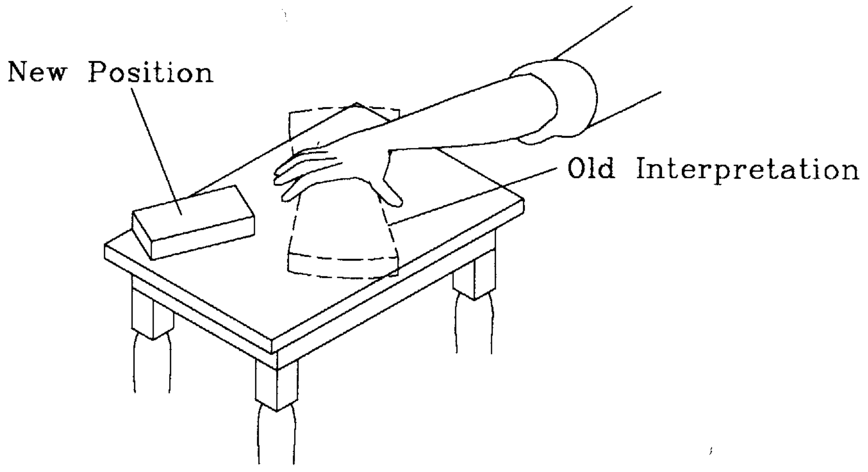


Figure 1-14. The object was unexpectedly moved such that an expected contact was not acquired.

1.6 Overview

Chapter 2 outlines the overall theoretical framework for the task-level control system. In this chapter we develop criteria for stable and convergence task-level behavior. Chapter 3 describes the architecture of the task-level control system. The architecture allows the rapid integration of multiple sensors and feature abstraction algorithms, task-level control routines, and debugging facilities. Chapter 4 introduces methods for the design and analysis of task-level controllers. Chapter 5 describes applications of the task-level control system, including a simulation of a planar hand/arm system and an implementation on the Salisbury Hand/PUMA arm robot system. Finally, chapter 6 provides a list of future research topics and robotic hardware.

Chapter 2

Theoretical Framework for Task-level Control

2.1 Introduction

In this chapter we will develop a theoretical framework for the task-level control scheme. We begin by partitioning the control of the robot into a number of discrete *tasks*, representing specific goals to be achieved for specific arrangements of the environment. Our definition of a task includes a *context*, that is a specific composition of the environment as it is known to the robot, a *goal*, that is a desired objective to be achieved within that context, and an *action*, that is either a further sequence of tasks, or a robot motion, designed to achieve the immediate goal, as shown in Figure 2-1.

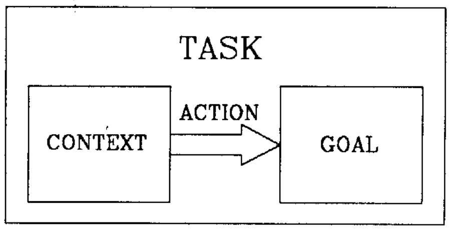


Figure 2-1. A *task* is composed of a *context*, that is a particular composition of the environment, a *goal*, that is an objective to be achieved, and an *action*, that is a further sequence of tasks, or a motion, designed to achieve the immediate objective.

The context traditionally has been defined rather broadly. For example, a common definition is of a collection of arbitrary polyhedral objects in random, possibly overlapping, positions. The difficulty with such a general description is the inherent complexity of object recognition and task planning. Alternatively we could describe the context as a more restrictive class of environments. Not only would this be consistent with the concept of a semi-structure environment, but would also greatly reduce the complexity of recognition and planning. For example, consider an environment composed of only isolated objects on a planar surface, each with opposing parallel faces. In order to grasp such an object, a recognition system need only approximate the location of these faces, so that a planning system may place antagonist fingers on opposite sides. Other examples, may include spatial arrangements of extruded solids, bundles of flexible linear objects, or aggregates of amorphous spheroids. At one extreme we could have a single model to represent all possible objects and environments, and on the other, a separate model for every situation the robot may encounter. Between these two extremes we can define a set of environmental contexts, which individually are as generic and simple as possible, and which corporately approximate a complete world description.

The definition of the context also includes information about the environment as it is known to the robot. For example, the task of retrieving a book from a table would be very different if we could not use vision. The acquisition of the book, however, could still be accomplished, but would require some exploratory motion. There are other perhaps less extreme examples were sensing provides only incomplete or approximate models of the physical world. Our objective is to include partial models into the definition of the context, so that robust behavior may be designed despite limited information.

The goal is defined as the achievement of a particular relationship among objects in the environment. For example, a goal may be the alignment of the principal axes of robot with those of an object, the mating between two parts, a conformational change of an articulated object, or the acquisition of an object through a force closure grasp. In any case, we represent all possible configurations in terms of a *state space*, that is the space of all possible positions of the robot and objects within the environment. We then define the goal as collection of subsets or regions within this state space. For example, if the objective is to align the palm of a robot hand with the principle axis of a revolute solid, then the goal is simply a subset of all configurations which produce this alignment. The definition of a goal thus provides an explicit definition of success, as well as an analytic basis for the evaluation of performance.

An action is defined as either a further sequence of tasks or a single motion designed to achieve the goal. If an action is described by a sequence of tasks, this results in a recursive structure as shown in Figure 2-2. In this case, the goal associated with a particular task is contained within the context of the subsequent task in the series. The result is a sequence of increasingly refined contexts and actions which carry the system from one context to the next, as shown in Figure 2-3. Suppose, for example, the task is to push an object which lying on a table. We could decompose this problem into a sequence of two tasks. The first is to move the hand into the plane of the table, and the second is to move within the plane to push the object. The context of the first task thus includes the full three dimensional space describing the location of the end effector, and the context of the second includes only the subspace defined by the plane of the table. In general, the purpose of the decomposition is to create a sequence of increasingly restrictive contexts in which the system is allowed to operate. This reduces the complexity of design and analysis of robot action by reducing consideration to action only within a particular context.

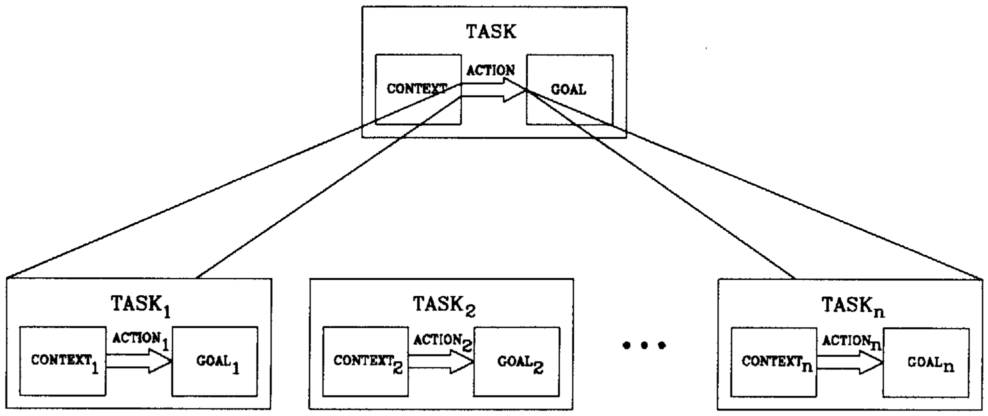


Figure 2-2. An action can be defined as either a sequence of tasks or a single trajectory designed to achieve the goal. If it is described by a task sequence this produces in a recursive structure, in which each of the subtasks has the same structure as the parent.

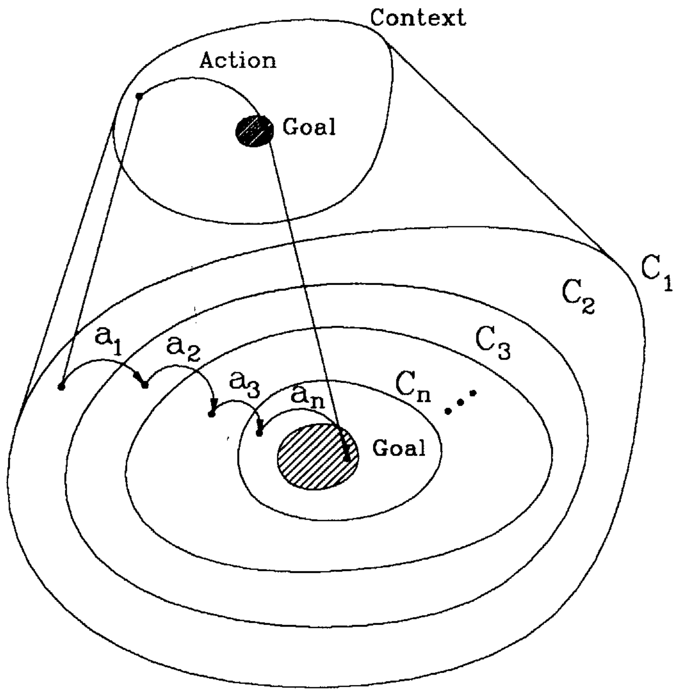


Figure 2-3. The goal associated with each task in the sequence is contained with the context of the subsequent task. Thus a task sequence represents an increasingly refined set of contexts together with associated actions which carry the system from one to the next. Note C_i denotes the context and a_i denotes the action associate with the i^{th} task in the sequence.

By recursively expanding the definition of the action, it is clear that all tasks are composed of a sequence of robot trajectories. It may seem more efficient to plan a single trajectory in the first place rather than this collection of discrete motions. However, the recursive structure has a number of advantages. First, it may not be possible to resolve the state of the environment with sufficient accuracy to generate a single rational plan from start to goal. Intermediate exploratory motion may be necessary to acquire enough information to achieve the task; and it is this possibility which we wish to include explicitly into the definition of the task. Second, unexpected changes in the environment may preclude the applicability of persistent plans or long-term trajectories. In the semi-structured model of the environment, we assume that unpredictable events occur in an otherwise static or quasi-static domain. Thus it is necessary to include in the definition of the task, the ability to compensate for sudden unexpected disturbances, and it is this ability that essentially defines the approach as a control system. Third, the definition of a specific action for specific context greatly reduces the complexity of design and analysis. One of the objectives of this approach is to decouple complex robot/object interaction into a sequence of discrete contexts and actions. Finally, the decomposition of a task into multiple independent entities provides an opportunity to create a reusable set of primitives which may serve as a basis for other composite tasks.

Although tasks may be specified in a sequence, as shown in Figure 2-2, the order in which they are executed does not depend on their position in the series, but on the recognition of the appropriate environmental context. For example, suppose the task is to retrieve a book from the shelf and place it on the table. A task sequence is specified as follows: locate book, acquire grasp, transport, orient, and release on the surface. However, suppose that during the retrieval process, the book slips and falls to the table. In this case, the task is done — albeit accidentally — and the robot should immediately proceed to some other problem. The natural sequence: transport, orient, and release, was circumvented by an unexpected event, and it is this type of event which we wish the robot to recognize and respond to. In general, however, the natural course of events tends to thwart rather than complete the task. Thus elements of the task sequence may have to be repeated or even abandoned in favor of some more fundamental objective. For example, suppose the book falls behind the table, in which case a more basic problem of clearing a path to the object must be undertaken.

Task execution may therefore be considered a control system, as shown in Figure 2-4. The environment, as perceived through the physical sensors, dictates the execution of the current task. If the system, that is the perceived states of both environment and the robot, are within the context of a particular task and not within the goal, the task continues to execute. The task-level controller determines the appropriate context by sampling the sensors at certain time intervals. For each sensor sample an interpretation is constructed, that is a partial model of the environment based only on the current sensor value, as shown in Figure 2-5. This model is then compared with one constructed from the previous sensor data to produce the current knowledge, that is the partial model of the environment based on the current interpretation and the previous environmental model. If the current knowledge satisfies the conditions for the context but not the goal, then that task is executed, that is the action associated with the particular task is executed. If the action is a task sequence, then this sequence of subtasks is enabled, as shown in Figure 2-6. We say that a task is enabled if the control system evaluates the associated context and goal. In other words, it is not necessary to evaluate all the tasks in the system, but only those pertinent to the current situation. However, if the action is a trajectory, the robot immediately begins to execute that motion. Again, this motion continues as long as the perceived environmental states remain within the context and not in the goal. Thus the sequence of task execution depends on the state of the environment, and may vary as depicted conceptually in Figure 2-7, where the solid arrows represent orderly transitions, and dashed arrows, unexpected disturbances. These disturbances may be advantageous or detrimental to the overall objective, but in either case, the task-level system response immediately to these perceived changes and generates the appropriate action.

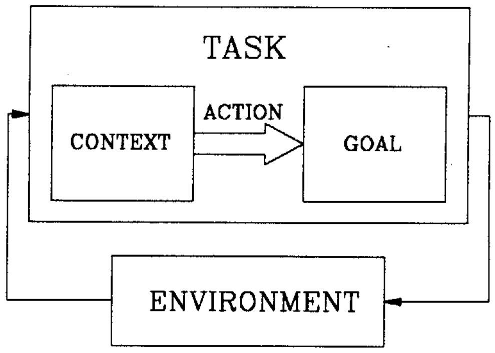


Figure 2-4. Task execution may be considered a control system. The environment as perceived through the physical sensors determines the context. If the system is within the context and outside goal, the associated task continues to execute.

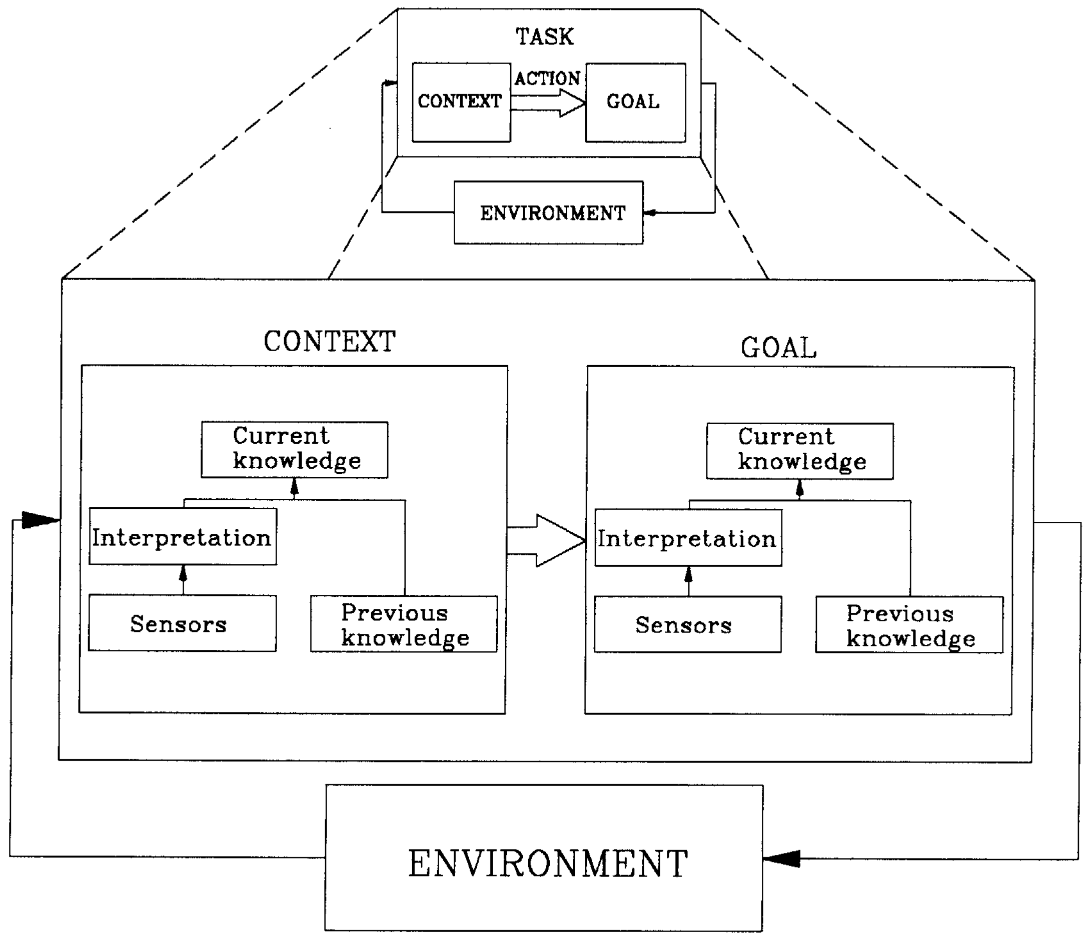


Figure 2-5. Sensor values are used to compute the interpretation, that is a model of the environment based on the most recent sensor data. This model is then compared with the previous knowledge, which is the model based on all previous sensor values, to determine the current knowledge. The action associated with the task is executed as long as the currrent knowledge remains inside the current context and outside the goal.

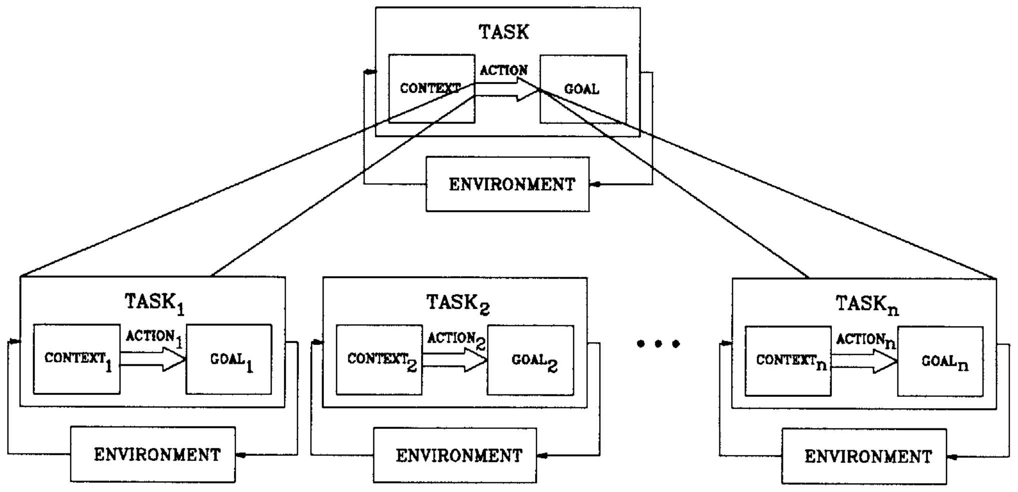


Figure 2-6. If the action associated with an active task is a task sequence, then each task in the sequence is enabled, each with its own context, action, and goal.

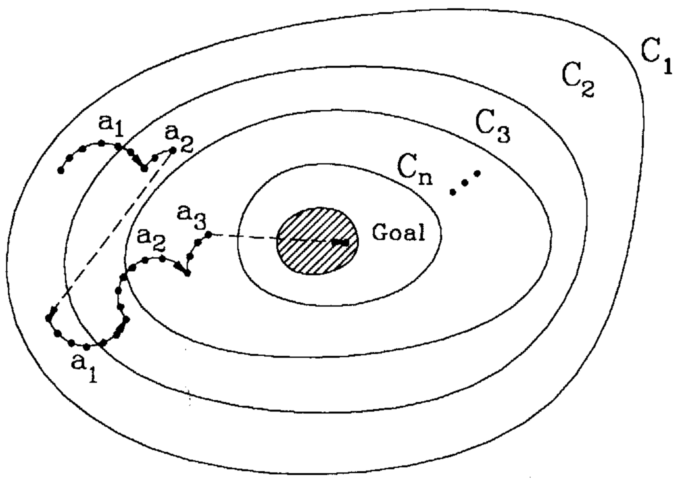


Figure 2-7. The sequence of execution depends on the particular context as perceived from the sensor data. Thus the actual path of execution may undergo discrete jumps resulting from unexpected occurances in the environment. Note C_i denotes the context and a_i denotes the action associate with the i^{th} task in the sequence. The nodes along the path represent time steps during which the current knowledge is reevaluated.

In the following sections, we will offer a formal definition for the *task*, along with its components, the *context*, *goal*, and *action*. We will also present a definition for *task-level control system* in general, as well as conditions for guaranteed stability and convergence. We also include, along with the definitions, an example which we will use throughout this chapter to clarify some of the concepts. The example we consider is a single link and disk, as shown in Figure 2-5. The link is equipped with a joint position sensor, and two binary contact sensors on opposing sides. The disk is free to move within a rectangle of width w and height h, though we initially assume it resides within the workspace of the link. Therefore if the disk is given by a location

$\mathbf{x} = (x, y)$ and a radius R, and, the link, by an angle θ and a length L, we initially assume $||\mathbf{x}|| \leq L$. Given these assumptions, the task is to simply touch the disk.

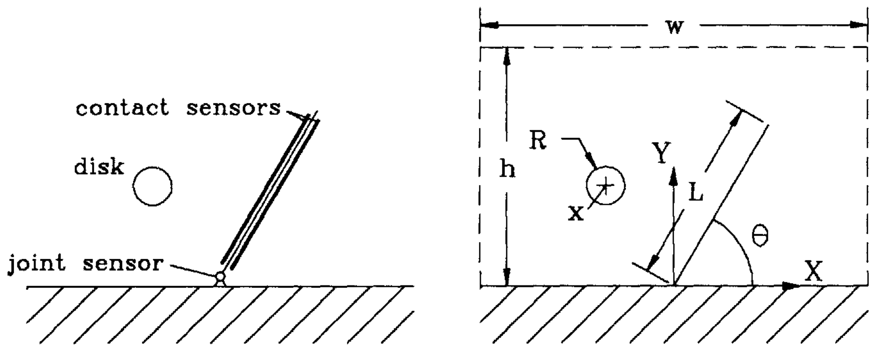


Figure 2-5. The task is to touch the disk with the link. We assume the link is equipped with joint position and binary contact sensors, and that the disk resides within the workspace of the link.

2.1.1 Assumptions

We make a number of assumptions in the example, and in the task-level control system in general. First, we assume the environment may change in unpredictable ways as a result of some unknown event. This compelled us to design robust behavior which compensates for accidents and errors which were not foreseen. Second, we assume the sensors on the robot always return true values, that is we assume the sensors return the correct value to within some bounded error. We do not allow the sensors to return random values, since it would be impossible to distinguish these from random changes in the environment allowed under the first assumption. The recognition of sensor failure based on constrained models of the environmental state would however be useful area for future research. Third, we generally assume quasi-static motion, that is motion in which the inertial forces are assumed to be negligible. In this thesis, we will concentrate on the geometric and kinematic aspects of sensing and manipulation. Finally, though not critical to the general definitions, we assume the environment is composed of discrete rigid solids.

2.2 Definition of the Context

The definition of the context embodies three ideas: first, the particular environment in which the robot operates; second, a particular relationship among objects within that environment; and third, the particular relationship as it is known to the robot. We allow the possibility of multiple environments, since it is our desire to reduce the complexity of the design and analysis of the robot action by restricting it to particular domains. In addition, we further constrain the operation to particular relationships among the objects within these environments. For example, the movement of the end effector in a plane, the alignment of a robot hand with a point, a rectangular solid with one edge on a line, a cylinder whose axis intersects a point, all define constraints on either the objects or the robot. Again, the idea is to simplify analysis of robot action, as well as maintaining relationships once formed among the objects and the robot. Finally, the context includes the idea of a relationship as it is known to the robot. In other words, the objective is to define a context which may be easily perceived by physical sensors available to the robot.

In order to provide a formal definition of the context, we begin by defining the state space as the space of all parameters of both the robot and the environment. In order to perceive these states, we must develop a method of estimating the current state from the physical sensor values. Therefore we begin with by discussing the physical sensors, and then describe an interpretation of their values. These interpretations must be reconciled with past information in order to generate a consistent knowledge space, that is a subset of state space describing the best state estimate. Finally, we must determine the context, that is to recognize whether the current knowledge satifies the relationship defined by the context.

2.2.1 State Space

The *state space* is the space of all parameters necessary to characterize the system. We describe the state space as the cartesian product of the states of the independent entities.

Definition. The **state space** $S = S^1 \times \cdots \times S^n$ is the set of all states of the system, where S^i is the state of all independent and complete parameters necessary to predict future events of the i^{th} entity. [1]

Example. In the example, the state space is given by $S = S^1 \times S^2$, where S^1 is the position of the disk

$$S^1 = \{(x,y)|x \in [-w/2+R, w/2-R] \text{ and } y \in [R, h-R]\},$$

where R is the disk radius, and h and w the width and height of the workspace, and S^2 is the position of the link

$$S^2 = \{\theta|\theta \in [0,\pi]\}.$$

2.2.2 Sensors

The physical sensors determine to a large degree the environmental features which may be perceived, as well as the types of tasks which may be performed and the methods used to achieve them. Thus rather than starting with a mathematical description

[1]The Cartesian product of sets A and B is the the set $A \times B = \{(a,b)|a \in A, b \in B\}$.

of the environment, we begin with by examining the physical sensing hardware and from this develop an environmental representation. Therefore we will briefly comment on the various types of sensing hardware available, the application of these sensors to particular tasks, and the effects of the sensors on the methods of task execution. We will then define a set representing the output of the physical sensors, and using this construct a representation of the environment.

Sensing hardware for robotics may be broadly classified as imaging, contact, and internal. We refer to imaging as those technologies which produce an array of data which can be used to construct a spatial representation of the surroundings, such as passive vision (Binford 1987, Horn 1983), active vision (laser light strip systems: Flynn 1987, Lozano-Perez 1989), tactile arrays (survey: Nicholls 1989, Howe 1992), and acoustic imaging (Steward 1988). Contact type sensing, on the other hand, refer to systems which provide partial information in the proximity of the sensor. Such systems include intrinsic contact sensors (Brock 1987a, Bicchi 1992), joint torque sensing (Eberman 1990, Gordon 1989), and proximity sensing (ultra-sonic: Mataric 1992, infered: Flynn 1988). We should note, that while not involving contact, proximity sensors provide similar information to contact sensors, and that tactile arrays have been used from for both imaging (e.g. tactile profiles: Hillis 1980) and contact (e.g. local curvature: Fearing 1989). Finally, internal sensors refer to systems which determine the internal state of the robot, such as joint position sensors (hall effect or potentiometer), torque (reverse EMF or instrumented gauges), orientation (compass and gyroscopes), and location (global positioning systems).

Imaging, contact, and internal sensors are generally applied to different tasks, or to different components of the same task. Image sensors provide an approximate representation of the surroundings, and are therefore useful in tasks which require coarse yet encompassing models, such as gross motion planning, navigation, and object discrimination. This is particularly true in cluttered environments composed to multiple occluded objects. In this thesis, however, we will generally consider isolated objects, whose geometry, position, and orientation lie within certain constraints. The intention is that by developing robust algorithms for partially modeled objects, we can incorporate these with coarse information produced from image sensing, and thereby reducing the reliance on high resolution, accurate models, complex algorithms, and expensive hardware. Contact sensors provide local information useful in tasks requiring partial guidance — particularly in manipulation, collision avoidance, and local object feature recognition. We will be primarily concerned in this thesis with the use of contact sensors to guide and select action. Internal sensors are primarily used to determine the states of the robot, but may, in some instances, abstract environmental features. For example, contact locations and forces between a robot and an object can be deduced from joint position and torque sensing (Eberman 1990).

Given a particular task, the type and arrangement of the physical sensors can effect the methods by which a task is executed. The objective is thus to determine the most efficient method to accomplish the task given the sensing capability of the robot. In the area of robot manipulation, visual sensing clearly provides information which would be difficult to acquire through contact sensing alone; however, the additional complexity and expense may limit its application. In the same way, tactile arrays provide information not available from force based contact sensors. For example, suppose a robot hand was equipped with a tactile array. Such a system could immediately provide an estimate of the contact location, radius, and orientation of a revolute solid from a single contact; whereas additional exploratory motion would be required using only instrinic contact sensing. The simplicity of force based contact sensors, however, in contrast to tactile arrays with their associated wires, connectors, electronics, and computation, generally compensates for the additional motion needed to accomplish a task. Furthermore, by considering minimal sensing systems, we develop fault tolerant capability, since these systems could continue to operate in the event more complex sensors are unavailable.

Not only do the sensors influence the tasks, but conversely, certain tasks imply particular sensors. For example, suppose the robot end effector was supplied with contact area sensors in addition to a contact force based system. Using this combination, the orientation of the cylinder could be approximated for a given radius. Other robot sensors have been constructed in a similar manner. For example, an interference sensor was used to detect the presence of an object within a gripper (Connell 1987, Ernst 1969), a mechanical probe to resolve proximal obstacles (Angle 1991), and a mechanical foot switch to determine impact events (Raibert 1989).

Definition. The *sensors* B is the set of all possible sensor output, and an element $b \in B$ is a particular sensor reading at a particular instant.

Example. In our example, we assume a joint position sensor provides the joint angle and contact sensors on opposing sides of the link provide binary contact information. Therefore we define

$$B = \{(b_1, b_2, b_3) | b_1 \in [0, \pi] \text{ and } b_2, b_3 \in \{0, 1\}\},$$

where b_1 is the joint angle, and b_2 and b_3 are the binary contact data, where b_2 is the sensor on the left or counterclockwise side of the link and b_3 is on the other.

2.2.3 Interpretation

Given a sensor reading $b \in B$, we wish to determine the state of the environment using this and any a-priori information. Therefore we define an *interpretation* $I = I(b) \subset S$ as a subset of the state space representing all states consistent with a single sensor reading. Hence the interpretation region I represents the knowledge gained through a sensor reading.

Definition. The **interpretation** $I = I(b) \subset S$ is the set of all states consistent with the sensor reading $b \in B$. Since the states are defined as a product space, the interpretation is defined in a similar way, that is $I = I^1 \times \cdots \times I^n$, where $I^i \subset S^i$.

Example. In the example, we assume the joint angle sensor returns the position of the link and that the contact sensors indicate the existence of a contact with an object. Thus there are only two possible situations: contact with the disk, $b_1 = (\theta, 1, 0)$ (by symmetry we need only consider contact with one side of the link), and no contact $b_2 = (\theta, 0, 0)$. Thus these cases span the range of possible input. In the first instance, $b_1 = (\theta, 1, 0)$, the interpretation is given by $I = I(b) = I^1(b) \times I^2(b)$, where

$$I^1 = \{(x,y)|x = \lambda c - Rs, y = \lambda s + Rc, \lambda \in [R(1-c)/s, L]\},$$

and

$$I^2 = \{\theta\},$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. The set I^1 is given by a line parallel to the link as shown in Figure 2-8.

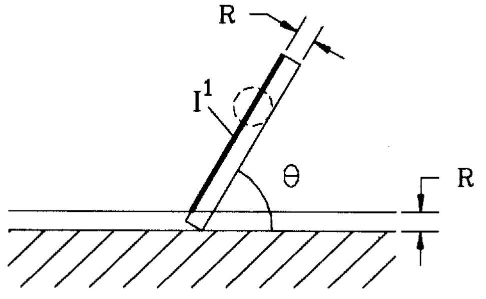


Figure 2-8. Contact with the manipulator constrains the disk to lie along a line parallel to the link and offset by the radius.

In the second case, $b_2 = (\theta, 0, 0)$,

$$\begin{aligned} I^1 \; = \; & \{(x,y)|cx + sy < 0, cx + sy > L, -sx + cy < -R, -sx + cy > R, \\ & x \in [R, w - R], y \in [R, h - R]\}, \end{aligned}$$

and

$$I^2 = \{\theta\},$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. The set I^1 is given by the shaded area as shown in Figure 2-9.

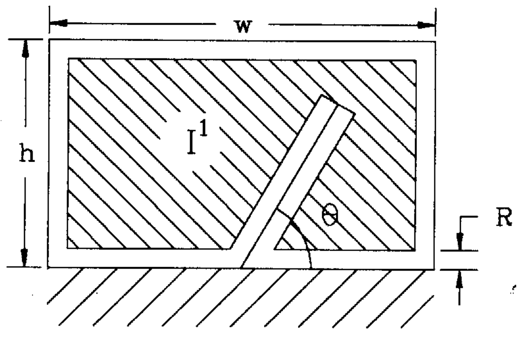


Figure 2-9. Non contact implies the disk lies within the workspace, yet outside of a region in the proximity of the link.

2.2.4 Knowledge

We define the *knowledge* K as a region in the state space S, representing the set of states currently known to the robot. For example, if the robot has perfect knowledge of the states, K would simply be a point in S, that is $K = \{s\}$ for some $s \in S$. It is more likely, however, that the robot will know the states of some entities to within some error, $K^i = \{s| \; ||s - s_o|| \leq e_i, s \in S^i\} \subset S^i$ for some i, while the states of others will be completely unknown; $K^j = S^j$ for some j, where $K = K^1 \times \cdots \times K^n$. In control theory, the knowledge, that is the current state estimate, is usually defined as a either single point or a point with some probablistic or indeterminate bounded error. However, here we define the current state estimate as a subset or even a subspace of the state space, in order to represent the partial knowledge gained from the sensory system. The objective is to then use this partial knowledge to design meaningful control action despite the limited information. As the robot proceeds in its task, we assume the knowledge K will become finer as the incoming sensory data is incorporated into the current model, and thus the resulting control action will become more delibrate.

Definition. The **knowledge** $K = K^1 \times \cdots \times K^n$ is the set of all states current known to the robot, where K^i is the set of states known about the i^{th} entity.

Suppose at times $t_0, \ldots, t_i$ we have acquired sensor data $b_0, \ldots, b_i$, where $b_i \in B$, and have constructed an estimate of the states $K_i = K_i^1 \times \cdots \times K_i^n$. Suppose at time t_{i+1} we obtain a new sensor reading b_{i+1}, and have formulated a new interpretation I_{i+1}. We wish to integrate this new information with the previous knowledge to construct a new estimate of the current states. Between times t_i and t_{i+1}, the robot will have executed some action a_i. (In the task-level control system a_i is depends only on the previous knowledge K_i. We will use this fact in both the theory and the implementation to simplify the evaluation of the new state estimate K_{i+1}^j as given below.) Thus the best estimate of the current states given only the previous knowledge K_i and the action a_i, is the forward projection $F_{a_i}(K_i)$ over the set K_i.

Definition. The **forward projection** $F_a(A) \subset S$ of an action a over a set $A \subset S$ is subset of the state space given by the set of points in A projected forward through some physical model.

Given the forward projection, $F_{a_i}(K_i^j) \subset S^j$, and the current interpretation $I_i^j \subset S^j$, we would expect some intersection. It is possible, however, not such intersection exists. This may happen, for example, because of some expected intrusion into the workspace — a distinct possibility when considering robots in human environments — or as a result of some unmodeled sensor or actuator error. Even small errors may lead to large unpredictable changes in the environment (Bradley 1990). Thus we will assume descrepancies between the forward projection and the current interpretation are a result of an unpredictable event, and will therefore ignore all previous information related to that entity and retain only the current interpretation, as illustrated in Figure 2-10. Hence we define the new knowledge $K_{i+1} = K^1_{i+1} \times \cdots \times K^n_{i+1}$, as

$$K^j_{i+1} = \begin{cases} F_{a_i}(K^j_i) \bigcap I^j_{i+1} & \text{if } F_{a_i}(K^j_i) \bigcap I^j_{i+1} \neq \emptyset \\ I^j_{i+1} & \text{otherwise.} \end{cases} \tag{2.1}$$

Realistically there may be multiple forward projections, which can be represented by collection of set of increasing size. For example, suppose a robot hand loses contact with an object which was in its grasp. The first estimate, or forward projection, for its position may be directly below the gripper. A second, more general, estimate may be the entire region under the robot, and a final, guess as to its location may be the entire workspace of the robot. Similarly we may define a family of forward projections, $F^0_{a_i}(K^j_i), \ldots, F^m_{a_i}(K^j_i)$, such that $F^k_{a_i}(K^j_i) \subset F^l_{a_i}(K^j_i)$ for $k < l$ and $F^m_{a_i}(K^j_i) = S^j$, as shown schematically in Figure 2-11. Thus we know at least one forward projection will intersect the current interpretation, and we base the new knowledge on the intersection of minimum size, that is

$$K^j_{i+1} = \min_k \{F^k_{a_i}(K^j_i) \bigcap I^j_{i+1}\}. \tag{2.2}$$

In addition to above equations, there are obviously various statistical and probabilistic methods for updating the current state estimate from the sensor data, previous knowledge, and system model. The Kalman filter is an example from traditional control theory. Thus an interesting area for future research may be the extension of the knowledge space estimates based on optimal probabilistic methods.

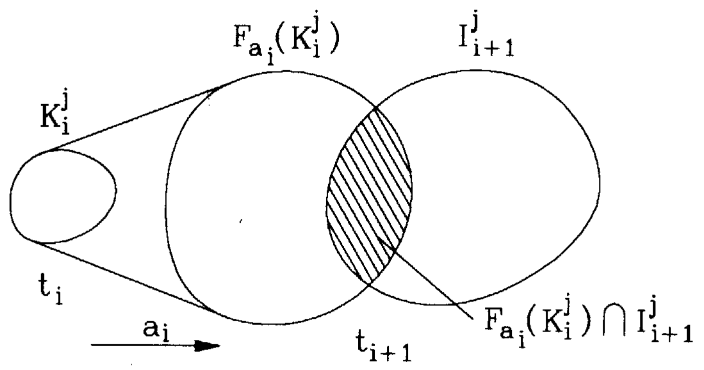


Figure 2-10. The current knowledge is given by the intersection between the forward projection of the previous knowledge and the current interpretation. If no such intersection exists we ignore all previous knowledge related to an entity and retain only the current interpretation.

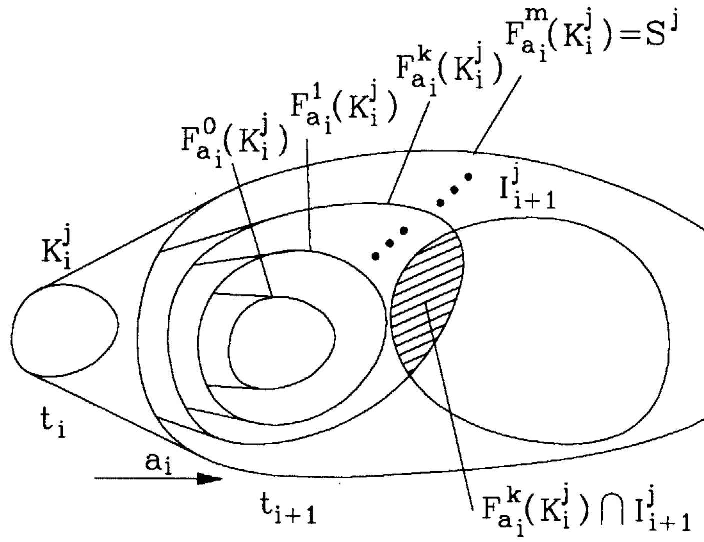


Figure 2-11. The current knowledge may be estimated as the minimum intersection between the family of forward projections and the current interpretation.

Example. Suppose the link is initially in contact with the disk, as shown in Figure 2-11. Thus the sensor data is $b_0 = (\theta, 1, 0)$, and the knowledge is equal to the current interpretation, that is $K_0 = K_0^1 \times K_0^2 = I_0 = I_0^1 \times I_0^2$, where

$$K_0^1 = \{(x,y) | x = \lambda c - Rs, y = \lambda s + Rc, \lambda \in [R(1-c)/s, L]\},$$

and

$$K_0^2 = \{\theta\},$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. Suppose between times t_0 and t_1, the link rotated by an amount $\Delta\theta$. If we neglect the dynamics, we may assume the disk remains either on the link or falls off, as shown in Figure 2-12. Thus the forward projection is given by $F_{a_1}(K_0) = F_{a_1}(K_0^1) \times F_{a_1}(K_0^2)$, where $F_{a_1}(K_0^1) = F_{\text{on}} \cup F_{\text{off}}$, where

$$F_{\text{on}} = \{(x,y) | x = \lambda c - Rs, y = \lambda s + Rc, \lambda \in [R(1-c)/s, L]\},$$

where $s = \sin(\theta + \Delta\theta)$ and $s = \cos(\theta + \Delta\theta)$,

$$\begin{aligned} F_{\text{off}} \quad = \quad & \{(x,y) | x = L\cos(\theta+\phi) - R\sin(\theta+\phi) + w/2, \\ & y = L\sin(\theta+\phi) + R\cos(\theta+\phi), \phi \in [0, \Delta\theta], \\ & x = R\cos(\psi) + L\cos(\theta), y = R\sin(\psi) + L\sin(\theta), \\ & \psi \in [\theta - \pi/2, \theta]\} \end{aligned}$$

and

$$F_{a_1}(K_0^2) = \{\theta + \Delta\theta\},$$

where a_1 is the link rotation. Suppose we obtain a new sensor reading $b_1 = (\theta, 1, 0)$, representing contact along the same side. The interpretation region I_1, is then the same as the original knowledge space K_0, but with θ replaced by $\theta + \Delta\theta$, and the intersection with the forward projection $F_{a_1}(K_0)$, yields the new knowledge $K_2^1 = F_{\text{on}}$ and $K_2^2 = \{\theta + \Delta\theta\}$, as shown in Figure 2-13. Alternatively, we may not have a contact, $b_1 = (\theta, 0, 0)$, in which case we assume the disk has fallen off the end. Intersecting the forward projection of the knowledge with the current interpretation implies the disk exists somewhere along the arc swept out by the endpoint of the link. Thus the new knowledge is given by $K_2^1 = F_{\text{off}}$ and $K_2^2 = \{\theta + \Delta\theta\}$, as illustrated in Figure 2-14. A final possibility is a contact with the opposing side, $b_1 = (\theta, 0, 1)$. This would unexpected and we must assume a result of some unkown event. Thus the current knowledge K_1 is given by the current interpretation $K_1 = I_1 = I_1^1 \times I_1^2$

$$I_1^1 = \{(x,y) | x = \lambda c + Rs, y = \lambda s - Rc, \lambda \in [R(1+c)/s, L]\},$$

and

$$I_1^2 = \{\theta + \Delta\theta\},$$

where $c = \cos(\theta + \Delta\theta)$ and $s = \sin(\theta + \Delta\theta)$, Figure 2-15.

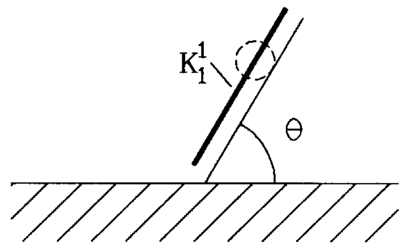


Figure 2-11. The initial knowledge space is given by a line parallel to the link and offset by the radius of the disk.

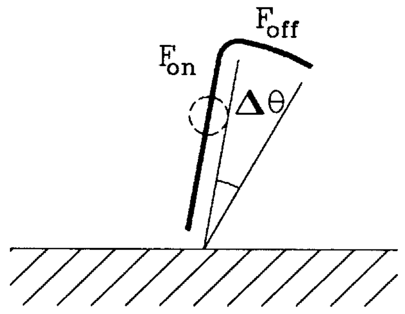


Figure 2-12. The forward projection of the knowledge space is given by a union of two sets, one composed of points along the link and the other of points along the arc.

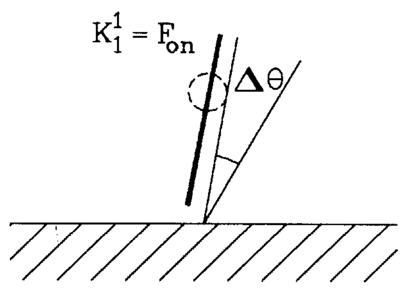


Figure 2-13. Contact implies the disk remains on the link. (Note if we assume a uniform pressure distribution under the disk, the disk will transverse a distance $R\Delta\theta$ along the link, independent of friction.)

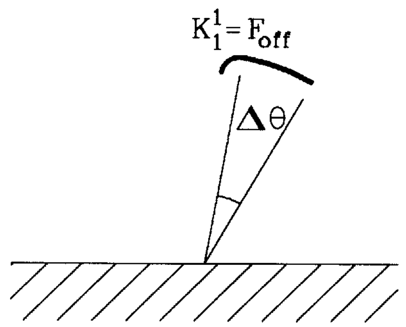


Figure 2-14. No contact implies the disk has fallen off the end of the link, and thus exists somewhere along the arc defined by the endpoint of the link swept through the angle $\Delta\theta$.

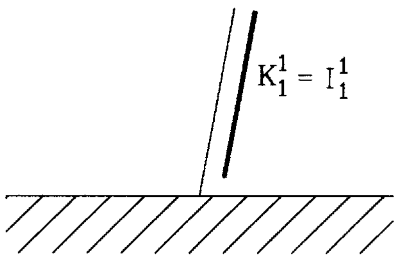


Figure 2-15. Contact along the opposing side is an unexpected event in that the interpretation does not intersect the forward projection. Therefore we must ignore all previous knowledge related to the entity and assume only the current interpretation, which in this case is a line on the oppose side.

2.2.5 Context

The *context* C is defined as an equivalence class of knowledge sets, that is a class of all elements $K \in \mathcal{P}(S)$ that are equivalent to some set $K_o \in \mathcal{P}(S)$ under some relation $R \subset \mathcal{P}(S) \times \mathcal{P}(S)$, where $\mathcal{P}(S)$ indicates the power set of the state space. In other words, the context is simply a collection of knowledge sets which satisfy some relation.

Definition. The **context** C is an equivalence class of knowledge sets, that is a class of all elements $K \in \mathcal{P}(S)$ that are equivalent to a set $K_o \in \mathcal{P}(S)$ under some relation $R \subset \mathcal{P}(S) \times \mathcal{P}(S)$ (where $\mathcal{P}(S)$ is the power set of the state space).

We say that the system is *within a particular context*, if the current knowledge is a subset of some element in context, that is if K is the current knowledge and C is a context, then the system is in C if $K \subset K_\alpha$ for some $K_\alpha \in C$.

Definition. The current knowledge is **in a context** C if $K \subset K_\alpha$ for some $K_\alpha \in C$.

Part of the motivation of the task-level control scheme is to decompose the control of the robot into a number of discrete tasks. It is desired that some tasks operate in coarse or more general contexts, and others in fine and specific situations. This is analogous to gross path planning and fine motion strategies (Lozano-Perez 1987). Therefore we will formalize the concept of one context being finer, or smaller, than another, and use this to say that one task operates in smaller context than another.

Definition. A context $C_j = \{K_b | b \in B\}$ is **smaller** or **finer** than another $C_i = \{K_a | a \in A\}$, denoted $C_i \succ C_j$, if

$$\bigcap_{b \in B} K_b \subset \bigcap_{a \in A} K_a$$

and for all $K_b \in C_j$, there exists $K_a \in C_i$ such that $K_b \subset K_a$.

We will see in the next sections, that the objective of the task-level control system is reduce the context, that is to move the system closer to the goal, which is the objective of the task, or to increase the knowledge of the states, which makes effective motion possible.

Example. Suppose that the position of the link is known, that is $K^2 = \{\theta\}$, and the disk lies within the workspace of the arm,

$$K^1 = \{(x, y)|x^2 + y^2 \leq L^2, y \geq R\}.$$

This situation can be described in general by a collection of knowledge sets, that is a context, which given as follows:

$$C_1 = \{K_\theta|K_\theta = \{(x, y, \theta)|x^2 + y^2 \leq L^2, y \geq R, R \leq -sx + cy \leq -R\}, \theta \in [0, \pi]\},$$

where $s = \sin(\theta)$ and $c = \sin(\theta)$, as illustrated in Figure 2-16. Suppose we have more refined knowledge, that is we know disk is on the link or to the right of the link. The context, in this case, may be described as

$$C_2 = \{K_\theta|K_\theta = \{(x, y, \theta)|x^2 + y^2 \leq L^2, y \geq R, -sx + cy \leq -R\}, \theta \in [0, \pi]\},$$

as shown schematically in Figure 2-17. Within this context we see that a simple motion, that is clockwise rotation, will produce a contact with the disk and solve the task. Thus the goal, that is a contact with the disk, can be described by a third context

$$C_3 = \{K_\theta|K_\theta = \{(x, y, \theta)|x = \lambda c \pm Rs, y = \lambda s \mp Rc, \lambda \in [R(1 \mp c)/s, L]\}, \theta \in [0, \pi]\},$$

as shown in Figure 2-18. The objective of the task-level control system is to move from a general context to the goal through a sequence of actions. In this case, given the initial context C_1 and the the goal C_3, we see that the contexts C_1, C_2, and C_3 form a natural sequence

$$C_1 \succ C_2 \succ C_3$$

representing both a reduction in state and an increase in knowledge.

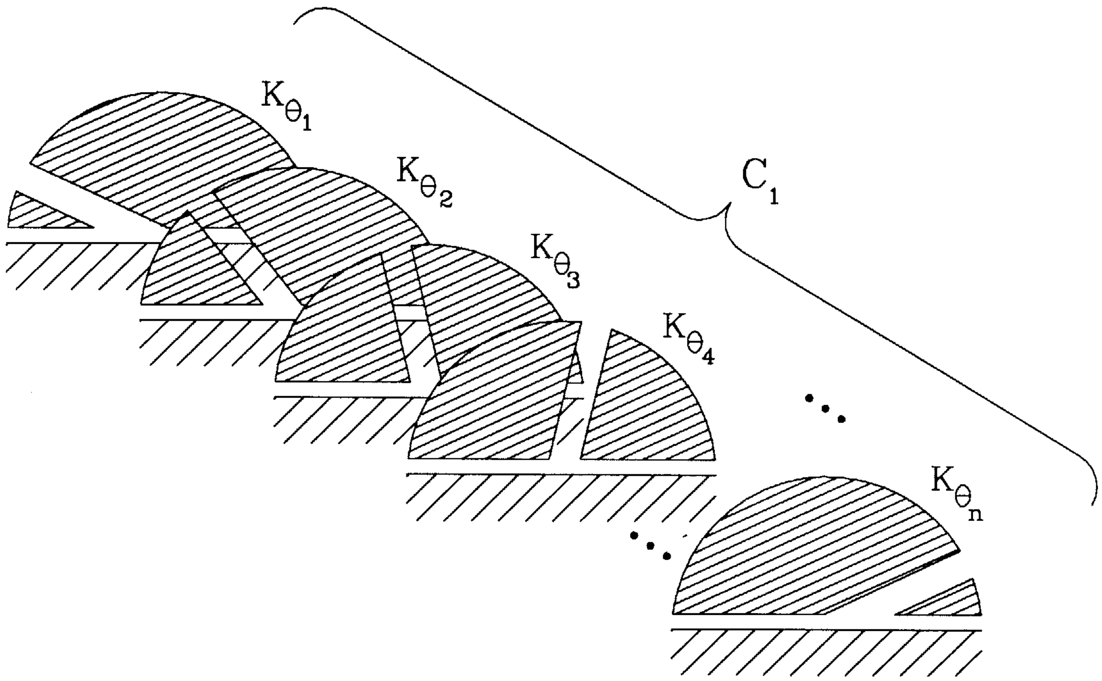


Figure 2-16. If we know the position of the link and the fact that the disk lies in the workspace of the arm, this information may be represented by a collection of knowledge sets.

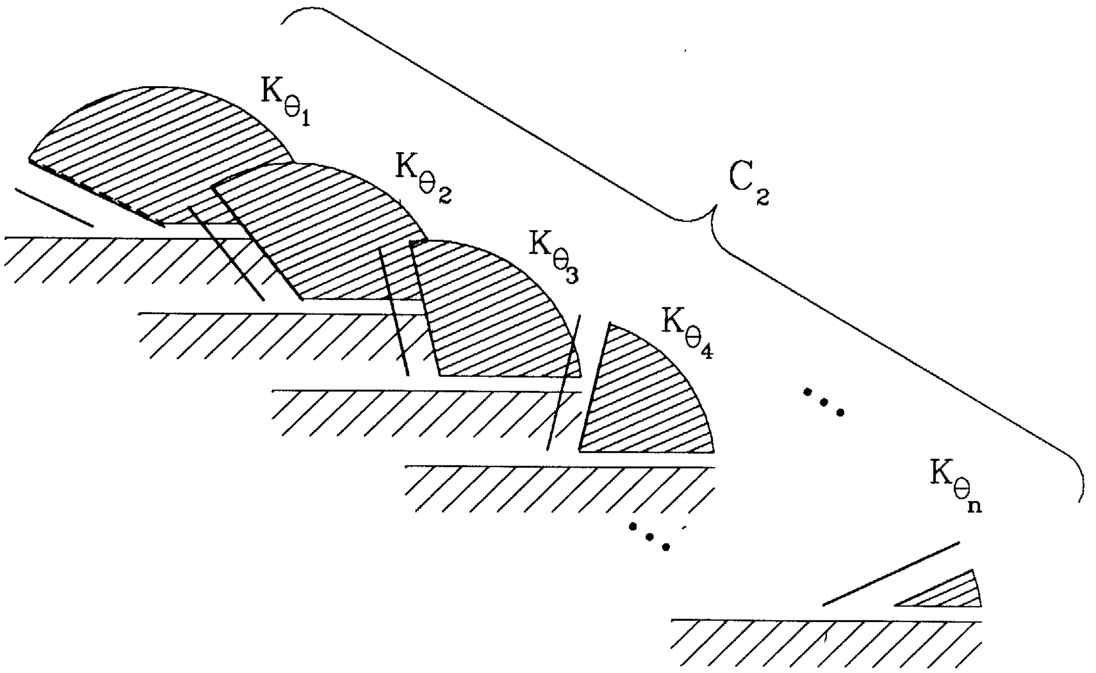


Figure 2-17. Knowing the disk lies to the right of the link represents an increase in knowledge from the former situation, and allows a simple motion, that is clockwise rotation, to contact the disk and solve the task.

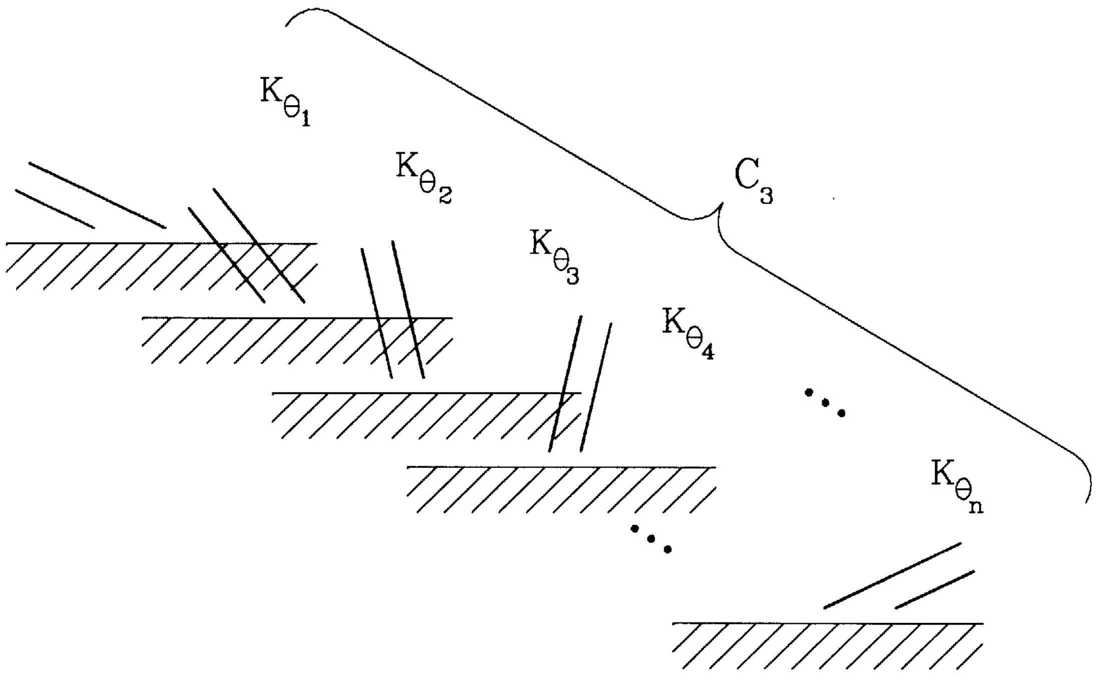


Figure 2-18. Contact with the link, on either the right or left side, represents the goal of the task.

2.3 Definition of the Goal

We define the *goal* in the same way as we defined the context. In other words, the goal is a desired relationship among objects in the environment which is known to the robot. Therefore the goal G is simply some context C.

Definition. The **goal** G is simply a specific context, that is the goal represents a desired relation among objects in the environment which is recognized by the robot.

Example. The goal is to contact the disk, thus $G = C_3$

2.4 Definition of the Action

Given a context C and a goal G, we wish to define some robot action a to achieve the goal. We define an *action* as either a sequence of tasks which operate on finer partitions of the context, or a simple trajectory in the joint space of the robot.

Definition. An **action** a is either a sequence of tasks $a = (t_1, \ldots, t_n)$ or a robot trajectory $a = (x_0, \ldots, x_n)$, where t_i are tasks and x_i are joint positions.

If the action a is a trajectory $a = (x_0, \ldots, x_n)$, we will describe its value $a(\lambda)$ on an interval $[\lambda_s, \lambda_e]$ as

$$a(\lambda) = (x_{i+1} - x_i)\lambda_c + x_i,$$

and

$$i = \left\lfloor n\frac{\lambda - \lambda_s}{\lambda_e - \lambda_s} \right\rfloor, \quad \lambda_c = n\frac{\lambda - \lambda_s}{\lambda_e - \lambda_s} - i,$$

where x_0 is the initial joint position.

Example. Let us define an action $a_1 = (t_1, t_2)$ composed of two tasks (where tasks will be defined in the next section), and actions $a_2 = (\pi)$ and $a_3 = (0)$ which represent simple sweeping motions of the arm, as shown in Figure 2-20. Together these motions span the workspace, and will thus guarantee a contact. We will use these actions in the next section to define the tasks.

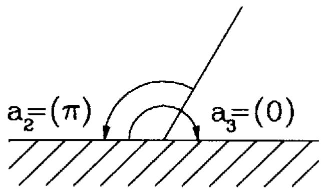


Figure 2-20. Trajectories specified by actions $a_2 = (\pi)$ and $a_3 = (0)$ first sweep the arm to one side and then the other to span the workspace and thus guarantee a contact with the disk.

Alternatively, we could define a slightly more complex trajectory $a_4 = (\pi, 0)$ which performs the same operation with a single motion, Figure 2-21.

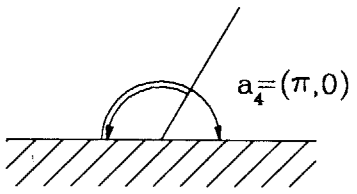


Figure 2-21. A simple trajectory, given by first rotating counterclockwise to one joint limit and then clockwise to the other, is guaranteed to produce a contact and solve the problem.

2.5 Definition of the Task

Intuitively we can think of a task as an action which only operates within a particular context to achieve a specific objective. More formally we say a task t is a set which contains a context C, goal G, and action a, that is $t = \{C, G, a\}$.

Definition. A **task** t on a state space S is a context C, goal G, action a, that is $t = \{C, G, a\}$.

For convenience, particularly in the implementation, we may partition the context C into a family of disjoint regions R_i, where

$$\bigcup_i R_i = C \text{ and } \bigcap_i R_i = \emptyset,$$

and associate an action a_i with each one, as illustrated in Figure 2-19.

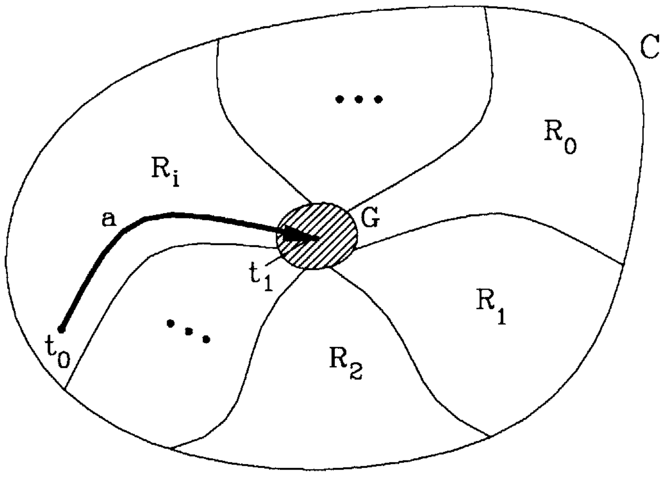


Figure 2-19. For convenience in analysis and implement the context C can be partitioned into distinct regions R_i, each one with an associate action a_i.

Example. Suppose the task is to touch the disk, and we know that the disk lies within the workspace of the arm. Then the task t is defined as $t = \{C_1, C_3, a\}$, where C_1 is the context specifying the disk is in the workspace, and C_3 is the goal specifying contact with the disk. We could solve this problem by defining an action $a = a_1 = (t_1, t_2)$, where t_1, t_2 are two task given by $t_1 = \{C_1, C_2, a_2\}$ and $t_2 = \{C_2, C_3, a_3\}$, where $a_2 = (\pi)$ and $a_3 = (0)$, as shown in Figure 2-22. Thus the task t_1 operates only within context C_1, has a goal C_2, and an associated action $a_2 = (\pi)$. The action simply sweeps the arm to one side which either establishes a contact, which is in context C_3, or the fact that the disk lies on the other side, that is context C_2. In either case the action a_2 will move the system from context C_1 to C_2. The second task t_2 operates within context C_2, has a goal C_3, and an associated action $a_3 = (0)$. Thus the disk is known to lie on the right and clockwise rotation will establish a contact. Alternatively we could have solve this problem more directly with a single action $a = a_4 = (\pi, 0)$, which sweeps the link through the workspace to produce a contact, as illustrated in Figure 2-23.

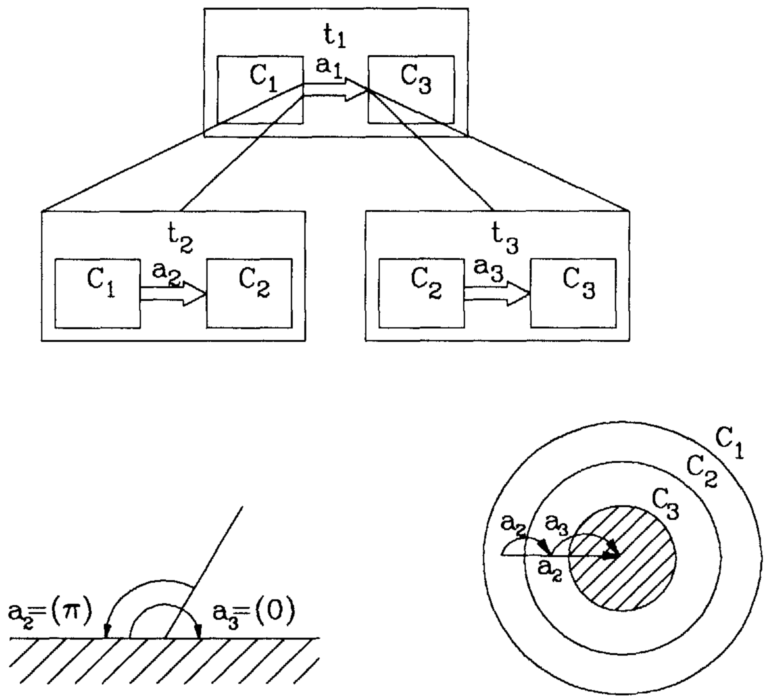


Figure 2-22. We can contact the disk by defining two tasks t_1, t_2. The first operates assuming the disk is within the workspace and moves the arm to the left. The second assumes the disk lies to the right thus rotates the arm clockwise to establish a contact.

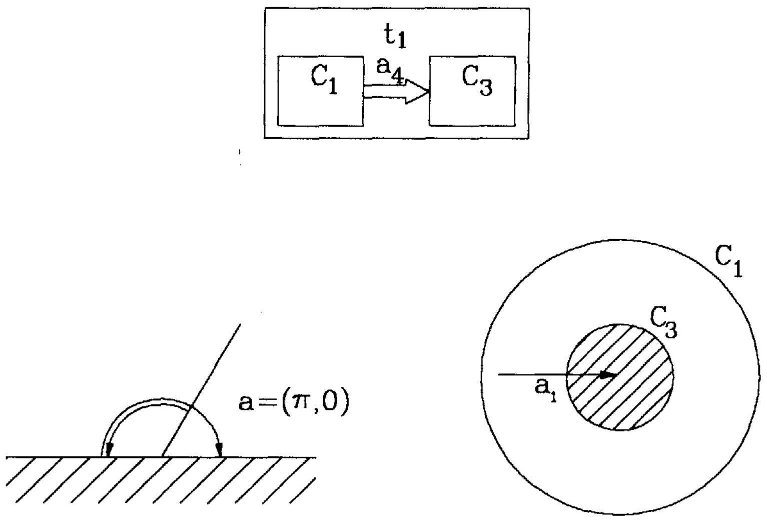


Figure 2-23. A single action given by first rotating counterclockwise to one joint limit and then clockwise to the other is guaranteed to produce a contact and accomplish the task.

2.6 Definition of the Task-Level Control System

We execute a task $t = \{C, G, a\}$ when the knowledge K is in the context C and not in the goal G. We say a task is *executed* when a trajectory is sent to the robot actuators if the action $a = (x_1, \ldots, x_n)$ is a trajectory, or when a set of tasks $t_1, \ldots, t_n$ is enabled when $a = (t_1, \ldots, t_n)$ is a task sequence. Also, we say that a task t is *enabled* when the control system checks whether the current knowledge is in the context, otherwise the task is ignored.

Thus the execution of a task depends on the current knowledge K, which, in turn depends, on both the previous knowledge and the sensor data. Since the sensor data is updated periodically (we use the discrete time model in this thesis), we have essentially defined a control system.

Specifically we define the *task-level control system* as a set of tasks $T = \{t_1, \ldots, t_n\}$ and the sequence of elements shown in Figure 2-24. Beginning from the left, we assume that at time t_i the system states assume some value $s_i \in S$. The states s_i then generate some sensor value $b_i \in B$, as a function of the measurement model, where the measurement model is a function $h : S \rightarrow B$ where $s_i \mapsto b_i$. The sensor data b_i and the previous knowledge K_{i-1} are then combined using a state estimate function to compute a new knowledge space K_i. The state estimate function can be though of as a generalized Kalman filter, and can be denoted as a function $f : B \times \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ which maps sensor data b_i and previous knowledge K_{i-1} to new knowledge K_i, that is $(b_i, K_{i-1}) \mapsto K_i$. Equations 2.1 and 2.2 from the previous section give examples of state estimate functions. We should note that in practice, we employ separate estimate functions f_t for each task $t_j \in T$. This makes sense, since environmental states which are important for one task may not be relevant to the others.

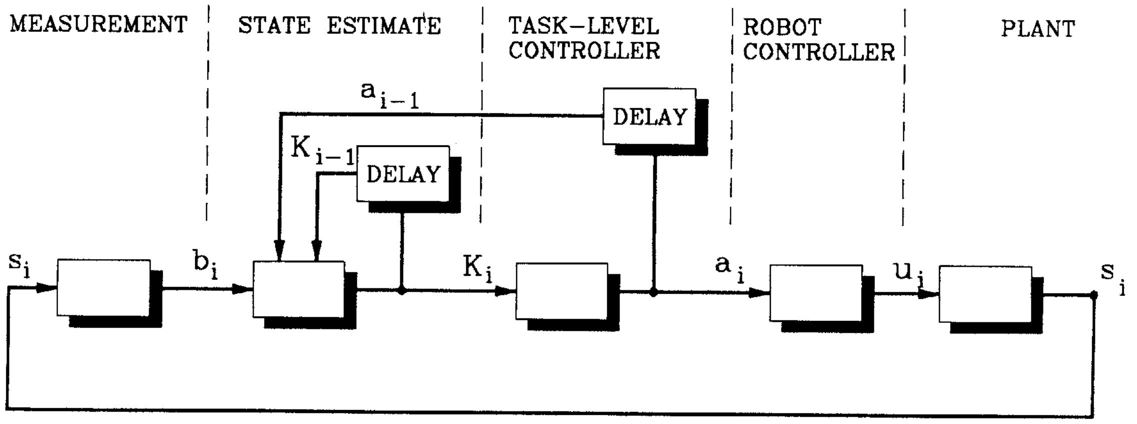


Figure 2-24. Sensor data and the previous state estimate are combined to form a current estimate of the system states. This state estimate, or ***knowledge space*** as defined in this thesis, is then issued to the task-level control system. The controller then determines whether the knowledge space resides within the context of all the enabled tasks. For all tasks for which the knowledge space is in the context and not in the goal, the task is executed.

The knowledge space K_i, representing the current model of the system states, is entered into the ***task-level controller*** T. If this estimate is within the context of a particular task and not within the goal, then the task is executed. An executing task may have an associated action, which is defined as a task sequence. In this case, every task in the sequence is enabled; which simply means the control system will check whether these tasks should execute. This procedure is represented schematically in Figure 2-25. In the figure the highlighted boxes represent enabled tasks and the shaded boxes executing tasks. As illustrated in Figure 2-26, a sudden change in the environment may cause a rearrangement in the pattern of execution at the lower level, yet not effect the overall task execution at the higher level. Again, the advantage of this structure is to allow the robot to response immediately to changes in the environment, and to generate corrective action to accomplish the tasks. Also the degree of change in the task execution is coupled with the degree of change in the perceived environment.

Finally, the trajectory a_i generated by the task-level controller is then delivered to the robot control system, which then executes the appropriate displacement d_i. Given displacement d_i and the previous state s_{i-1}, the physics of the system determines the subsequent state s_i, and the loop is complete.

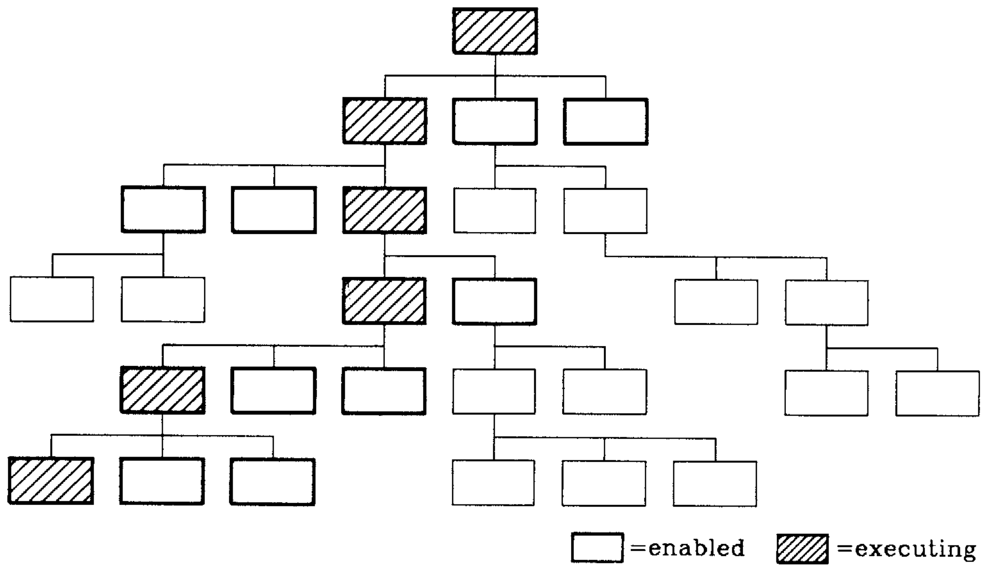


Figure 2-25. If the system state estimate, or knowledge space, is within the context of a particular task and not within the goal, the task is executed. If the action associated with an executing task is a task sequence, every task in the sequence is enable, which simply means the control system checks whether these tasks should be executed.

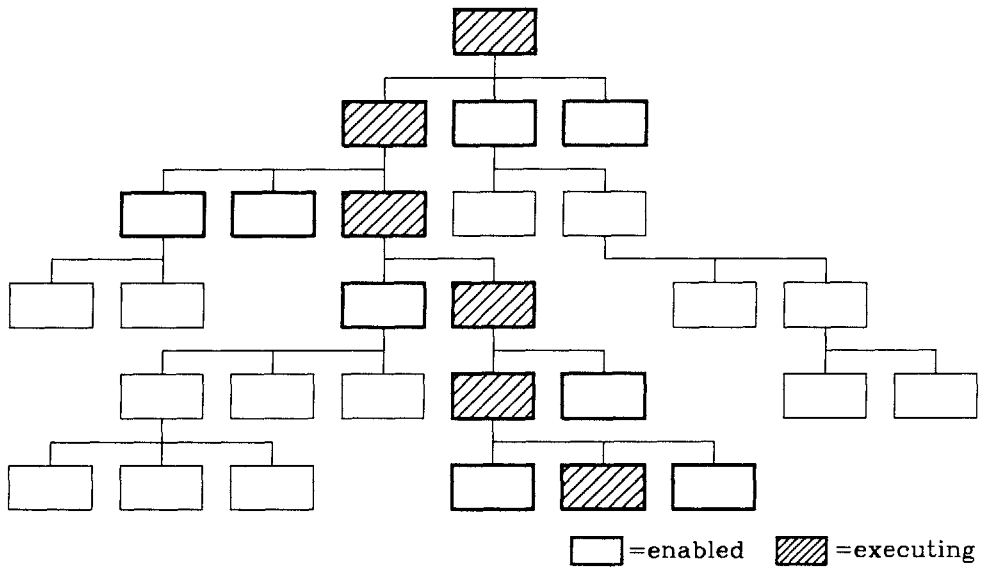


Figure 2-25. A sudden change in the environment causes a rearrangement in the pattern of execution at the lower level, but does not effect the overall task execution at the higher level. Thus the degree of change in the task execution is coupled with the degree of change in the perceived environment.

Example. Let us define the collection of tasks $T = \{t_1, t_2, t_3\}$ as illustrated in Figure 2-23, where $t_1 = \{C_1, C_3, a_1\}$, $t_2 = \{C_1, C_2, a_2\}$, and $t_3 = \{C_2, C_3, a_3\}$, and implement the control loop as given in Figure 2-24. Figure 2-26 shows the progress of events for a particular example. The link is initially unaware of the location of the disk, except that it is known to lie within the workspace of the arm. In 1, the knowledge space K_1 is given by all positions disk positions not intersecting the link. Since K_1 is in context C_1 and not in C_2, task t_1 and t_2 are active and the arm executed trajectory $a_1 = (\pi)$. In 2, the knowledge space is refined but is still not contained in any context C_i $i > 1$, thus a_1 continues to execute. In 3, a contact is established, hence K_3 is in C_1, C_2, and C_3, and task t_1 is terminated. In 4, the disk moves or is moved suddenly and unexpectedly to a new position. The new sensor data is inconsistent with the previous knowledge, thus the old information is discarded and the new knowledge space is given by K_4 as shown in the figure. The arm executes action $a_1 = (\pi)$, until, in 6, it is established the disk is to the right and task t_3 is active. Task t_3 executes action $a_3 = (0)$ until a contact is finally established in 7.

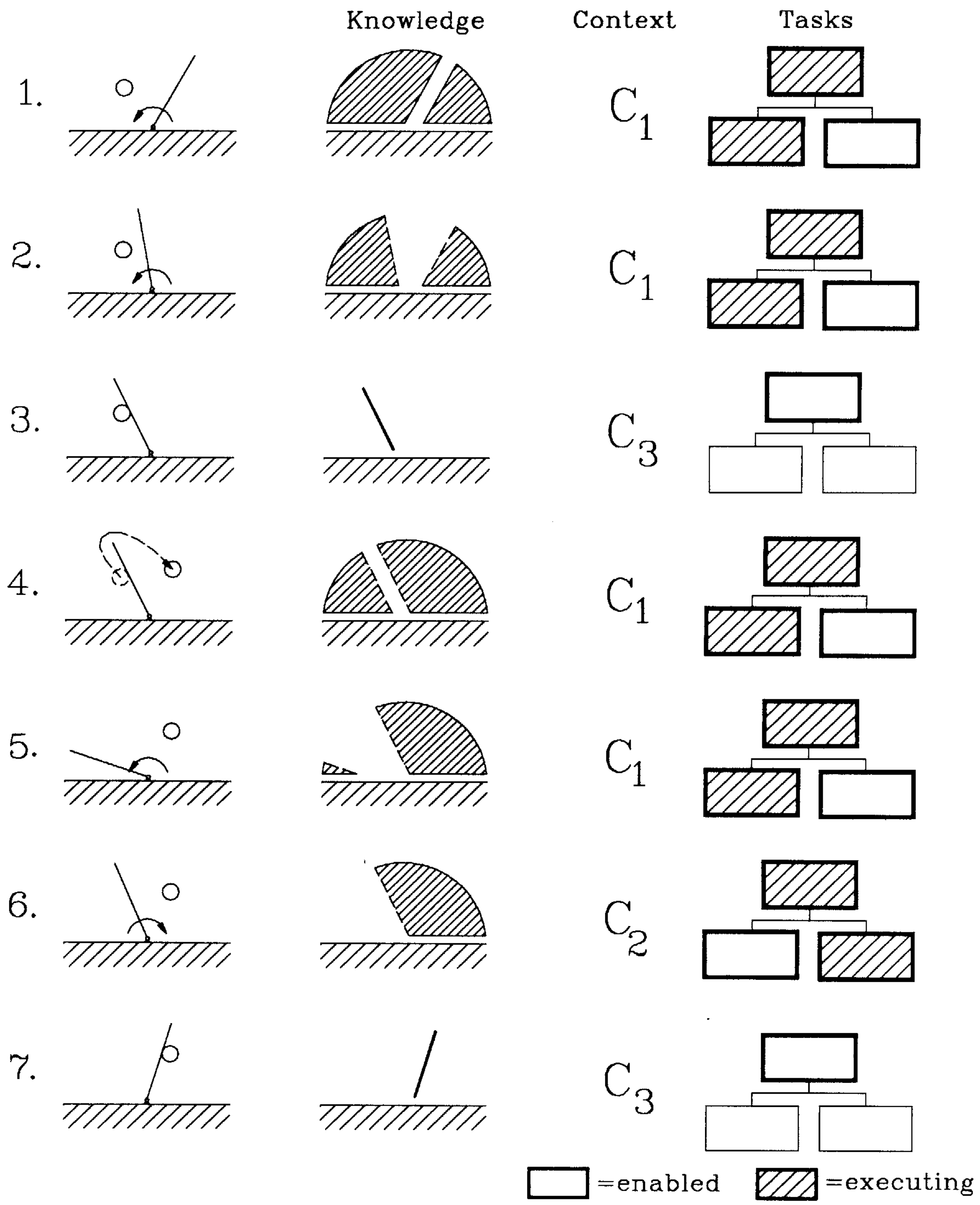


Figure 2-26. The task-level control system reacts immediately to perceived changes in the environment and executes an action associated with the task appropriate to the current situation.

2.7 Stability and Convergence

One of the major issues in any control problem is provable stability and convergence. In other words, is it possible to guarantee that the system will actually solve the task and do so uniformily. Part of the difficulty with behavior based systems is the inability, in many cases, to guarantee the system would solve the task, and avoid chattering instabilities and limit cycles. A visualization tool discussed in the next section was designed to study stability and convergence issues for behavior based systems, as well as offer guidelines in the formulation of system behavior. However, part of the motivation for task-level control system was to intrisically provide stability and convergence, as well as allow various design tools based on the parial representations which describe the contexts. Proofs of stability and convergence are possible because the robot actions for the task-level control system are initiated in response to explicit partial representations of the environment rather than particular sensory events and memory states.

We have stated that the task-level control system operates in a semi-structured environment, and in our assumption of a semi-structured environment we have allowed the states to change arbitrarily and at random times. Clearly we cannot guarantee a stable control system with this assumption. However, we would like to demonstrate that in an otherwise static environment, the task-level control system remains stable and converges to the goal in a finite time. Since the control system is composed of a collection of discrete tasks, each one operating within a particular context and executing a particular action, we would like to develop some criteria on the tasks, such that they cooperate together to accomplish the overall objective. Therefore in this section we will develop the ideas of *stability* and *convergence* for a task-level system and offer criteria on the tasks which guarantee these qualities. However, before the issues of stability and convergence can be addressed, we must establish criteria for a consistent control system, that is a control system which generates one and only one trajectory for any given input.

2.7.1 Comment on Stability and Convergence of Behavior Based Systems

A behavior based system may be described as a control system which generates a robot motion based on the sensor values and the internal memory state. Since the states of the system, that is the states of both the robot and the environment, determine the values of the sensors, within the limits of noise, then given an internal memory state we can uniquely determine the action of the robot. Therefore we can describe these effects as a vector or trajectory field on a configuration space formed as a product of the state space and the memory states. We will call such a map a *behavior diagram* (Brock 1991) since it completely describes the behavior of the system in a given environment. Although similar to potential or gradient fields used previously (Payton

1990, Arkin 1990), these diagrams extend to include both trajectories and memory state transitions. An important aspect of these maps is that a sudden environmental change can be represented as a sudden shift in position within the behavior diagram.

Example. As an example, we will consider a mobile robot (Mataric 1989, 90) which employs the subsumption architecture through use of the behavioral language (Brooks 1990). The cylindrical mobile robot translates and rotates independently and has twelve sonar range sensors spaced evenly about its circumference, Figure 2-27. Significant regions about the robot are described by thresholding sensor values or comparing their ratios. By considering both the robot and these critical regions we describe an extended object, Figure 2-28. The position of this extended object within a particular environment describes an extended configuration space. This is similar to the extended configuration space proposed by (Villarreal 1991), which incorporates the effects of compliance during assembly. The present definition, however, also includes aspects of sensing and control as well as compliance. In Figure 2-29 we show a projection of the extended configuration space for the robot encountering a wall. The vertical axis represents its displacement from the wall and horizontal axis its rotation. The control algorithm, assigns a displacement vector or state transition for each critical region. This is reflected in the behavior diagram, Figure 2-30. The behavior of the robot is thus described as the motion of a point through this diagram and a sudden environmental change as a shift in this point.
Figure 2-30 shows the effects of the lower level "stroll" behavior. The length of the vectors define the translational or rotation displacements and the state transition regions define shifts to alternate diagrams. In this case, positions in the RECENT STOP region cause transitions to the right diagram. Many aspects of the system behavior become readily apparent, such as a collision region ①, limit cycle ②, and a stable region, line ③.
These diagrams also offer implications for behavioral design. The actions, that is the displacement vectors or trajectories, must be compatible between adjoining regions to avoid chattering. Specifically, the components of the command pvectors along the boundary normal in neighboring regions must be in the same direction. Design may also be possible by examining the extended configuration space and assigning vectors or trajectories to the disjoint regions.

In the subsumption architecture, state transitions are represented by triggered monostables, which are variables that when set remain true for a specific length of time. Since these variables represent knowledge states gained through sensing, their persistence for a fixed length of time may not be desirable, since newly acquired sensor values may negate their validity. The most recent sensor values should be compared with the previous knowledge state to yield a consistent representation.

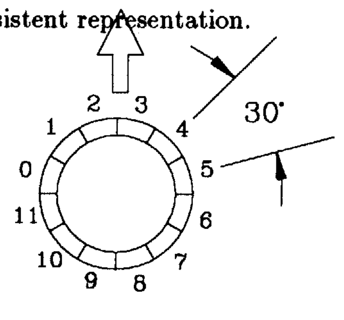


Figure 2-27. The cylindrical mobile robot translates and rotates independently and has twelve sonar range sensors space evenly about its circumference.

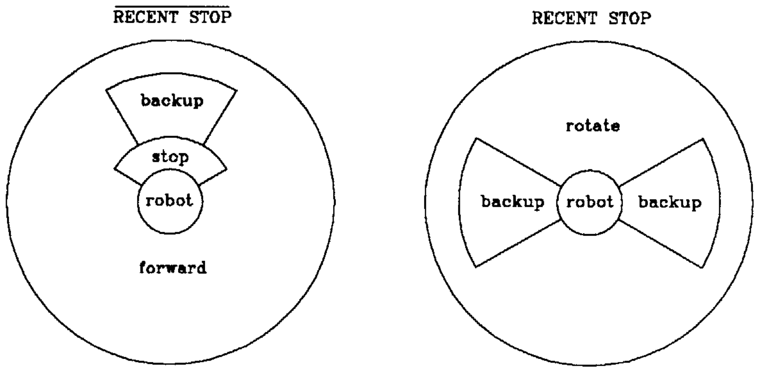


Figure 2-28. The robot and critical regions surrounding it represent an extended object which can be used to construct a configuration space map.

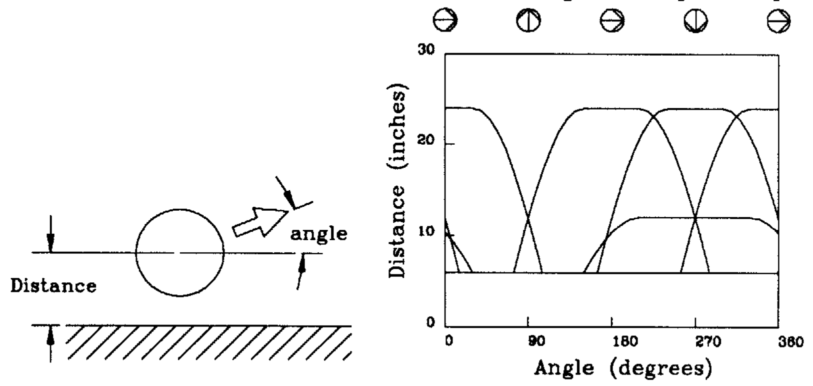


Figure 2-29. The extended configuration space is shown for the robot encountering a wall. The space is parameterized by the distance and inclination from the wall.

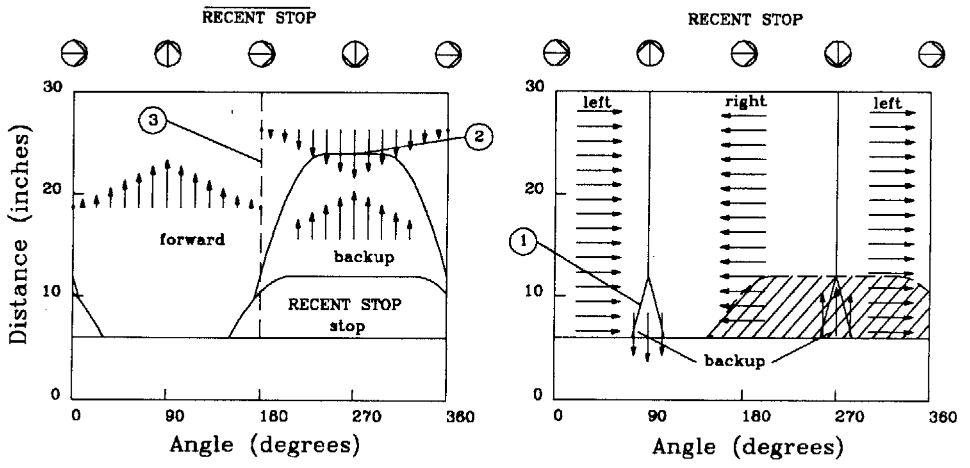


Figure 2-30. The diagrams illustrate the operation of the "stroll" behavior. The length of the vectors define the length of the displacement or angle relative to the wall. In the left diagram, positions in the RECENT STOP region cause a transition to the right diagram. A point remains in the right diagram for a fixed period of time after leaving the shaded region. This may cause aberrant behavior for moving obstacles, such as the case for region ① which causes the robot to collide with the wall. Also incompatible vectors at the interface of the boundaries cause chattering instabilities ②. On the other hand, line ③ represents a stable region, in which the robot neither moves toward nor away from the wall.

Example As part of a grasping behavior, the Salisbury Robot Hand is required to align with an object given only the binary contact information available from a palm sensor (see Figure 5-4), as shown in Figure 2-31. Based on this information, Figure 2-32 shows the behavior diagrams and the experimental results for three different robot behavior. In the first case a poorly designed motion fails to converge to the goal; in the second, convergence is achieved but the incompatiable across the switching surface causes the system to chatter; and in the last case, a correctly design motion causes the system to smoothly converge to the goal. The incompatiability of the vector fields across switching surfaces can be easily observed in these diagrams.

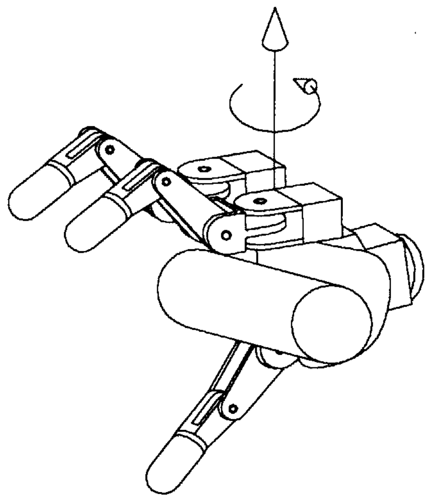


Figure 2-31. The Salisbury Robot Hand is required to align with an object given only binary contact information Available from a palm sensor (see Figure 5-4), as shown in Figure 2-31.

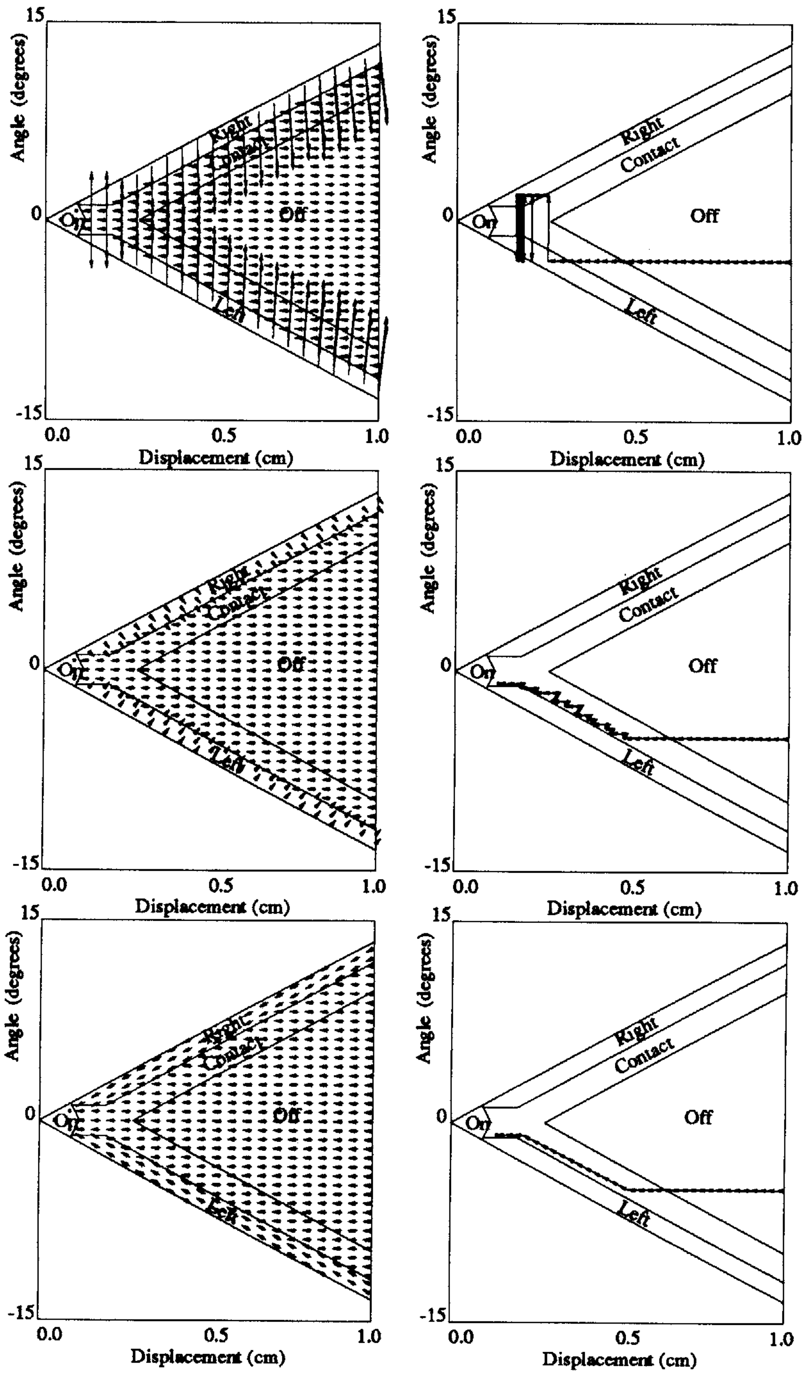


Figure 2-32 The behavior diagram combines the extended configuration space and associated actions. Smooth convergence, chattering instability and limit cycling are demonstrated. The horizontal axis represents the distance of the palm to side of cylinder and vertical axis the angle between them. The plots to the right are result from experimental trials.

2.7.2 Consistent Task-Level Control

Definition. A task-level controller T is **consistent** if it produces no output, or one and only one parameterized action $a(t)$, for all input $K \subset S$.

The task-level controller consists of a set of tasks $T = \{t_1, \ldots, t_n\}$, where each task t_i is given by a context, action, and goal, that is $t_i = \{C_i, G_i, a_i\}$. The action a_i is defined as a trajectory $a_i = (x_1, \ldots, x_n)$ or a task sequence $a_i = (t_{i1}, \ldots, t_{in})$. If the action a_i is a task sequence, we may consider the task t_i to be a node and the subtasks, $t_{i1}, \ldots, t_{in}$, a set of branches, as shown Figure 2-33. More formally, we can describe t_i as a vertex and the subtasks as a set of directed edges $(t_i, t_{i1}), \ldots (t_i, t_{in})$ in a directed graph, see appendix A section 1 for definitions and terminology of graph theory. Continuing in this way, we may describe the entire control system as a directed graph $G_T = (T, E)$, where the set of tasks T are represented by vertices, and $E = \{(t_{i_1}, t_{j_1}), \ldots, (t_{i_n}, t_{j_n})\}$ by edges. By describing the task-level controller in this way, one criteria that is clearly necessary is that G_T should be an acyclic graph, which is to say G_T is a *forest* or a graph of *trees*, as in the example shown in Figure 2-34. Since G_T is a directed graph it will contain a set of *roots* $T_r \subset T$. We define a root as a task $t_r \in T$ such that there exist no $(t, t_r) \in E$. Let us define a subgraph G_{t_r} of G_T given by the set tasks connected which are connected to the root t_r, as shown in Figure 2-35. In order to avoid conflict in the task-level controller, we require one and only one root to be enabled at a time. This implies that the robot has only one task to accomplish at any one time. Thus the control system may be considered as a single tree G_{t_r} with a root given by the enabled task t_r.

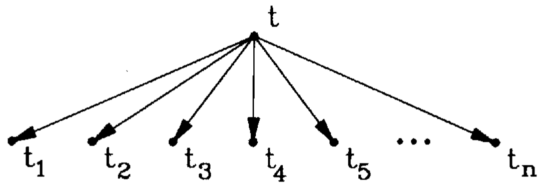


Figure 2-33. If the action is a task sequence, the task t may be considered a node and the subtasks, $t_1, \ldots, t_n$, branches.

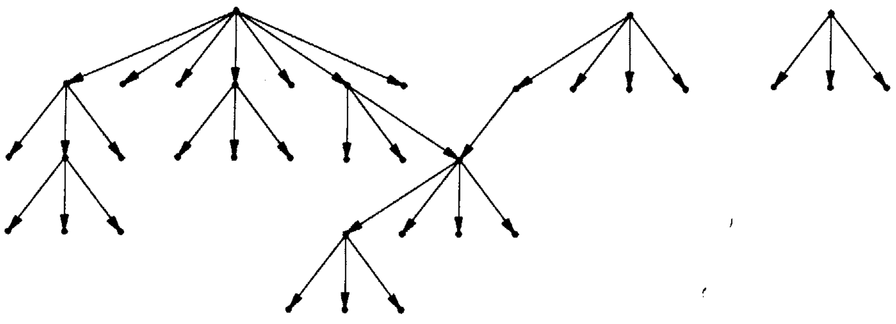


Figure 2-34. In order to avoid obvious loops the task structure should be acyclic

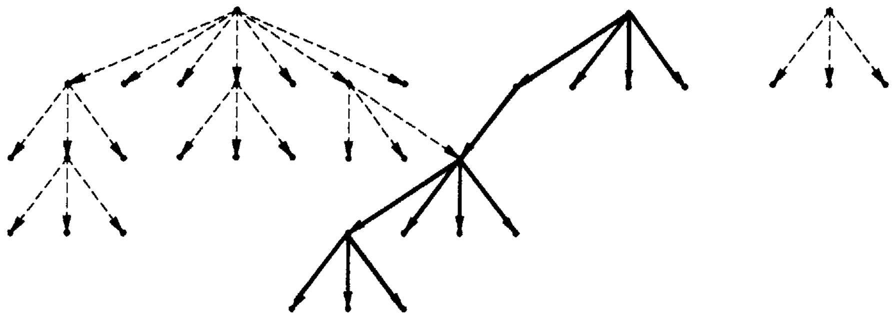


Figure 2-35. The set of tasks which are enabled at any one time form a tree.

Lemma. Suppose $C_1 \succ \cdots \succ C_n$ is a sequence of contexts, suppose $K \subset S$ in C_j and not in C_{j+1} for some j, then K is in C_i and not in C_k for all $i < j < k$.

PROOF. First, by induction, given K in C_j, assume K in C_i for $i < j$, and show K in C_{i-1}. Since K in C_j, there exists some $K_b \in C_j$ such that $K \subset K_b$. Since $C_{j-1} \succ C_j$, there exists some K_b such that $K_b \subset K_a$, hence $K \subset K_b$ and K in C_{j-1}.
Second, again by induction, given K is not in C_{j+1}, assume K is not in C_k for $k > j+1$, and show K is not in C_{k+1}. By contradiction, assume $K \in C_{k+1}$. This implies there exists some $K_b \in C_{k+1}$ such that $K \subset K_b$. Since $C_k \succ C_{k+1}$, there exists some $K_a \in C_k$ such that $K_b \subset K_a$. Hence $K \subset K_a \in C_k$ a contradiction □.

Definition. A sequence of tasks $t_1, \ldots, t_n$ is **consistent** if no task executes or one and only one task executes.

Lemma. (2.1) A sequence of tasks $t_1, \ldots, t_n$, where $t_i = \{C_i, G_i, a_i\}$ such that $C_{i+1} = G_i$ and $C_i \succ G_i$ is consistent for all $K \subset S$.

PROOF. If K is not in C_1 or K is in G_n, no task executes. Therefore assume K is in C_1 and not in G_n. Thus the sets sets $C_1, \ldots, C_n, C_{n+1} = G_n$, form a sequence of contexts $C_1 \succ \cdots \succ C_{n+1}$. Let $\mathcal{C} = \{C_{i_1}, \ldots, C_{i_m}\}$ be a set of contexts of the sequence for which K is not in C_{i_j}. The set $\mathcal{C}$ contains at least one element since K is not in G_n, and not all elements, since K is in C_1. Let $C_m \in \mathcal{C}$ be the largest context in the set. Therefore K is in C_{m-1} and not in C_m, and thus the task t_{m-1} executes. By the previous lemma K is in C_i and not in $C_{j'}$ for all $i < m-1$ and $j > m$, so that t_{m-1} is the only task executed. □

Definition. A task-level controller has a **regular structure** if

The graph of T, G_T, is acyclic.

G_T has one and only one root t_r which is enabled.

For each task $t = \{C, G, a\}$ in G_{t_r}, the action a is a trajectory or a task sequence $a = (t_1, \ldots, t_n)$, such that $C = C_1$, $G = G_n$, $C_i \succ G_i$, and $G_i = C_{i+1}$.

Proposition. If a task-level controller T has a regular structure it is consistent.

PROOF. Let $K \subset S$, and $t_r = \{C_r, G_r, a_r\}$ be the root task. If K is not in C_r or is in G_r, the system produces no output. Therefore assume K is in C_r and not in G_r, and thus t_r executes. If a_r is a trajectory the proposition is proved, so assume a_r is a task sequence. By induction, we are given t_r executes and that the action $a_r = \{t_{r1}, \ldots, t_{rn}\}$ is a consistent task sequence. Thus only one task t_{ri} executes. Assume that t_k is an executing task, such that $t_r, t_1, \ldots, t_k$ is a path of executing tasks in the graph G_{t_r}. Again if a_k is a trajectory, the inductive step is proved; therefore assume $a_k = (t_{k1}, \ldots, t_{kn})$ is a task sequence. Since a_k is a consistent, only one task executes and the tasks $t_r, t_1, \ldots, t_k, t_{k+1}$ is a path of executing tasks in G_{t_r}. Finally, since the task-level controller is finite, only one trajectory executes. □

2.7.3 Stable Task-Level Control

The task-level control system is defined as a set of tasks $T = \{t_1, \ldots, t_n\}$, where each task $t = \{C, G, a\}$ has a context C, goal G, and action a. The intuitive notion of *stability* for such a system is that the effect of the action a contains the system within its context. In other words, once the system has arrived in a particular context, the actions are such to keep it there. This is somewhat analogous to the bounded-input bounded-output (BIBO) stability criteria for linear time-invariant (LTI) systems, in that an action (in this case a bounded input $x[n]$ where $|x[n]| < B$ for all n) produces a bounded output

$$|y[n]| \leq B \sum_{k=-\infty}^{+\infty} |h[k]| \text{ forall } n,$$

if the unit response is absolutely summable,

$$\sum_{k=-\infty}^{+\infty} |h[k]| < \infty$$

(Oppenheim 1983). In other words, the effect of the action mantains the system within a certain bound for all time, which in this case is defined as the context C.

Definition. A task $t = \{C, G, a\}$ is **stable** if the effect of the action a maintains the system within the context C.

Definition. A task-level control system T is **stable** if all tasks $t \in T$ are stable.

Let us first define the *forward task projection*, that is a task $t = \{C, G, a\}$ and a knowledge space K in C, the forward task projection describes the effect of the knowledge space K as the result of the action a.

Definition. Let $t = \{C, G, a\}$ be a task, where $a(\lambda)$ is trajectory parameterized on an interval $[\lambda_s, \lambda_e]$, and assume a discretization $\lambda_s = \lambda_1, \ldots, \lambda_n = \lambda_e$. Then given a knowledge space K in C, **forward task projection** is a collection of knowledge spaces $F_t(K) = \{K_0, \ldots, K_n\}$, where $K_{i+1} = F_a(\lambda_i)(K_i \cap K_g)$, for all $K_g \in G$.

For a particular task to be stable, the elements of the forward task projection must be within the context. In other words, given a task $t = \{C, G, a\}$, the effect of the action a maintains the system within within the context C. For the task-level control system to be stable, every element in $F_t(K)$ for K in C, must be in C.

Proposition. A task-level control system T is stable, if it is consistent and the forward task projection for every task t containing a trajectory is smaller than its context, that is for $t = \{C, G, a(\lambda)\}$, $C \succ F_t(K)$, for all K in C.

> **PROOF.** Since the T is stable, we know that either T produces no output, or one and only one trajectory executes. If it produces no output T is stable by definition. Therefore suppose T receives some input K, then one and only one trajectory $a(\lambda)$ executes. Since the forward task projection $C \succ F_t(K) = \{K_0, \ldots, K_n\}$, for all $K_i \in F_t(K)$, $K_i \subset K_\alpha$ for some $K_\alpha \in C$. Thus for all input K the effect of the executing task $t = \{C, G, a\} \in T$ mantains the system within the context C.

2.7.4 Convergent Task-Level Control

Intuitively, we say a task-level controller converges if it accomplishes the task in finite time.

Definition. A task $t = \{C, G, a\}$ **converges** if it is stable and if the effect of the action a moves the system into the goal G.

Definition. A task-level control system T **converges** if it is stable and all the tasks $t \in T$ converge.

Proposition. Let $t = \{C, G, a\}$ be a stable task where the action a is a trajectory $a(t)$. The task t will converge if the exists an element K_i in the forward task projection $F_t(K)$ such that K_i in G.

PROOF. For all K in C not in G, let $F_t(K) = \{K_0, \ldots, K_n\}$ be the forward task projection. Since t is stable $K_j \in F_t(K)$ is in C, for $j = 0, \ldots, n$. If K_j is in G, for $j < i$, the proposition if proved. If K_j not in G, the task continues to execute and for $j = i$, K_j is not in G by hypothesis, thus the task converges.

Proposition. A task-level control system T converges if it has a regular structure and all tasks whose actions are trajectories converge.

PROOF. By induction. Given that tasks whose actions are trajectories converge, assume a task $t = \{C, G, a\}$, where $a = (t_1, \ldots, t_n)$, is such that all tasks $t_i = \{C_i, G_i, a_i\}$ converge. Show t converges. Let K be in C not in G. Since T is regular, $C = C_1 \succ \cdots \succ G_n = G$, and by lemma 2.1, only one task t_i executes. Since t_i converges by hypothesis, then there exists a K' in $F_t(K)$, such that K' is in C_{i+1}. If K' is in C_{n+1}, then the proposition is proved. Assuming K' is not in C_{n+1}, continue the above procedure. Since n is finite there will exists some K' such that K' is in $C_{n+1} = G_n$. □

Chapter 3

Architecture of the Task-Level Control System

3.1 Introduction

In this chapter we will describe the computational architecture of the task-level control system, which has been implemented in simulation and on a robot hand/arm system. The computational architecture reflects the recursive task organization described in the previous chapter. Again, control of the robot is decomposed into a set of discrete tasks, in which each task represents, essentially, a complete control loop. The control loop has its own model, that is the *context*; a control law, that is the associated *action*; and a set point, which is the *goal*, as shown in figure 3-1. Since the task-level control system is composed of multiple tasks — each operating within a specific context — the entire control system may be considered a collection of unique controllers, which are turned on and off as necessary to achieve specific objectives. In the following sections we will describe the components of the task-level control architecture, and provide examples to illustrate their operation.

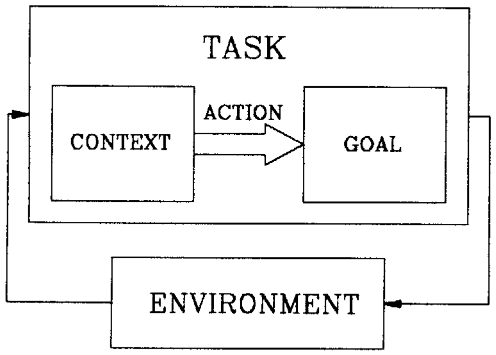


Figure 3-1. The computational architecture is represented by a collection of discrete tasks. Each task represents essentially a complete control loop, where the *context* is analogous to the system model, the *action* to the control law, and the *goal* to the set point.

3.2 Structure of the Task

The computational structure of the task is illustrated in figure 3-2, and includes elements which represent the **context**, **goal**, and **action**. These elements will be described in more detail in the following sections. The definition also includes the boolean values **on**, which allows the activity of the task to be controlled externally; **enabled**, which indicates a task should be evaluated; **evaluated**, indicating that a task has already been evaluated; and **executed**, which signifies that the task has been executed.

```
typedef struct task {
  char          *name;            /* task name */
  char          *description;     /* task description */
  BOOLEAN        on;              /* on */
  BOOLEAN        enabled;         /* enabled */
  BOOLEAN        evaluated;       /* evaluated */
  BOOLEAN        executed;        /* executed */
  CONTEXT       *context;         /* context */
  GOAL          *goal;            /* goal */
  ACTION        *action;          /* action */
} TASK;
```

Figure 3-2. The structure of the task includes elements which represent the **context**, **goal**, and **action**

At each time step, all the tasks in the task-level control system are examined. If a task is both **on** and **enabled**, the task is evaluated. Evaluation of a task means that the knowledge space, or the current state estimate, appropriate for that task is constructed. If this knowledge space satisfies the conditions for the context and not those of the goal then the task is **executed**, which simply means that the action associated with the task is initiated. If the action associated with the executing task is a task sequence, then every task in that sequence is enabled; and if the action is a trajectory, then that trajectory is sent to the robot servo controller.

We can consider two extremes for the practical construction of the knowledge space. In the first case, illustrated in figure 3-3, we could consider a single modeling function, which produces a unique representation common to every task in the hierarchy. The advantage of this approach is the inherent simplicity of a single representation, as well as the ability to decouple the modeling from the control algorithms. However, the disadvantage is the every task would use the same representation, independent of its context or purpose. The result is that the model may be overly general in order to accomodate every possible task, thus producing more complex recognition and planning algorithms and limiting the possibility of real-time control. At the other extreme, we could employ a separate modeling function for every task, as shown in figure 3-4. The advantage would be that every task could use a different representation of the environment, as well as separate recognition algorithms tailored for its particular purpose. Furthermore only enabled tasks would execute their respective modeling functions, thus eliminating the computation of inappropriate, and potentially expensive, algorithms. The disadvantage, is that similar tasks would produce similar information, thus resulting in redundant and wasteful computation. The solution adopted in this thesis is a hybrid approach, in that individual tasks construct their own unique knowledge space, that is each task employs a unique representation of the environment that is built separately for that particular task. However, common algorithms associate with multiple tasks are extract and organized into a dependent hierarchy, which we will describe in detail in the following sections.

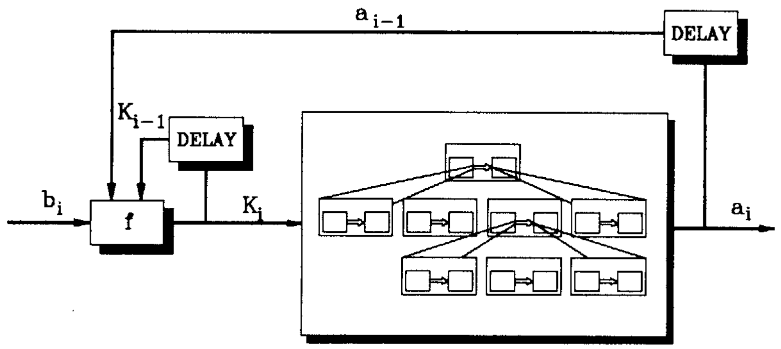


Figure 3-3. The practical construction of the knowledge space could be achieve with a single modeling function. The advantage is the inherent simplicity of a single representation, as well as the ability to decouple the modeling from the control algorithms. However, the disadvantage is that every task must employ the same representation independent of its context or purpose. In addition, the overly general model may increase the complexity of the recogniton and planning algorithms, thus limiting the possibility of real-time control.

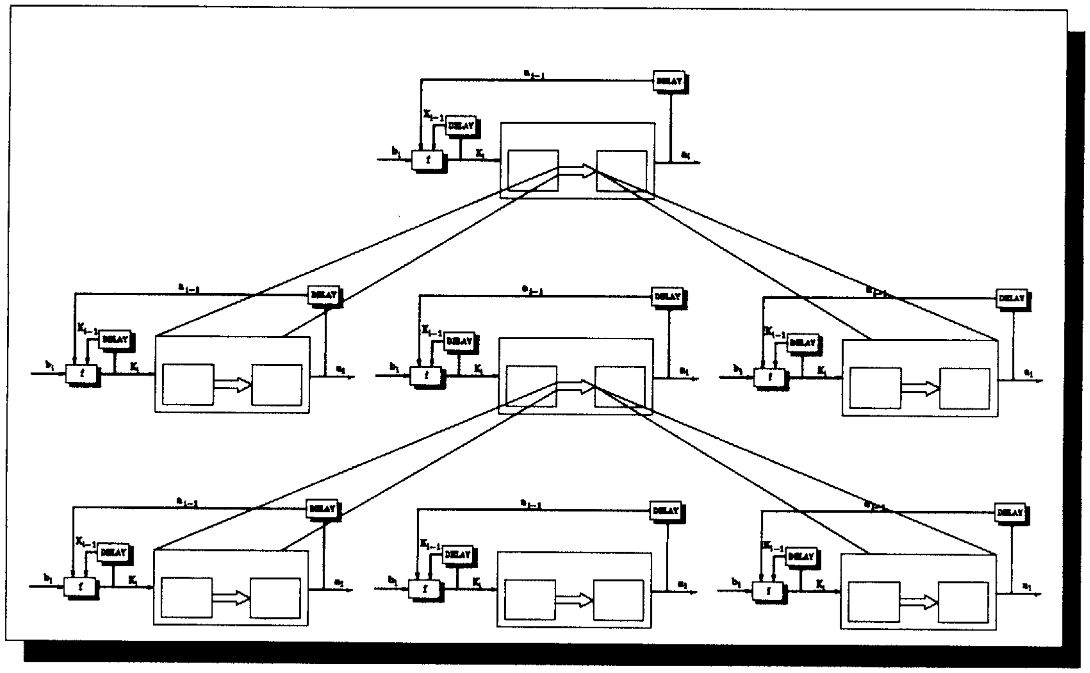


Figure 3-4. Separate recognition algorithms could be employed for every task in the hierarchy. The advantage is that every task could employ unique environmental representations and modeling algorithms tailored for its particular purpose. However, the disadvantage is that similar tasks may produce similar information, thus resulting in redundant and wasteful computation.

3.3 Evaluating the knowledge space

Each task has an associated **context**, whose structure is given in figure 3-5. The context contains a list of **perception units** which represent common algorithms shared among various tasks. The structure and the organization of the perceptual units are given in the next section. The context also contains a unique **function** which uses the data generated from the perception algorithms to construct the current **knowledge**. If this current knowledge is within the context, the function sets the boolean element **in**, indicating that the perceived state of the environment satisfies the conditions of the context. In addition, the structure contains the integer **region** representing the particular partition within the context in which the perceived knowledge space resides. The **goal**, as stated before, is defined in the same way as the context; thus the structure, as shown in figure 3-6, contains many of the same elements as the context. However, the knowledge and region elements are not included in the structure, since in definition of the task-level control system simply satisfying the conditions of the goal are all that is required. The **knowledge** space is represented by the structure shown in figure 3-7. It contains an element indicating a generic type and a list of **parameters** which describe the specific knowledge; that is, these parameters essentially encode geometry of possible positions of objects in the environment. In our implementation, the generic type is given by an integer whose increasing value indicates either a refinement of knowledge of the environmental states or an increase in constraint between the robot and the environment. This value thus corresponds to the sequence contexts used in the definition of the tasks.

```
typedef struct context {
  char          *name;              /* context name */
  int           n_percept;          /* number of perception functions */
  char          **percept_str;      /* perception list string */
  PERCEPTION    **percept;          /* perception list */
  char           *func_str;         /* function string */
  int           (* func)();         /* function */
  BOOLEAN         active;           /* context active */
  int             region;           /* partition */
  KNOWLEDGE     *knowledge;         /* knowledge space */
} CONTEXT;
```

Figure 3-5. The computational structure of the context contains elements function used to construct the knowledge space, as well as a list of common algorithms share by various other tasks.

```
typedef struct goal {
  char          *name;              /* goal name */
  int            n_percept;         /* number of perception functions */
  char         **percept_str;       /* perception list string */
  PERCEPTION   **percept;           /* perception list */
  char          *func_str;          /* function string */
  int          (* func)();          /* function */
  BOOLEAN        active;            /* goal active */
  KNOWLEDGE     *knowledge;         /* knowledge space */
} GOAL;
```

Figure 3-6. The goal is given by a structure similar to the context.

```
typedef struct parameter {
  int           id;                /* numerical identifier */
  char          *name;             /* name of parameter */
  int           n_floats;          /* number of floats */
  double        *f_val;            /* floats */
  int           n_ints;            /* number of integers */
  int           *i_val;            /* integers */
  int           n_bools;           /* number of booleans */
  int           *b_val;            /* booleans */
} PARAMETER;

typedef struct knowledge {
  int           type;              /* type */
  int           n_parameters;      /* number of parameters */
  PARAMETER     **parameter;       /* parameter list */
} KNOWLEDGE;
```

Figure 3-7. The knowledge space structure contains a integer **type**, whose increasing value indicates an increase in knowledge of the states or an increasing restriction of those states. The structure also contains a list of parameters which explicitly describes the partial environmental representation.

3.4 Perceptual units

In practice the construction of the current knowledge may require a number of levels of processing, as shown in figure 3-8. This includes, at the lowest-level, the physical sensors; at intermediate levels, data processing, and feature abstraction algorithms; and at higher-levels, the interpretation, and the current knowledge. The interpretation, that is the state estimate based only on the current sensor values, is combined with the forward projection of the previous knowledge space to yield the current knowledge. In other words, the estimate of the system states is based on both the sensor data and the previous estimate projected forward through the robot action. This is analogous to the Kalman filter, in which state estimate is based on both the measurement data and previous estimate projected forward through the system model.

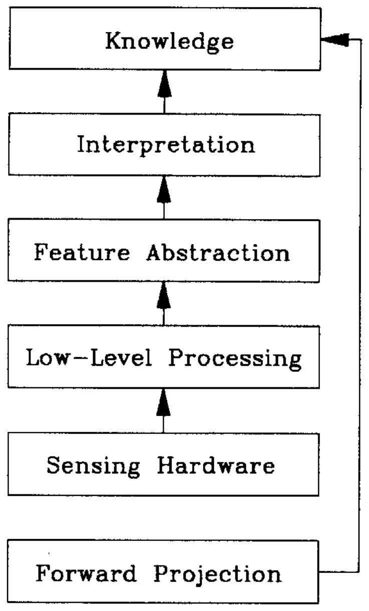


Figure 3-8. In practice, the estimate of the system state may require a number of levels of processing, including sensor data acquisition, low-level processing, feature abstraction, feature interpretation, and knowledge construction.

Suppose, for example, the task is to push an object which lies on a table, and suppose this task is only executes when the robot hand is actually on the surface. Thus the context for this task may be defined as the set of all positions such that the robot hand lies in the plane defining table. To actually determine the position of the hand, however, the robot may require a number of physical sensors, such as a contact force, contact area, tactile, proximity, and joint position sensors. The data generated from these sensors may require a some initial processing to reduce noise and other spurious readings. After this, feature abstraction algorithms may be necessary to identify particular aspects of the processed data. For example, the identification of local curvature using tactile array, contact force and contact area sensors, may be necessary to determine whether an acquired contact actually represents the flat surface of the table or the curved surface of some object which lies on it. The sensory data and abstracted features are then combined to construct the current interpretation, that is an environmental model based only on the current sensory information. The current interpretation, in this example, is simply an estimate of the position of the hand relative to the surface of the table. Finally, this interpretation is combined with the forward projection of the previous knowledge to produce the current knowledge. These processing steps are illustrated in figure 3-9.

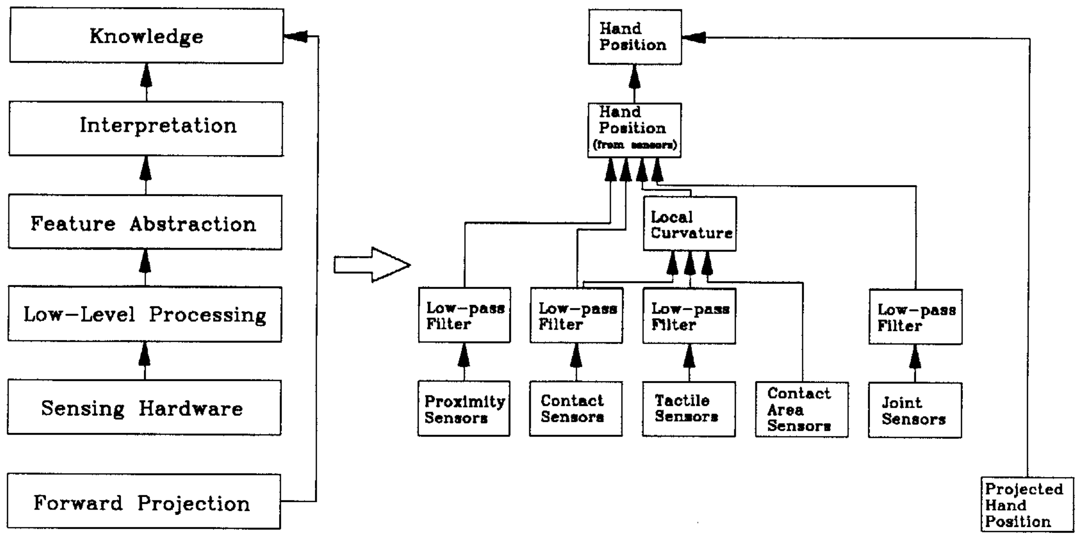


Figure 3-9. The task is to push an object on the table, and the context for this task is the set of all hand positions which actually lie in the plane of the table. To determine the position of the hand relative to the surface, the robot requires a number of physical sensors, as well as low-level processing and feature abstraction algorithms. The position estimated from the current sensor data is combined with the previous position to yield the best current estimate of the hand position.

The sequence of processing steps outlined above may be generalized as a hierarchy of computational elements. These computation elements — called **perceptual units** — include, at the lowest-level, the sensing hardware; at intermediate levels, data processing, and feature abstraction algorithms; and at the highest-level, feature interpretation, and the knowledge space, as shown in figure 3-10. Although the perceptual units process different data at different levels, they are define as computationally identical. Figure 3-11 outlines the structure of a perceptual unit.

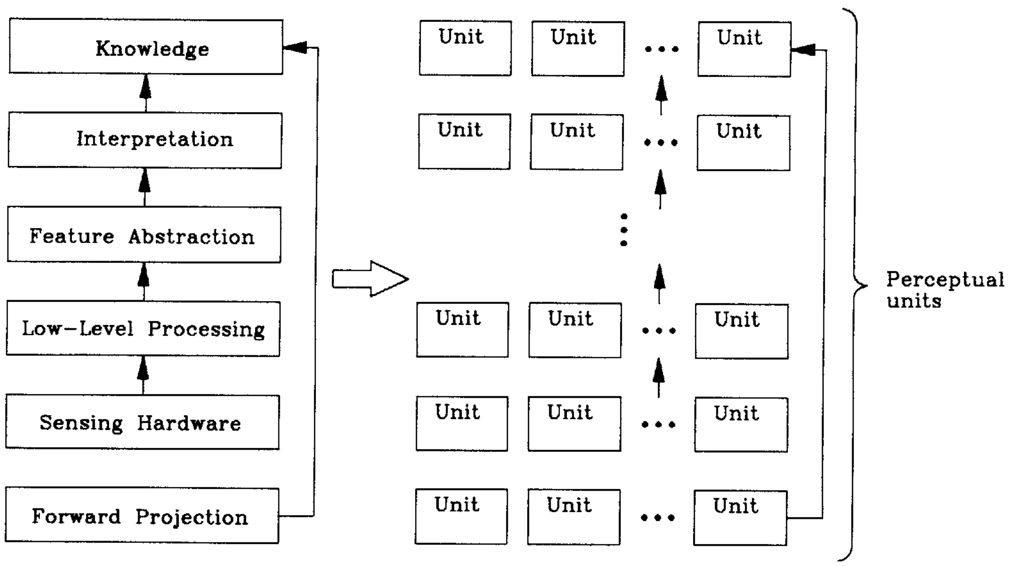


Figure 3-10. The construction of the system state is generalized as a hierarchical network of computational elements, or **perceptual units**, which include, at the lowest-level, the physical sensors; at intermediate levels, data processing, and feature abstraction algorithms; and at the highest-level, feature interpretation, and the knowledge space.

```
typedef struct perception {              } PERCEPTION;
  struct perception   *p;                 /* pointer */
  char                *name;              /* name */
  int                  id;                /* numerical idenifier */
  int                  enabled;           /* enabled */
  int                  active;            /* active */
  int                  changed            /* changed */
  int                  executed;          /* executed */
  int                n_depend;            /* number of dependencies */
  char               **depend_name;       /* dependency list names */
  struct perception  **depend;            /* dependency list */
  char                *func_name;         /* function name */
  int                (* func)();          /* perceptual function */
  int                n_parameters;        /* number of parameters */
  PARAMETER          **parameter;         /* parameter values */
  KNOWLEDGE           *knowledge;         /* knowledge space */
```

Figure 3-11. Structure of the perceptual unit.

When a perceptual unit is **enabled**, it implies that its output should be reevaluated on the next cycle through the control algorithm. The output of a perceptual unit, however, may depend on the output of a number of other perceptual units. For example, a perceptual unit which computes the radius of curvature of an acquired contact may depend on the output of a tactile array, contact force, and contact area sensor, as shown in figure 3-12. Therefore each perceptual unit contains a list of **dependencies**, that is a list of other perceptual units on which its computation depends. Thus when a perceptual unit is enabled, it sends a message to every perceptual unit in the list of dependencies and enables them. Each perceptual unit in the list, in turn, sends a message to the units in their lists enabling them, and so on. This continues until the entire dependent structure is enabled, as shown in figure 3-13.

The reason for this approach, is that specialized sensors and processing algorithms should only be executed when the relevant tasks require them.

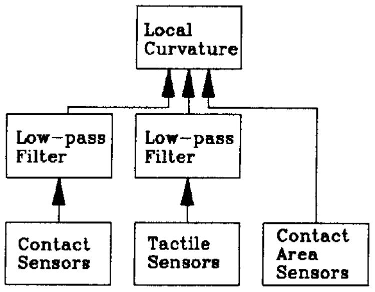


Figure 3-12. A perceptual unit which computes the radius of curvature of a revolute object which pressed against the robot surface depends on three other perceptual units which compute the output of a tactile array, contact force, and contact area sensor.

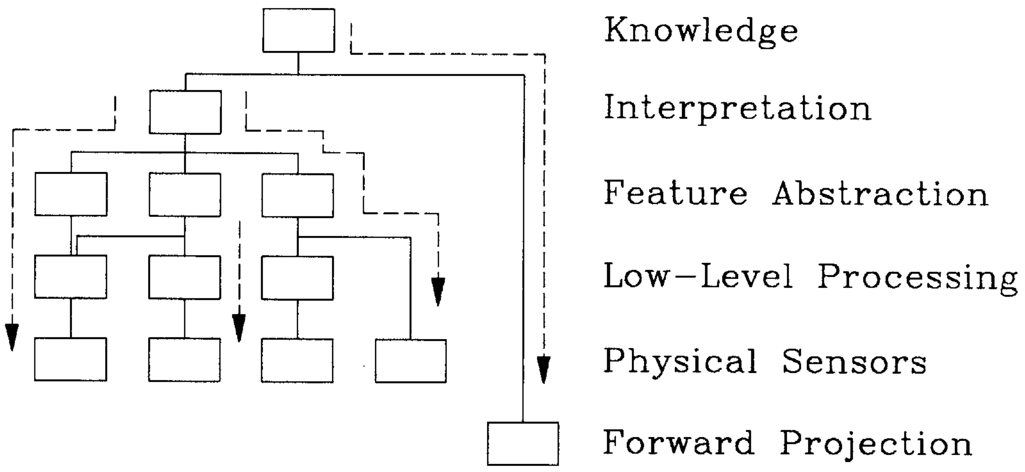


Figure 3-13. When a perceptual unit is enabled, it sends a message each perceptual unit in its list of dependencies, enabling them. This continues until the entire dependent structure is enabled.

The lowest-level units in the perceptual hierarchy are the physical sensors (and the forward projection of the previous knowledge which we will treat as a physical sensor). When the perceptual units are enabled they automatically recompute their values. In the case of the physical sensors, the raw data from the A/D converters is read into the system. If the sensor values have changed since the last time they were read, the boolean flag **changed** is set. Proceeding up the hierarchy, each perceptual unit recomputes its value only if there is a perceived change among the output of the perceptual units in its list of dependencies, as shown in figure 3-14. Hence, if there is no change, the unit will not execute, thus saving valuable computation resources. However, if a perceptual unit does recompute its output, the flag **executed** is set, thus ensuring the unit will only be computed once during a time step. Given this structure, we see that the perception of the environment, and thus the behavior of the robot, is driven only by *changes* in the sensory data or the abstracted environmental features. The detection of change is thus critical in the selection of action for the robot. We should note, that consistent with this idea is some recent research which focuses on the identification of change or significant events in the incoming sensory data stream (McCarragher 1990, Eberman 1993).

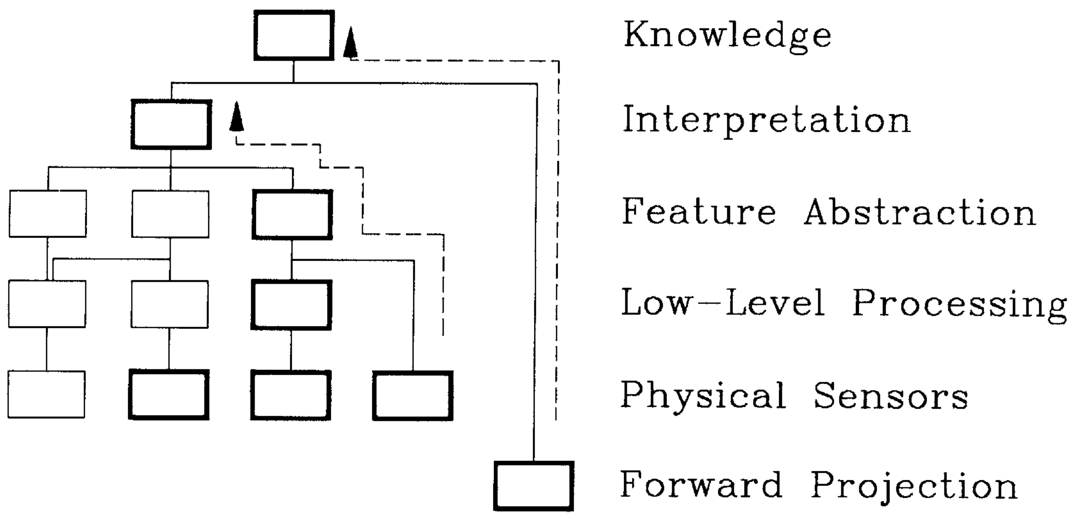


Figure 3-14. A perceptual unit is recomputed only if there is a change among its list of dependencies. Therefore, perception of the environment, and hence the behavior of the robot, is driven only by *changes* in the sensory data or the abstracted environmental features.

The output of the perceptual unit is stored in the **knowledge** element. At lower levels this represents simply the sensory values or abstracted features, and at higher levels this is the current interpretation and knowledge space. Based on the data from the list of dependencies, the **function** in the perceptual unit actually determines the output. The perceptual unit also contains a list of **parameters** which represent easily accessible values used in the computation of the output. For example, such values may include threshold and polynomial coefficients of a low pass filter, or tolerances and geometric variance in higher level units.

The decomposition of the state estimate into a hierarchical structure of perceptual elements has a number of advantages. First, common modeling and feature abstraction algorithms are shared among higher level units. Second, the dependent organization recomputes only when necessary, thus conserving valuable computational resources. Third, new algorithms and sensing hardware can be added to the system without interfering with the other units. Fourth, the system facilitates rapid development, program modification, and debugging. Finally, the structure provides a complete representation of the environment, from the lowest level sensor values, to abstracted feature, and explicit environmental representation.

3.5 Definition of the Action

The **action** is defined as either a sequence of tasks or a robot trajectory, as represented by the computational structure given in figure 3-15. The structure includes an **initialization function** which is executed when the action is first activated. This function calculates the way-points of the trajectory based on the information stored in the knowledge structure of the context. In other words, the particular motion generated is a function of the particular model of the environment. Since the action is based on a recognition of a particular state of the environment, the construction of the forward projection is most conveniently computed within structure of the action. Therefore the structure contains the function, **project**, which computes the forward projection. Finally, the structure also included, the list of tasks in the task sequence, or the points of the trajectory, as given in figure 3-16.

```
typedef struct action {
  char           *name;             /* action name */
  int             id;               /* numerical identification */
  int             type;             /* action type either: TASK_SEQ or TRAJ
  char           *init_str;         /* initialization function string */
  int           (* init)();         /* initialization function */
  int            n_tasks;           /* number of tasks in sequence */
  char          **task_str;         /* names of tasks in task sequence */
  struct task   **task;             /* tasks in task sequence */
  int            n_traj;            /* number of trajectories */
  TRAJECTORY    **traj;             /* trajectory */
  char           *proj_str;         /* projection function string */
  int           (* proj)();         /* projection function */
} ACTION;
```

Figure 3-15. The action associated with the task is given by a sequence of tasks or a robot trajectory.

```
typedef struct traj_point {
  int               n_points;       /* number of indices */
  double           *template;       /* trajectory point */
  int              *type;           /* (relative or absolute) */
  int               direction;      /* for revolute joints (long or sho
  double            speed;          /* speed (deg/s) */
  double           *point;          /* absolute trajectory point */
  double           *rate;           /* rate */
  KNOWLEDGE        *project;        /* forward projection */
} TRAJ_POINT;

typedef struct trajectory {
  int              n_points;        /* number of trajectory points */
  int               new;            /* new trajectory point */
  int               current;        /* current point */
  TRAJ_POINT      **start;          /* starting point */
  KNOWLEDGE        *project;        /* forward projection */
} TRAJECTORY;
```

Figure 3-16. The definition of the robot trajectory includes elements for a set template points and forward project. Both of these can be used for predetermine stereotypical motion.

Chapter 4

Task-level Control System Design and Analysis

4.1 Introduction

The design of the task-level control system involves the decomposition of a task into a collection of discrete subtasks. This decomposition continues until an individual task may be achieved with a single robot motion. Since a task is comprised of a context, goal, and action, the design of the task-level control system thus involves the determination of discrete contexts in which the robot operates and associated actions, either sequences of tasks or trajectories, which achieve the specified goals. The analysis of the task-level control system employs geometric and kinematic models to verify that the actions, once defined, achieve the goals within the given contexts. Thus, the purpose of this chapter is to construct a set of *practical* contexts and goals, and then derive actions, which through geometric and kinematic analysis, can be shown to achieve the goals within the specified contexts.

The construction of the context depends on a number of factors. First, there is the physical environment in which the robot operates. This includes both the objects which populate the environment, and the kinematic and obstructive constraints which restrain these objects. Hence the first issue in the construction of the context is the identification of the specific environment in which the robot will operate and the models used to represent it. Since one of the objectives of the task-level control architecture is to divide robot control into discrete components, there may be multiple, perhaps disparate, environmental representations. For example, one representation may be rectangular solids on a planar surface; another, cylinders in free space; and a third, flexible planar objects in a stack. The particular representation used by the task-level control system depends on the real-time recognition of that particular context. Second, the specific task the robot must perform affects the definition of the context. For example, if the task is to grasp a box, the representation may only require an approximate model of the overall geometry; however, if the task is to open the box, the representation must include some model of the articulated lid, as well as the operation of the mechanism used to secure it. Therefore given an identical object in an identical situation, the definition of the task will determine how this object is represented. Third, since the context is defined as a relationship among objects *which is known to the robot*, the robot kinematics and sensing hardware also determines how the context is defined. For example, a parallel planar gripper may only require an object model which specifies opposing surfaces, while a multi-fingered robot hand may require a more general representation. Sensing hardware also greatly effects the way in which the context is defined. For example, contact and tactile sensors provide only incomplete or partial information about the environment. Thus the context must incorporate these partial or constrained models in its representation.

The construction of the action depends on the specific context and goal. It would seem that given a context, it would be possible to construct a robot path to achieve the goal. However, because of the uncertainty inherent in some contexts and the dynamic variability in some environments, a single robot motion may not be sufficient. Exploratory or tentative motion may be required to constrain the context to a point where a single trajectory will be successful. In this case, the definition of the action would be a sequence of tasks, each with its own context, goal, and action.

Throughout the design and analysis of the task-level control system we will employ traditional geometric and kinematic models to describe the contexts, goals, and actions. Each context implies a particular relationship among objects in the environment, which we will model as a collection of geometric solids. The uncertainties in object position and the partial constraints resulting from incomplete sensor information will be represented by swept volumes constructed by moving the objects through a range of possible positions consistent with the sensor information. For example, suppose we acquired a single contact on a block as shown in figure 4-1. We can construct a region of possible object positions by taking the union of volumes produced by moving the block through the range of positions consistent with the contact data, as shown in figure 4-2a. Conversely, by intersecting these volumes, as in figure 4-2b, we can construct a region in which part of the object is known to lie. Thus the union represents the minimum volume in which the object is contained and the intersection maximum volume which contains part of the object. Hence by constructing an action which avoids this union, collision free motion is assured; while an action which passes through the intersection guarentees an additional contact. We will use these volumes to design trajectories which either avoid collisions, produce partial alignments, or create additional contacts.

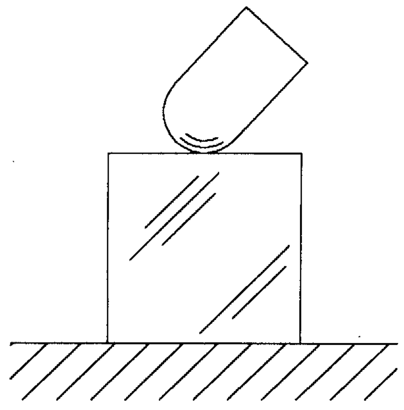


Figure 4-1. A single contact partially constrains the position of a block.

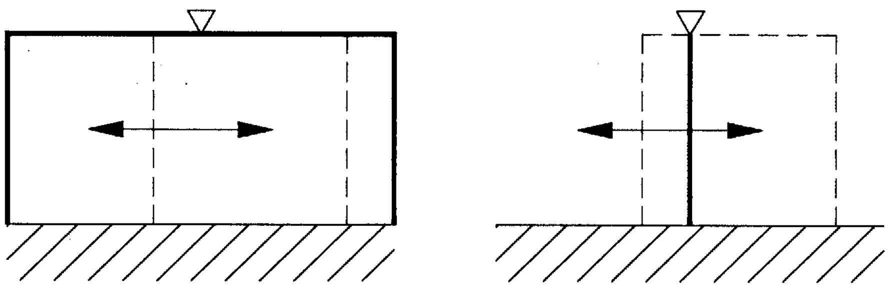


Figure 4-2. By taking the union of volumes consistent with the contact, we construct a region which, when avoided, guarantees collision free motion. Conversely by taking intersections, we generate a volume through which an additional contact is guaranteed.

4.2 Construction of the Context

In this section we will develop contexts which are useful in the task of grasping. Again, the context is defined as a collection of knowledge spaces, that is particular arrangement of objects in the environment which is known to the robot. The construction of the context depends on three factors: the physical environment in which the robot operates; the overall task (in this case grasping), and the manipulator, including both its kinematics and sensing hardware. Thus in the following sections we will examine the physical environment, task of grasping, and kinematics and the sensing hardware of the robot. From this analysis we will propose some practical contexts which will be used to derive robot action appropriate to the task of grasping.

4.2.1 Physical Environment

It is useful, at some point, to consider the actual physical environment in which the robot operates. We will broadly classify robot environments as *structured*, *unstructured*, and *semi-structured*. Although not a strict classification, this decomposition provides a general outline for the analysis of physical objects and their associated constraints.

Structured domains refer to those environments in which the objects and their positions are orderly and predictable. The concept of a structured environment evolved from development of robotic systems for automated assembly and computer controlled machinery (Anderson 1973, Boothroyd 1982). Because of the predictable and uniform nature of a structured environment, persistent and preplanned motion for obstacle avoidance and grasp acquisition is feasible. Offline programming techniques based on manual or automatic way-point generation produce acceptable results for robot motion (Dombre 1984, Gordon 1983, Latombe 1983, Lieberman 1977). The difficulty with applying similar techniques outside the structured domain is the possible variation in the character of the objects, as well as the dynamic nature of many environments. For example, the desire to apply automation to undersea exploration (DePeiro 1986), extraterrestrial manipulation (Townsend 1988, Schenker 1988), and hazardous environments (Stansfield 1989, Johnson 1987, Simmons 1991) necessitates the development of alternative representations.

The *unstructured* environment removes the assumption of predictable objects and uniform locations, and replaces it with arbitrary objects in random positions. In order to describe objects in such a domain, it is necessary to develop a representation of sufficient generality to model objects with arbitrary geometry. Such representations include arbitrary polyhedral solids (Lozano-Peres 1979), quadric surfaces (Franklin 1981), and piecewise smooth manifolds (Canny 1987, Richards 1985). With such assumptions, motion planning algorithms which guarantee obstacle avoidance and stable grasping essentially provide complete solutions for almost every situation the robot may encounter. The difficulty, however, is that such algorithms become substantially complex in an effort to accommodate the generality of these assumptions (Canny 1987, Nguyen 1986). In addition, the algorithms necessary to recognize and location these objects also become quite complex, as well as computationally expensive. The issue is even more pronounced when considering real-time control in dynamic environments. The solution proposed in this thesis is to reexamine the underlying assumptions of the physical domain. Many environments, far from being arbitrary, display a great deal of order and uniformity. Thus rather than ignoring the inherent structure in many environments, the objective is to exploit the underlying symmetry to reduce the complexity of recognition and planning, and to ease the demands on a real-time control system.

The *semi-structured* environment, introduced in this thesis, describes a physical domain composed of regular objects whose geometry and position can be approximated within certain constraints. This representation is derived from observations made of the physical world, in both man-made and natural domains, in that many objects have regular geometries and uniform constraints. Examples of such domains include human environments, such as machine rooms, hospitals, offices, homes, or laboratories, as well as natural environments, such as forests, fields, gardens, etc. Such assumptions greatly reduce the complexity of task planning and robot manipulation, but implies the use of multiple simultaneous representations. The semi-structured environment also assumes that while most objects remain stationary, there remains a possibility of some unpredictable displacement resulting from either an unexpected intrusion into the robot workspace or an unanticipated effect of the robot motion. In the following sections, we will consider examples of semi-structured environments, along with the objects which compose them, and the kinematic and obstructive constrain which restrain the objects.

Objects in Semi-structured Environments

In order to develop a reasonable representation of the objects in semi-structure environments, it is necessary to examine the objects which actually exist within them. In tables 4-1 to 4-3 we enumerate the objects which populate a particular machine shop, office, and kitchen. An attempt was made to remove as much bias as possible by listing every object in these domains, not just those which may be easily manipulated. We should note that these enumerations serve only as examples and guidelines for model selection rather than a complete taxonomy of all physical objects in semi-structured domains. Based on these limited observations, however, we will propose a prototype taxonomy which may serve as a basis for a more complete description. We should also note that this taxonomy is related to object manipulation and grasping, as opposed to some other perhaps more complex task.

Many aspects become immediate apparent upon examine these lists. First, many objects cannot be adequately described as single rigid solids. They fall into such categories a flexible sheets (e.g. paper, cloth, mats), flexible linear objects (e.g. cord, wire, string, rope, and tubes), articulated objects (e.g scissors, books, staplers), and large composite structures (e.g. oven, milling machine, stool, file cabinet).

Second, of the those objects which may be considered rigid solids, many — both man-made and natural objects — display a great deal of symmetry and regularity. The regularity of man-made objects proceeds, by a large degree, from the manufacturing processes used to produce them. Metal products are generally formed through casting, turning, milling, punching, welding, and forging; while plastics are constructed through blow molding, thermoforming, compression molding, extrusion, transfer molding, and injection molding. These techniques produce symmetric objects as a result of either the direct machining process or the associated molds. Furthermore many of these objects can be approximated by simple geometric solids, such as cylinders and rectangular solids. Natural objects also display symmetric structures as a result of the underlying biological and physical processes. These symmetries may be described as bilateral, radial, crystallographic, and spherical (Weyl 1952).

Third, most objects in human environments are designed (as can be expected) for human interaction. Objects such as the computer keyboard, telephone, screw driver, or paint brush, have specific purposes and are intended for specific methods of manipulation. Since in this thesis, we are mainly concerned with object acquisition, these specific functions are not immediately relevant. Manipulation of objects consistent with their intended function is an interesting area of further research and has been studied to some extent with respect to grinding (Asada 1988).

Finally, while many objects display a significant degree of geometric variance, the method of manipulation or interaction with these objects remains consistent. Consider, for example, the variation in the geometry of pushbuttons and toggle switches, yet the same, or nearly the same, motion operates them all. In natural environments, geometric disparity is exaggerated to an even greater degree, yet many of the same stereotypical motions produce similar results, such as a grasping cylindrical branches.

By considering not only the physical environment, but also the specific task, we can reduce the complexity of the object representation by modeling only those features relevant to the specific operation. Since our focus is on grasping, many characteristics of the objects may be neglected. For example, object articulations, such as the lid of a jar, cap of a pen, or joint of a stapler, may be irrelevant to simple task of acquisition and transportation. In addition, many geometric details, such as knurling, ribs, slots, indentations, or protrusions, may be superfluous in the analysis of object acquisition.

```
Rectangular Objects:
--------------------------------------------------------------
Objects: (quantity, h x w x d, description)
  N  1/4 to 4 x 60            aluminum bar stock
  1  1/2 x 1 x 4              oil stone
  1  1/8 x 2 x 5              hole gauge
  N  1/4 x 1/4 to 1 1/2 x 6   parallels
  10 1/20 x 1/2 x 6           metal rulers
  1  3/4 x 1 3/4 x 2 1/2      collet blocks
  13 1/2 x 2 x 2              measuring tape
  2  2 x 4 x 6                drill set holder
  N  varies                   metal scrap
  22 3/4 x 12 x 72            pressed board shelves
Boxes:
 Cardboard with mating lid  (quantity, h x w x d, contents)
  4  1 3/4 x 2 3/4 x 3 3/4  sheet metal punch
  9  2 x 4 1/2 x 4 1/2      sheet metal punch
  3  4 1/2 x 4 1/2 x 4 1/4   sheet metal punch
  1  1 1/2 x 5 x 6 1/2      helix coil threads
 Cardboard with hinged lid  (quantity, h x w x d, contents)
  1  2 x 3 1/4 x 3 1/2      socket cap screws
  1  2 x 3 x 4              socket cap screws
  1  1 3/4 x 3 x 4          socket cap screws
  3  1 x 7 x 9              snap ring boxes
  1  1 1/2 x 4 1/4 x 9      rollpin boxes
  3  1 1/2 x 2 3/4 x 4 1/4  rollpins
  2  6 x 6 1/2 x 7 1/4      belt sander belts
  1  2 x 2 x 3              replacement chuck
 Cardboard flaps  (quantity, h x w x d, contents)
  6  1 1/2 x 9 x 9          sand paper
  1  6 1/2 x 4 3/4 x 7      kimtex wipes
  3  2 1/2 x 5 x 14         steel wool
 Plastic with hinged lid (quantity, h x w x d, contents)
  12  3/4 x 1 x 1           snap rings: small
  3  1 x 1 1/4 x 1 1/4      snap rings: large
  1  1 1/4 x 4 x 8          multicompartment: hitch pins 1/16D to 3/16D
  1  1 1/4 x 4 x 8          multicompartment: machinery keys
  1  1 1/4 x 4 x 8          multicompartment: woodruff keys
  1  1 1/4 x 4 x 8          multicompartment: hex keys
  1  1 1/4 x 4 x 8          multicompartment: cotter pins
  1  2 1/4 x 8 1/2 x 11     multicompartment: O rings
  1  2 1/4 x 9 1/2 x 13 1/2snap ring plier kit
  1  2 1/4 x 8 1/8 x 13     thumb screws
  1  ???                       hydralic knockout punch driver
  3  5/8 x 2 3/4 x 4 1/4    cassette tape boxes
 Plastic with mated lid (quantity, h x w x d, contents)
  1  4 x 1 1/4 x 4 1/2      nut driver set
 Metal boxes with hinged lid (quantity, h x w x d, contents)
  1  2 x 10 x 14            multicompartment: stock springs and washers
  1  2 1/2 x 5 1/2 x 8 3/4  tap set
  1  1 1/2 x 2 1/2 x 10 1/2socket set 1/4 drive
  2  1 1/2 x 3 x 17         socket set 3/8 drive
  1  3 x 2 1/2 x 17         socket set 1/2 drive
  2  2 1/2 x 10 x 16        gauge pins 1/20 to 1/2
 Wooden boxes with hinged lid (quantity, h x w x d, contents)
  1  1 1/4 x 4 3/4 x 11     depth micrometer
  1  1 1/2 x 5 x 19         large caliper
  1  3/4 x 8 1/2 x 14       micrometer set
  1  1 1/2 x 5 3/4 x 8 1/2  machinists square
  1  3 x 7 x 7 1/2          hex & square collect block set
  1  3/4 x 4 1/2 x 12 1/2   angle block set
  1  5 x 10 x 27 1/2        vertical caliber
  1  7 x 5 1/2 x 7          granite block
Containers:
  1  1/2 x 2 1/2 x 5        oil can
  25 1 1/2 x 4 x 5          tap fluid
  8  9 1/4 x 4 x 6 1/2      lubricant
                              1 3/4 x 1/2 threaded cap

Cylindrical Objects:
--------------------------------------------------------------
Objects: (quantity, diameter, length, description)
  N  1/4 to 4 x 5     aluminum round stock
  2  1/4 x 5          flet tip pen
  6  1/4 x 4          wood pencils
  N  1/2 to 2 x 2     knock out punches (mated)
  N  1/16 to 1        end mills
  N  1 to 3           shell mills
  N  1/16 to 1 1/2    reamers
  N  1/4 to 1         drifts
  N  3/4 to 1         punches
  N  1/2 to 1         tapered shank drills
  2  3 x 1/2          electrical tape
  N  1/2 to 1 x 6     lathe arbors
  1  1/2 x 4          edge finder
  6  3/16 x 4 1/2     scribes
  1  1/4 x 5          jewelers screwdriver
  8  3/4 x 5          center punches
  3  1 x 4            deburring tools
  1  1/2 x 6          air chisels (x 6)
  1  1 x 4            pipe dies
  8  1 x 4            collets: milling machine
  20 1 x 4            collets: lathe
  N  1/20 x 1         drills
  N  1/2 x 2 to 8     hold down set threaded rods
  4  1 to 5 x 1/4 to 3 masking tape
  2  1 x 7            round brush
  1  6 x 1            buffing wheel
  1  1 x 7/8          buffing wheel insert
  1  6 x 1            wire brush wheel
  2  10 x 1/2         buffing wheel
  3  2 x 4            buffing composition
  3  2 x 15           cardboard cylinders
Containers:
 Metal with press fit lid (quantity, dia x length, contents)
  3  5 x 4            paint
  4  3 1/4 x 4        paint
  1  3 1/4 x 4        flux
  3  4 1/4 x 5        primer
  2  6 1/2 x 6        lubricant
 Metal with threaded lid (quantity, dia x length, contents)
  1  2 1/2 x 3 3/4    plastic pipe cement
  1  2 1/2 x 3        flux
 Metal with cap (quantity, dia x length, contents)
  2  2 1/2 x 5 1/2    cleaning solvent
  1  2 3/4 x 7 1/2    spray adhesive
  14 2 3/4 x 7 1/2    spray paint
  4  2 3/4 x 7 1/2    lubricant
 Metal with mating skirt (quantity, dia x length, contents)
  3  3 x 2 1/2        abrasive tape
  4  2 1/4 x 2 1/2    abrasive tape
  6  2 1/2 x 3/4      abrasive cord
 Metal
  96 2 1/2 x 4 3/4    soda cans
 Paper (quantity, dia x length, contents)
  1  2 x 9            grease gun cartridge
  2  2 x 11           stick wax lubricant
 Plastic (quantity, dia x length, context)
  1  2 1/4 x 5        wood glue
  1  2 3/4 x 6        oil
  1  3 1/4 x 8        tool block cap 3/4D x 1/2
  3  3 1/4 x 8        thinner
  1  6 x 10           jug
```

```
Composite Structures
--------------------------------------------------------------
Large:
  stock cabinet
    overall 3' x 6' with 18 3x5, 96 2x3 drawers
    contains bolts, nuts, washers: 2-56 to 3/4
  stock cabinet
    overall 4' x 1' with 18 2 x 3, 2 x 5 1/2 drawers
  floor mounted vice
  horizontal band saw
  vertical band saw
  sheet metal brake
  tool cabinet 42 x 28 x 32
  milling machine (x 2)
  lathe (x 2)
  wooden table
  wooden platform
  floor mounted pipe bender
  sheet metal bender
  mobile stand
  benches (x 2)
  bench stools (x 4)
  shop vacuum
  lathe tool storage cabinet
Medium:
  fire extinguisher
  small drill press
  push broom
  flat broom
  knock out punch cabinet
  radio
  fish scale
  jig saw
  lathe coolant system
Small:
  hacksaw
  wooden dust brush (x 2)
  telephone
  paint brush (x 2)
  small wire brush (x 2)
  snap ring pliers (x 3)
  squares
  knife
  right angle attachment
  file cleaners (x 4)
  dust brush
  die handles (x 7)
  dies large 2 x 1/2 (x 7)
  dies small 2 x 1/2 (x 3)
  air chisel set
  hammer (x 7)
  hammer sledge
  tap handle
  lathe tools
  vibro marker tool

Fixed Objects
--------------------------------------------------------------
rectagular:
  wall electrical sockets
  heavy duty electrical switch (x 2)
  thermostat (x 2)
cylindrical:
  water pipe 1 1/2D
  electrical conduits 1D

Small objects:
--------------------------------------------------------------
  bolts   2-56 to 3/4 FLT, CAP, PAN, BUT HD
  nuts    2-56 to 3/4
  washers 2-56 to 3/4

Articulated:
--------------------------------------------------------------
  clip board
  sheet metal cutting pliers
  telephone books
  flashlight
  flint starter
  vernier caliper
  vernier caliper, large
  depth micrometer
  protractors
  compass
  plastic face shields
  dial indicator
  dial indicator extensions
  magnifiying glasses
  squares
  v-block brackets
  collet block wrench
  measuring tape
  hex key set large (x 8)
  hex key set small (x 5)
  indexing table
  puller set
  lathe chucks
  rotex marker

Flexible/compliant objects:
--------------------------------------------------------------
Planar:
  hack saw blades
  trash can liner
  band saw blades
  abrasive pads (x 30)
  sand paper (x N)
  sand paper rolls (x 3)
  belt sander belts (x 2)
  wipes
  rubber floor mat
  gloves
  paper 8 1/2 x 11
  shim stock
  vacuum bags
  brass mesh
  vacuum filters
Linear:
  power cord
  abrasive tape
  abrasive cord
  air hose
Spheroid:
Amorphous:
  steel wool

Anthropomorphic
--------------------------------------------------------------

Other Solids
--------------------------------------------------------------
Medium:
  plastic bucket
  trash can
  dust pans (x 3)
  plastic tray
  crow bar (x 3)
Small:
  sheet metal scrapes
  v-block large 2 1/2 x 2 1/2 x 3
  v-block small 1 1/2 x 1 1/2 x 2
  angle block
  granite block
  metal files: sqr, rat, half rnd (x 10)
  hold down set brackets
     3/4 x 1 1/4 to 3 to 6
  hold down set triangular solids
     1 1/4 x 2 1/4 x 4 to
     1/2 x 1 x 1
  sheet metal bender brackets
```

Table 4-1. Objects within a machine shop.

```
Rectangular Objects:
--------------------------
  fish tank
  automatic timer
  fish tank filter
  Kem wipe box
  first aid kit
  Vaseline jar
  drawing storage tubes
  name plate
  postit note tray
  trash container
  computer monitor
  plastic ruler
  shelves
  video tapes
  ceiling tiles

Cylindrical Objects:
--------------------------
  door handle
  paper cup
  fish food container
  medicine container
  tetracycline container
  honey container
  oil container
  aluminum cylinderical stock
  soda cans
  white board marker
  torque sensor
  bicycle pump
  mechanical pencil
  pen
  plastic cap
  pot
  flourscent lights

Composite Structures
--------------------------
Large:
  door
  book case
  file cabinet
  desk
  computer table
  office chair
  window blinds
  white board
Medium:
  transmission model
  robot model
  telephone
  track shoes
  umberlla
  light covers
  tennis racket
Small:
  hex bolt
  reference calculator
  hole punch
  rolodex
  scissors
  tape dispenser
  stapler

Fixed Objects
--------------------------
rectagular:
  shelve mounts
  electric socket
cylindrical:
  screws in wall
```

```
Articulated:
--------------------------
  hindges
  notebooks
  technical manuals
  computer keyboard

Flexible/compliant objects:
--------------------------
Planar:
  jacket
  tie
  poster
  file folders
  poster board
  newpaper
  plastic file holders
  postit notes
  large pert chart
  letters
  photographs
  ellipse template
  telephone directory
  shirt
  shorts
Linear:
  electrical cable
  window shade cords
Spheroid:
Amorphous:
  large plant
  cactus
  fish

Anthropomorphic
--------------------------
  watering jar
  skates

Other Solids
--------------------------
Medium:
Small:
  can opener
  coat hanger
  shelve brackets
  lunch tray
```

Table 4-2. Objects within an office.

```
Rectangular Objects:
------------------------
  Picture Frames
  Cutting Board
  Cereal Box
  Glass
  Knapkin Holder
  Recipe Box

Cylindrical Objects:
------------------------
  Trash Container
  Pen
  Soda Bottle
  Vase
  Tuperware Jar
  Wine Bottles
  Cooking Spray
  Spice Container
  Pepper Container
  Liquid Spray Cleaner
  Lotion Container
  Baby Bottles
  Thermometer
  Light
  Lamp
  Plates

Composite Structures
------------------------
Large:
  Refrigerator
  Oven
  Potted Plant
Medium:
  Sink
  Wine Rack
  Chair
  Toaster Oven
  Mixer
  Stool
  Foundue Set
  Cuisinart
Small:
  Coffee Maker
  Tea Kettle
  Dust Buster
  Telephone
  Spatule
  Brush
  Burner Covers
  Beaters
  Hammer
  Dish Brush

Articulated:
------------------------
  Sissors
  Tongs
  Cook Books
```

```
Flexible/compliant objects:
------------------------
Sheets:
  Magazine
  Papers
  Calender
  Dish Towels
  Rug
  Nail File
  Diaper
  Cloth
  Paper Plates
  Gloves
Linear:
  Electric Cord
Spheroid:
  Onions
  Tomatoes
  Oranges
  Apples
Amorphous:
  Dryed Spices
  Trash
  Wreath

Anthropomorphic
------------------------
  Ladle
  Spoon
  Fork
  Knife
  Drainer

Other Solids
------------------------
Medium:
  Antique Broom
  Large Bowl
Small:
  Refrigerator Magnets
  Glove Holders
```

Table 4-3. Objects within a kitchen.

Based on the enumerations outlined above, we present a prototype taxonomy based on object geometry as it relates to grasping and manipulation. The objects, listed in tables 4-1 to 4-3 are organized roughly into one or more of the categories outlined below.

Prototype Taxonomy

Rectangular Many objects (or major components) encountered in the environments can be approximated as rectangular solids, as illustrated in figure 4-3. Rectilinear objects such as boxes, erasers, desks, cabinets, doors, etc., acquire their shape from the simple processing of stock material.

Cylindrical A second major category is the cylindrical object. Examples such as jars, glasses, cans, tubes, pencils, shown in figure 4-3, obtain their geometry from turning operations on the objects themselves or the associated molds, as well as from circular extursions, rolling, or spinning.

Composite Composite structures are defined here as those objects whose major components consist of simple geometric forms. For example, a hammer may be roughly approximated as two cylinders joined at right angles, or a telephone receiver as two cylinders and a rectangular solid, as shown in figure 4-3. Composite structures are further classified into sizes, which are defined relative to the size of the manipulator. For this classification we will assume the Salisbury Robot Hand, which is approximately of anthropomorphic scale.

- **Small** A small composite structure refers to one which may be grasped readily by the manipulator.
- **Medium** A medium sized composite structure refers to an object which can be grasped or transported through manipulation of its components, as would be the case for a pushbroom, trash can, fire extinguisher, and chair.
- **Large** A large composite structure is one which cannot be manipulated as a whole by the robot. Examples of Such structures include a tool cabinet, milling machine, operating table, and refrigerator.

Articulated Although not as important for simple grasping operations, a number of objects are composed of major components connected by joints. Examples include scissors, boxes, computer keyboards, books, etc.

Fixed Fixed objects refer to those which cannot be manipulated in any way. These include electrical outlets, water pipes, thermostats, and electrical conduits.

Small Small objects refer to those whose small size is beyond the scope of manipulation. Examples of such objects include mechanical pencil leads, suturing needles, threads, small washers, etc. Again, we should note that size, as defined here, refers only to the scale of the object relative to the manipulator, which in this case the Salisbury Robot Hand.

Flexible/compliant A large class of compliant and flexible objects cannot be adequately described as rigid solids. These are decomposed into planar, linear, spheriodal, and amorphous flexible/compliant objects. Although some research has been applied to the manipulation of these objects, further exploration into the handling and interaction with these objects could prove both interesting and of practical importance.

- **Planar** Examples of planar flexible objects include clothing, plastic sheets, paper, carpet, etc.
- **Linear** Linear flexible objects refer to such objects as cords, ropes, electrical cables, pneumatic tubes, etc.
- **Spheroidal** Spheroidal compliant objects describe soft objects with a discernable geometry, such as sponges, steel wool, pillows, etc.
- **Amorphous** Amorphous compliant objects refer to those which have variable or indistinct geometry. These may include especially natural objects, such as plants, which are not classified by any other category.

Anthropomorphic Anthropomorphic objects refer specifically those objects which cannot be classified elsewhere and designed solely for human interaction.

Other The miscellaneous category is included to accommodate all remaining objects.

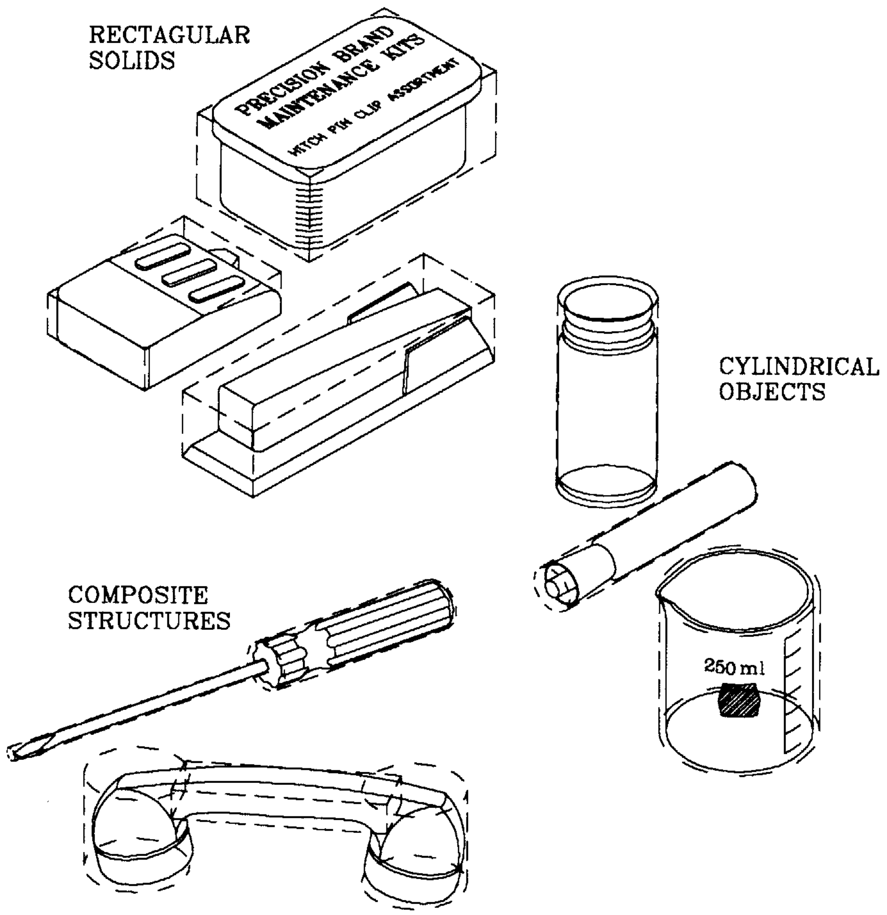


Figure 4-3. A majority rigid solid objects in common environments seem to be composed of geometric primatives — especially cylinders and rectangular solids. If such an approximation is possible, then the complexity of recognition and planning can be greatly reduced. Part of the motivation for the task-level control system was to take advantage of such a possibility by employing numerous simultanteous representations appropriate to the predomainant object types encounterd in an environment.

Although the above analysis is somewhat cursory, it is clear that many environments have distinct classes of objects. One objective of the task-level controller is to take advantage common object types by incorporating multiple simultaneous representations, recognition, and planning algorithms within the same system, which are appropriate for the distinct object classes. Another advantage is that modeling and control functions for the individual object types can be decoupled and analyzed separately. As will be seen in the following sections, this decomposition continues, including not only distinct object types, but also distinct object contraints, robot tasks, and levels of knowledge about these objects.

However, assuming the environment can be decomposed in the manner described above, we see a large number of objects can be modeled with simple geometries, particularly cylinders and rectangular solids. Again, this representation relates less to the specific object geometry than to the overall shape as it relates to the task of grasping. Also, since an objective of this thesis is the decomposition robot control into an aggregate of discrete contexts, goals, and actions, additional object representations can be added incrementally without effecting the current implementation. Therefore, as an initial set of object models, we will consider an approximate cylinder and rectangular solid, as shown in figure 4-4 and 4-5. In the first case, we assume the object is bound by two cylinders: a smaller one, of radius r_1 and height h_1, inscribed in the object, and a larger, of radius r_2 and height h_2, circumscribing it. In the second case, an inscribe rectangular solid has height h_1, width w_1, and length l_1, while a circumscribed solid has dimensions h_2, w_2, and l_2.

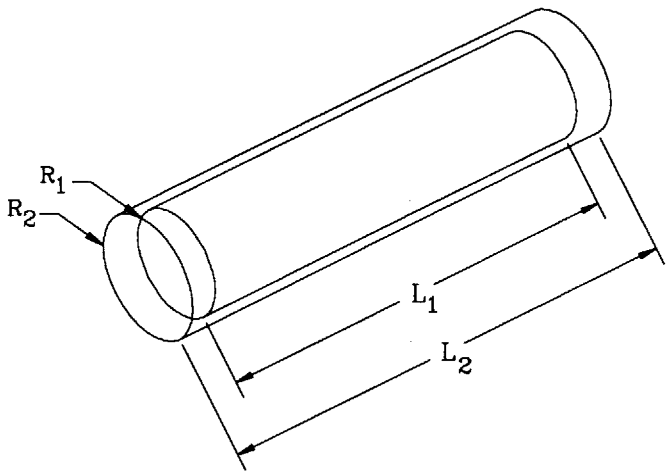


Figure 4-4. Cylindrical approximation

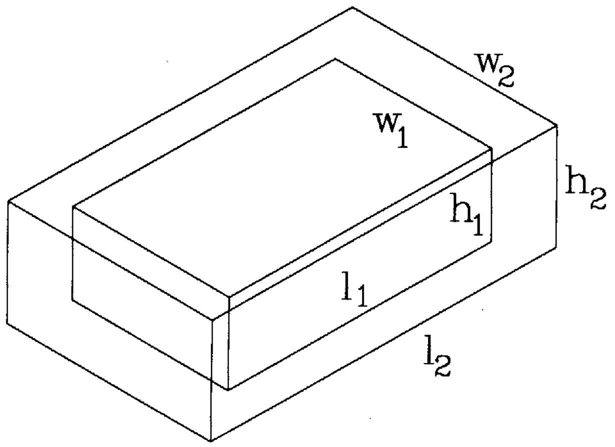


Figure 4-5. Rectangular solid approximation

Kinematic Constraints in Semi-structured Environmênts

Kinematic constraints refer to limitations on object motion imposed by contacting objects. Examples of such constraints include drawers, doors, knobs, switches, slides, and screws. We will enumerate and classify kinematic constraints using the same techniques employ in manipulator kinematics.

Since the position of one object relative to another may be specified by as a point in the configuration manifold $x \in SE(3)$, given by the Euclidean group of 3-dimensional space, $SE(3) = \Re^3 \times SO(3)$; and the instantaneous displacement v by a point in the tangent space, $(x, v) \in T_x SE(3)$, then the constraint imposed by two objects in contact can be represented locally as a k-dimensional linear subspace of the tangent space $T_x SE(3)$. Thus by enumerating subspaces of dimensions $k = 1, \ldots, 5$, we span the space of all possible kinematic "joints" which existing between two objects. A basis for this space is given by the set of lower order kinematic pairs, shown in figure 4-6, (see Angeles 1982 for a more complete description). Although joints with different degrees of freedom, such as the two-dimensional condyloid joint formed by orthogonal revolute hyperboloids (an example of which is found in the metacarpel-trapezium joint of the human thumb), the lower order pairs specific all joints for which the objects contact along a surface. All other joints contact along a point or a line.

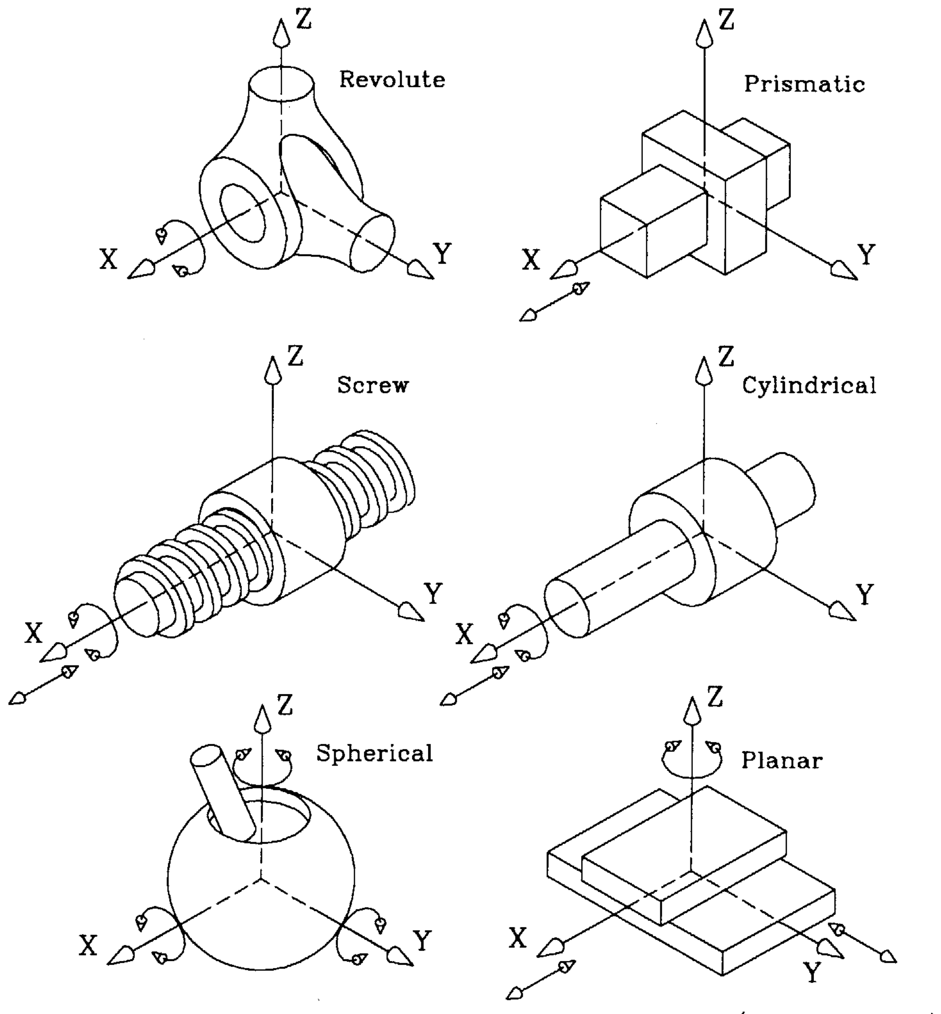


Figure 4-6. The space of all possible joints is spanned by a basis given by six lower order pairs, which include the revolute, prismatic, screw, cylindrical, spherical, and planar joint.

Examples of all lower order pairs are observed in real semi-structured environments. The planar pair is perhaps the most common, since any stationary independent object in a gravitational field will naturally form a local planar joint with any supporting surface. Revolute joints including doors, knobs, and valves, and prismatic joints such as drawers and buttons, are common, as well as screws in any mechanical setting. Spherical and cylindrical kinematic pairs are perhaps less common, though clearly cannot be ignored. Thus, the development of a practical set of context must include the set of kinematic couplings given by lower order pairs.

Obstructive Constraints in Semi-structured Environments

Obstructive constraints refer to workspace limitations imposed by neighboring obstacles which are not in direct contact with the object to be grasped. It is clear, that in many situations, other objects place boundaries around the object of interest, restricting both its motion and accessibility. To solve the complete obstacle avoidance problem, it is necessary to employ an entire geometric model of the environment, and develop the necessary algorithms to maneuver without collision within this model (Canny 1986, 1987; Lozano-Perez 1980). However, if it were possible to categorize obstacles into a number of generic types, as in the case of kinematic couplings, then the problem of obstacle avoidance could be greatly simplified. In this case, the control system need only recognize a particular type of obstruction, and generate an appropriate stereotypical action to avoid the obstacle. In many practical situations, this generalization seems possible. For example, tables, walls, ceilings, cabinets, and shelves, all present similar constraints which are common to many situations. Therefore, as a first assumption, we will model the obstacles around an object may as a set of orthogonal planes, as shown in figure 4-7. Figure 4-8 also illustrates some common examples of obstructive constraints. In the first case, a fixed object has no surrounding obstructions, representing perhaps a pole, pipe, or structural support; in the second, an object is supported by a single plane, representing a number of scenarios in which an object lies on a surface; and in the third, five orthogonal planes place restrictions on an object, representing perhaps a object within a cabinet or on a shelf.

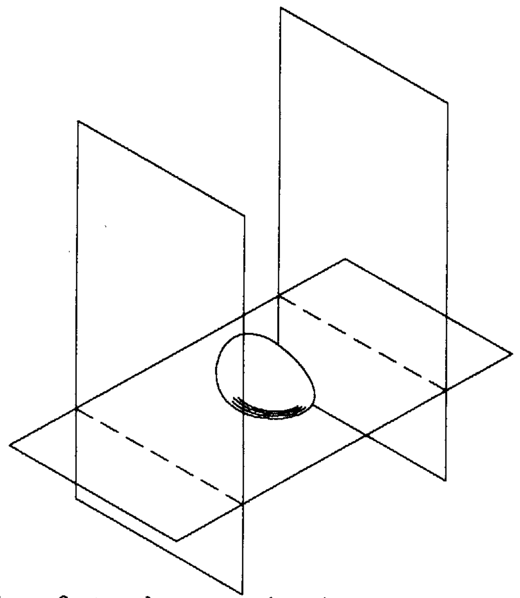


Figure 4-7. As a first order approximation, we will assume that many common obstacles can be approximately modeled as a set of orthogonal planes surrounding the object of interest. Employing this assumption rather than a more general one, allows the resulting recognition and planning algorithms to be greatly simplified.

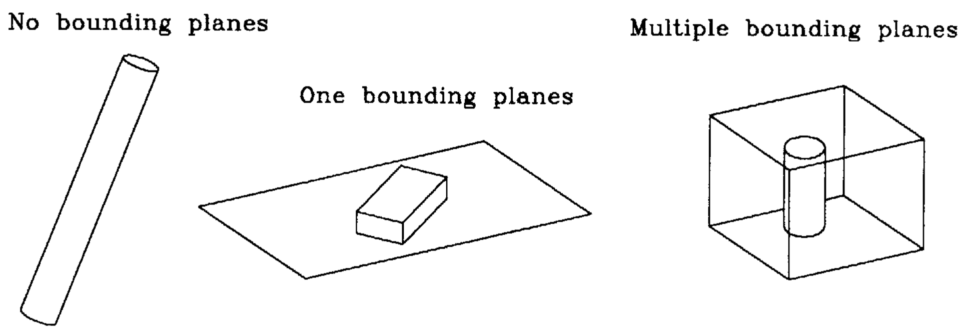


Figure 4-8. Examples of common obstacle types include an object in free space, an object on a plane, and an object bound within a box. These examples, as well as others, serve as prototypes in the construction of the context.

4.2.2 Overall Task

In addition to the objects and their constraints, the definition of the context also depends on the particular task the robot is required to perform. In the example given at the beginning of this section, the representation of a box was dependent on the particular task performed with it. If the task was to acquire and transport the box, the representation only includes the overall geometry; however, if the task was to open the box, the model must include a representation of the articulated lid. In general, different tasks will require different models to represent the same objects. Since, in this thesis, we are mainly concerned with robot grasping, we will employ gross geometric models to represent the objects in the environment. The next chapter, however, provides a few examples of tasks and object representations which do not involve grasping.

4.2.3 Manipulator

Since the context is defined as a particular relationship among objects in the environment *as perceived by the robot*, the particular kinematic structure and sensing hardware of the manipulator will affect the construction of the context. For example, suppose the robot was equipped with a tactile array (Dario 1987, Berkemeier 1990), then contact with the side of a cylinder would determine the cylindrical axis — though the center of mass would be unresolved, as shown in figure 4-9a. Tactile arrays, however, have only been applied to robot control with some difficultly because of the associated complexity of the wiring and sensors. Contact force sensing, on the other hand, has been applied successfully in a number of robotic systems (Brock 1985, 1989; Bicchi 1991, 1993). A single contact with a contact force sensor, however, can only resolv a single point and the surface normal. Thus contact with the side of a cylinder would constrain the object to lie within a disk centered at the contact point, as shown in figure 4-9b. If errors were present in the sensor reading, this would constraint would be somewhat larger as represented in figure 4-9c. Binary contact sensors, such as those in figure 5-x and 5-y, have also been employ as an economical and robust means of extracting useful information from the environment (Brock 1989). A single contact with the cylinder on one of these sensors, would produce a fairly large region of possible positions, as shown in figure 4-9d. Finally, torque sensing at the robot joints has been used to infer some contact information on the links (Eberman 1990). Using only torque information, the estimated position of the cylinder is shown in figure 4-9e.

We see that the estimate of the environmental states varies as function of the type of sensors used on the robot. In general, a single sensor reading $b \in B$ produces a set of consistent system states, referred to as an interpretation region $I = I(b)$ in chapter 2. The intersection of multiple interpretation regions define the knowledge spaces K, which thus forms the basis for the construction of the context.

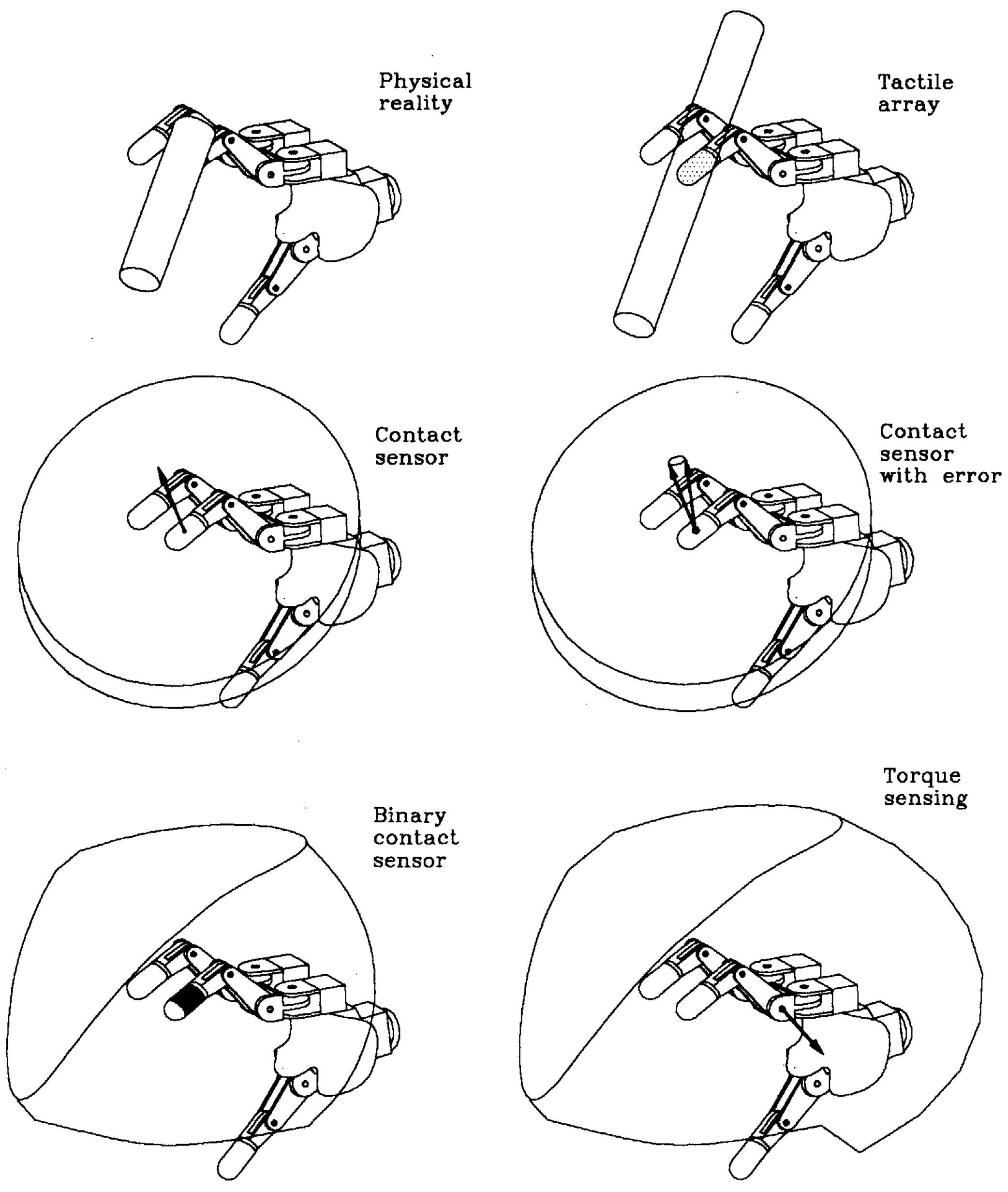


Figure 4-9. The interpretation of the environment varies depending on the types of sensors used on the robot. Given a single contact with the side of a cylinder, the above examples show volumes representing cylinder positions consistent with the sensor data.

4.2.4 Examples of contexts

Given a representation of the objects in the environment, the overall task, and the kinematic and sensing hardware of the robot, we are in a position to construct the contexts, which essentially form the basis of the task-level control system. In order to analyze the contexts and design appropriate actions, we must use some method to explicitly represent these geometric entities. Therefore for the purposes of designing and analyzing the task-level control system, we will construct computer representations of the geometric volume which describe the contexts. We begin by constructing a model of the manipulator, as shown in figures 4-10 through 4-12. The contexts then represent by geometric volumes which exist in relation to the manipulator. For example, suppose a cylinder contacts a force sensing fingertip, as in figure 4-9c, then the resulting set of cylindrical positions can be represented by the disk shown in figure 4-13. Using a fingertip binary contact sensor, a single contact with a rectangular solid on a planar surface produces the set of consistent positions shown in figure 4-14. The robot in figure 4-15 contacts a large vertical surface, (e.g. a wall, cabinet, door, etc). Using only joint torque information, the location of the surface can be limited to the reqion shown in the figure.

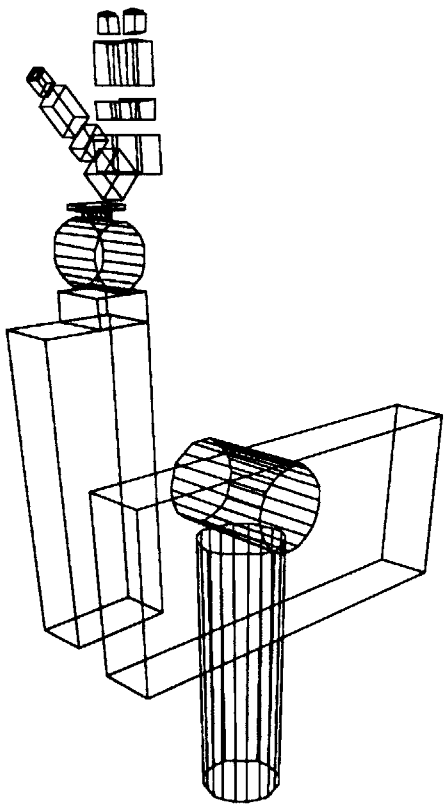


Figure 4-10. A coarse solid model of the Salisbury Robot Hand and PUMA arm system are shown. The software used to generate these models can simulate the trajectory of the manipulator and test for possible collisions with any other volume

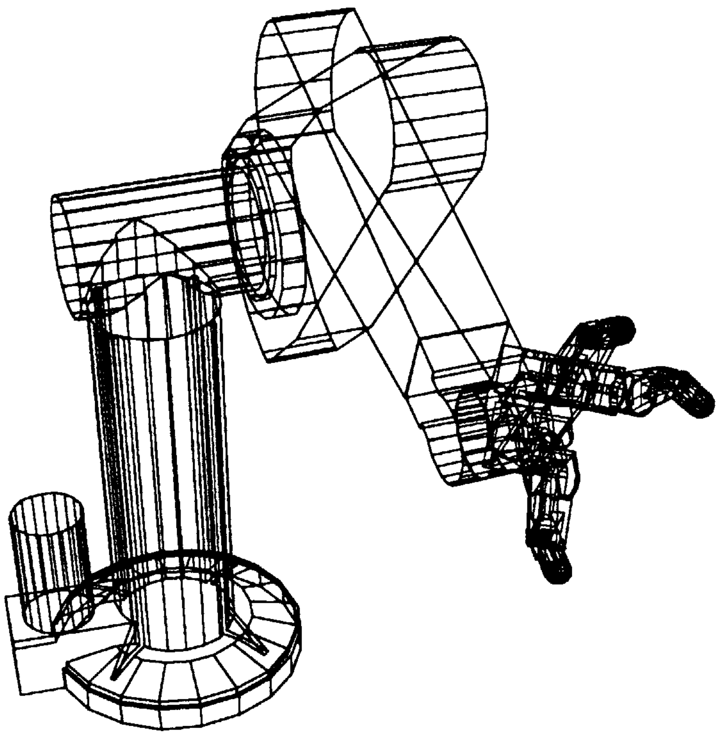


Figure 4-11. A detailed model of the Salisbury Robot Hand and PUMA arm system allows percise calculations of volume intersections, but carries substantial computational cost.

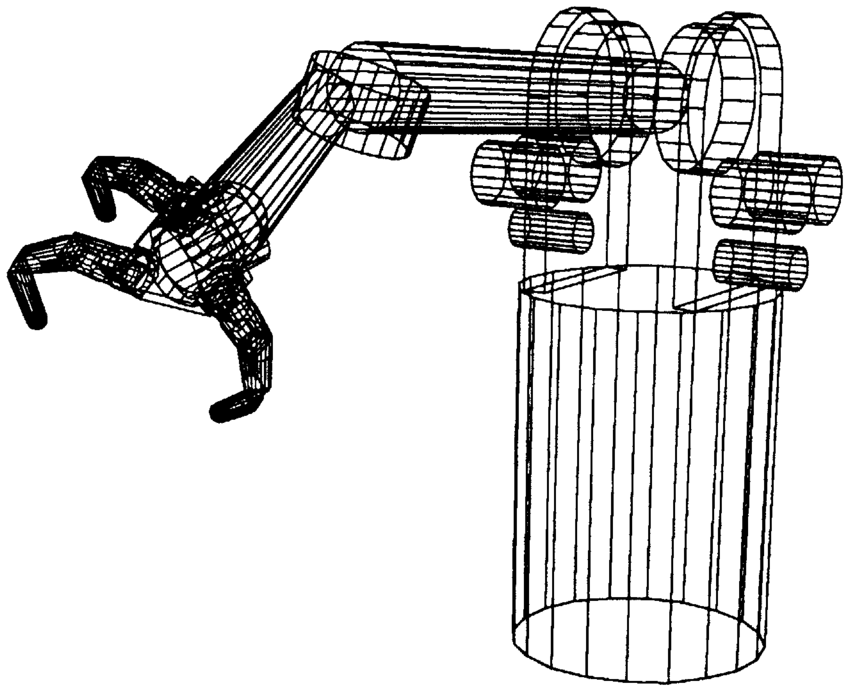


Figure 4-12. The Whole Arm Manipulator (WAM) robot (Salisbury 1987, Townsend 1988) is shown with a hypothetical hand based on the prototype finger (see figures 6-3 through 6-6).

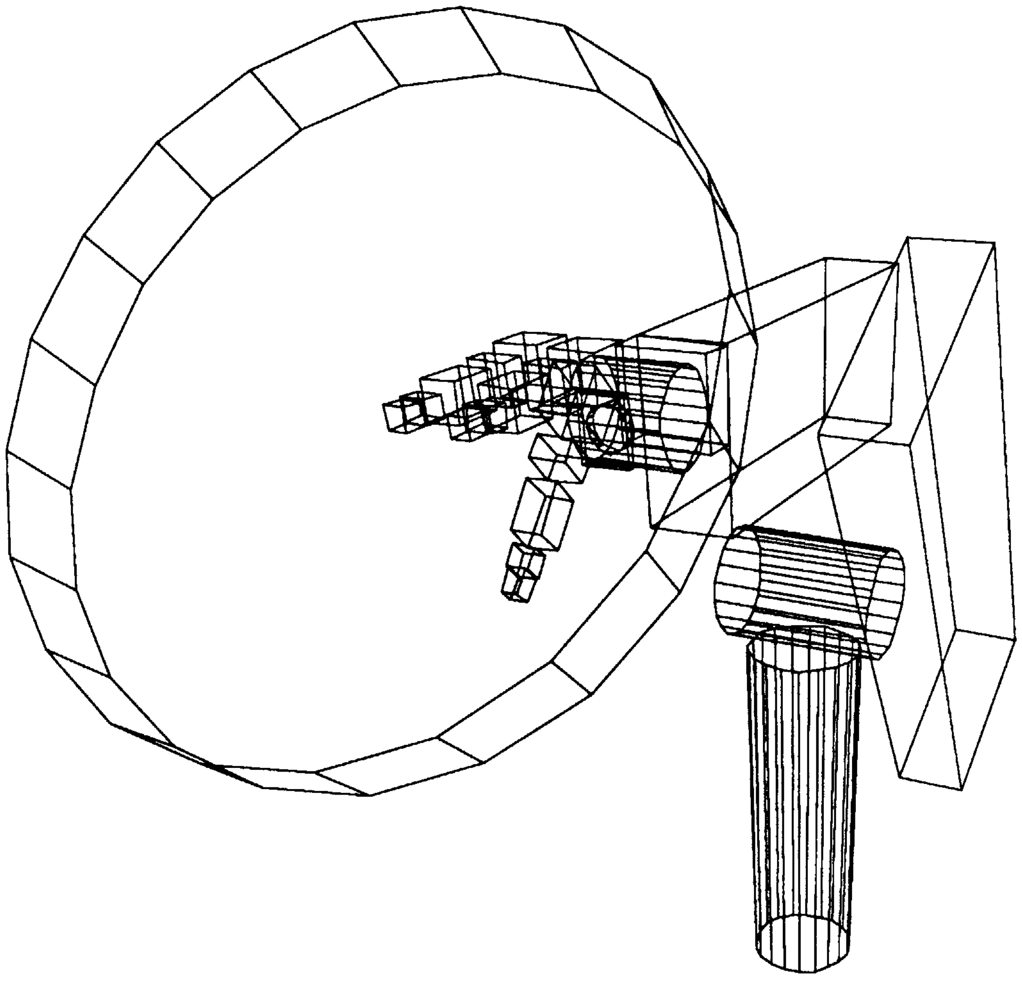


Figure 4-13. Contact with a force/torque contact sensor limits the position of a cylinder to a set of positions which exist within a disk centered at the contact point.

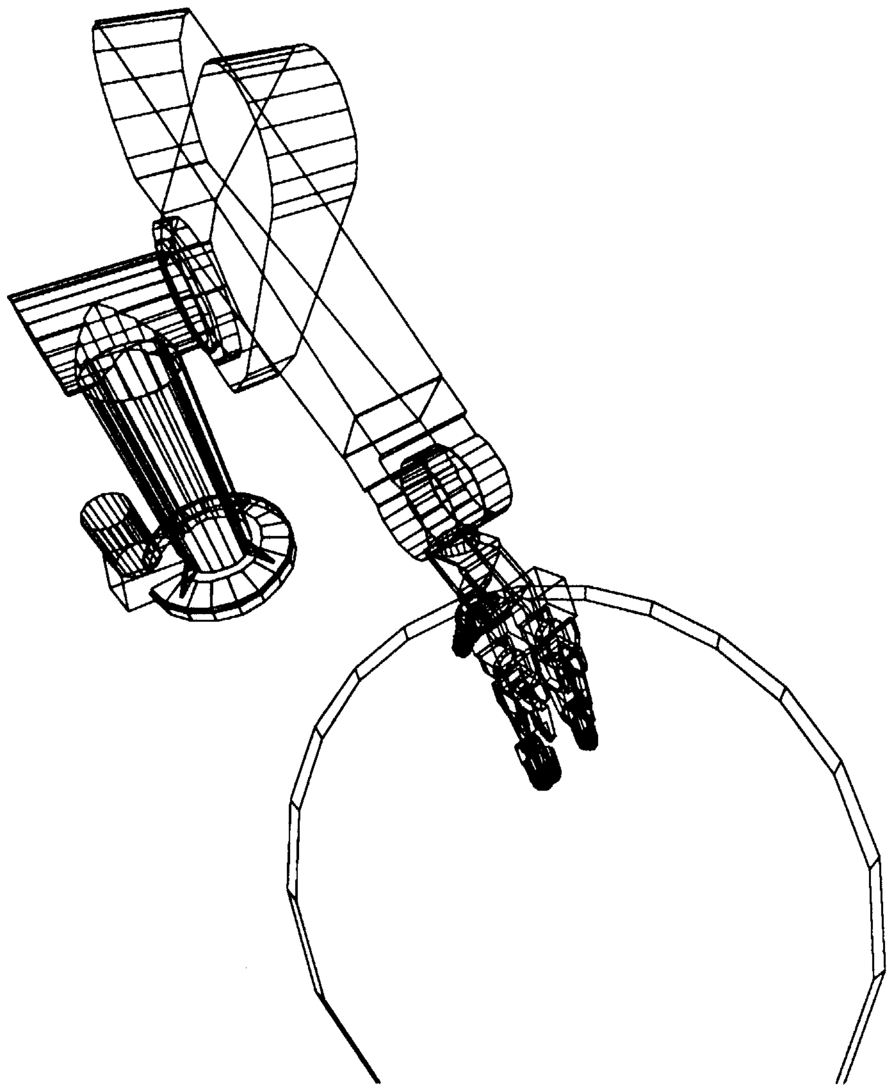


Figure 4-14. A single contact with a rectangular solid limits its position to a disk.

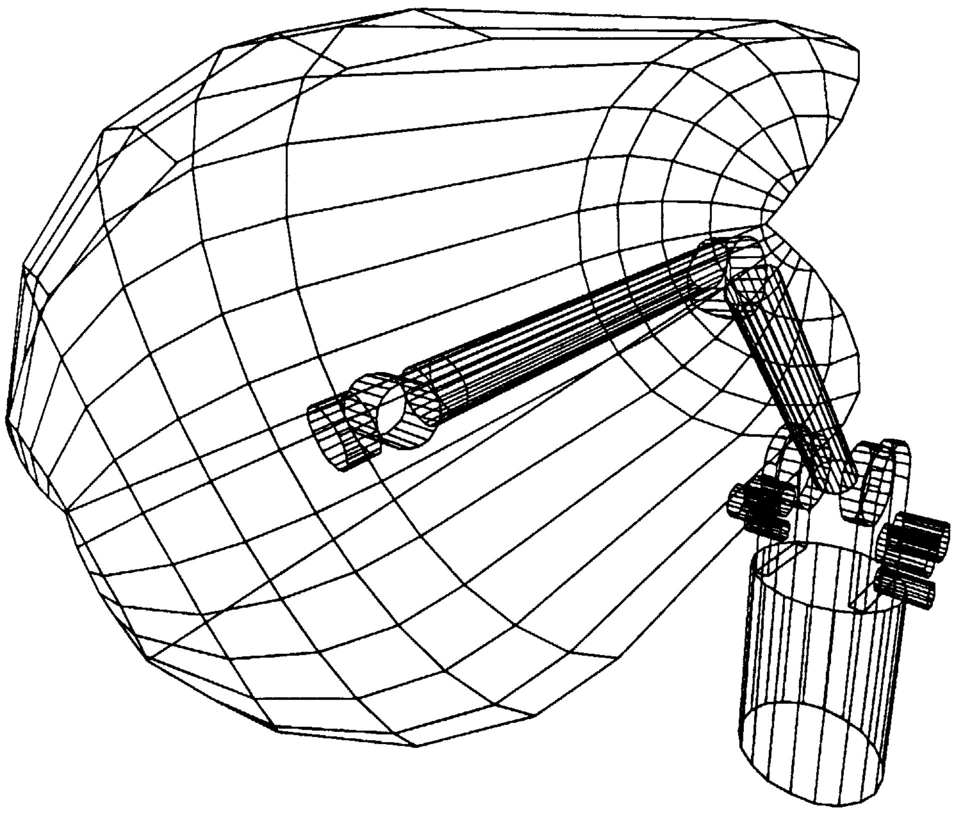


Figure 4-15. Using only torque sensing in one joint, the WAM robot can limit the position of a rectangular solid to the region shown above.

4.3 Construction of the Action

An action is defined as either a task sequence or a trajectory, both of which designed to achieve the goal within a given context. Both the context and the goal are defined in the same way, that is both are collections of knowledge spaces. In other words, the context and the goal represent particular relationships among objects in the environment which are known to the robot. Given a context and goal, however, there is no fixed rule for the construction of the action. Even the choice between a task sequence and a trajectory cannot be easily determined. For example, the relatively complex task of aligning an L shaped block from an arbitrary initial position can be achieved with a single fixed trajectory (Erdmann 1988). However, the seemingly simple task of grasping an object which is placed in the hand and moving it some fixed distance, cannot be achieved with a single motion. The transition between the grasping and transportation, in this case, depends on the knowledge of a secure grasp, which, in turn, depends on finger force and/or torque information. There are also many tasks which can be solved with a single trajectory, but may be solved more efficiently with a task sequence. For example, the problem of acquiring an object on the table, as outline in the first chapter, could be accomplished with a single motion; that is, the hand could be moved over the surface of the table in an effort to sweep the object into the grasp. However, it would be more efficient to use the intermediate sensor information to grasp the object as soon as it contacts the fingers.

In general, we will use the contexts, as constructed in the previous section, as guidelines in the design of the actions. The contexts represent geometric entities formed by sweeping an object through the set of positions consistent with the sensor data. These swept volumes represent weak constraints on the desired trajectory of the robot. Given these weak constraints, we will determine a set of equivalent trajectories to achieve the desired goal.

Chapter 5

Implementation of the Task-Level Control System

5.1 Introduction

The primary motivation for the task-level control system was to develop a system which takes full advantage of the available sensing hardware, which guarantees stability and convergence, and which responses immediately to perceived changes in the environment. These attributes make the task-level control system particularly attractive for practical implementation on existing robot systems. In order to demonstrate these concepts, we will consider some autonomous systems, both in simulation and physical hardware, which must achieve a specific task given limited sensing.

5.2 Planar Hand/Arm Simulation

As an example which illustrates many of the key concepts of the task-level control system, we will consider the simple system shown in figure 5-1. This planar hand/arm system has binary contact sensors on the links, position sensors in the joints, and a binary sensor in the goal region, as shown in figure. The task for this manipulator is simply to place the disk into the goal region. Even for this seemingly simple task, a number of difficulties become evident. First, it is not always possible to determine the position of the disk exactly, although it is precisely the position of the disk which the system must control. Second, there numerous ways in which this task can be achieved, including pushing, grasping, or a combination of both, to move the disk into the goal. Third, the disk may at any time experience a random displacement. This assumption is included to model unexpected intrusions into the robot workspace or unmodeled consequences of robot action. Thus the manipulator must perceive these changes, and generate immediate action to achieve the goal. Fourth, even though this is a simple example, it is fairly high dimensional. Six parameters are needed to describe the state space, four for the position of the manipulator and two for the disk, as shown in figure 5-2.

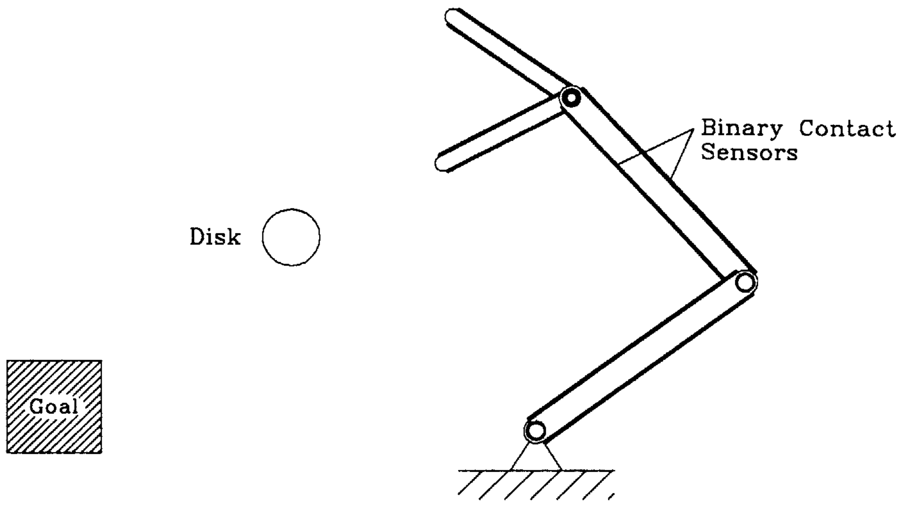


Figure 5-1. A simulated planar hand/arm system has binary contact sensors on the surface of every link, position sensors at the joints, and a binary sensor in the goal region.

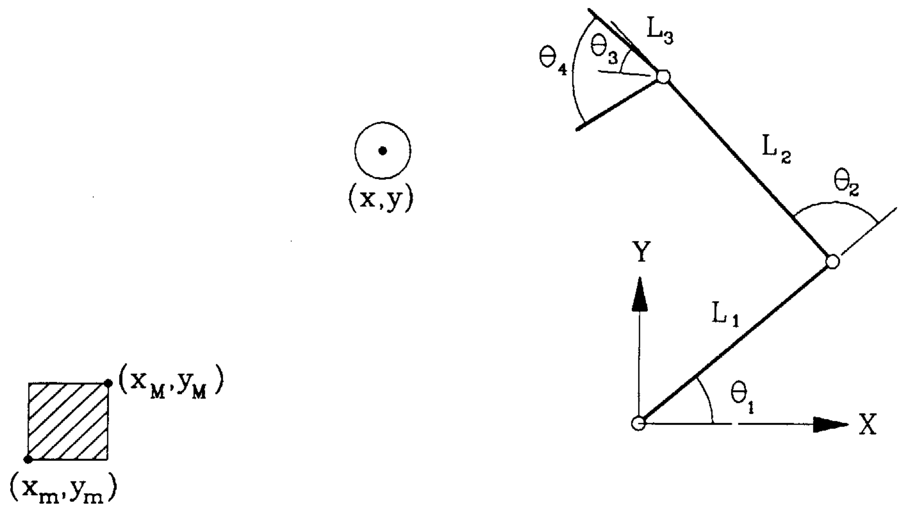


Figure 5-2. This planar system requires six parameters to describe its state, four for the position of the manipulator and two for the disk.

We will assume the position of the disk is initially unknown, expect that it lies within the workspace of the arm. We will also assume in this example, quasi-static motion; that is the motion dictated only by frictional and contact forces. Inertial forces are assumed to be negligible. Therefore the set of system states is given $S = S^1 \times S^2$, where $S^1 = \{x, y\}$ is the position of the disk, and $S^2 = \{\theta_1, \theta_2, \theta_3, \theta_4\}$ the position of the manipulator. The context, in this case, can be described by

$$C = \{K | K = K^1 \times K^2\},$$

where $K^1 = \{\mathbf{x} = (x, y) | ||\mathbf{x}|| < L_1 + L_2 + L_3\}$ $K^2 = S^2$, and the goal by

$$G = \{K | K = K^1 \times K^2\},$$

where $K^1 = \{\mathbf{x} = (x, y) | x \in (x_m, x_M), y \in (y_m, y_M)\}$ and $K^2 = \mathbf{0}$.

In other words, the context is simply the knowledge that the position of the disk lies within the workspace of the arm, and the goal is the knowledge that the disk is in the goal region and the arm is at the null position, as represented schematically in figure 5-3.

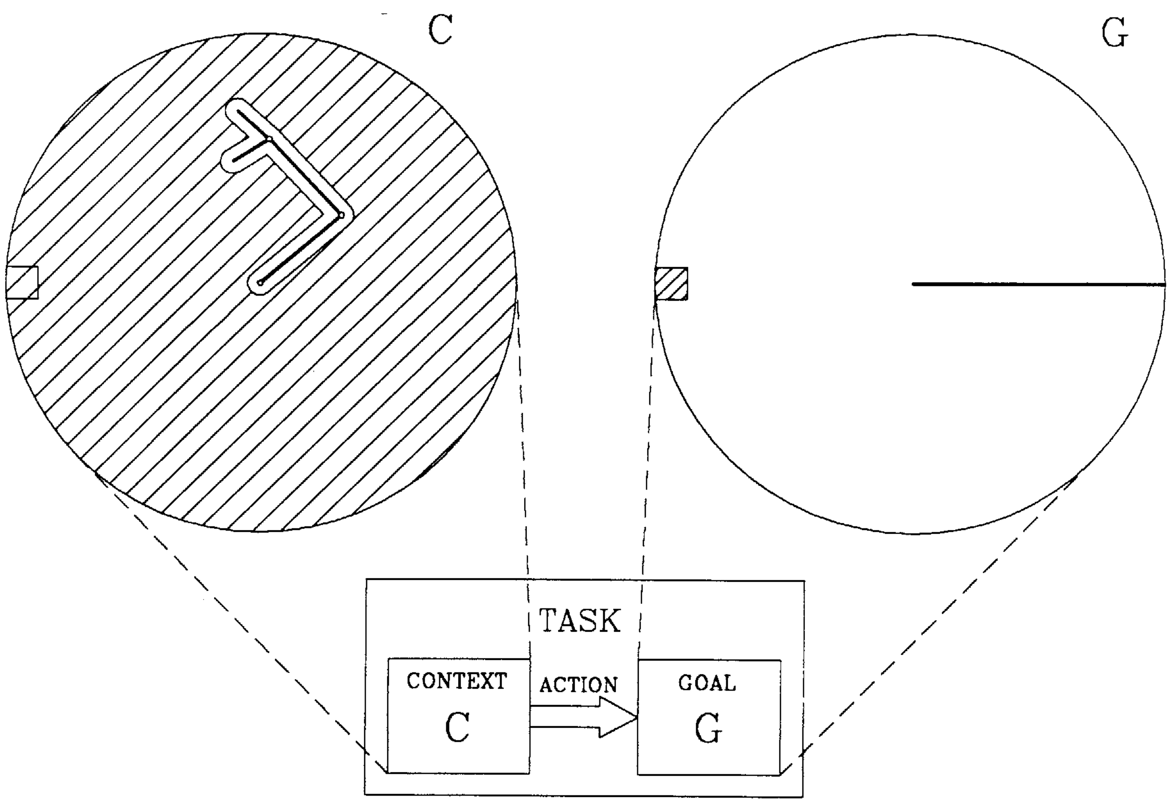


Figure 5-3. The context is defined as the disk in the workspace and the goal as the disk in the goal region and the manipulator in the null position.

There are many possible solutions to this task. For example, the arm could be extended and simply rotate about the workspace until the disk was pushed into the goal. This type of solution is consistent with the concept of sensorless manipulation (Erdmann 1984, Goldberg 1989, Mason 1986); however, it does not take advantage of the available sensing. Employing the procedure of the previous chapter, we can construct a sequence of contexts based on the information produced from the available sensors. We can also construct contexts by constraining the variability of the known states; that is given partial information it may be possible to incrementally achieve a given task. Based on these concepts a sequence of contexts are constructed as shown in figure 5-4. In the first case, a single contact with the disk decreases the uncertainty of its position from a plane to a line. A second contact, refines this knowledge from a line to a point. With the position of the disk known, it is possible from the manipulator to simply grasp the object and place it in the goal region. Finally, the position of the disk in the goal implies a single arm motion to the null position. This sequence of contexts thus form the basis for a task-level controller, as shown in figure 5-5.

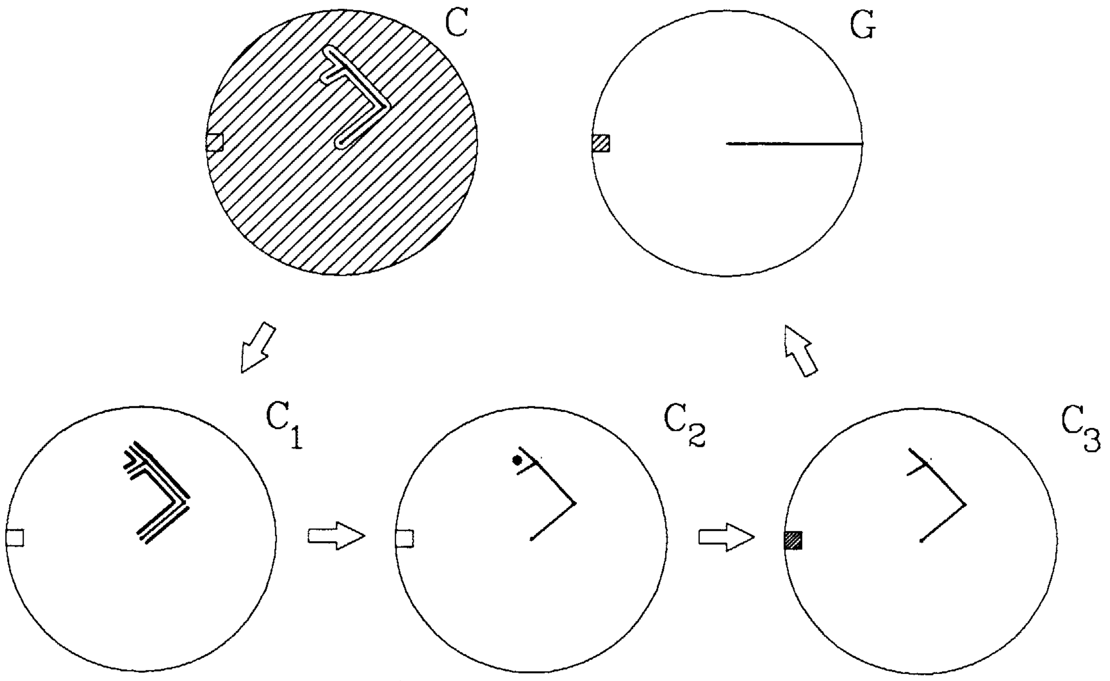


Figure 5-4. Based on the available sensors, as well as incremental constraints representing partial solutions of the task, we can construct a sequence of contexts representing increased knowledge or increased restrictions on the system states.

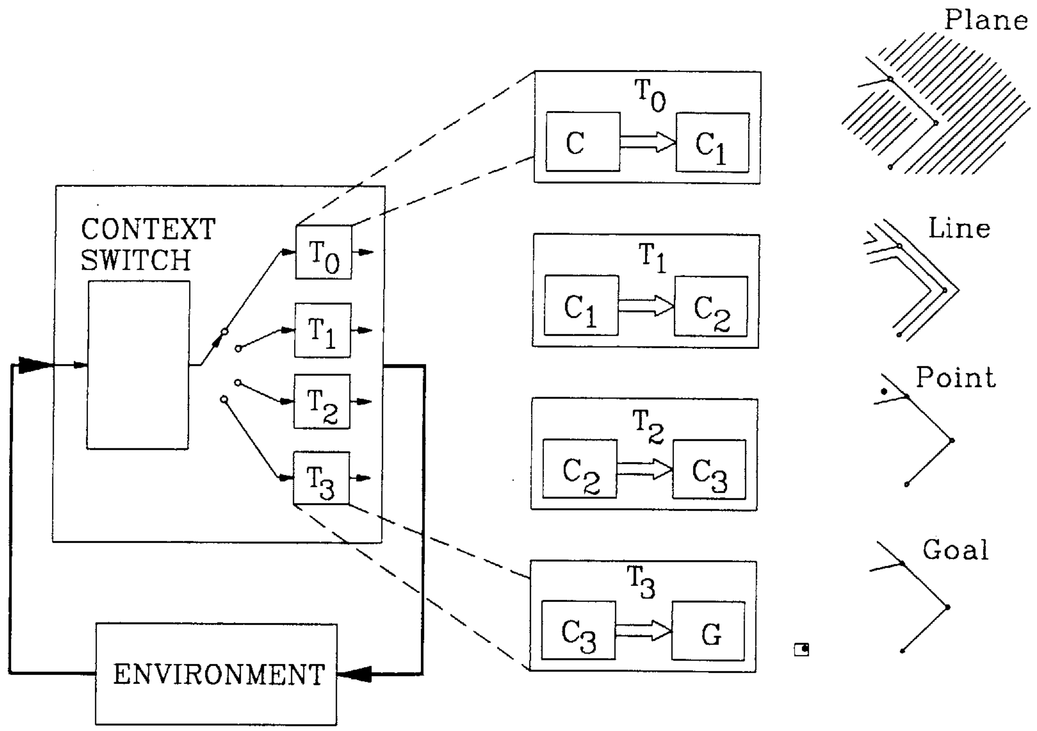


Figure 5-5. The sequence of contexts based on the available sensor information form the basis for the construction of a task-level controller.

Given a sequence of contexts, we must now construct an associated set of actions which move the system from one context to the next. In the first case, a single contact can be acquired by extending the arm and rotating it through the workspace. Once a contact is established, the position of the disk is now known to lie along a line adjacent to the associated link. This context can be partitioned into smaller ones, in which the disk is known to lie next to a particular link, as shown in figure 5-6. By moving the hand through line possible disk positions, we guarantee an additional contact, and thus transition to the subsequent context. The establishment a second contact constrains the known disk position to a point, and by closing the fingers about this point we guarantee a grasp. Finally, with the disk in the grasp, the manipulator can simply move to the goal region, release the object, and move to the null position. Since every step involved an increase in knowledge of the states or an increase restriction of the known states, we are guaranteed a stable and convergent system.

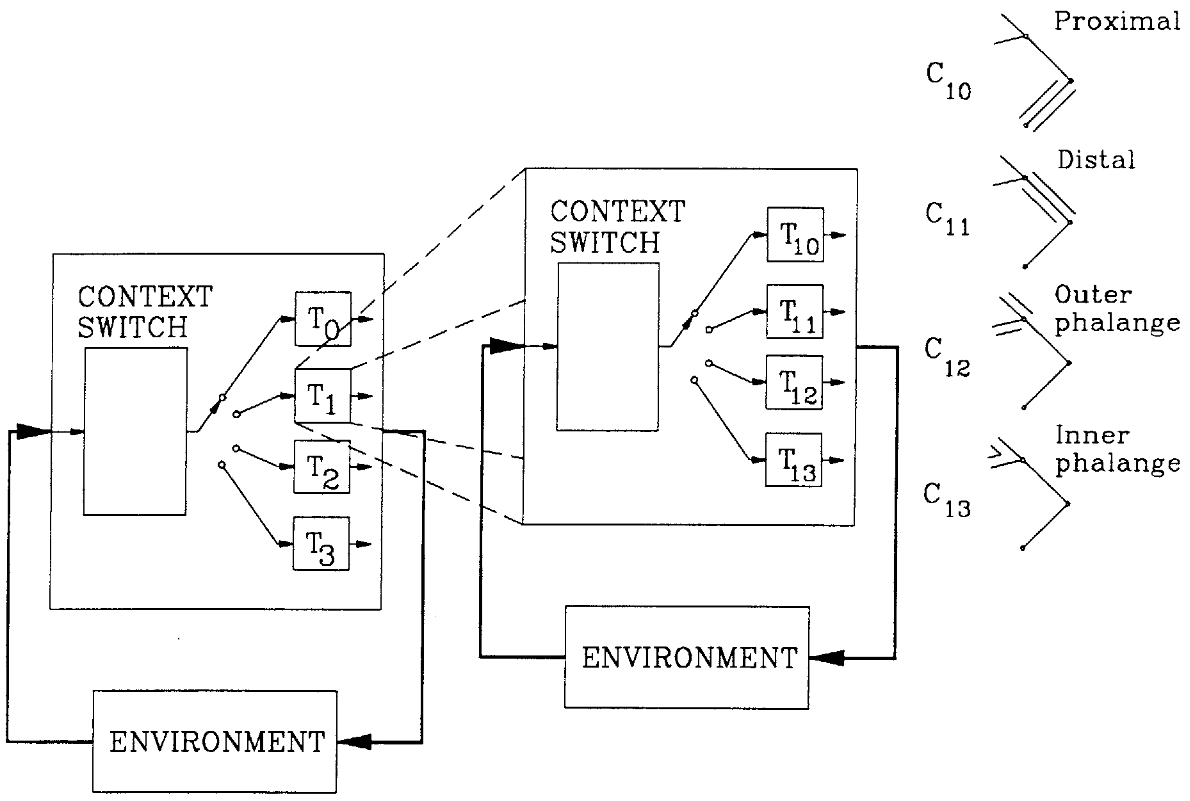


Figure 5-6. The context describing contact with the disk can be partitioned into separate regions corresponding to contact along a particular link.

An important point, is that the task-level control system constructs an estimate of the system states in real-time, and this estimate is described by a region or subset of state space, rather than a particular point. This estimate is then compared with sets describing the contexts given above. If the set satisfies the conditions of the context and not the goal of a particular task, then the action associated with that task is executed.

This particular system and the task-level control system described above were simulated, and a time history of a particular run is shown in figure 5-7. The graph includes a plot of the sensory events, the distance of the disk from the goal, and level of knowledge of the system states. Notice that as sensory events occur, the level of knowledge is increased and the appropriate action is executed. The arrows at the top of the chart represent random displacements of the disk. As soon as a discrepancy is recognized between the expected position and the position consistent with the sensor values, the previous information is discarded, and the current interpretation is retained.

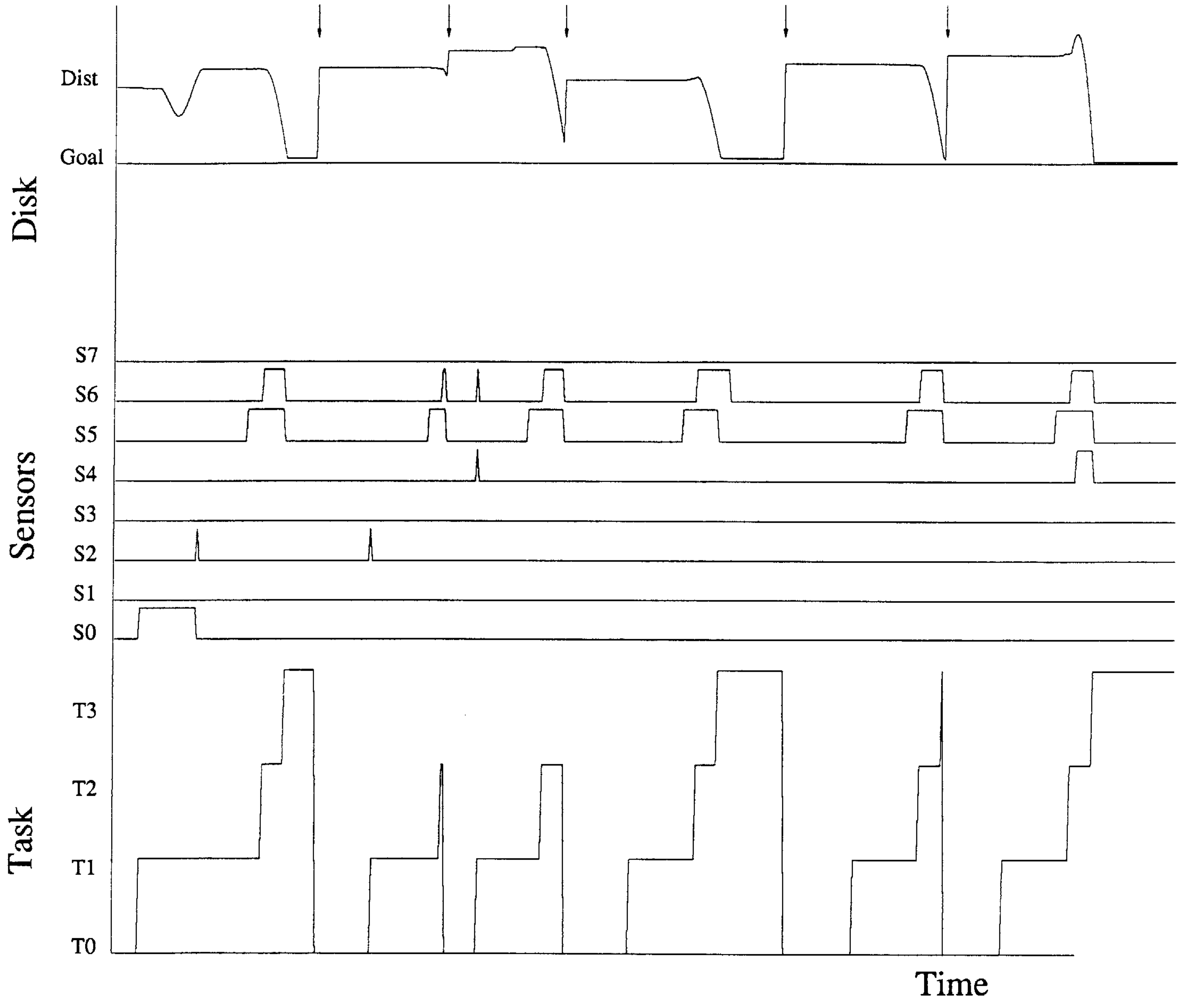


Figure 5-7. A time-line history of the planar hand/arm simulation. The plot shows the distance of the disk from the goal, as well as the sensory events, and the executing tasks. The task correspond to the level of knowledge of the disk location. The arrows at the top of the chart indicate external random disturbances in the position of disk. Notice the corresponding lose of knowledge when the interpretation of thé sensory events conflict with the previous model.

5.3 Example with Robot Hand/Arm System

As a second example, we will consider the problem of grasping a cylinder with Salisbury Robot Hand/PUMA arm system shown in figure 5-8 (Salisbury 1982).

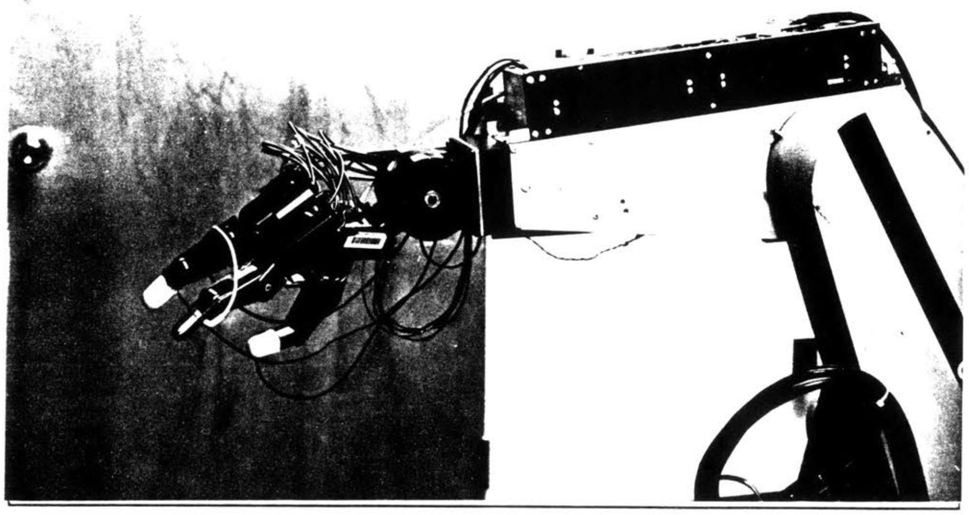


Figure 5-8. The Salisbury Robot Hand with three fingers each with three degrees of freedom is mounted on a six degree of freedom PUMA Robot Arm (Salisbury 1982).

5.3.1 State Space

The state space in this case is 20 dimensional and is described by $S = S^1 \times S^2 \times S^3$, where

$$S^1 = \{(\hat{l}, \tilde{l})\}$$

describes the position of the cylinder in Plücker line coordinates. [1],

$$S^2 = \{[\ j_1 \quad j_2 \quad j_3 \quad j_4 \quad j_5 \quad j_6\ ]^T,$$

describes the position of the arm, and

$$S^3 = \left\{ \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \right\}$$

the position of the hand.

[1]The first three elements $\hat{l}$ represent the normalize direction of the line and the last three $\tilde{l}$, the moment of the line about the origin of the reference system.

5.3.2 Available Sensors

Since the contexts on which the task-level controller depends are derived directly from the sensing capability of the physical hardware, we must first consider the sensors available on the robot. The Salisbury Hand/PUMA Arm system has a number of sensors including joint position, joint torque, fingertip, palm, and phalange sensors. The force/torque based fingertip shown in figure 5-9 can resolve the magnitude, direction, and location of an applied force (Brock 1985, Bicchi 1992). Binary contact sensors, shown in figure 5-10, were constructed as a less expensive alternative to the intrinsic force sensors. These sensors, however, provide less specific information resulting in a larger interpretation region. Phalange sensors based on PTFE, a piezoelectric polymer, have recently been constructed (Morrell 1993), figure 5-11. These sensors provide transient force information, which can be used to determine the formation and termination of contact. As with the binary contact sensors, however, the phalange sensors can determine only a region of contact. Finally, a contact sensing palm was constructed, which consists of a curved plate supported by mechanical springs, figure 5-12. Four infrared emitter/detector pairs measure the displacement of the corners of the plate and thus deduce the approximate location of contact (Brock 1990).

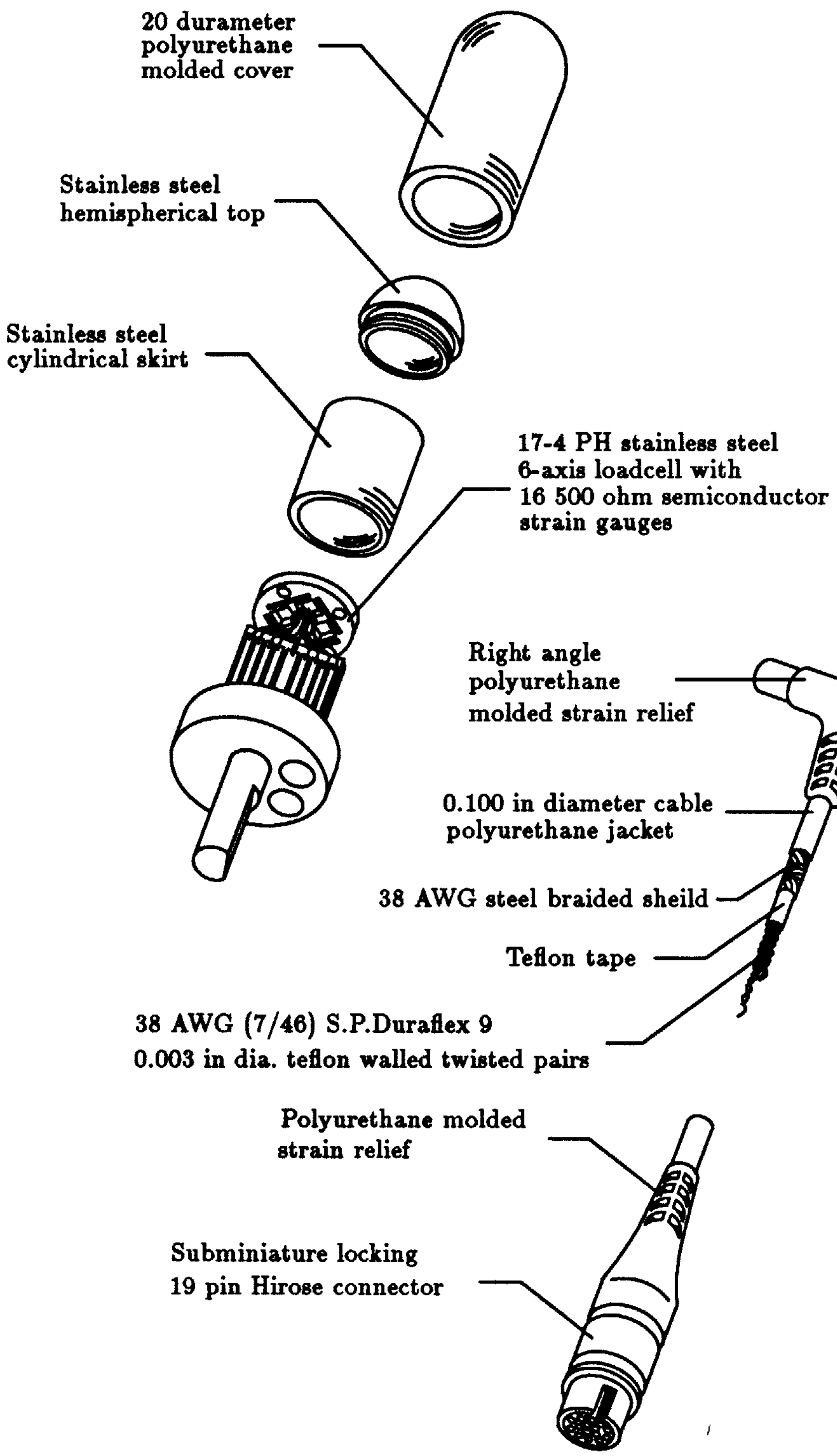

20 durameter polyurethane molded cover
Stainless steel hemispherical top
Stainless steel cylindrical skirt
17-4 PH stainless steel 6-axis loadcell with 16 500 ohm semiconductor strain gauges
Right angle polyurethane molded strain relief
0.100 in diameter cable polyurethane jacket
38 AWG steel braided sheild
Teflon tape
38 AWG (7/46) S.P.Duraflex 9 0.003 in dia. teflon walled twisted pairs
Polyurethane molded strain relief
Subminiature locking 19 pin Hirose connector

Figure 5-9. The force/torque base fingertip sensor can determine the magnitude, the location, and the direction of a force applied on the surface through a point contact (Brock 1989). Small semiconductor strain gauges mounted on small steel flexure are used to resolve the forces and moments on the fingertip shell.

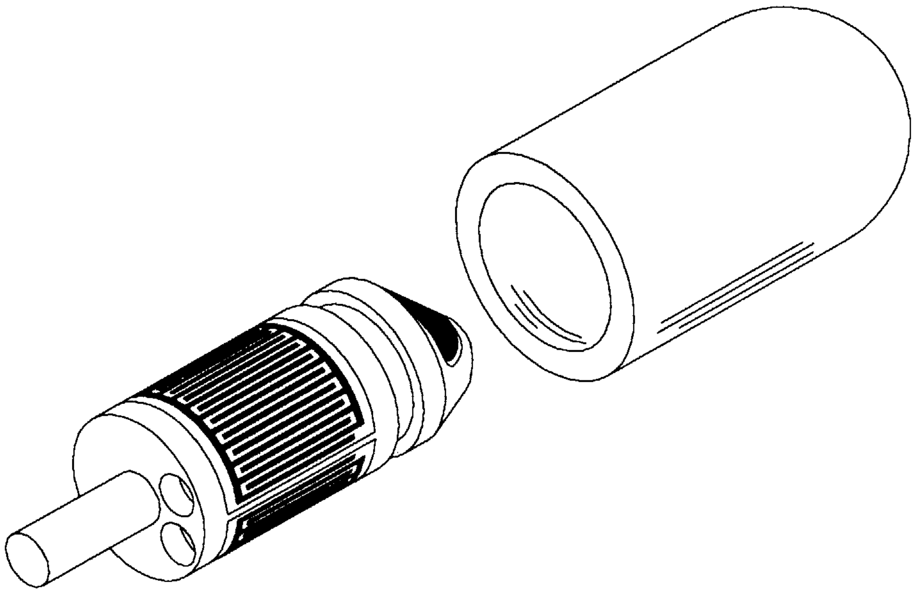


Figure 5-10. A binary contact sensor, based on commercial contact sensors, provides simple contact information which is useful for simple task-level control systems.

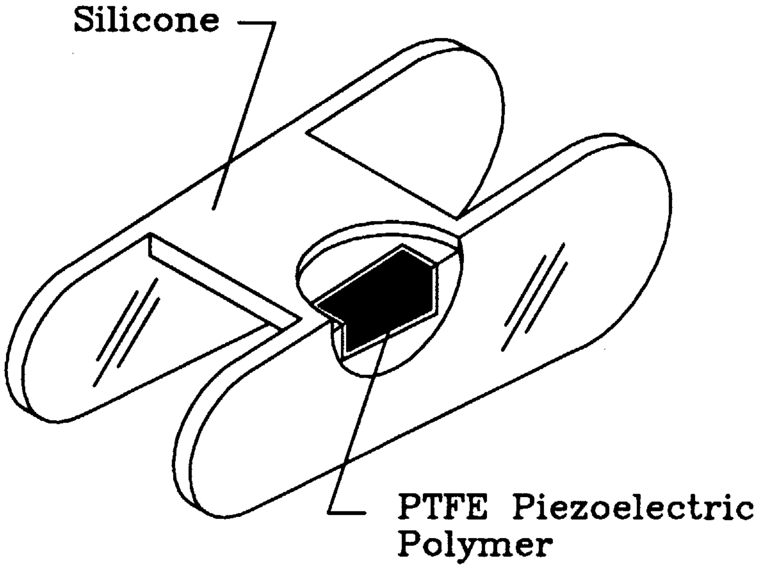


Figure 5-11. Phalange sensors based on PTFE, a piezoelectric polymer, provide transient force information, which can be used to determine the formation and termination of contact.

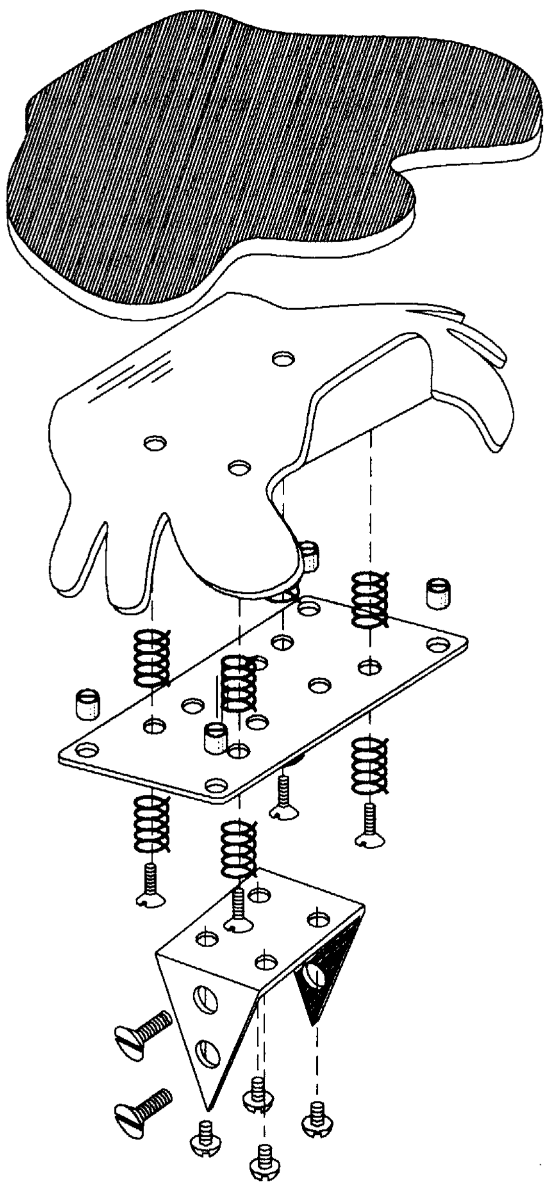


Figure 5-12. Optical range sensors detect the position of the palmar surface supported by simple mechanical springs.

5.3.3 Initial Context and Goal

We assume the cylinder lies within the workspace of the hand and that its position is initially unknown. Thus the context is described by five dimensions, representing the unknown values of the position of the cylinder. The goal in this case is a stable grasp; however there are a number of robot/cylinder configurations which produce equally acceptable force closure grasps. We could, for example, grasp the cylinder at the fingertips, between a pairs of opposing phalanges, or around the entire circumference.

While there are many solutions to this problem, the objective is to derive a sufficient solution and then develop a set of actions which generate this solution. Therefore we will specify an enclosure or "wrap" grasp as the objective of this task-level control system.

In addition to the various arrangements of the manipulator which produce force closure grasps, there are also multiple configurations resulting from the symmetry of the task. Since the cylinder possesses radial, linear and reflective symmetry, a grasp anywhere along its length, about its circumference, or in either direction (palm up or palm down) are equally acceptable. Workspace boundaries and singularities will of course place limit the possible configurations, but, at the least, symmetry considerations provide a greater range of possible solutions. In addition, considering small motions within the boundaries of the workspace, the symmetry of the task can reduce the dimensionality of the problem and ease the complexity of control system design and analysis. Also, in keeping with the idea of a semi-structured environment, we will generally consider objects with at least some degree of symmetry. Thus the example presented here can serve as a basis for the analysis of other tasks.

Taken together, multiple configurations and symmetry produce an equivalent set of acceptable solutions for both the robot and object. For our example, we will consider an enclosing grasp in which the fingers are wraped around the surface. Given radial, linear, and reflective symmetry this produces an equivalent set of configurations and defines the goal region as

$$G = \{\mathbf{p}, \mathbf{h}, \mathbf{l}\},$$

where $\mathbf{h}$ is the hand coordinate frame given by

$$ {}^{R}_{H}T = \begin{bmatrix} {}^{R}\hat{X}_{H} & {}^{R}\hat{Y}_{H} & {}^{R}\hat{Z}_{H} & {}^{R}P_{HORG} \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

where ${}^{R}P_{HORG}$ is the origin of the hand frame,

$$ {}^{R}P_{HORG} = {}^{R}_{L}T\,{}^{L}P, $$

where ${}^{R}_{L}T$ is the transformation from a line coordinate system given by

$$ {}^{R}_{L}T = \begin{bmatrix} {}^{R}\hat{X}_{L} & {}^{R}\hat{Y}_{L} & {}^{R}\hat{Z}_{L} & {}^{R}P_{LORG} \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

where

$$ \begin{aligned} {}^{R}\hat{X}_{L} &= \tilde{l}/\|\tilde{l}\| \\ {}^{R}\hat{Y}_{L} &= \hat{l} \times \tilde{l}/\|\tilde{l}\| \\ {}^{R}\hat{Z}_{L} &= \hat{l} \\ {}^{R}P_{LORG} &= \hat{l} \times \tilde{l} \end{aligned} $$

and ${}^{L}P$ is an arbitrary point on the surface of the cylinder

$${}^{L}P = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ z \end{bmatrix},$$

where θ is some angle about the z-axis in the line coordinate frame and z is an arbitrary point along its length. Finally, the axes of the hand coordinate frame are given by

$${}^{R}\hat{X}_H = \pm\hat{\imath} \quad {}^{R}\hat{Y}_H = \begin{bmatrix} -\cos(\theta) \\ -\sin(\theta) \\ 0 \end{bmatrix} \quad {}^{R}\hat{Z}_H = {}^{R}\hat{X}_H \times {}^{R}\hat{Y}_H.$$

Note the positive or negative orientation allowed for ${}^{R}\hat{X}_H$ due to the reflexive symmetry of this system. The goal in the example is a pair of two dimensional spaces, parameterized by an angle about the cylinder, a position along its length and a boolean value specifying the orientation of the palm.

5.3.4 Constructing Contexts from Sensor Data

Proceeding, as in the planar case, a single contact on the fingertip sensor, resolves a single contact point and contact normal, thus providing partial constraint on the position of the cylinder. Given the contact location and normal direction and the position of the robot from the joint angle sensors, we can determine a point through which the axis of the cylinder must pass, as shown in figure 5-13. The the context is describe by a set of knowledge spaces given by $K = K^1 \times K^2 \times K^3$, where $K^2 = \{\mathbf{p}\}$, $K^3 = \{\mathbf{h}\}$, and $K^1 = \{l\}$, where

$$l = {}^{R}l = {}^{R}_{C}P\,{}^{C}l = \begin{bmatrix} {}^{R}_{C}R & 0 \\ {}^{R}_{C}X\,{}^{R}_{C}R & {}^{R}_{C}R \end{bmatrix} {}^{C}l,$$

where ${}^{R}_{C}R$ is the 3×3 rotation matrix from the contact coordinate frame C to the reference coordinate frame R, ${}^{R}_{C}X$ is the 3×3 antisymmetric matrix which yields the cross product of the origin of R with respect to C with ${}^{R}_{C}R$, and ${}^{C}l$ is the line defined with respect to the contact coordiante frame,

$${}^{C}l = \frac{1}{\sqrt{x^2+y^2}} \begin{bmatrix} x & y & 0 & -yR & xR & 0 \end{bmatrix}^T,$$

where (x, y) is the contact point defined in the contact coordinate frame. Finally, a second contact uniquely constrains the position of the cylinder, as shown in figure 5-14.

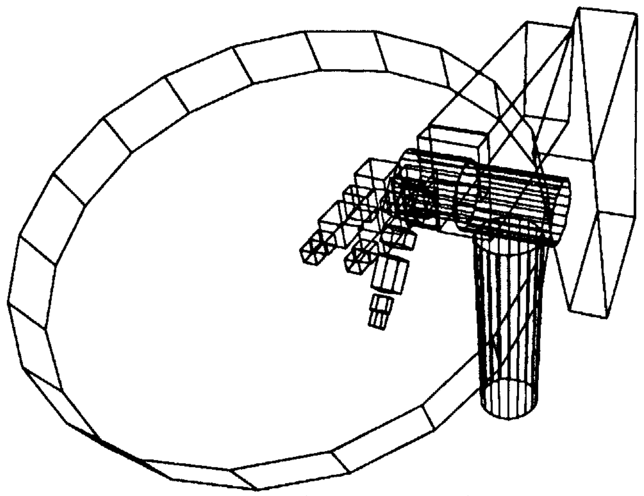


Figure 5-13. A single contact reduces the dimension of the context from five to one; that is the direction of the cylinder in the plane has yet to be defined.

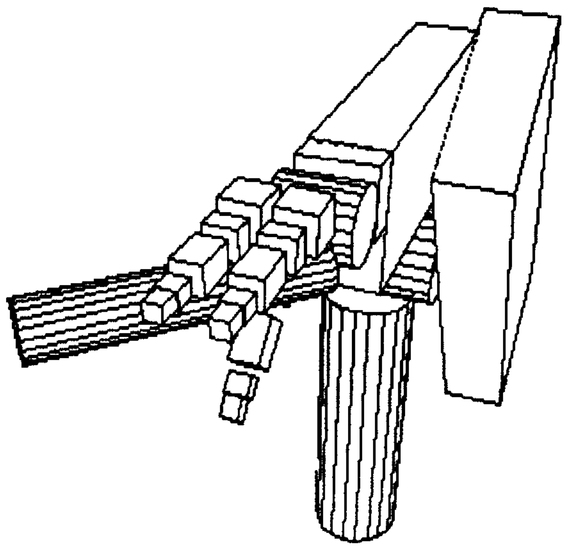
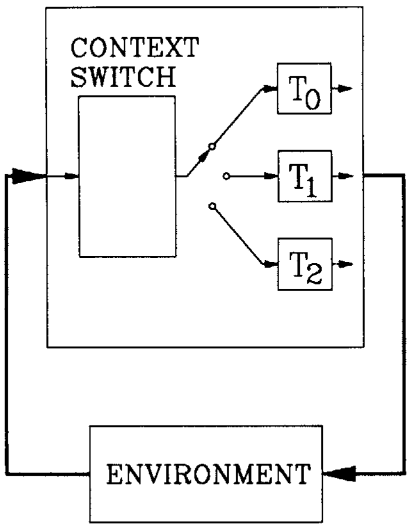


Figure 5-14. Two contacts constrain the position of the cylinder.

5.3.5 Task Sequence

Since the robot is unaware of the position of the cylinder, the only possible action is to search the space of possible positions. The establishment of a single contact, however, partially constrains the position of the cylinder. At this point the robot could probe in a region surrounding the contact in order to acquire a second point and thus fix the location of the cylinder. Alternatively, the robot could move in a more efficient manner to not only acquire a second contact, but to also reduce the dimensionality of the task and partially achieve the goal. If the hand is moved into the space of possible positions and aligned with the plane defining the partial constraint, the robot is guaranteed to simultaneously establish an additional contact while partially achieving the task. With a second contact achieved, the palm can move directly to alignment with the cylinder while closing the fingers around the circumference. The sequence of tasks described above is illustrated in figure 5-15.



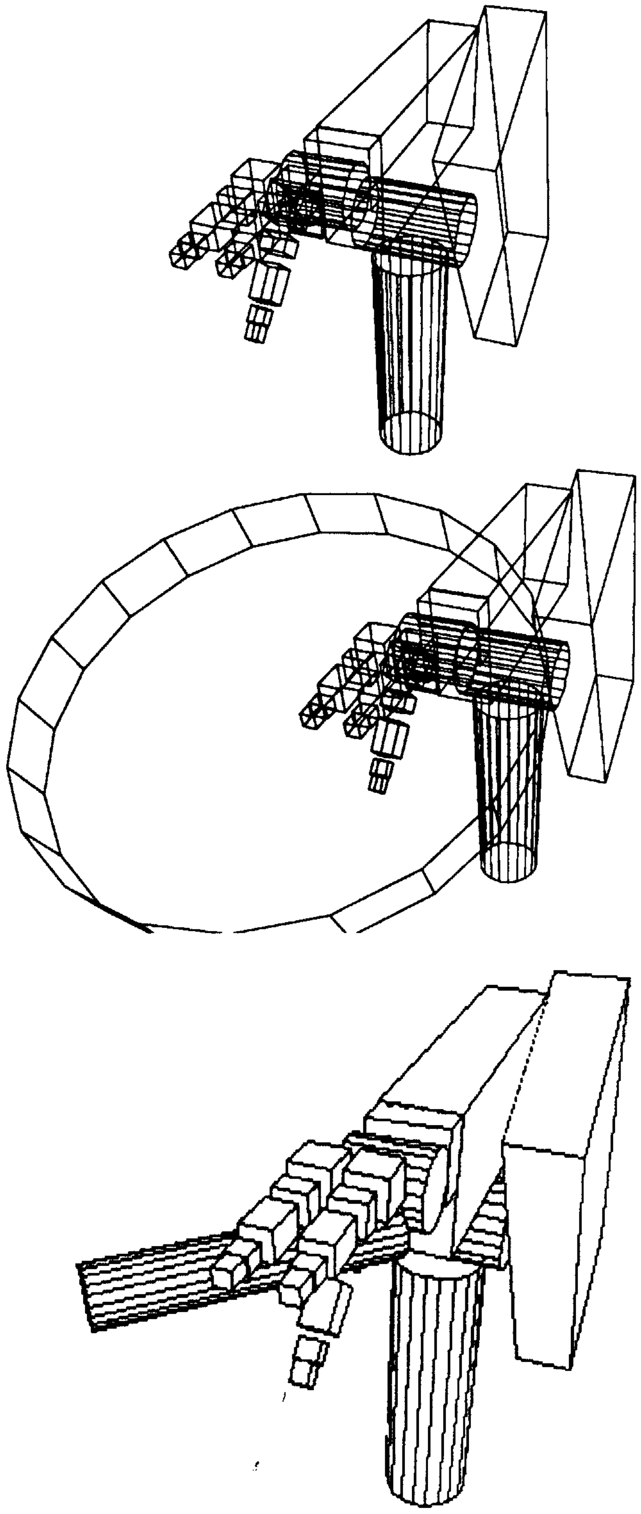


Figure 5-15. Acquisition of the cylinder is possible with a sequence of three tasks representing either an increase in knowledge of the object position or an increase constraint on the known states.

Chapter 6

Conclusion

In this thesis, we have developed a task-level control system which can respond immediately to perceived changes in the environment and generate the necessary action to achieve the goal. Rather than simple error detection and correction, this ability to react is an intrinsic part of the task-level control structure. The system also uses explicit models of both the robot and the environment, thus allowing precise proofs of performance, as well as analytic tools for the design of robot action. Also fundamental in the definition of the task-level control system is a representation of the actual information available from the physical sensing hardware. This allows the design of the task-level control system to be coupled directly to the sensing ability of the physical robot. In this thesis, we have developed the theoretical basis for the task-level control system, as well as practical implementations in both simulation and physical hardware.

6.1 Review

First, we developed a general theory for the task-level control. We partitioned the control of a robot into a number of discrete *tasks* representing specific goals to be achieved for specific arrangements of the environment. Each task included a *context*, that is a specific composition of the environment as it is known to the robot, a *goal*, that is a desired objective to be achieved within that context, and an *action*, that is either a further sequence of tasks, or a robot trajectory. We then developed criteria for the tasks which guarantee stability and convergence. Second, a computational architecture was developed based on the theoretical development, in which individual task modules execute a specific action based on the output of the context module. The context module is a computation element, associated with a each task, which constructs the current knowledge of the system states using a hierarchical organization of perceptual units. These perceptual units form a dependent hierarchy which, at the lower-levels, include the physical sensors and raw data processing, and, at the higher-levels, feature abstraction and environmental modeling. Third, we presented tools for the analysis and design of the task-level control system and were applied primarily to the task of robotic grasping. Fourth, we implemented a task-level control system in both simulation and physical hardware.

6.2 Contributions

- **Task-level Control System** The main result of this thesis is a task-level control architecture which can respond immediately to perceived changes in the environment, as well as maintain persistent and rational sequences of action. The system also employs explicit models of both the robot and the environment, thus allowing exact proofs of stability and convergence. Inherent in the definition of the task-level control system is an exact representation of the limited information available from actual physical sensing systems. This feature is critical, since it allows rapid implementation of task-level control systems on physical hardware.

- **Stability and Convergence Criteria** Conditions were developed which guarantee the task-level control system remains stable and converges to the goal. Traditionally the development of stability criteria and proofs of performance were difficult for task-level reactive systems. However, the introducing explicit models of the partial information gained from the limited sensing hardware, allowed these stability and convergence conditions to be developed. These conditions were based on the requirement that the resulting robot action either increase the knowledge of the system states or further restricts the allowed variance of the known states.

- **Design Tools** Geometric design tools were introduced, based on the partial environmental models, to allow robot actions to be constructed which satisfy the conditions for stability and convergence. These tools now clarify the design of previous ill-defined problems in the intelligent control of robot manipulation.

- **Software Architecture** A software architecture was developed for the task-level control system which allows incorporation of multiple disparate sensors, feature interpretation algorithms, object representations, and planning procedures.

6.3 Future Research

The development of the task-level control system suggests a number of areas for further research. These areas were roughly divided into theoretical issues and design tools, and robotic hardware and sensory systems.

6.3.1 Theoretical Issues and Design Tools

- **Symmetry** It was observed that many objects, as well as the robots which manipulate them, display a significant degree of geometric symmetry. This symmetry could be used explicitly to reduce the complexity of path planning and object recognition. The symmetry of the objects, robots, and their interaction can be represented by algebraic symmetry groups (Poppelstone 1987). Robot trajectories can then be generated in the symmetric subspaces, thus reducing the dimension of the path planning problem. These ideas could be applied not only to trajectory generation, but also to robot construction and sensor configurations

- **Multiple Objects** In this thesis, it was assumed that the context can be constrained to a single object/robot interaction. However, an area of further research — and one of practical importance — would be the development of task sequences which restrict the contexts to the point where a single object/robot interaction is possible.

- **Multiple Forward Projections** In the practical implementation, we assumed a sensor interpretation inconsistent with the forward projection previous knowledge resulted produced a complete lose of knowledge within a particular context. However, an area of future research would be the application of multiple forward projections, as in equation 2.2,

  $$K^j_{i+1} = \min_k \{F^k_{a_i}(K^j_i) \bigcap I^j_{i+1}\},$$

  where I^j_{i+1} was the current sensor interpretation, and $F^k_{a_i}(K^j_i)$ for $k = 1, \ldots, n$ was a sequence of forward projections. The increasing size of the forward projections represent more general — and less likely — consequences of the robot action.

- **Task-level Kalman Filter** In the implementation, the forward projection of the previous knowledge space was simply intersected with the current sensory interpretation. This process was analogous to a generalized Kalman filter. However, it may be possible to apply the analogy further by actually a performing a weighted intersection of the forward projection and interpretation based on the statistics of the measured state.

- **Sensor Fault Tolerance** In this thesis, we assume the sensor always provide data accuracy to with some undetermined bounded error. However, as in the case of binary contact sensors, aberrant sensor data is indistinguishable from random environmental disturbances. Thus the incorporation of probabilistic models could be applied to resolution of sensor error from environmental disturbance, thus providing essentially sensor fault tolerance.

- **Search strategies** Many of the actions performed by the robot involved increasing the knowledge of the systems states. These actions were usually performed by a direct and deterministic motion. However, it may be possible to apply random search techniques for increasing state knowledge, allow more rapid statistical convergence than the deterministic method (Erdmann 1990).

6.3.2 Robotic Hardware and Sensory Systems

The task-level control system allowed the robot to respond immediately to perceived changes in the environment. However, to perceive these changes and react to them, the robot requires a greater extent of both sensing hardware and manipulation surfaces. To explore the possibility of extending the manipulation surfaces, a prototype robotic fingers were developed which presented compliant surfaces and sensing over the entire exterior, appendix A.

6.4 Conclusion

This thesis was motivated by three observations. The first was of a robot, which after having dropped a part accidentally, continued to move, vacously inserting the phantom object into a fixture. It seems reasonable that the robot should respond appropriately to any situation — either catastrophic or serendipitous. The second, was that in many domains, objects have regular and uniform geometries. Thus it was desired to create a task-level robot control system which exploits these characteristics. Finally, in agreement with behavior based systems (Brooks 1986), it seems that both humans and animals employ different strategies in disparate situations. This implies a granularity of robot contexts and tasks necessary for effective operatation in practical environments.

REFERENCES

1. Abel, J.M., Holzmann, W., and McCarthy, J.M., "On Grasping Planar Objects with two Articulated Fingers", *Proceedings of the IEEE International Conference on Robotics and Automation,* St. Louis, MS, March 1985, p. 576-581.

2. Ambler, A.P. and Popplestone, R.J., "Inferring the positions of bodies from specified spatial relationships," *Artificial Intelligence*, Vol. 6, No. 2, pp. 157-174, 1975.

3. Angle, C., "Design of an Artificial Creature," S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1991.

4. Angeles, Jorge, *Spatial Kinematic Chains,* Springer-Verlag, New York, 1982.

5. Arimoto, S., et. al., "A Feasible Approach to Automatic Planning of Collision-Free Robot Motions", *Proceedings of the International Society of Robotics Research*, 1987, p. 163-172.

6. Arkin, R.C., "Intergrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation," *Designing Autonomous Agents*, ed. Maes, P., 1990.

7. Asada, H. and Asari, Y., "The Direct Teaching of Tool Manipulation Skills Via the Impedance Identification of Human Motions," *Proceedings of the IEEE International Conference on Robotics and Automation,* Philadelphia, PN, 1988, pp 1269-1274.

8. Bajcsy, Ruzena and Goldberg, Kenneth Y., "Active Touch and Robot Perception", *Cognition and Brain Theory*, Vol. 2, Summer 1984.

9. ——— and Tsikos, Constantinos, "Perception via Manipulation", *Proceedings of the International Society of Robotics Research*, 1987.

10. Berkemeier, M.D. and Fearing, R.S, "Determining the Axis of a Surface of Revolution Using Tactile Sensing," *Proceedings of the IEEE International Conference on Robotics and Automation,* Cincinnati, OH, May 1990.

11. Bicchi, A.C., Salisbury, J.K., and Brock, D.L., "Experimental Evaluation of Friction Characteristics with an Articulated Robotic Hand," International Conference on Experimental Robotics, June 1991.

12. Bicchi, A.C., Salisbury, J.K., and Brock, D.L., "Contact Sensing from Force Measurements," *International Journal of Robotics Research*, Volume 12, Number 3, June 1993.

13. Binford, Thomas, "Generic Surface Interpretation: Observability Model", ISRR 1987, p. 287-294.

14. Bonivento, C., Faldella, E., and Vassura G., "The U.B. Robotic Hand Project: Current State and Future Developments," ICAR, Pisa, 1991.

15. Bradley, E., "Causes and Effects of Chaos," *MIT AI Memo* 1216, MIT Artificial Intelligence Laboratory, Dec. 1990.

16. Brock, David and Chiu, Stephen, "Environment Perception of an Articulated Hand Using Contact Sensors", ASME Winter Annual Conference, Feb 1985.

17. ——., "Enhancing the Dexterity of a Robot Hand Using Controlled Slip," *Artificial Intelligence Laboratory Technical Report*, 1987.

18. ——, "Enhancing the Dexterity of a Robot Hand Using Controlled Slip," *Proc. of IEEE International Conference on Robotics and Automation*, Philadelphia, PA, April 1988.

19. ——, "Sensing and Control on the Salisbury Robot Hand," Sandia Report SAND88-7159, Sandia National Laboratories, November 1989.

20. ——, "Contact Sensing Palm for the Salisbury Robot Hand," Sandia Report, Sandia National Laboratories, July 1990.

21. —— and Salisbury, J.K., "Implementation of Behavioral Control of a Robot Hand/Arm System," *Proc. of International Conference on Experimental Robotics*, Toulouse, France, June 1991.

22. Brooks, Rodney A., "Solving the Find-Path Problem by Representing Free Space as Generalized Cones," *MIT AI TR* 674, MIT Artificial Intelligence Laboratory, May 1982.

23. —— and Lozano-Pérez, Tomás, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," *MIT AI TR* 684, MIT Artificial Intelligence Laboratory, December 1982, and *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-15, No. 2, March 1985.

24. ——, "Planning Collision Free Motions for Pick and Place Operations," *MIT AI Memo* 725, MIT Artificial Intelligence Laboratory, May 1983.

25. ——, "A Robust Layered Control System for a Mobile Robot," *MIT AI Memo* 864, MIT Artificial Intelligence Laboratory, September 1985.

26. ——, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-2, March 1986.

27. ——, "Achieving Artificial Intelligence Through Building Robots," *MIT AI Memo* 899, MIT Artificial Intelligence Laboratory, May 1986.

28. ——, Connell, J. H., and Flynn, A. M., "A Mobile Robot with Onboard Parallel Processor and Large Workspace Arm," *Proceedings of AAAI-86*, Philadelphia, PA, August 1986, p. 1096-1110.

29. —— ——, "Asynchronous Distributed Control System for a Mobile Robot," *SPIE Proceedings*, Vol. 727, October 1986, p. 77-84.

30. ——, "A Hardware Retargetable Distributed Layered Architecture for Mobile Robot Control," *Proc. IEEE Robotics and Automation, Atlanta, GA,* March 1987.

31. ——, Flynn, A. M. and Marill, T., "Self Calibration of Motion and Stereo Vision for Mobile Robot Navigation," *MIT AI Memo* 984, MIT Artificial Intelligence Laboratory, August 1987.

32. ——, Connell, J. H. and Ning, P., "Herbert: A Second Generations Mobile Robot," *MIT AI Memo* 1016, MIT Artificial Intelligence Laboratory, January 1988.

33. ——, "A Robot that Walks: Emergent Behaviors from a Carefully Evolved Network," *Proceedings of the IEEE International Conference on Robotics and Automation,* Philadelphia, PN, 1988.

34. ——, "A Robot Being," MIT Aritificial Intelligence Laboratory, October 1989.

35. Brost, R. C., "Automatic Grasp Planning in the Presence of Uncertainty," *Proceedings of the IEEE International Conference on Robotics and Automation,* San Fransico, CA, April 1986.

36. Caine, M. E., "Chamferless Assembly of Rectangular Parts in Two and Three Dimensions," S.M. Thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, 1985.

37. Canny, John, "Collision Detection for Moving Polyhedra", *IEEE Transactions of Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 2, March 1986.

38. ——, "Exact Solution of some Robot Motion Planning Problems," *Proceedings of the International Society of Robotics Research*, 1987.

39. —— and Donald, Bruce, "Simplified Voronoi Diagrams," *MIT AI TR* 957, MIT Artificial Intelligence Laboratory, April 1987.

40. ——, *The Complexity of Robot Motion Planning*, ACM Doctoral Dissertation Award 1987, MIT Press, 1988.

41. Chammas, Camille, "Automatic Grasping," NASA Report 1989.

42. ——, "Analysis and Implementation of Robust Grasping Behaviors," *MIT AI TR* 1237, MIT Artificial Intelligence Laboratory, 1990

43. Chiu, Stephen L., "Generating Compliant Motion of Objects with an Articulated Hand", MIT MS Thesis 1985.

44. Connell, Jonathan, "Creature Design with the Subsumption Architecture," *Proceedings of IJCAI-87*, Milan, Italy, August 1987.

45. ——, "A Behavior-Based Arm Controller," *MIT AI TR* 1025, MIT Artificial Intelligence Laboratory, June 1988.

46. ——, "A Colony Architecture for an Artificial Creature," *MIT AI TR* 1151, MIT Artificial Intelligence Laboratory, August 1989.

47. ——, et. al., "Grasping as a Contact Sport", *Proceedings of the International Society of Robotics Research*, 1987, pp. 191-197.

48. Dario, P., et. al., "Tactile Perception in Unstructured Environments: A Case Study for Rehabilitative Robotics Applications," *Proceedings of the IEEE International Conference on Robotics and Automation,* Raleigh, NC, 1987, pp., 1506-1507.

49. Donald, B. R., "Error Detection and Recovery for Robot Motion Planning with Uncertainty," *MIT AI TR* 982, MIT Artificial Intelligence Laboratory, July 1987.

50. Drumheller, Michael, "Mobile Robot Localization Using Sonar", MIT AI Memo 826, Jan 1985.

51. Eberman, B. S. and Salisbury, J. K., Determination of Manipulator Contact Information from Joint Torque Measurements," In Hayward, V., and Khatib, O. (eds.), *Lecture Notes in Control and Information Sciences,* New York, Springer-Verlag, 1990.

52. Eberman, B.S., "Application of Change Detection to Dynamic Contact Sensing," *Proc. of International Conference on Experimental Robotics*, Kyoto, Japan 1993.

53. Erdmann, M. A., "On Motion Planning with Uncertainty," *MIT AI TR* 810, MIT Artificial Intelligence Laboratory, August 1984.

54. ——, "On Probabilistic Strategies for Robot Tasks," *MIT AI TR* 1155, MIT Artificial Intelligence Laboratory, 1990.

55. —— and Mason, M. T., "An Exploration of Sensorless Manipulation," *IEEE Journal of Robotics and Automation*Vol. 4, No. 4, pp. 369-379.

56. Ernst, H.A., "MH-1, A Computer-Operated Mechanical Hand." Sc.D. Thesis, MIT, Department of Electrical Engineering.

57. Fearing, Ronald S., "Simplified Grasping and Manipulation with Dextrous Robot Hands", MIT AI Memo 809, Nov, 1984.

58. ——, "Implementing a Force Strategy for Object Re-orientation," *Proceedings of the IEEE International Conference on Robotics and Automation,* San Fransico, CA, April 1986.

59. Franklin, W.R. and Barr, A.H., "Faster Calculation of Superquadric Shapes," *IEEE Journal of Computer Graphics*, July 1991.

60. Flynn, A. M. and Brooks, R. A., "MIT Mobile Robots - What's Next?" *Proceedings of the IEEE International Conference on Robotics and Automation,* Philadelphia, PN, 1988.

61. Gaston, Peter C and Lozano-Pérez, "Tactile Recognition and Localization Using Object Models: The Case of Polyhedra on a Plane," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-6, No. 3, May 1984, p. 257-266.

62. Gesell, Arnold, *Infant Development - The embryology of early human behavior* Greenwood Press, Westport, CT, 1952.

63. Goldberg, K., "Probabilistic Grasping Strategies," Ph.D. School of Computer Science, Carnegie Mellon University, 1989.

64. Gordon, Steven J. and Townsend, William T., "Integration of Tactile Force and Joint Torque Information in a Whole-Arm Manipulator," Proc. IEEE International Conference on Robotics and Automation, Scottsdale AR, April 1989.

65. Gould, James L., Ethology - The Mechanisms and Evolution of Behavior, W.W. Norton and Company, New York, 1982.

66. —— and Marler, Peter, "Learing by Instinct", Scientific American, pp. 74-85, January 1987.

67. Greiner, Helen, "Passive and Active Grasping with a Prehensile Robot End-Effector," MIT Master Thesis 1990.

68. Grimson, Eric L. and Lozano-Pérez, T., "Model-Based Recognition and Localization from Sparse Range or Tactile Data," *IEEE Journal of Robotics and Automation*, Vol. 3, No. 3, Fall 1984, p. 3-35.

69. —— ——, "Recognition and Localization of Overlapping Parts from Sparse Data", MIT AI Memo 841, June 1985.

70. Grupen, Roderic A., "Behavior based control for autonomous robotic manipulation" University of Massachusetts, Dec. 13, 1988.

71. Hillis, Daniel W. "Intelligence as an Emergent Behavior; or, The Songs of Eden", Daedalus Journal of the American Academy of Arts and Sciences, Winter 1988, pp. 175-190.

72. Horn, Bertold K. P., and Ikeuchi, Katsushi, "Picking Parts Out of a Bin", MIT AT Lab Memo No. 746., 1983.

73. Howe, R. D. and Cutkosky, M. R., "Touch Sensing for Robotic Manipulation and Recognition," in Khatib, O., Craig, J., Lozano-Pérez, T., (eds.), *Robotics Review 2*, Cambridge, MA, MIT Press, pp. 55-112.

74. Iberall, Thea, "The Nature of Human Prehension: Three Dextrous Hands in One", *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, p. 396-401.

75. Jacobsen, Stephen C., et., al., "Behavor Based Design of Robot Effectors", Center for Engineering Design, Department of Mechanical Engineering, University of Utah, 1987.

76. Jameson, J. W., "Analytic Techniques for Automatied Grasps," PhD Thesis, Stanford University, January 1985.

77. Jones, Joseph L., "The Handey Grasp Planner," MIT Memo, May 1988.

78. Kawabe, Shinji, Okano, Akira, and Shimada, Kenji, "Collision Detection among Moving Objects in Simulation", *Proceedings of the International Society of Robotics Research*, 1987, p. 173-179.

79. Laugier, C., "A Program for Automatic Grasping of Objects with a Robot Arm," *Proceedings of the 11th International Symposium on Industrial Robots,* 1981.

80. Li., A. and Sastry, S., "Optimal Grasping by Multifingered Robot Hands," *Proceedings of the IEEE International Conference on Robotics and Automation,* Raleigh, NC, 1987.

81. Livingstone, Margaret S., "Art, Illusion and the Visual System", Scientific American, pp. 78-85., Jan 1988.

82. Livingstone, Margaret, and Hubel, David, "Segregation of Form, Color, Movement, and Depth: Anatomy, Physiology, and Perception", Science, May 6, 1988, Vol 240, P. 740-749. 1989

83. Lozano-Pérez, Tomás, "The Design of a Mechanical Assembly System," *MIT AI TR* 397, MIT Artificial Intelligence Laboratory, 1976.

84. —— and Wesley, M. A., "An Algorithm for Planning Collision Free Paths Among Polyhedral Obstacles," *Communications of the ACM,* Vol. 22, No. 10, October 1979, p. 560-570.

85. —— , "Spatial Planning with Polyhedral Models," PhD Thesis, MIT Department of Computer Science and Electrical Engineering, June 1980.

86. —— , "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man, and Cybernatics,* Vol. SMC-11, No. 10, October, 1981, p. 681-689.

87. ——, Mason, M. T., and Taylor, R. H., "Automatic Synthesis of Fine-Motion Strategies for Robots," *International Journal of Robotics Research,* Vol. 3, No. 1, Spring 1984.

88. —— , et. al, "Handey: A Task-Level Robot System," *Proceedings of the International Society of Robotics Research,* 1987, p. 123-130

89. —— , et. al., "Task-Level Planning of Pick-and-Place Robot Motions," *IEEE Computer,* March 1989, p. 21-29.

90. Markenscoff, X. and Papadimitriou, C. H., "Optimum Grip of a Polygon," *International Journal of Robotics Research*Vol. 8. No. 2., April 1989.

91. Mason, M. T., "Manipulator Grasping and Pushing Operations," *MIT AI TR* 690, MIT Artificial Intelligence Laboratory, June 1982.

92. Mataric, Maja J., "Qualitative Sonar Based Environment Learning for Mobile Robots," *Proceedings of the SPIE Conference on Mobile Robots,* October 1989.

93. Mataric, Maja J., "A Distributed Model for Mobile Robot Environment Learing and Navigation," *MIT AI TR* 1228, MIT Artificial Intelligence Laboratory, 1991.

94. McCarragher, "A Discrete Event Dynamic Systems Approach to Robotic Assembly Tasks," MIT MS Thesis 1992.

95. Oppenheim, A. V., and Willsky, A. S., with I. T. Young, *Signals and Systems,* Prentice-Hall, Englewood Chiffs, NJ, 1983.

96. Nicholls, Howard R. and Lee, Mark H., "A Survey of Robot Tactile Sensing Technology," Internation Journal of Robotics Research, Vol. 8, No. 3, June 1989, pp. 3-30.

97. Nguyen, Van-Duc, "The Synthesis of Stable Force-closure Grasps," *MIT AI TR* 905, MIT Artificial Intelligence Laboratory, July 1986.

98. —— , 'The Synthesis of Stable Grasps in the Plane," *Proceedings of the IEEE International Conference on Robotics and Automation,* San Fransico, CA, April 1986.

99. —— , "Constructing Force-Closure Grasps," *Proceedings of the IEEE International Conference on Robotics and Automation,* San Fransico, CA, April 1986.

100. —— , "Constructing Stable Grasps in 3-D," *Proceedings of the IEEE International Conference on Robotics and Automation,* Raleigh, NC, 1987.

101. ——, "Constructing Force-Closure Grasps in 3-D," *Proceedings of the IEEE International Conference on Robotics and Automation,* Raleigh, NC, 1987.

102. Payton, D.W., "Internalized Plans: A representation for action resources," *Designing Autonomous Agents*, ed. Maes, P., 1990.

103. Pertin-Troccaz, Jocelyne, "On-line Automatic Programming: A Case Study in Grasping," *Proceedings of the IEEE International Conference on Robotics and Automation,* Raleigh, NC, 1987.

104. Pollard, Nancy S., "The Grasping Problem: Toward Task-Level Programming for an Articulated Hand," MIT Masters Thesis, May 1989.

105. Poppelstone, Robin, "Group Theory and Robotics," 1987.

106. Raibert, Marc, et. al, "Dynamically Stable Legged Locomotion," *MIT AI TR* 1179, MIT Artificial Intelligence Laboratory, 1989.

107. Richards, Whitman, Koenderink, Jan J., and Hoffman, D.D., "Inferring 3D Shapes from 2D Codons", MIT AI Memo 840, Apr 1985.

108. Salisbury, Kenneth J. and Craig, John J., "Articulated Hands: Force Control and Kinematic Issues," *International Journal of Robotics Research*, Vol. 1., No. 1, Spring 1982, pp. 4-17.

109. ——, "Kinematic and Force Analysis of Articulated Hands", Ph.D. Thesis, Stanford University, July 1982.

110. ——, Brock, David, and Chiu, Stephen, "Intergrating Language, Sensing, and Control for a Robot Hand", *Proceedings of the International Society of Robotics Research*, 1985.

111. ——, "Whole Arm Manipulation," *Proceedings of the International Society of Robotics Research*, 1987.

112. ——, et. al., "The Design and Control of an Experimental Whole-Arm Manipulator," *Proceedings of the IEEE International Conference on Robotics and Automation,* Philadelphia, PN, 1988.

113. Schenker, P., "NASA Research and Development for Space Telerobotics," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 24, No. 5, Sept 1988, pp. 523-534.

114. Simmons, R. and Krotkov, E., "An Integrated Walking System for the Ambler Planetary Rover," *Proceedings of the IEEE International Conference on Robotics and Automation,* Sacremento, CA, April 1991, pp. 2086-2091.

115. Stansfield, S. A., "Reasoning About Grasping", AAAI-88.

116. ——, "Robotic Grasping of Unknown Objects: A Knowledge-Based Approach," Sandia Report SAND89 - 1087, June 1989.

117. ——, "Knowledge-based Robotic Grasping," *Proceedings of the IEEE International Conference on Robotics and Automation,* Cincinnati, OH, May 1990, p. 1270-1275.

118. Steward, W. K., "A Model-Based Approach to 3-D Imaging and Mapping Underwater," *Proceedings of the ASME Conference on Offshor Mechanics and Artic Engineering,* Vol. 6, pp. 61-71, Houston, TX, February 1988.

119. Tomovic, R., Bekey G. and Karplus, W., "A Strategy for Grasp Synthesis with Multifingered Robot Hands," *Proceedings of the IEEE International Conference on Robotics and Automation,* Raleigh, NC, 1987.

120. Townsend, William T., "The Effect of Transmission Design on Force-Controlled Manipulator Performance," Ph.D. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1988.

121. Villarreal, A. and Asada, H., "A Geometric Representation of Distributed Compliance for the Assembly of Flexible Parts," *Proceedings of the IEEE International Conference on Robotics and Automation,* Sacremento, CA, April 1991.

122. Weyl, Hermann, *Symmetry,* Princeton Science Library, 1952.

123. Whitney, D. E., "Quasi-Static Assembly of Compliantly Supported Rigid Parts", Journal of Dynamic Systems, Measurement, and Control, Vol. 104, March 1982, p. 65-77.

124. Zeki, S. and Shipp, S., "The functional logic of cortical connections", Nature Vol. 335, pp. 311-317, Sept 22, 1988.

Appendix A

Prototype Robotic Hardware

In order to accomodate unforeseen changes in the environment, it was clear the robot must manipulate objects with other surfaces besides the fingertips. For this reason a prototype finger was constructed which presented compliant on the entire exterior, figure A-1. The finger was constructed from thin walled steel, and maintained same kinematic structure and cable routing scheme as the Salisbury Robot Hand. The covering was composed of a foam core and latex skin, containing various viscous fluids, producing a compliant, viscoelastic surface with similar force/displacement characteristics of a human finger, figure A-2. This rheological finger provided a number of advantages including tendon lubrication, load distribution, comformability and mechanical interlock around contacting objects, and energy dissipation, as well as continuous and uniform surfaces. Although the rheological finger provided convenient mechanical surfaces for manipulation, it was not designed for active control or contact sensing.

A second prototype was constructed which provided both compliant exteriors and contact sensing, as shown in figures A-3 to A-6. Again the finger maintains the same kinematic structure, cable routing, and mechanical freedoms as the Salisbury Robot Hand, though with the addition of compliant surfaces, intrinsic force/torque sensing (Brock 1985; Bicchi 1989, 1992, Bonivento 1991), and coarse tactile sensing, figure A-7. Internal electrical cable conduits free the exterior surface for manipulation and sensing. The goal is the eventual inclusion of the finger as part of a hand, figure A-8, or a hand/arm system, as in figure 4-12.

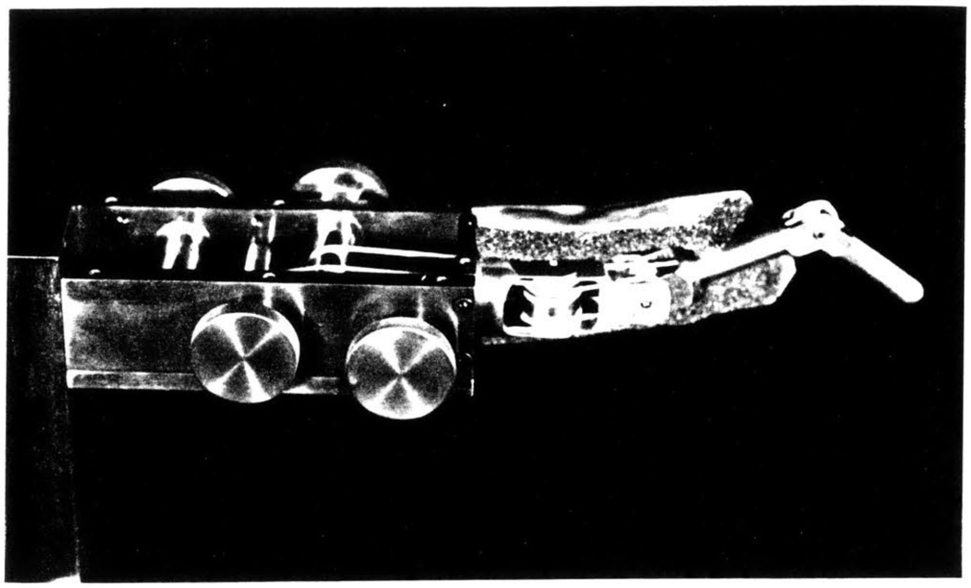


Figure A-1. In order to manipulate objects in unexpected locations, a prototype finger was constructed which provides a compliant exterior on all exposed surfaces. The covering was composed of a foam core and latex skin, which contained a viscous fluid, thus providing a compliant and viscoelastic interface with the environment.

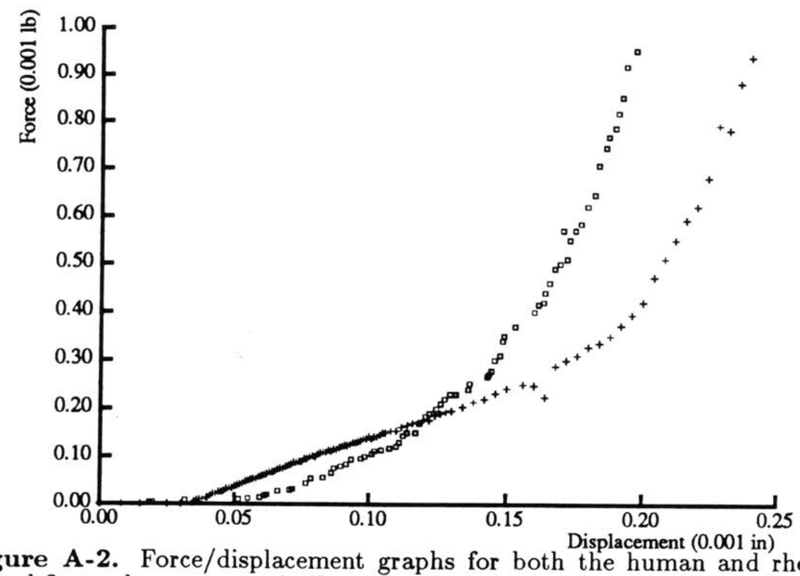


Figure A-2. Force/displacement graphs for both the human and rheological finger demonstrate similar characteristics, where □ = Human finger pad and + = Rheological fingertip.

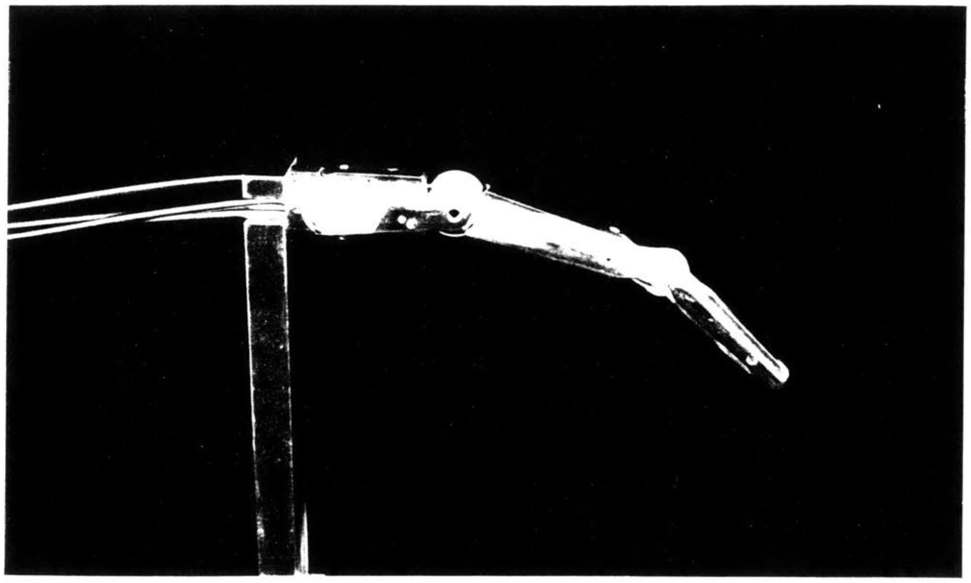


Figure A-3. A second prototype was constructed, providing both compliant surfaces and contact sensing. The finger maintains the same kinematic structure, cable routing, and mechanical freedoms as the Salisbury Robot Hand, though with the addition of compliant surfaces, intrinsic force/torque sensing (Brock 1985; Bicchi 1989, 1992; Bonivento 1991), and coarse tactile sensing.

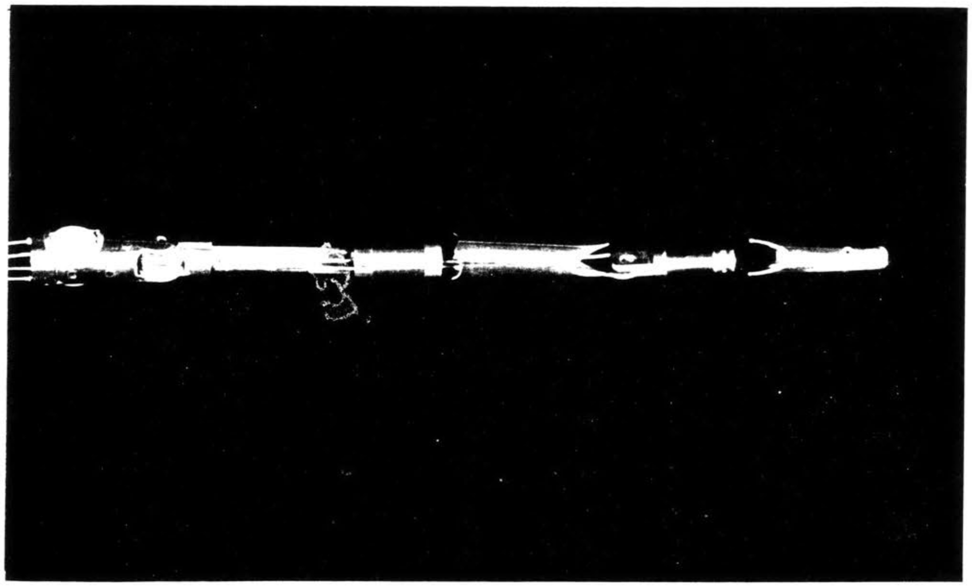
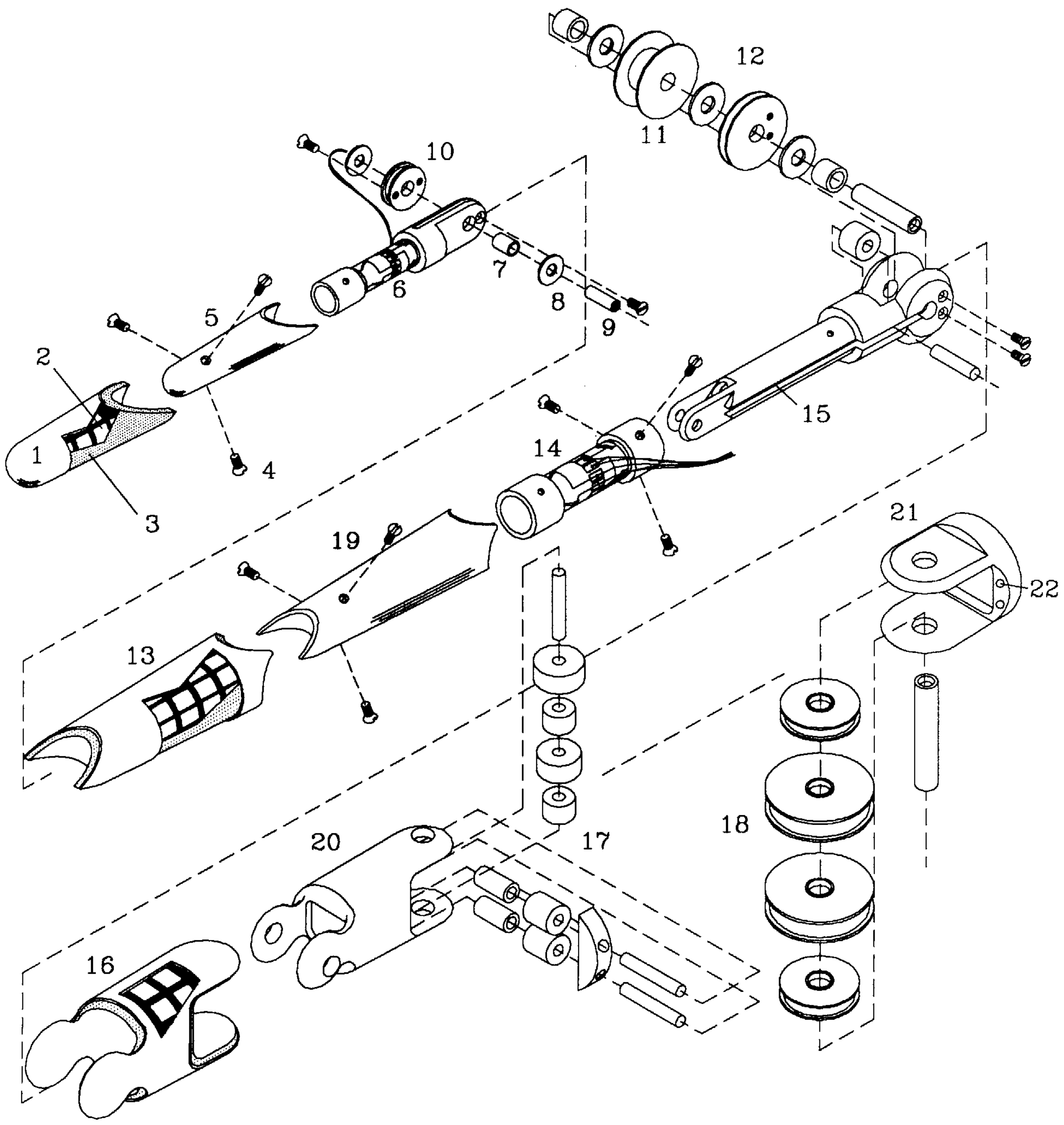


Figure A-4. Intrinsic force/torque sensing is provided on the distal and medial phalanges.



1. Latex Cover
   Killian Latex, Inc.
2. Capactitive based coarse
   tactile sensing,
   Metal flexures with
   polyurethane spacing
3. Moldable open core foam
   IPI Isofoam Systems, Inc.
   Color Die, Milicon, Inc.
4. 0-80 Flat head screws
5. Distal Phalange
   7075-T6 Aluminum
   Industrial Aluminum Co., Inc.
6. Distal six-axis loadcell
   Eight 500 Ohm Semiconductor
   strain gauges
   Micron, Inc.
7. DU self-lubricating bearing
   steel case, porous bronze,
   PTFE-lead overlay
   02DU03, Garlock
   Bearings, Inc.
8. Telfon bushing,
   .004 thickness
   Enflo, Inc.
9. Stainless steel
   percision ground shafts
   Wilfred M. Berg, Inc.
10. Distal pulley assembly
11. Delran idler pulley
12. Medial pulley assembly
13. Medial tactile array
14. Medial six-axis loadcell
15. Electrical cable conduit
16. Proximal tactile array
17. Proximal guide
    pulley assembly
18. Proximal idler
    pulley assembly
21. Finger base
22. Tendon conduits
    Spectra 1000 tendon
    BS10/12, Cortland Cable
    Company, Inc.

Figure A-5. A prototype finger was constructed which has both sensing and compliant manipulation surfaces over the entire exterior.

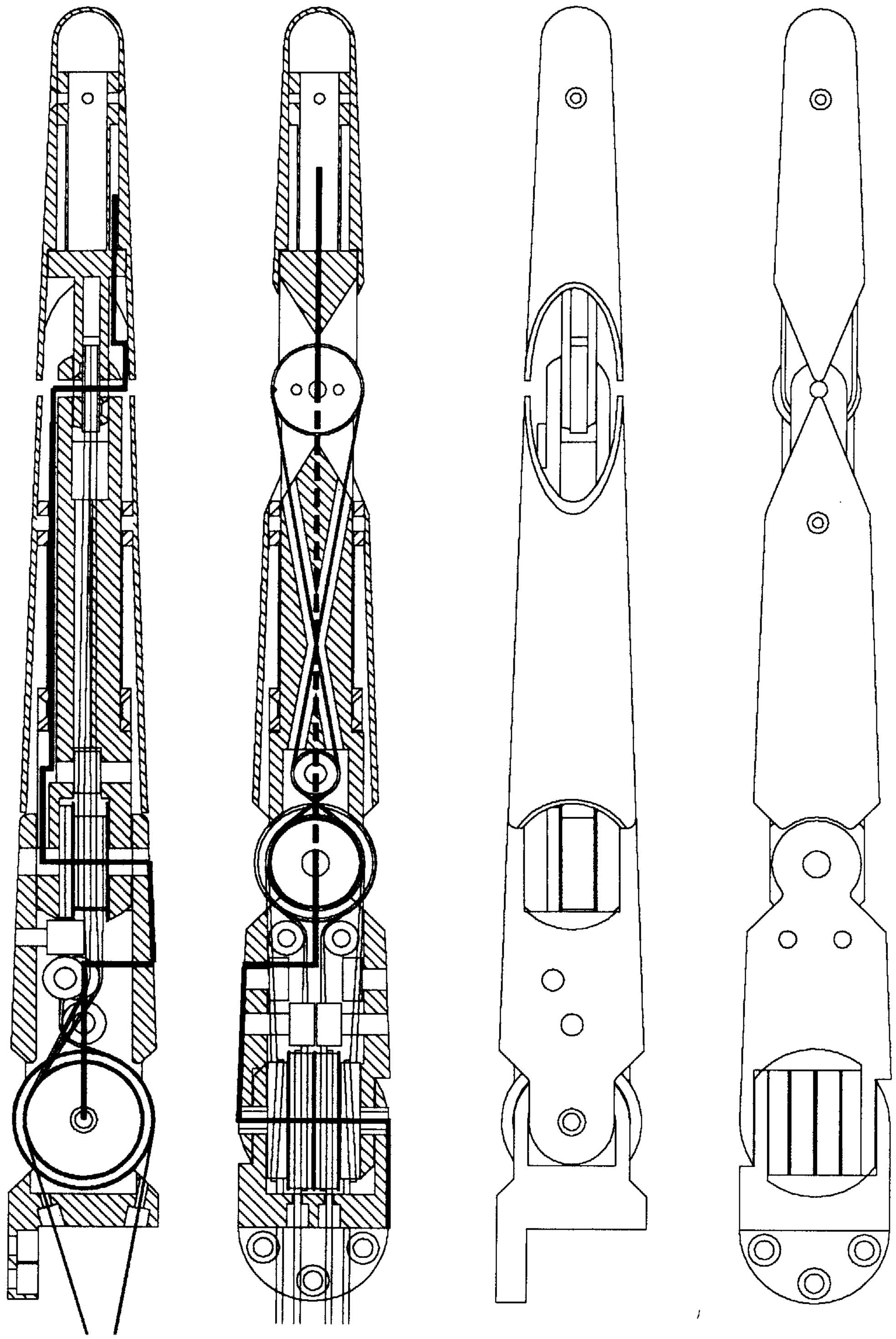


Figure A-6. Internal electrical cable conduits allow sensory signals to be conducted through the kinematic structure.

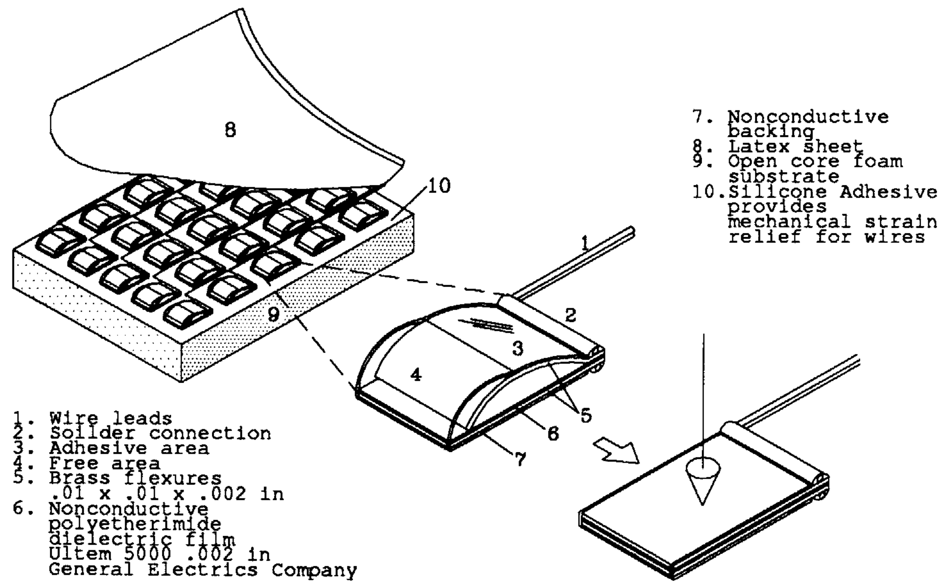

8
10
1
2
3
4
5
6
7
9
1. Wire leads
2. Soilder connection
3. Adhesive area
4. Free area
5. Brass flexures .01 x .01 x .002 in
6. Nonconductive polyetherimide dielectric film Ultem 5000 .002 in General Electrics Company
7. Nonconductive backing
8. Latex sheet
9. Open core foam substrate
10. Silicone Adhesive provides mechanical strain relief for wires

Figure A-7. Capacitive tactile sensing based on small (0.1 in x 0.1 in) metal flexures reduces hysteresis common in polymer based systems. Also relatively large vertical displacements from 0.01 in to 0.001 in provide adequate changes in capacitance.

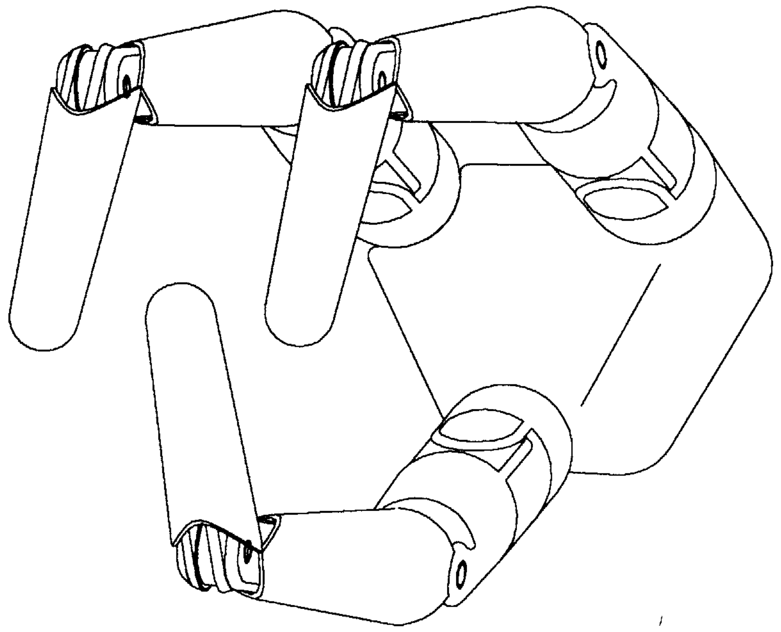


Figure A-8. The goal is the eventual inclusion of this finger as part of a hand or a hand/arm system.