Explanation-Based Manipulator Learning: Acquisition of Planning Ability Through Observation

Alberto Maria Segre
Gerald DeJong

Artificial Intelligence Research Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

This paper describes a robot manipulator system currently under development which learns from observation. The system improves its problem-solving capabilities through the acquisition of task-related concepts. The system observes manipulator command sequences that solve problems currently beyond its own planning abilities. General problem-solving schemata are automatically constructed via a knowledge-based analysis of how the observed command sequence achieved the goal. This learning technique is based on *explanatory schema acquisition*. It is a knowledge-based approach, requiring sufficient background knowledge to *understand* the observed sequence. The acquired schemata serve two purposes: they allow the system to solve problems that were previously unsolvable, and they aid in the understanding of later observations.

1. Introduction - Why Robbie Can't Learn

The introduction of robot manipulators has had an enormous impact on the manufacturing community. The scope of this impact is indeed remarkable if one considers that today's manipulators do not exploit their full potential. This lacuna can be attributed to the fact that while manipulators are indeed *general purpose*, they are still far from *flexible* [1].

This distinction does not refer to any shortcoming in the actual design of the mechanical arm, but rather is meant to reflect the gap that exists between the theoretical capabilities of a manipulator and the difficulties involved in preparing it to perform some new task. We call this problem the *retraining problem*.

This paper addresses the retraining problem in a novel fashion. We have implemented a system which acquires planning concepts through observation. The system monitors the actions of a manipulator. When it recognizes completion of an important goal, the system generalizes the observed manipulator sequence to create a new concept.

1.1. The State of the Art in Robot Teaching

There are two methods in general use today for the programming of manipulator sequences: *teaching* (also termed *guiding* or *tape-recorder mode*) and *manipulator-level programming languages*. Each approach suffers from different weaknesses.

Teaching refers to the practice of guiding the gripper through the actions to be executed, either manually or by use of a joystick. The human teacher marks important intermediate positions in the sequence so that the manipulator may move to these positions during execution. While this method is easy to use, it is obviously inflexible with respect to initial piece placements. In addition, the burden of providing a complete sequence with no superfluous gripper movements rests on the teacher: the manipulator cannot judge the quality of the sequence.

The development of manipulator-level programming languages was motivated in large part by the introduction of sensory capability on many newer arms, as well as a desire to provide some limited flexibility through the use of algorithmic choice points. The use of one of these languages still requires the robot programmer to completely state the task solution *a priori*, including decision points, conditional operations, and the like. Nevertheless, by providing a way to incorporate various sensory inputs (e.g. visual feedback, force feedback) these robot programs ([2, 3] among others) are adequate for a great number of applications.

The problem with manipulator-level programming languages is that developing a program for even a simple application requires a skilled robot programmer. Like the teaching method described above, the burden of providing a good program rests on the programmer: but unlike the teaching method, development of new manipulator programs is neither easy nor quick.

There have been some recent attempts to mix these two methods in order to profit from the advantages of each [4]. These systems extend the teaching method through the use of an interactive dialogue with the teacher. This essentially permits the robot to generate a sequence which looks very much like a manipulator-level program, complete with preconceived sensor strategies, without the intervention of a skilled robot programmer. While these systems permit some flexibility in, for example, layout of the workspace (by relying on sensors to find pieces within a certain margin of error), they do not address the retraining problem. These systems still place the burden of designing new task sequences on the teacher, and, as such, are still unable to deal with situations not predicted by the teacher.*

1.2. Explanation-Based Manipulator Learning

To correct this problem, we propose to replace robot teaching with robot learning: to shift the burden of designing new task sequences from the operator to the robot by endowing the manipulator with a *conceptual level* for task specification and learning.

Lozano-Perez [5] divides robotics research into two distinct areas: *robot control* and *robot planning and programming*. Robot control refers to work in subfields of robotics such as kinematics, trajectory planning, compliance, and feedback control. Robot

This work was supported in part by the Air Force Office of Scientific Research under grant F49620-82-K-0009 and in part by the National Science Foundation under grant NSF-IST-83-17889.

* This refers to unforeseen situations far more complex than, for example, slight deviation from a particular initial piece placement. Here we are referring to cases such as serendipitous situations where part of the task is already complete, or where a piece cannot be picked up due to other pieces obstructing it.

planning and programming address the retraining problem."

Our approach is based on work in *explanation-based learning*. Explanation-based systems [6-9] differ from more traditional correlational machine learning paradigms [10-12] in that it is possible to acquire useful knowledge on the basis of a single problem-solving episode. For this to be possible, explanation-based systems rely on extensive domain-specific knowledge both while observing the sample problem-solving episode, and in guiding the generalization process. In our system, domain knowledge is encoded in a knowledge structure called a *schema* [13-16]. A schema represents the sytem's knowledge on a particular topic. For example, a schema for a block would include its dimensions, grasping techniques, its current location, etc. In our system there are three kinds of schemata: physical object schemata, mechanism schemata, and schemata representing actions and action sequences.

2. System Architecture

Figure 1 shows the architecture of our explanation-based system. Observed input is in the form of a sequence of primitive gripper commands, together with a goal specification. The gripper commands are fed to both the arm emulator and the justification analyzer. A software arm emulator is used in place of a real robot arm.

The justification analyzer constructs a causally complete *explanation* of how the observed gripper commands achieve the

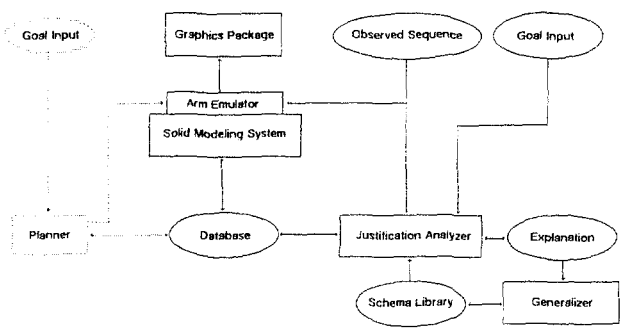

Figure 1 -- System Architecture

specified goal. The justification phase is similar to the processing of other *understanding* systems [6, 17-19], with one important difference. In addition to the causal links interrelating the input actions, the explanation includes the set of constraints crucial to the successful completion of the task.

The generalizer operates on the explanation to produce a new schema for the schema library which can then be used by the planner to achieve this goal in similar situations. The constraints are used to limit the extent of generalization so that the resulting schema is causally correct (e.g., a square peg does not fit a round hole).

The planner applies schematic knowledge from the schema library when presented with a new situation. The dotted lines in the figure indicate the planner and justification analyzer do not operate at the same time. Schemata from the schema library are indexed by the goals they achieve. Once a schema is selected, the planner formulates its preconditions as a series of subgoals to be achieved.

** The term robot planning should not be confused with work in artificial intelligence which has also been termed "robot planning." The latter really has very little to do with planning manipulator tasks, but rather addresses the problem of planning command sequences for an idealized robot moving about in a simple idealized environment (e.g., consisting of several rooms with doors, boxes, etc.). For historical reasons, we shall continue to refer to such work as "robot planning" while adopting henceforth the term "manipulator planning" for what Lozano-Perez terms "robot planning."

3. Knowledge Representation Issues

An explanation-based system relies on domain-specific knowledge in order to be able to learn from a single problem-solving episode. In our system, domain-specific knowledge falls into three general areas: physical objects, mechanisms, and tasks, each represented as a schema type. The present implementation is capable of adding new task schemata. Note that the system does not make any distinction between built-in schemata and schemata acquired automatically.

3.1. Physical Object Schemata

We have adopted a solid modeling approach for the representation of the physical objects manipulated by the system. Our simplified solid modeling system is a modified *Constructive Solid Geometry* (or *CSG*) system [20-22]. This particular CSG system supports two *primitive pieces*, the block and cylinder; although other primitives can be added with a reasonable effort. By applying set operations on instances of the primitive pieces, we can form more complex *composite pieces*. By accepting limitations on both the operator set and the primitive set, the complexity of the modeling system is substantially reduced.

A *surface* corresponds to the face of a piece. Each surface contains a cartesian coordinate system with the Z-axis normal to the face of the solid. The cartesian coordinate system is centered on the face so as to be aligned with the primary axis of the solid. In order to avoid potentially expensive CSG to *boundary representation* conversion algorithms, each primitive has precomputed surfaces associated with it. Two special cases of surfaces are important to the example discussed later: a *socket* is a hole resulting from the set difference of two pieces, while a *bore* is a socket which extends through the solid.

3.2. Mechanism Schemata

We define a *link* to be a rigid solid body, with an arbitrary number of cartesian coordinate systems, or *hooks*, affixed to the solid to serve as reference points. A *joint* (also a *low-order pair*) relates two links by specifying a hook from each link together with a parameterized transformation matrix giving the legal relative positions between hooks. The number of independent variables in the transformation matrix indicates the number of *degrees of freedom* in the joint. A *mechanism* is a set of links together with a set of joints defined on those links [23-25].

We can construct a *taxonomy of joints* based on the degrees of freedom they allow. Two unconstrained rigid bodies have six degrees of freedom: three translational degrees of freedom and three rotational degrees of freedom. Two rigidly constrained bodies (such as a peg in a snug hole) have zero degrees of freedom. Some common joint types are:

RigidJoint - zero degrees of freedom.
PrismaticJoint - one translational degree of freedom.
RevoluteJoint - one rotational degree of freedom.
CylindricalJoint - one rotational and one translational degree of freedom.
ScrewJoint - same as a **CylindricalJoint** with only a single degree of freedom. Translational and rotational movement are dependent.
UniversalJoint - two rotational degrees of freedom.
SphericalJoint - three rotational degrees of freedom.
NullJoint - all six degrees of freedom.

It is important to realize that there are only a finite number of combinations of degrees of freedom possible, and, therefore, only

* This primitive set is obviously impoverished in comparison with solid modeling systems seeing commercial use (e.g., in such areas as static interference analysis, finite-element meshing, and automatic verification of machine tool numerical control programs). It is, however, adequate for the representational demands our examples make on the solid modeling system.

a finite number of joint types. However, there may be many ways to realize a given joint.

There is a schema for each of the common joint types discussed above. Each joint schema contains two hooks, one on each piece involved in the joint, as well as a joint transformation matrix . The transformation matrix contains some variables, the number of which depends on the number of degrees of freedom this particular joint type allows. The joint schema also contains bounds on these variables, along with an indication of what kind of bound occurs at each end of the variable's range. When the mechanism physically prohibits a variable from exceeding a bound, it is termed a *hard bound*, otherwise it is termed a *soft bound*. While a mechanism is physically prohibited from exceeding a hard bound, it may exceed a soft bound. Exceeding a soft bound, however, will cause the mechanism to fail in some catastrophic fashion.

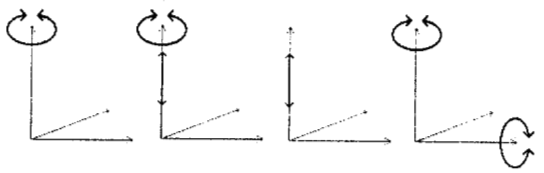

Figure 2 – Sample Joint Types

3.3. Task Schemata

Task schemata relate a goal with a more detailed task specification. The task specification may be in terms of low-level gripper primitives and/or other task schemata. Task schemata include preconditions which must be true for the schema to be applicable. They also include a set of side-effects which specify how the world changes as a result of the schema's application. These schemata are used by the planner when it generates a low-level gripper command sequence to accomplish a given goal, as well as by the justification analyzer when it constructs an explanation to account for observed inputs in a problem-solving episode. An explanation is constructed by linking preconditions and effects, yielding a network of instantiated schemata.

An example of a task schema is **PickUp**, which specifies how to approach and grasp a piece. Preconditions of **PickUp** include that gripper be empty and opened. Effects include that the piece is held by the gripper, and that the piece is no longer supported by the workspace surface.

4. An Implemented Example

The task we will consider is to attach two pieces using a friction fit peg as an axle (see figure 3). Our system does not have prior knowledge of such axle structures or of how to construct them. The system monitors a conventional arm as it goes about the rote construction of this assembly. From the low-level input sequence our system acquires a task schema which can be used to accomplish this assembly as well as other similar mechanisms, irrespective of the starting locations of the constituent pieces.

The system is implemented in LOOPS [27], a programming language built on top of Interlisp. LOOPS adds data-oriented, object-oriented and rule-oriented programming to Interlisp's existing procedure-oriented programming.

** Our system uses 4x4 matrices to represent homogeneous coordinate transforms. For a discussion of homogeneous coordinate system transformations, see [26].

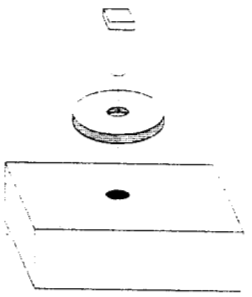

Figure 3 – Sample Assembly Task

4.1. Gripper Model

The gripper position and orientation is specified by a coordinate system in three space. The coordinate system is located at the point directly between the gripper fingers, the gripper *hot-spot*.

There are five low-level gripper primitives:

Close - close the gripper fingers.
Open - open the gripper fingers.
Rotate(unitVector, theta) - rotate by theta degrees around axis determined by the gripper hot-spot and given unit vector.
Translate(unitVector, delta) - translate by delta units along an axis determined by the gripper hot-spot and given unit vector.
MoveTo(position) - move to a new position/orientation following any (unspecified) collision-free path. The arm emulator simply ignores collisions when executing this primitive. If a real robot arm were to replace the emulator, a suitable path finding algorithm would be necessary [28, 29].

4.2. Goal Specification

For generalization to be successful, the system must construct an explanation that justifies the specified goal [7, 30]. In this example, we specify that the observed assembly sequence realizes a **WheelAndPegAssembly*** in a work space of four pieces, whose CSG representations and initial placements are also given. The system inserts **WheelAndPegAssembly** as a semantically empty class in the LOOPS inheritance lattice. When the system prompts for a definition of this class, the class is defined as a **RevoluteJoint** between piece A and piece C. This is the functional specification of **WheelAndPegAssembly**. The system still does not know how to build a **WheelAndPegAssembly**. It simply knows a matrix of possible transformations between pieces A and C which describe it.

4.3. Input Sequence

The initial piece placement is as shown in figure 4. Our system observes the low-level gripper primitives which drive the conventional arm to the completion of this task. The sequence contains noisy inputs, such as unnecessary arm movements, and operations on pieces which have nothing to do with the final structure. In addition, the input sequence is tailored to both the physical characteristics of these particular pieces and their initial locations.

* The actual name used is unimportant to the system. Note that **WheelAndPegAssembly** is actually a misnomer, since there is no part of the goal specification that implies a wheel is necessary. It is, however, convenient to use this name for purpose of discussion, as opposed to, say, **TwoRevolvingBodiesWithPegAxleAssembly**.

The task is accomplished by 19 low-level gripper commands which correspond to:

(1) Orienting the socket in piece A upwards (4 commands).

(2) Clearing the top of piece C by stacking piece D temporarily on top of piece B (6 commands). During this operation, piece D is moved along each of the three cartesian axes in turn. The reader should note that piece D might just as well have been moved with a single (and computationally cheaper) **MoveTo** rather than three successive **Translates**. In addition, block B plays no part in the final assembly; piece D might just as well been placed aside directly on the work surface.

(3) Stacking piece C on piece A while aligning the bore in C with the socket in A (4 commands).

(4) Moving piece D first to a position above the aligned bore and socket and then inserting it through piece C into piece A (5 commands). The reader should note that the **Translate** here is indeed necessary since a **MoveTo** would not guarantee insertion. This makes this particular **Translate** crucial to the success of the task, while the three previous **Translates** are not.**

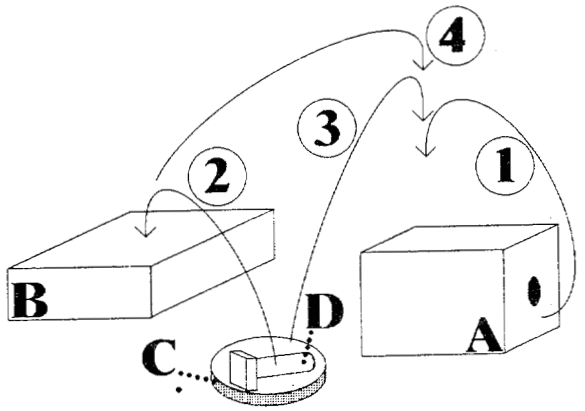

Figure 4 -- Observed Manipulator Sequence

4.4. Constructing an Explanation

The explanation is constructed from task schemata in a bottom-up fashion. As the number of schemata in the system increases, we are quickly faced with the combinatorially explosive *frame selection problem* [14, 31, 32].

The schema suggestion/activation/cancellation/closure mechanism is our solution to the frame selection problem. As each input is received, it may *suggest* certain schemata, each with a particular set of bindings. This suggestion mechanism is a way of focusing attention and thus improving efficiency. The *activation condition* for each suggested schema is evaluated with respect to these bindings. If valid, an instance of the schema is *activated* and linked with the nascent explanation. To avoid excessive numbers of suggested schemata, a *cancellation* algorithm removes suggested schemata which can no longer be relevant.

In our example, the **OrientSocketUp** schema is suggested when piece A is the object of a **Drop** schema, which was in turn suggested by the low-level gripper input **Open**. The activation conditions for **OrientSocketUp** require that the piece being dropped have a socket, and that the new orientation leave this socket pointing upwards. The check for these activation conditions is made on the emulated arm's workspace. If true, this instance of **OrientSocketUp** is activated. In this way the system hypothesizes that the reason for **Drop** was to accomplish **OrientSocketUp**. At this point, the newly activated **OrientSocketUp** may make further suggestions.

Joint schemata are also suggested and activated in the same manner. Knowledge about *how* joints are physically realized allows us to suggest the proper joint schema based on task schemata. For example, an **Insert** schema suggests a **CylindricalJoint** schema. The **CylindricalJoint** schema has additional activation conditions which check the shape and size of the shaft against the socket. When the joint schema is activated, it may suggest other joint schemata. The other joint schemata suggested generally indicate the loss of degrees of freedom via substitution of the activated schema's soft bounds with hard bounds.

In our example, the activation of **CylindricalJoint** suggests the possibility of a **RevoluteJoint**, which has one less (translational) degree of freedom. When the **RigidJoint** is activated between piece D and piece A, the **CylindricalJoint**'s soft bound is replaced with a hard bound. This change in bound type in this particular case also reflects a change in strictness of the bound; the wheel no longer has as much travel space on the shaft of the peg. Indeed the new bound reduces travel distance to the point that the suggested **RevoluteJoint**'s activation conditions are met, and an instance of the schema between pieces A and C is added to the explanation.

4.5. The Generalizer

By applying domain-specific background knowledge to the explanation constructed, the system produces a generalized task schema which does not reflect the peculiarities of the observed problem-solving episode. It is independent of the initial piece placements and is free of redundant steps. Any observed gripper command which does not lead towards the previously specified goal is removed before generalization and thus does not appear in the schema.*

The current implementation produces a task schema which realizes **WheelAndPegAssembly** efficiently from arbitrary initial states. This schema reflects the fact that piece B plays no part in the mechanism, and that piece D can be cleared to the workspace surface (removal of redundant pieces). Furthermore, the new task schema substitutes a single **MoveTo** for the three **Translates** used to clear piece D from the top of piece C (removal of redundant commands in the observed input). It does not, of course, remove the **Translate** from the final assembly step.

The task schema produced by the system was applied to a number of different initial placements, two of which are shown

** The following are English descriptions of the gripper commands:

1. **MoveTo** (grasping-position-for-piece-A)
2. **Close**
3. **MoveTo** (socket-up-position-for-piece-A)
4. **Open**
5. **MoveTo** (grasping-position-for-piece-D)
6. **Close**
7. **Translate** (minus-X-axis, units)
8. **Translate** (Z-axis, units)
9. **Translate** (Y-axis, units)
10. **Open**
11. **MoveTo** (grasping-position-for-piece-C)
12. **Close**
13. **MoveTo** (stacking-position-piece-C-on-piece-A)
14. **Open**
15. **MoveTo** (grasping-position-piece-D)
16. **Close**
17. **MoveTo** (alignment-position-piece-D-with-pieces-A-and-C)
18. **Translate** (minus-Z-axis, units)
19. **Open**

* In the new implementation currently under development, the resulting schema will also be applicable to pieces with differing size and shape but similar functionality.

in figures 5 and 6**. For the initial placement in figure 5, the generated sequence was of only 6 gripper commands, reflecting the power of this approach in taking advantage of serendipitous situations during planning. For the initial placement in figure 6, the generated sequence was of length 22, demonstrating the ability of the system to untangle a more complicated initial placement and still successfully accomplish the construction task.

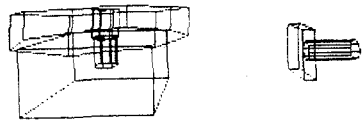

Figure 5 -- Wheel on Block, Socket and Bore Aligned

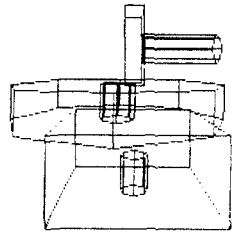

Figure 6 -- Peg on Wheel, Wheel on Inverted Block

5. Future Directions

As currently implemented, the system only demonstrates the feasibility of explanation-based learning. Of the three types of schemata in the system's schema library (task schemata, mechanism schemata, and physical object schemata) the system currently learns only task schemata. It is our goal to endow the system with the ability to acquire all types of schemata.

The next step is for the system to acquire the concept of **Peg**. Recall that the system has no built in knowledge about the class of pieces which could fulfill the role of **Peg** in this assembly: it simply has the CSG representation of this particular peg. From this single instance, by an analysis of the physical constraints imposed on the object which must fulfill this role (e.g., the shaft must fit snugly into the socket of piece A, but only loosely in the bore of piece C), we hope to obtain this new Class of physical objects in the LOOPS inheritance hierarchy.

6. Conclusion

In this paper we discussed adding a *conceptual level* to a robot manipulator system. The conceptual level allows the system to solve assembly problems specified purely in terms of the desired structure's functionality. Furthermore, such a system can improve its own performance abilities through observation. When shown a single example illustrating a new assembly technique, the system constructs a general concept that can be applied to later problems. These later problems need bear little resemblance to the observed problem in terms of the number of parts, size or shape of the parts, their initial locations, or the manipulator command sequence that embodies the solution. In fact, the problems need only be similar at the conceptual level.

** Figures 5 and 6 are taken from a crude graphics package interface we have implemented in LOOPS. As such, only wireframe drawings are rendered without the benefit of a hidden line removal algorithm. The new implementation under development will have a better graphics interface, although it is obviously not the central focus of our work.

Our approach differs from the BUILD system of Fahlman [33] which planned block manipulation tasks in that ours can extend its processing through learning. An early system that learned robot plans (as opposed to manipulator plans) was STRIPS [34]. However, it learned by generalizing and remembering successful problem-solving episodes rather than observing and understanding the actions of others. Thus it was limited by its own problem-solving ability. Learning from the observation of other's intelligent actions does not suffer from such a self-imposed limit. STRIPS also differs in that it worked in a much simpler world where grasping and lifting objects was not permitted. Nor was it possible to construct new objects from combinations of existing ones. Note that the STRIPS generalization process was unable to remove irrelevant temporal orderings. Nonetheless, there is a strong flavor of the MACROPS approach in our work.

More recently Levis and Selfridge [35] have constructed a manipulator teaching system. That system puts a much greater load on the user who must designate *watch points* to denote important assembly states; we rely on the system's understanding of the functionalities associated with structures it manipulates to focus its attention. Another recent system [4] provides a clever way to permit tape-recorder mode teaching to utilize prepackaged sensor strategies. This can be viewed as an extension of current manipulator teaching technology rather than a new way of addressing the retraining problem.

The approach we have discussed results in a system which can improve its effectiveness while greatly decreasing the human programming time of robot manipulators. A manipulator can be taught by leading it through a task. The observed sequence need not be a perfect sequence: it must simply be effective in accomplishing the stated goal. The system acquires a general solution which not only enables it to react to variations and exceptional conditions (an ability previously bestowed only by trained robot programmers through hours of work), but also attempts to provide a more efficient solution by removing redundant or needlessly inefficient steps. This combines the ease of instruction of teaching systems with the flexibility of manipulator-level language systems; and as an added bonus shifts the programming burden from the human teacher to the robot learner.

In the long term view of robotics, introducing a conceptual level for learning and task specification takes a needed first step in the direction of intelligent manipulators. We feel that the future of robotics lies with systems that understand *what* they are doing and *why* each command achieves its desired goal. These understanding manipulators will obviate much of the down time for retooling and enable a flexibility in manufacturing never before possible, truly fulfilling the robot's promise as a *flexible* general-purpose manufacturing tool.

References

1. R. Tilove, "Personal communication," , August 1984.
2. V. Hayward and R. Paul, "Introduction to RCCL: A Robot Control 'C' Library," *Proceedings of the Institute of Electrical and Electronics Engineers International Conference on Robotics*, Atlanta, GA, 1984, 293-297.
3. B. Shimano, C. Geschke and C. Spalding, "VAL-II: A New Robot Control System for Automatic Manufacturing," *Proceedings of the Institute of Electrical and Electronics Engineers International Conference on Robotics*, Atlanta, GA, 1984, 278-292.
4. P. D. Summers and D. D. Grossman, "XPROBE: An Experimental System for Programming Robots by Example," *The International Journal of Robotics Research* 3, 1 (Spring 1984), 25-39.

5. T. Lozano-Perez, *A Tutorial on Robotics*, American Association for Artificial Intelligence, Pittsburgh, PA, 1982.
6. G. DeJong, "Explanation Based Learning," Technical Report, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1984.
7. T. Mitchell, "Learning and Problem Solving," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983, 1139-1151.
8. B. Silver, "Learning Equation Solving Methods from Worked Examples," *Proceedings of the 1983 International Machine Learning Workshop*, Urbana, IL, June 1983, 99-104.
9. S. Minton, "Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, 251-254.
10. P. H. Winston, "Learning Structural Descriptions from Examples," AI Technical Report-231, MIT Artificial Intelligence Lab, Cambridge, MA, 1970.
11. R. Michalski, "Pattern Recognition as Rule-Guided Inductive Inference," *Pattern Analysis and Machine Intelligence* 2, 4 (July 1980), 349-361.
12. J. R. Quinlan, "Learning Efficient Clasification Procedures and their Application for Chess Endgames," in *Machine Learning*, R. Michalski, J. Carbonell, T. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983.
13. W. Chafe, "Some Thoughts on Schemata," *Theoretical Issues in Natural Language Processing 1*, Cambridge, MA, 1975, 89-91.
14. M. L. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. Winston (ed.), McGraw-Hill, New York, NY, 1975, 211-277.
15. E. Charniak, "On the Use of Framed Knowledge in Language Comprehension," *Artificial Intelligence* 2, 3 (1978), .
16. R. Schank and R. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1977.
17. E. Charniak, "MS. MALAPROP, A Language Comprehension System," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977.
18. R. E. Cullingford, "Script Application: Computer Understanding of Newspaper Stories," Technical Report 116, Department of Computer Science, Yale University, New Haven, CT, January 1978.
19. M. Dyer, *In Depth Understanding*, MIT Press, Cambridge, MA, 1983.
20. J. Boyse, "Preliminary Design for a Geometric Modeler," Technical Report GMR-2768, GM Research Laboratories, Warren, MI 48090, July 1978.
21. J. Boyse and J. Rosen, "GMSOLID - A System for Interactive Design and Analysis of Solids," Technical Report GMR-3451, GM Research Laboratories, Warren, MI 48090, October 1980.
22. I. Braid, "The Synthesis of Solids Bounded by Many Faces," *Communications of the Association for Computing Machinery 18*, 4 (April 1975), 209-216.
23. R. Tilove, "Extending Solid Modeling Systems for Mechanism Design and Kinematic Simulation," *Institute of Electrical and Electronics Engineers Computer Graphics and Applications 3*, 3 (May 1983), 9-19.
24. J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Mechamisms," *ASME Journal of Applied Mechanics 22*, (June 1955), 215-221.
25. P. N. Sheth and J. J. Uicker, "A Generalized Symbolic Notation for Mechanisms," *ASME Journal of Engineering for Industry 93*, 1 (February 1971), 102-112.
26. W. Newman and R. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, NY, 1973.
27. D. G. Bobrow and M. Stefik, "The LOOPS Manual," reference manual, Xerox PARC, Palo Alto, CA, 1983.
28. T. Lozano-Perez and M. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the Association for Computing Machinery 22*, 10 (October 1979), 560-570.
29. S. M. Udupa, "Collision Detection and Avoidance in Computer Controller Manipulators," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, August 1977.
30. G. DeJong, "Generalizations Based on Explanations," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C. , Canada, 1981, 67-60.
31. E. Charniak, "With Spoon in Hand this Must be the Eating Frame," *Theoretical Issues in Natural Language Processing 2*, Urbana, IL, 1978.
32. S. E. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, MA, 1979.
33. S. Fahlman, "A Planning System for Robot Construction Tasks," *Artificial Intelligence 5*, (1974), 1-49 .
34. R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, (1972), 251-288.
35. A. Levis and M. Selfridge, "A User-Friendly High-Level Robot Teaching System," *Proceedings of the Institute of Electrical and Electronics Engineers International Conference on Robotics*, Atlanta, GA, 1984, 413-416.