STATE SPACE MODELS OF REMOTE MANIPULATION TASKS*

Daniel E. Whitney
Assistant Professor, Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts

Summary

A state variable formulation of the remote manipulation problem is presented, applicable to human supervised or autonomous computer-manipulators. A discrete state vector, containing position variables for the manipulator and relevant objects, spans a quantized state space comprising many static configurations of objects and hand. A manipulation task is a desired new state. State transitions are assigned costs and are accomplished by commands: hand motions plus grasp, release, push, twist, etc. In control theory terms the problem is to find the cheapest control history (if any) from present to desired state. In theorem proving terms it is to combine predicates and axioms to prove (or disprove) the proposition "The task. . .is possible." Each corresponds to finding the shortest path to the desired state. A method similar to Dynamic Programming is used to determine the optimal history.

The system is capable of obstacle avoidance, grasp rendezvous, incorporation of new sensor data, remembering results of previous tasks, and so on.

Keywords:

Remote Manipulation
Control Theory
Problem Solving
Theorem Proving

Introduction and Problem Statement

Remote manipulation involves a human operator and a machine together performing a task which could be performed more easily and efficiently by the man alone, were the task or Its environment not too large, small, distant, ponderous, delicate, obscure, dangerous or some combination of these. Manipulators are used In quite complex hot lab experiments,[17] for underwater retrieval, for complete operation and maintenance of large radioactive research installations for extended periods of time,[8] and in robot research,[11] to name a few examples.

To overcome the handicaps of time, scale or distance barriers, two lines of research have developed which involve combining a manipulator and a computer. One is the Human Supervised Remote Computer-Manipulator,[14] depicted in Figure 1. Here, a human operator is aided by the computers in planning and executing the task. The other approach is the autonomous robot, which is intended to maneuver and manipulate unaided in a distant environment.[11*1]

*Work supported by NASA Grant NsG 107-61

Major problems in the design of human supervised or autonomous computer-manipulators are

1) Model and real world of hands, objects and obstacles.
2) Define the notion of manipulation task.
3) Develop methods which generate plans of manipulative action which are relevant to the task environment (as modelled in 1)) and which are capable of achieving the stated goal (as expressed In 2)).

To solve these problems, we shall consider the manipulator's hand and the task site as a system to be controlled by an operator or other intelligence. This approach differs from previous work[1*15] in which only the hand is included in the system model. A state vector is defined, containing not only variables which describe the manipulative device, but also vital parameters of task site, possible including locations of relevant objects and obstacles. This vector, suitably quantized, spans a discrete state space whichcontains many different static configurations of the manipulator and the objects to be manipulated. A manipulation task is then defined as a new state which the "operator" (human or computer) wishes the manipulator-objects-obstacles system to occupy. State transitions are accomplished by commands: quantized basic motions of the manipulator's jaws plus grasp, release, push, twist, and so on.

One may Interpret the resulting problem in Modern Control theory terms as follows: given that each state transition costs a known amount, find the cheapest (shortest) path from the present state to the desired state. This path represents the optimal control policy for accomplishing the given task.

Alternatively, one may view the state space in terms of problem solving or theorem proving: each state transition represents an instance of a predicate or an axiom such as "If the jaws move one inch to the right, then the new state is related to the old state by. . .," or "If the jaws are grasping object A and the jaws move one inch to the right, then object A moves the same distance and the new state is related to the old state by. . •" Each instance is specified by arguments Including the current state, the name of a relevant object, and so on. Since the problem is purely geometric, the result of applying any axiom or predicate is unambiguously related to the goal. The resulting path may then be thought of as the solution to the problem (or the proof of the theorem) "Can the specified task be accom-

pUshed?"

Tasks and Commands

At the outset ve should distinguish two types of tasks, those which can be planned ahead and those which cannot• An excellent example of the latter is the task "Find the pencil." Execution cannot be planned open loop, along the lines of: Move jaws to location X, pick up pencil. Rather, execution consists of continual interaction between the manipulator-computer and its environment* The type of plan needed closes a loop through the manipulator's sensors, and has many of the features of a numerical algorithm rich in "If" statements. Such problems have been studied by Ernst4 and Barber1. Larson and Keckler[21] have used Dynamic Programming to find search and motion strategies for a robot in an unexplored environment*

Aside from searches and other tasks whose history and outcome are almost unknown £ priori, most manipulation tasks can be planned out In advance* The likelihood of the plan's success depends greatly on the accuracy of a priori information concerning location of named places and objects, and also on plain luck. Less than certain success is not, however, any reason for not planning. It is the ability to formulate and attempt execution of plans that makes an automatic or semi-automatic manipulator different from and superior to a manually controlled manipulator. Furthermore, lack of certainty in the a priori information can be compensated for by execution routines which combine obedience to the plan with minor-range searches. Greater sophistication in the execution routines will allow more uncertainty in the £ priori information without degrading the certainty of overall task completion..

Of all tasks which can be planned out in advance, we shall consider what for ordinary manipulation constitute the great bulk, namely those in which the positions or orientations of objects and effectors (jaws, tools, etc*) are changed. Thus ve specifically exclude such activities as bouncing a ball or balancing a stick on end. By ignoring velocities and accelerations, we may concentrate on the geometric constraints fundamental to manipulation: obstacle avoidance, rendezvous of a jaw-borne object and its destination for pushing, releasing, pouring, inserting, and so on. (Granted a jaw-borne object has a velocity: nevertheless what is important about carrying Is that jaws and object maintain a fixed geometric relation to each other throughout.) Then one may state the planning problem as that of finding a sequence of intermediate configurations for the task site to occupy on the way, so to speak, from the initial configuration to the desired one.

To enable the computer to solve such problems, we must equip it with a model of manipulation tasks so that it can determine what configurations may follow from a given configuration, or equivalently what changes can be made to a given configuration. This requires a way of expressing geometric constraints or equlvalently manipulative predicates and how they are altered by changes in the task site. More generally, we must recognize that some configurations are "near neighbors" while others are not, but rather are separated by one or more intervening configurations, some of which are near neighbors. Near nelghborllne88 of two configurations may be defined as a (usually bilateral) attribute implying that one single simple motion of the manipulator jaws, probably of limited extent, will carry the task site from one configuration to the other. Working with a small, well chosen set of such subnotions, we can plan a wide variety of tasks. To be precise, call the submotlons atomic commands. For example:

Move jaws left one Inch
Move jaws right one inch
Open jaws
Close jaws

In what follows we shall describe a method by which a task site may be modelled, the "operator" may request a task, and the computer can devise a sequence of atomic commands which, to the degree of precision of the task model, can accomplish the task.

The State Space Model

The problem we have posed is to find a sequence of finite elements which has a particular property. We may dispense with enumeration of the possibilities, since there are far too many, and far too few of them are worth considering.

Let x be a vector containing the position of the manipulator jaws, the positions of relevant objects and any other variables of Interest, such as object orientations. Then the manipulator-task system is governed by the equation

$$\begin{aligned} \underline{x}(k+1) &= \underline{x}(k) + A(\underline{x}(k))\ \underline{u}(k) \\ \underline{x}(0) &\text{ given} \\ k &= 0,1,2.... \end{aligned} \qquad (2)$$

where tj(k) is a vector of admissible controls such as equation (1), and A is a state dependent matrix which expresses the geometric task constraints mentioned in the previous section. These constraints are more easily visualized on a finite graph-[5] in which each node represents a state x. and each branch leading out to another state indicates an allowed command at state x_. States connected by a single branch are thus near neighbors. The totality of nodes, some connected by branches, constitutes the state space. Its nodes represent all the configurations which the task site can assume as a result of the execution of arbitrary strings of atomic commands. Alternatively, the state space represents the limited set of Instances of the manipulative predicates being considered, the instances differing in the arguments associated with each, and the set bounded by the limits on each state variable. We

may then say that paths through the space represent strings of atomic commands (controls or proof steps) which make coherent (though not necessarily purposeful or efficient) changes in the task site. For example, consider the task site in Figure 2. The Jaws may move from point to point along the line, open and close, but may not move the block. (This last capability is added below.) Given the atomic command set (1), the state space corresponding to Figure 2 is shown in Figure 3. The coordinates on the axes are xT, jaw position, and H, Jaw status. *Xj* and H are the state variables. The configuration of Figure 2 (jaws in location 4, closed) is represented by the flag at state [4,1] in Figure 3. The vertical lines indicate that the Jaws may open or close at any x, while the horizontal lines indicate allowed jaw movement. Movement directly into [2,1] from [3,1] or [1,1] is forbidden since this would involve collision between jaws and object. Thus [3,1] and [2,1] are not neighbors in the sense defined above.

If we wish the jaws to move to location x-1, jaws closed, we ask the computer to find a satisfactory path from state [4,1] to state [1,1].* (Naturally we want the computer to derive by itself the fact that the jaws must straddle the object on the way.) Such a path, if it exists, can be translated immediately into a string of atomic commands suitable for accomplishing the task, since the path tells the sequence of neighboring intermediate configurations through which the task site should pass on the way to the desired configuration. There are countless possible paths, most of which go nowhere purposive. But a shortest path cannot go nowhere and in particular cannot loop. So let us find shortest paths.

Many algorithms are available for finding shortest paths in networks, among them Dynamic Programming,2 Ford's algorithm,6 and the Hart-Nilsson-Raphael algoritlun. ** Of more interest to us is the general interpretation we can give to "shortest": A path may be short in time, fuel, risk, lack of information, or some (normalized) combination of these, for example. A state space may well be costly in fuel all over, but costly in risk only in certain areas. Sometimes a given command, like carrying, is costly everywhere, while at other times the cost of a command may depend on the state at which it is being executed (for example, carrying through a crowded region of physical apace). By specifying the dimension, magnitude and distribution of the'lengths" of lines between points in the state space, the operator

♦Suitable computer routines can generate the value of the desired state from a less formalized input command, such as "Go to the left of the object"

**For tasks at the complexity level considered here, this analytic approach seems superior, in terms of computer time and likelihood of success, to similar work employing Heuristic Programming to elicit strategies.16

can to some degree affect the nature or "style" of the resulting solution. The arrangement of lengths shown in Figure 4 results in the path indicated by the arrows. (This schedule of lengths suppresses unnecessary motions of the open jaws.) The corresponding work plan is:

Move left one inch
Open
Move left one inch
Move left one inch
Close

A path which allows the jaws to grasp the object is shown in Figure 5. The corresponding path for the case where the object is in location x-1 is shown in a new state space in Figure 6. By making the object's location a new state variable, y0, we may represent carrying and pushing in a larger state space, Figure 7. This figure is made by combining Figures 5, 6, and others like them, each corresponding to a particular value of y0. Note that pushing is not expressed 88 a transition analogous to that from [3,1] to [2,1] in Figures 3, 4, and 5, for example. This type of state transition is still forbidden since it tells nothing about what happens to the pushed object. Pushing is properly expressed in Figure 7 as a variation of carrying.

Physical Demonstration

The ideas of the previous section were implemented on a three degree of freedom manipulator converted from a plotting table. Square objects could be grasped and moved about in a region 15 inches on a side. The manipulator jaws were equipped with grip sensors inside and contact sensors outside. No jaw rotations were possible. A Digital Equipment Corp. PDP-8 computer contained a 2000 word program and a 500 word state space.* The state vector consisted of (Xj, Yj, H), while objects and obstacles were kept track of in a separate list. The 2000 word program contained all I/O, a path finding algorithm, touch sensor evaluation routines, stepping motor control, plus interpretation of commands such as

```
operator designates object at (x,y) with
  name. . .
pick up object with name. . .
take it to location (x,y)
go to location (x,y)
```

If a new object was discovered by the touch sensors while a path was being executed, the system estimated its location, asked the operator for a name and then computed a new path to the original goal, incorporating knowledge of the new object. This object could be referred to later by name, picked up, carried, and so on. A more complete description appears in .

*Twelve bits per word.

A More Complex Example

Suppose we wish to move a long thin spar through a crowded two-dimensional environment. (Picture carrying a sofa from the living room to the porch.) Here the interactions between the environment and the spar's position and orientation are of the most interest. To map motions of the spar once it is grasped by a rotating pair of jaws, we choose state variables

x = x coordinate of spar, $1 \leq x \leq 5$
y = y coordinate of spar, $1 \leq y \leq 3$
α = orientation of spar

$$\alpha = \begin{cases} 0 \text{ if spar is parallel to x axis} \\ 1 \text{ if spar is parallel to y axis} \end{cases}$$

The allowed commands are

Move ± x direction one unit
Move ± y direction one unit
Rotate 90°

Thus both position and orientation of the spar are quantized.

The physical space is shown in Figure ft. Walls are shown as open rectangles, while the two possible orientations of the spar are shown by cross lines at each possible position. The challenge is provided by the doorways, which allow the spar to pass axlally but not athwart. This constraint and the presence of walls are shown in the state space by deleting the lines corresponding to the forbidden transitions.

The state space appears in Figure 9. We assume for illustration that each "move" is of length 2, each "rotate" of length 3. Let the spar be initially at location (2,2) in the physical space, oriented parallel to the y axis, and say we want it moved to (3,3), ending up oriented parallel to the x axis. Then the initial state is [2,2,1] and the final state is [3,3,0]. These are marked Start and End, respectively, on Figure 9.

There are two equal length solution paths, shown In Figure 10 and visualized on a sketch of the task site in Figure 11. These paths do not "look like" the most direct route. Closer examination, however, reveals that these paths, by initially moving the object away from the final state, are able to save two rotations by spending a little more distance. Again, if we read a solution path, we get a list of the required moves and rotates in the correct order. A more general solution to this problem which includes grasping and releasing the spar in arbitrary (quantized) positions and orientations, may be found in Chapter V of Reference [8].

Discussion of the State Space Model and its Implications

The main feature of the State Space Models above is their quantization. This Is a direct consequence of the kind of atomic commands we allow and of our Interest In the main motion features of tasks for planning purposes. The atomic commands may be thought of as task differentials, but ultraflne quantization Is neither practical nor necessary, especially if good sensors are available. In fact, since more complex tasks require, In principle, state spaces of higher dimension, quantization poses staggering computer storage problems. (6 state variables, 10 points per axis - 10^6 points.) Three factors mitigate such difficulties:

1) Only a handful of these points need be in live (core memory) storage at any one time. In fact, the problem of finding shortest paths is, by default, one of State Increment Dynamic Programming, 9 which latter may be greatly speeded by algorithms such as that of Hart, Nilsson and Raphael, which pursue only the currently most promising path.

2) The state space need not be built and held whole and intact in storage, but rather only those sections needed as a particular path is pursued. The state space is just a logical consequence of a list of object and jaw locations, sizes and orientations, plus extremely local "knowledge" of what circumstances prohibit a given command. The needed portions of the state space may be built to order, using the information in the list. Then, merely by concentrating on one state and its immediate neighbors at a time, the computer can plan tasks which involve hundreds (or any number) of states. In most algorithms, moreover, progress is monotone so that a state, once considered, is never considered again. These notions suggest that the state space *ie*_ a list and that list-processing computer languages may be useful in dealing with it.

3) A state space describing a complex task is of high dimension only because the space stores the relations for all tasks which could be performed by manipulating the objects and Jaws in question. No command, however, asks for all tasks, and because rearrangement tasks consist of repeated sequences such as "Move empty Jaws to location X, grasp, carry object to location Y, release," it should be clear that only the state variables actually involved in one such sequence need be considered variable at one time. This is obviously equivalent to considering only a limited and much lower dimension cross section of the original space. Great savings in computer time and storage may be effected by treating sequences of such cross sections, Indeed considering them as atomic actions on a much higher level than the atomic commands which underlie them. We are thus afforded two levels of planning, an upper level in which gross motion goals are selected, and a lower level in which detailed strategies for such motions are evolved and their costs evaluated. Thus we can consider more complex tasks without recourse to unmanageable state spaces. In this way, competing gross plans may be judged for cost-effectiveness and the best one (for the given cost structure) selected. It is worth speculating that

methods of Heuristic Programming5 would be useful at the higher level, but some comparisons19 show that for actually finding detailed paths, even the relatively inefficient Ford algorithm is greatly faster than Travis'16 heuristic path finding methods.

Extending the Power of the State Space Method

Now we have reduced some basic manipulation tasks to shortest path problems. In this section we show how to make use of the basic pick-up-and-carry capability demonstrated in Figure 7 to program more complex tasks without recourse to enormous state spaces, preserving the "operator's" ability to shape the general features of the result without having to specify details.

We think of commands as functions whose arguments (specified by the operator) may be place or object names, and whose values are paths in one or more related state spaces. The basic function "Take" is evaluated by a state space such as Figure 7. Formally:

Take (A,X,Y) = carry object A to location X, leaving the jaws in location Y (3)

To exchange objects A and B, we may use "Take" to define "Switch", using an appropriate interpretive computer language:

Switch (A,B,X,Y) = Take (A,X,Y)
+ Take (B,A',Y)
+ Take (A,B',Y) (A)

in which "+" means "followed by", X is a location between A and B (chosen by the "operator"), Y is a location near X, A' is A's old location and B' is B's old location. Switch may be "optimized" if we allow the operator to define fuzzy areas $\underline{X}$ and $\underline{Y}$ in which points X and Y, respectively, are to be sought by the computer so as to achieve minimum total task length:

$$\text{Switch } (A,B,X,Y) = \min_{\{x_i\}\{y_j\}} \begin{Bmatrix} \text{Take } (A,x_i,y_j) \\ +\text{Take } (B,A',y_j) \\ +\text{Take } (A,B',y_j) \end{Bmatrix} \quad (5)$$

in which $\{x_i\}$ and $\{y_j\}$ denote the sets of locations comprising regions $\underline{X}$ and $\underline{Y}$ respectively.

The evaluation of (5) requires the establishment of a tree representing the result of choosing each x_i and y_j, the free subgoals. See Figure 12, in which we let $\underline{X} = x_1, x_2$ and $\underline{Y} = y_1, y_2$. The cost of an arc on the tree is deduced by constructing a state space similar to Figure 7 (simpler alternatives using even lower dimension spaces may also be used[20]) and finding the length of the shortest path which does the required job. This space, consistent with remarks in the previous section, may be thought of as a cross section through a much larger space, which we never bother to construct. In this way, only those subtasks which the "operator", in his greater wisdom, thinks are worth considering will actually be evaluated. When all the possibilities in $\underline{X}$ and $\underline{Y}$ have been priced, the cheapest x_i and y_j may be chosen, completing the evaluation of (5).

The advantage of (5) over (4) is that the "operator" need not have precise knowledge concerning $\underline{X}$ and $\underline{Y}$ in order to request a switch. $\underline{X}$ and $\underline{Y}$ correspond to the statement "over there somewhere", a convenient sort of statement to be able to make. In a working system, one could make this statement by pointing one's finger, a light pen, a Lincoln Wand[13] or similar I/O device at the task site or a representation of it on a TV or CRT screen.

Function definition is an obvious way of increasing the manipulative capabilities of a computer-manipulator. We may build layer on layer, using the existing functions to define new ones. We have a way of telling the computer "how" to do new tasks, such as assembling involved structures or making complex selection and ordering of subgoals. As an example of the latter, consider the task site in Figure 13. Two objects block a narrow doorway. We wish to move object A to location X. Both blockers must be moved from the doorway before A can be carried through. It is possible to define a recursive function which enables the computer to decide, on the basis of minimum total length, which blockers to move, the order in which to move them, and where to, in order to get A to X. The operator, when defining the function, says in effect, "These are the blockers, here is a region where they might be put; move A to X." If the cost of a unit move empty is one, the cost of a unit carry is two, and no diagonal moves are allowed, then the tree corresponding to the evaluation of this function is shown in Figure 14. The cost shown on each arc of the tree is deduced from a two dimensional state space based on the appropriate version of Figure 13. 53 such evaluations are required, employing vastly less computation and storage than the general 8 dimensional space which could be defined. The two equal length solution paths differ only in that B_1 and B_2 exchange roles. This problem is discussed more fully in Chapter V of [18].

Conclusions

The state space method has been shown to be capable of describing a wide variety of the logical and physical constraints which comprise manipulation. Using it, we can plan some non-trivial tasks and gain insight into the nature of tasks and commands. Combined with a flexible input language and carefully composed execution routines, it can provide a basis for Supervisory Controlled or Autonomous manipulation.

References

1. Barber, D.J., "MANTRAN, A Symbolic Language for Supervisory Control of an Intelligent Remote Manipulator," S.M. Thesis, M.I.T., Department of Mechanical Engineering, May, 1967.

2. Bellman, R.E., and S.E. Dreyfus, Applied Dynamic Programming, Princeton: Princeton University Press, 1962.

3. Berge, Claud, Theory of Graphs and Its Application, New York, JOhn Wiley and Sons, 1962.

4. Ernst, H.A., "A Computer-Operated Mechanical Hand," Sc.D. Thesis, M.I.T., Department of Electrical Engineering, December, 1961.

5. Feigenbaum, E.A., and J. Feldman, editors, Computers and Thought, New York: McGraw-Hill Book Co., 1963.

6. Ford, La.R., Jr., "Network Flow Theory," Rand Corp. Paper P-923, August 14, 1956.

7. Hart, P.E., N.J. NIlsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans, on Systems Science and Cybernetics, V. SSC-4 (2), July, 196S, *pp.* 100-107.

8. Johnsen, Edwin, Discussant at 11th Annual Meeting of the Human Factors Society,. September 25-28, 1967.

9. Larson, R.E., "Dynamic Programming with Reduced Computational Requirements," Trans. IEEE Auto. Control, V AC-10 (2), pp. 135-43, April, 1965.

10. Mergler, H.W., and P.W. Hammond, "A Path Optimization Scheme for a Numerically Controlled Remote Manipulator," 6th Annual Symposium of the IEEE Human Factors in Electronics Group, May, 1965.

11. Minsky, M.L., and S.A. Papert, Research on Intelligent Automata, Status Report II, Sept., 1967, M.I.T. Project MAC.

12. NIlsson, N., and B. Raphael, "Preliminary Design of an Intelligent Robot," in Computer and Information Sciences - II, New York: Academic Press, 1967.

13. Roberts, L.G., "The Lincoln Wand," FJCC 1966, *pp.* 223-26.

14. Ferrell, W.R., and T.B. Sheridan, "Supervisory Control of Remote Manipulation," IEEE Spectrum, V 4 (10), *pp.* 81-88, October, 1967.

15. Tomovlc, R., and G. Bonl, "An Adaptive Artificial Hand," IRE Trans. Automatic Control, AC-7, April, 1962, pp. 3-10.

16. Travis, L.E., "Experiments with a Theorem-Utilizing Program," SJCC 1964, *pp.* 339-58.

17. Weinberg, Alvln, "Transuranlc Elements and the High-Flux Isotope Reactor," Physics Today, V 20 (1), p. 29.

18. Whitney, Daniel E., "State Space Models of Remote Manipulation Tasks," Ph.D. Thesis, M.I.T. Department of Mechanical Engineering, January, 1968.

19. Whitney, D.E., op. cit., p. 106.

20. Whitney, D.E., op. cit., pp. 113-123.

21. Larson, R.E., and V.G. Keckler, "Optimum Adaptive Control in an Unknown Environment," IEEE Trans. Auto. Control, V AC-13 (4), August, 1968, pp. 438-9.

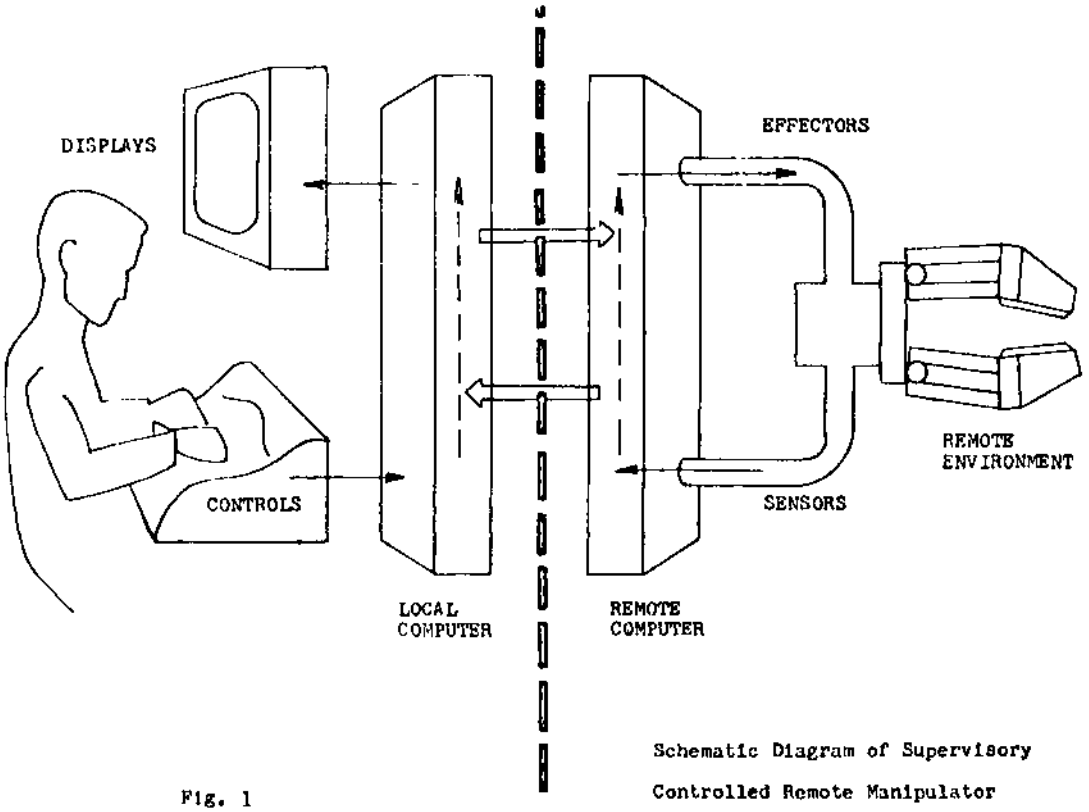


Fig. 1

Schematic Diagram of Supervisory Controlled Remote Manipulator

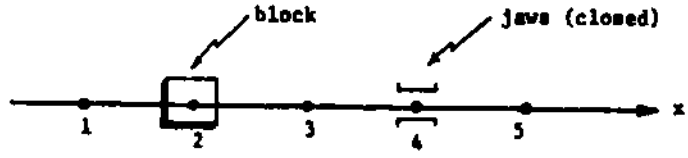


Figure 2. One-Dimensional Task Site

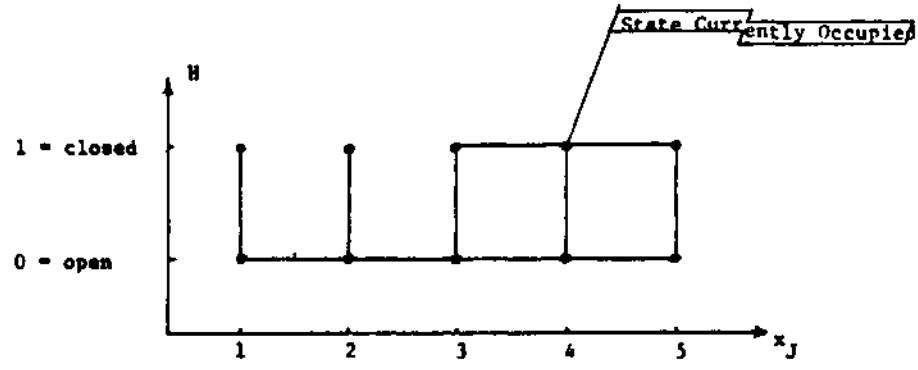


Figure 3. State Space Corresponding to Figure 2

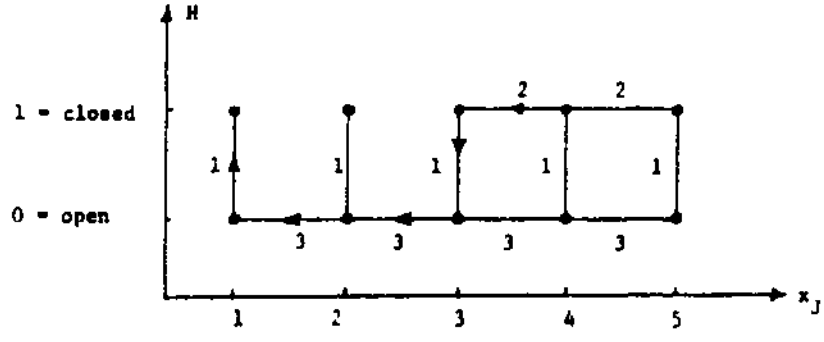


Figure 4. State Space with Assigned Lengths and the Shortest Path from $\begin{vmatrix} 4 \\ 1 \end{vmatrix}$ to $\begin{vmatrix} 1 \\ 1 \end{vmatrix}$

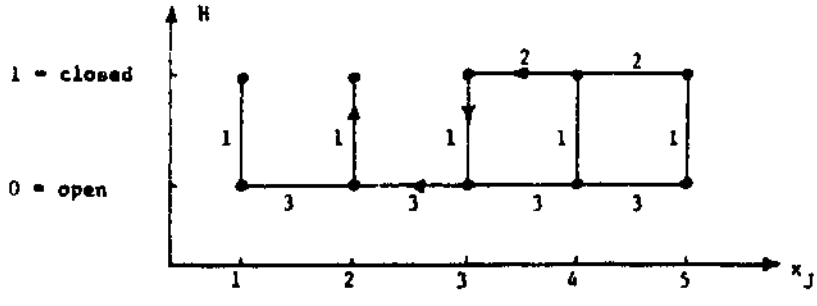


Figure 5. Path Which Represents Grasping the Object

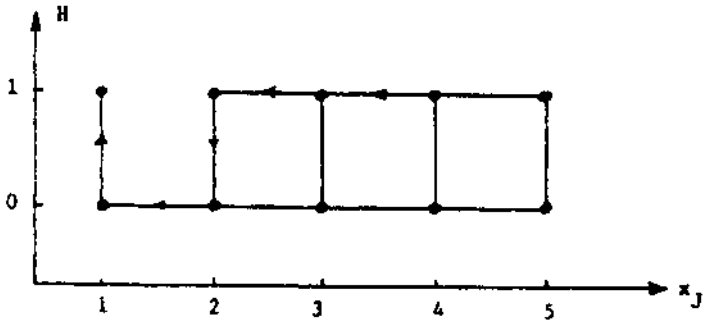


Figure 6. Path for Grasping the Object in Location 1

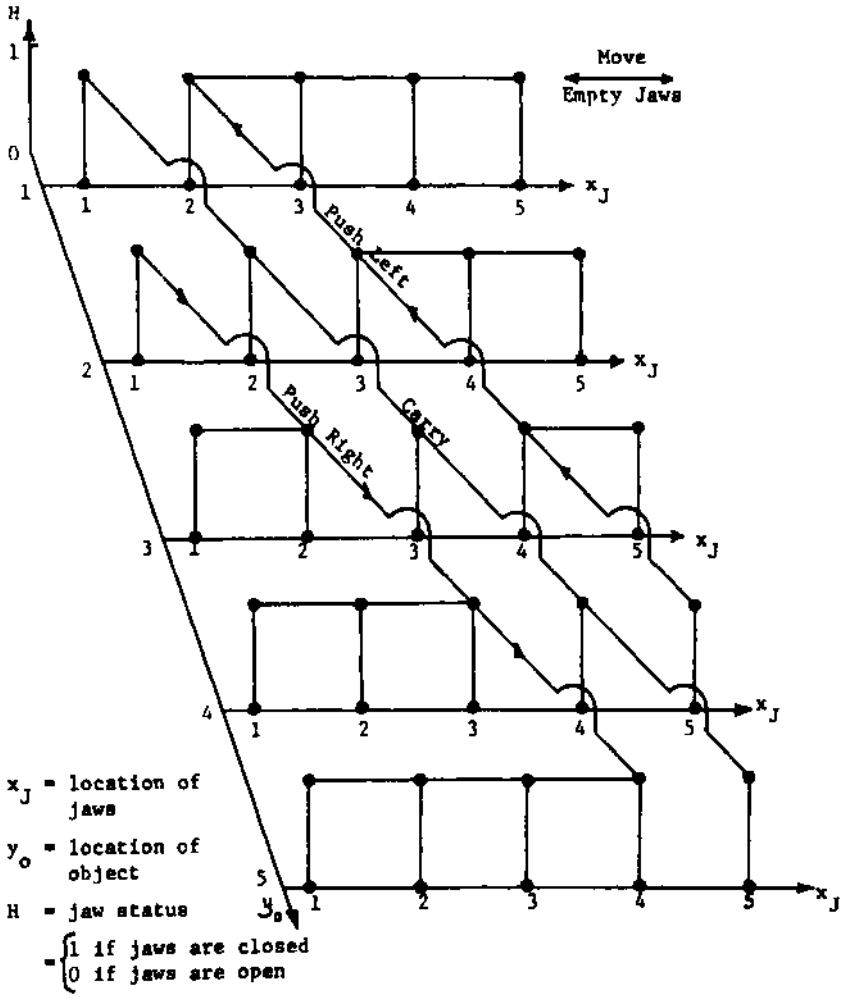


Figure 7

STATE SPACE ALLOWING JAWS TO PUSH THE OBJECT

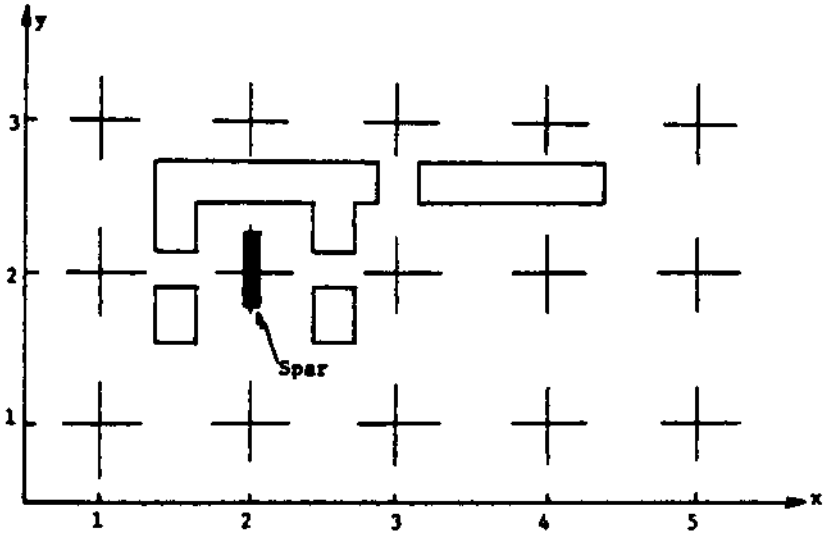


Figure 8. Physical Space for the Spar Problem

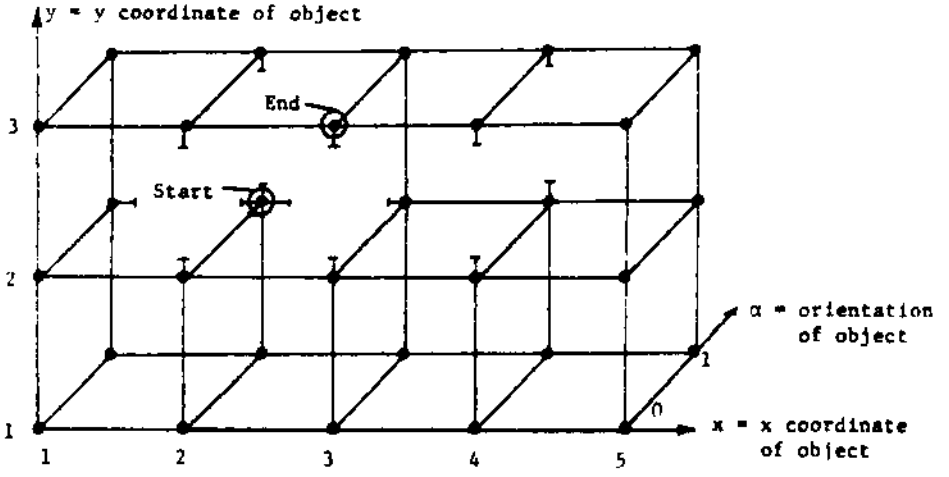


Figure 9

GRAPH OF THE PROBLEM IN FIGURE 8

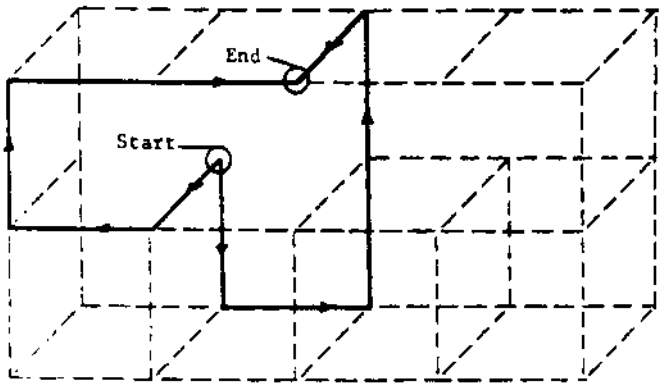


Figure 10

SOLUTION TO THE GRAPH OF FIGURE 9

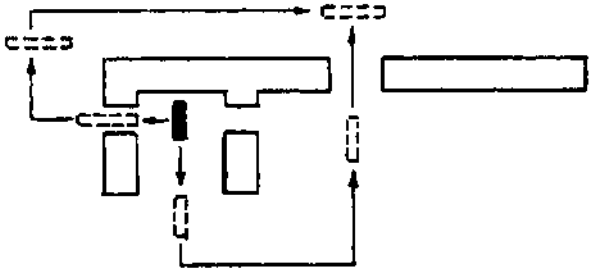


Figure 11. Visualization of Solution Path for Figure 10

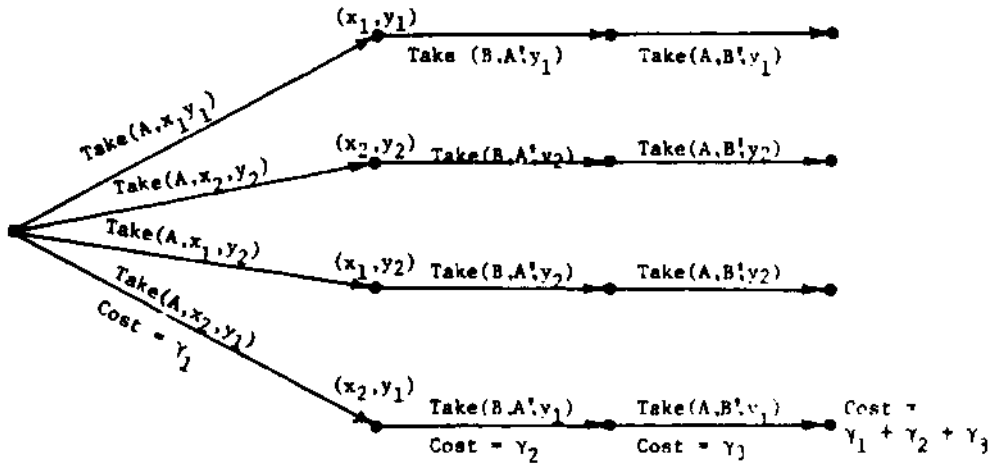
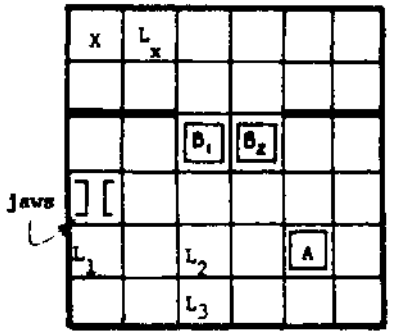


Figure 12. Tree for Evaluation of Equation (5).



Y_1, Y_2 = [illegible]

L_1, L_2, L_3 = Places to put blocks

X = Desired location for object A

L_x = Final location for jaws

Figure 13. Physical Space for the Blocks World Model.

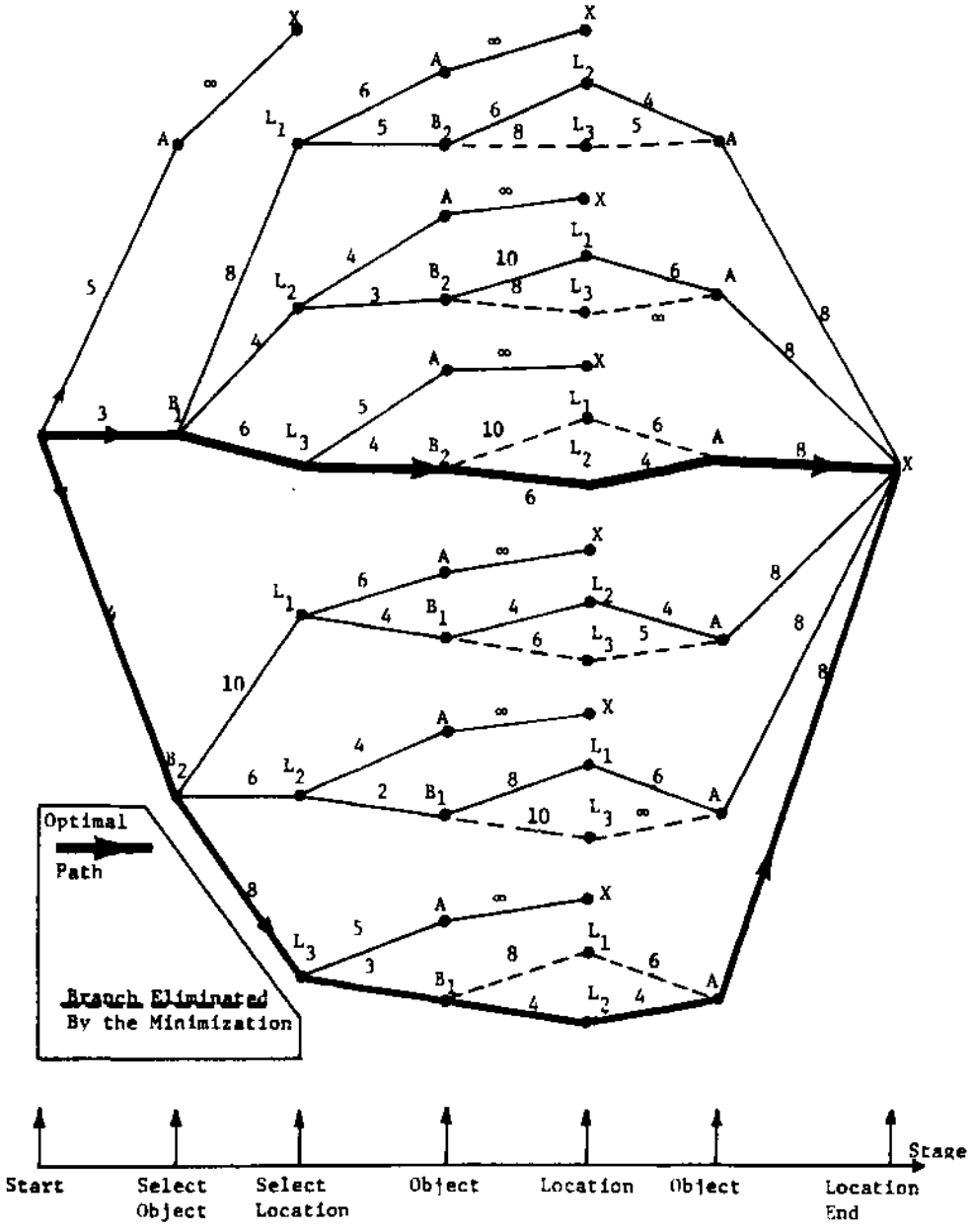


Figure 14

TREE CORRESPONDING TO SOLUTION OF THE BLOCKED DOORWAY PROBLEM IN FIGURE 13.