File No. UIUCDCS-F-86-963

Smooth Path Tracking through Symbolic Computations

By

Kaihu Chen

Artificial Intelligence Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign

July 1986
ISG 86-13

This work was supported in part by the National Science Foundation under grant DCR 84-06801, the Office of Naval Research under grant N00014-82-K-0186, and the Defense Advanced Research Projects Agency under grant N00014-K-85-0878.

Smooth Path Tracking through Symbolic Computations

Kaihu Chen

Artificial Intelligence Laboratory
Department of Computer Science
University of Illinois

ABSTRACT

The problem of robotic manipulator path tracking has been traditionally solved with numerical methods. An alternative approach that employs the techniques of symbolic computations is proposed. When given tutorial examples of path tracking, the method generates decision rules through inductive generalization from training instances. The decision rules can then be used to generate approximate solutions for smooth path tracking. The method permits the integration of task modules with low level sensor/actuator control modules. Parallel processing in the path tracking phase is also possible.

1. Introduction

The applications of the techniques developed in artificial intelligence to robotics have been mostly limited to task level problems such as planning or problem solving. One major reason for this situation is because lower level problems in robotics (such as manipulator path tracking), are numerical in nature, while artificial intelligence concerns itself mostly with symbolic computations.

Recent developments in inductive learning [Michalski, 1983; Quinlan 1983; Mitchell, 1977] indicate the possibility of the application of symbolic computations to these type of problems. Instead of solving problems with numerical methods, the problem can be transformed into *qualitative* descriptions, and then *qualitative* solutions are found using symbolic computations. Here we seek to find qualitative solutions for smooth path tracking of an arbitrary robot arm that can be characterized as a *generalized robot arm* through the technique of inductive inference.

Why would "inexact" qualitative solutions be desirable when exact solutions can be found with existing numerical methods? There are several reasons for that:

(1) Sometimes consideration over the efficiency and flexibility of the solutions overrides precision.

(2) Generalized decision rules can be obtained through inductive learning during the learning phase, so that solving the problem of manipulator path tracking can be as efficient as the time needed to retrieve and apply the decision rules.

(3) The interaction between the task level and manipulator level modules can be greatly simplified, since both modules can now speak the same language (i.e. symbolic representations). It becomes possible for task level modules to interact directly with very low level attributes such as motor speed or sensor signals. For example, given a path tracking solution in the form of a decision rule:

> If condition A holds,
>
> then activate the robot arm in a way as specified by B.

It is easy to attach to the rule as many extra conditions as the task modules call for:

> If condition A holds,
>
> and the reading of sensor #4 is less than 30,
>
> and there is no danger of knocking down anything with the elbows,
>
> then activate the robot arm in a way as specified by B.

The method proposed here is designed to solve the path tracking problem for those robot arms that can be abstractized as *generalized robot arms*. The problem of smooth path tracking is defined here as the simultaneous control of the velocity of each joint, so that the velocity at the end effector is always tangent to the given path.

2. Generalized Robot Arm

A generalized robot arm is defined with three tuples: a *configuration tuple*, a *control tuple*, and a *reference tuple*. A *configuration tuple* is a N–tuple of three dimensional vectors, where N is the number of links (degrees of freedom) :

$$<\vec{v}_1\ \vec{v}_2\ \vec{v}_3\ \cdots\ \vec{v}_N>$$

where $\overrightarrow{v_i}$, $1 \leq i \leq N$, are three dimensional vectors that represent the geometry of individually controllable links of the arm. Each vector $\overrightarrow{v_i}$ is characterized by its polar coordinate representation[1] $(r_i\ \theta_i\ \phi_i)$, with angles measured in radians. The parameter r_i defines the length of $\overrightarrow{v_i}$, while θ_i and ϕ_i jointly define the orientation of $\overrightarrow{v_i}$. The coordinates of $\overrightarrow{v_i}$ are measured relative to its reference frame R_i in the corresponding *reference tuple*. Ordinarily, the reference frame of $\overrightarrow{v_i}$ is either $\overrightarrow{v_{i-1}}$, or the shoulder (the fixed base) of the robot arm. A configuration tuple of the form:

$$<(5\ \pi/2\ 0)\ (4\ 0\ \pi/2)\ (3\ 0\ 0)>$$

indicates that the robot arm has three links, $\overrightarrow{v_1}$, $\overrightarrow{v_2}$, and $\overrightarrow{v_3}$. The first link $\overrightarrow{v_1}$ has a length of 5, pointing (with respect to its reference frame) in the direction $\theta=\pi/2$ and $\phi=0$. The second link $\overrightarrow{v_2}$ has a length of 4, pointing (with respect to its reference frame) in the direction $\theta=0$ and $\phi=\pi/2$. The third link $\overrightarrow{v_3}$ has a length of 3, pointing (with respect to its reference frame) in the direction $\theta=0$ and $\phi=0$. Connecting the world coordinates representation of these N vectors head to tail represents an abstraction of the robot arm, as shown in Figure 1. The tail (emanating point) of $\overrightarrow{v_i}$ is called the ith *joint*.

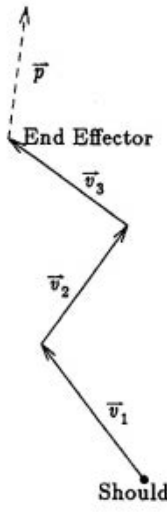

Figure 1. An abstractized three–link manipulator

[1] When superimposed with a Cartesian coordinate frame, θ_i is the angle between the projection of $\overrightarrow{v_i}$ on the x–y plane and the positive x axis. ϕ_i is the angle between $\overrightarrow{v_i}$ and the x–y plane.

The *control tuple* of a generalized robot arm is an N-tuple that takes the following form:

$$<x_1\ x_2\ x_3\ \cdots\ x_N>$$

where x_i (called *control parameters*), $1 \leq i \leq N$, is one of r, θ, or ϕ. X_i defines the dimension for which the ith link is to be controlled. For example,

$$x_4 = r$$

indicates that the fourth link is controlled by changing its length. Note that a *generalized robot arm* assumes that a link has at most one control parameter. For robot arm with joints that have more than one control parameter, pseudo links of zero length can be used to indicate the additional control without altering the physical geometry of the abstractized arm.

A typical three joint Cartesian arm as shown in figure 2 can be defined as follows[2]:

Configuration Tuple = $<(r_1\ \theta_1\ \pi/2)\ (r_2\ 0\ 0)\ (r_3\ \theta_3\ \pi/2)>$
Reference Tuple= $<shoulder\ v_1\ v_2>$
Control Tuple = $<r\ r\ r>$

A typical three joint cylindrical arm as shown in figure 3 can be defined as follows:

Configuration Tuple = $<(r_1\ \theta_1\ \pi/2)\ (0\ \theta_2\ 0)\ (r_3\ 0\ 0)>$
Reference Tuple= $<shoulder\ v_1\ v_2>$

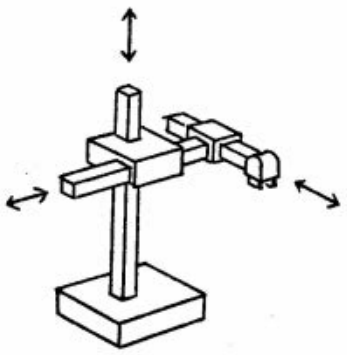

Figure 2. A typical Cartesian arm

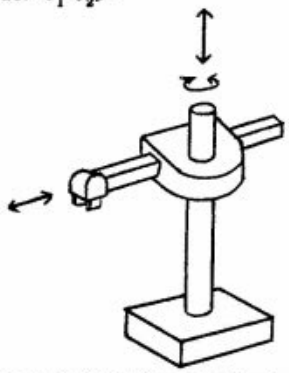

Figure 3. A typical cylindrical arm

[2] In general, the configuration tuple can be defined in more than one way, depending on the orientations of the reference frames. This is not a problem as long as the reference frame are interpreted consistently.

Control Tuple $= <r\ \theta\ r>$

The net effect of $\dot{x}_i$ (the velocity of the ith link) to the end effector, is given as a function $\vec{E}(\dot{x}_i)$ (abbreviated as $\vec{E}_i$), which defines the instantaneous direction and speed that the end effector will travel, provided that the ith link is activated with velocity $\dot{x}_i$, with all the other links being inactive.

Given the geometry of a robot arm, and the desired direction and speed of travel (expressed as a vector $\vec{p}$ emanating from the end effector of the arm), **manipulator path tracking** can be defined as the problem of finding the velocities of each link $\dot{x}_i$, such that

$$\sum_{i=1}^{N} \vec{E}(\dot{x}_i)=\vec{p}$$

The end effector will travel in the direction and speed indicated by $\vec{p}$ if each link is activated simultaneously with velocity $\dot{x}_i$, assuming instantaneous accelerations are possible[3]. Since given a set of solutions (called *solution tuple*)

$$<\dot{x}_1\ \dot{x}_2\ \dot{x}_3\ \cdots\ \dot{x}_N>$$

for a particular path tracking problem, arbitrary speed can be obtained by proportionately scaling the velocities of each link $\dot{x}_i$, in the following sections we shall concentrate only in the acquisition of solutions that generate an unit length $\vec{p}$.

3. Inductive Learning

Inductive learning is a process of acquiring knowledge by drawing inductive inferences from given examples. This process involves operations of generalizing and transforming knowledge representations in order to accommodate given examples and satisfy various additional criteria. The generalized knowledge can be viewed as the condensed version of the given examples.

---

[3] The problem of dynamics is ignored here.

Our approach to the problem of robotic manipulator path tracking begins with a *learning phase* that acquires generalized decision rules from tutorial examples, then followed by a *problem solving phase* that actually solves the path tracking problem. During the learning phase, the system accepts as input preclassified training examples for tracking a given path at a certain time instant, and produces as output generalized decision rules for each decision class. The control of the arm is decided in the problem solving phase with the rules learned from the learning phase.

Each tutorial example is given as a set of attributes $\{A_i\}$, plus a solution tuple S that can generate a unit length $\overline{p}$ at the end effector. The pair $(\{A_i\}, S)$ represents an instance of a correct decision in the following sense:

> If the world (of the robot arm) has attributes $\{A_i\}$,
>
> then activate the arm in a way as specified by S
>
> to generate an instantaneous end effector velocity
>
> of unit size as specified by $\overline{p}$.

The set of attributes can be viewed as the properties of the training instance, and the solution tuple is the class designator for the training instance. A solution tuple that doesn't generate unit end effector speed can be proportionally scaled before being fed to the learning algorithm. Training instances can be obtained through either teach pendants, lead–through actions, off–line data dump, or even directly from spontaneous random exercise of the robot arm.

Remember that a solution tuple is an N–tuple of real numbers, which corresponds to a N–dimensional space. Generating a class designation from this N–dimensional solution space can be achieved by dividing the space into many small hypercubes, with each hypercube corresponding to a class. For example, the velocity of each link $\dot{x}_i$, can be quantized into three ranges: positive, negative, or zero, according to the sign of its value. In this way, 3^N classes can be generated. Characteristic descriptions [Michalski, 1983] for each class can be then generated with an inductive algorithm.

The natural question that comes to mind is: can sensible solutions be found with this approach? The worst form of inductive learning could deteriorate to rote learning if no meaningful generalization can be found. The crux of the problem lies in the selection of the right kind of attributes for the inductive learning algorithm. The simplest case occurs when the true description for each class is *non–disjoint*, *convex*[4], and *non−overlapping* with the descriptions of other classes. By *non–disjoint* we mean that the description for each class (the conditions of a decision rule) contains no disjunctive terms; by *convex* we mean that the closing–gap generalization is truth preserving[5]. It is a known fact that our approach can guarantee neither the non–disjointness nor the non–overlapping properties. However, we do believe that a limited form of convexity property can be obtained through a technique called *quadrant separation*, which will be explained in the following sections.

Choosing the right attributes for an inductive learning system is a profoundly difficult problem. Robot arms with redundant links, such as a Cartesian arm with two links both controlling the x–dimension, poses additional difficulty since what's important is not the individual velocities of the two x–dimension links, but the *sum* of the two. The problem of constructing interesting attributes by the arithmetic combinations of the existing attributes has been an intensively investigated topic in machine learning [Falkenhainer 1984; Langely, Bradshaw & Simon, 1981]. Although this technique is not used here, it is clear that advancement in this field can be readily introduced to improve the performance of the system. A set of basic attributes that enable the system to acquire meaningful rules for uncomplicated cases is given in the following sections.

---

[4] Here we are concerned only with the *convexity* with respect to the closing–gap generalizations [Michalski, 1983], since only real numbers are involved.

[5] That is, given two rules

$$R_1 \equiv Conditions_1 \rightarrow Decision_k$$
$$R_2 \equiv Conditions_2 \rightarrow Decision_k$$

that are both true, the following must also be true:

$$closing-gap(Conditions_1, Conditions_2) \rightarrow Decision_k.$$

Convexity implies non–disjointness, but the reverse is not necessarily true.

3.1. Representation

The description of each class can be represented in the form of a decision rule

$$Conditions \rightarrow Decision$$

Conditions is represented as a VL_1 complex [Michalski, 1983]. A VL_1 complex is a conjunction of relational statements called *selectors*. A *selector*, representing a logical statement concerning the value range of an attribute, is of the following form[6]:

$$[Attribute_i = lower-range \; .. \; upper-range]$$

where both *lower−range* and *upper−range* are real numbers. For example, the VL_1 complex

$$[Angle = 35.3 \; .. \; 40.0][Length{=}3.4 \; .. \; 5.5]$$

indicates that the value of the attribute *Angle* falls inclusively in the range between 35.3 and 40.0, *and* the value of *Length* falls inclusively in the range between 3.4 and 5.5. The conjunction between adjacent selectors is implicit. Training instances also take the form of decision rules. In the following sections the term *rules* are used to refer specifically to the generalized decision rules derived from training instances. A training instance $C_1 \rightarrow D_1$ is *covered* by a rule $C_2 \rightarrow D_2$ if and only if D_1 and D_2 are identical, and C_1 tautologically implies C_2. For example, the event

$$[Angle = 4 \; .. \; 5][Length{=}6 \; .. \; 8] \rightarrow D_5$$

is covered by the rule

$$[Angle = 2 \; .. \; 5][Length{=}4 \; .. \; 12] \rightarrow D_5.$$

The **closing-gap generalization** can be characterized as the operation of eliminating the gap between two ranges. For example, the following two decision rules

---

[6] The definition of *selector* given here is only a small subset of its permissible forms defined in [Michalski, 1983].

$$[Angle = 35.3 \,..\, 40.0][Length{=}3.4 \,..\, 5.5] \rightarrow D_5$$

$$[Angle = 20.5 \,..\, 25.5][Length{=}4.4 \,..\, 7.8] \rightarrow D_5$$

can be generalized to

$$[Angle = 20.5 \,..\, 40.0][Length{=}3.4 \,..\, 7.8] \rightarrow D_5.$$

Note how the value ranges of the same attribute in the lefthand side of the two decision rules are "merged" to generate a new generalized hypothesis for D_5.

3.2. Attributes and Algorithm

The selection of proper attributes for the inductive learning algorithm is of utmost importance in any learning system. The attributes must be chosen in such a way such that simple class descriptions can be found. For example, if a learning algorithm is given only the absolute coordinates of the arm geometry, then it is highly unlikely that any meaningful generalized decision rule can be found.

We choose to use attributes that manifest the relationship between the desired direction of travel $\vec{p}$, and the status of the arm. Specifically, the following attributes are used:

(1) Attributes EP_i, $1 \leq i \leq N$ (N is the total number of links), where EP_i is the angle between $\vec{E}_i$ and $\vec{p}$.

(2) Attributes VP_i, $1 \leq i \leq N$, where VP_i is the angle between $\vec{v}_{iN}$ and $\vec{p}$, $\vec{v}_{iN}$ is the vector from the ith joint to the end effector.

(3) Attributes L_i, where L_i is the length of $\vec{v}_{iN}$.

The EP attributes represent the relationships between the effects of link velocities and the desired direction of travel $\vec{p}$. The VP and L attributes represent the relationships between the status of the robot arm and $\vec{p}$. Angles, measured in radians, are positive real numbers between 0 and π. The value of an angular attribute, which represents the angle between $\vec{p}$ and e.g. $\vec{E}_i$, can be divided into two *quadrants* (the other two quadrants are unused). The first quadrant has a value range between 0 and $\pi/2$, and the second quadrant has a value range between $\pi/2$ and π. The two quadrants represent the two qualitative categories of the relationship between $\vec{E}_i$ and $\vec{p}$. A first quadrant angle indicates that $\vec{E}_i$ has a positive

component on $\vec{p}$. Activating the ith link with positive velocity will cause the end effector to move more in the direction of $\vec{p}$. A second quadrant angle indicates exactly the opposite. It is thus wise to avoid generalization across quadrant boundaries, since class descriptions are unlikely to scatter across quadrant boundaries. This is called the *Principle of Quadrant Separation*.

Since there is usually more than one solution for path tracking[7], each training instance can belong to more than one class. Each training instance is a *positive* example for its designated class, but cannot be viewed as a *negative* example for other classes. For this reason, an inductive learning algorithm that is capable of generating *characteristic description* is required. A simple algorithm based on the closing-gap generalization and the *Principle of Quadrant Separation* is given below:

(1) Initialize the rulebase RB to nil.
(2) Input a training instance $T \equiv C \rightarrow D$.
(3) Let IR be the set of rules in RB that have the same righthand side as T:

$$IR \equiv \left\{ R_i \mid R_i \equiv C_i \rightarrow D \right\}$$

(4) Generalize R_i to

$$R_i \equiv closing-gap(C, C_i) \rightarrow D$$

if the *principle of quadrant separation* is not violated.
Replace all generalized R_i back into RB.
If no rule in IR has been generalized, then insert T into RB.
(5) Exit if there is no more input. Otherwise go to (2).

The quantization of the solution tuples into qualitative classes also merits some discussion here. In general, the total number of classes is exponentially proportional to the number of levels used in the quantization of each dimension of the solution space. The finer the quantization, the more precise the final solutions will be, and the more burden the inductive engine must bear. For a solution space that is unbounded (i.e. $-\infty \leq \dot{x}_i \leq \infty$) due to normalization, logarithmic or variable quantization schemes can be used.

The efficiency of the learning algorithm relies greatly on the quality of the tutorial examples. Good tutorial examples allow the learning algorithm to skip unpopulated parts of the solution space, and con-

[7] This problem is also common in the similar inverse kinematics problem.

centrate on finding characteristic descriptions only for those classes that are present in the tutorial examples. Thus although the number of potential classes might seem to be formidable at first sight, actually only a small portion of them will be explored.

4. Path Control with Decision Rules

Equipped with the decision rules generated from the learning phase, we are now ready to solve the manipulator path tracking problem for a given robot arm. Given the status of the robot arm and the desired direction of travel $\overline{p}$ at a certain time instant, the attributes can be calculated. If the conditions of a decision rule are satisfied by the calculated attributes, then the righthand side of the rule represents an approximate solution to the problem. Those decision rules whose conditions are satisfied by the calculated attributes are said to be *invoked*. Path tracking solutions can be recalculated at regular time intervals to allow the robot arm to trace the path smoothly. It is obvious that it is most likely that more than one rule will be invoked. Multiple invocation of rules indicates the presence of alternative solutions. Alternative solutions can be handled in one of two ways:

(1) Select the solution that satisfies predefined criteria, e.g. select the solution that is most efficient, most commonly used, doesn't violate the arm geometry or working space geometry, etc.

(2) The average of the invoked solutions can be used in order to achieve greater precision.

It is also possible that no rule is invoked due to insufficient data points during the learning phase. In this case, a distance function can be defined so that the rule whose conditions that is closest to the computed attributes is invoked.

5. Example and Performance Evaluation

A simple two dimensional, two–link robot arm is given below to demonstrate the proposed method. This two dimensional robot arm with two rotary joints can be characterized as follows:

$$\text{Configuration Tuple} = \langle (5\ \theta_1)\ (4\ \theta_2) \rangle$$
$$\text{Reference Tuple} = \langle \textit{shoulder}\ v_1 \rangle$$
$$\text{Control Tuple} = \langle \theta\ \theta \rangle$$

Note that the length of the first link is 5, and the length of the second link is 4. The solution space, an unbounded two dimensional space, is quantized into 0.02 by 0.02 squares. The unboundness of the solution space is due to a singular point at the position where the two links are aligned in a straight line. Only two directions of movement at the end effector are possible at this position. It takes great link velocities to generate a unit size $\overline{p}$ near the singular point, where computation errors are also magnified. Although there are potentially infinite number of classes, a relatively small number of rules were generated. The robot is simulated with a program written in Franz Lisp running on a Sun–2/170 workstation. The status of the arm and the desired direction of travel $\overline{p}$ were generated randomly, its solution tuple was then computed numerically in order to generate a complete training instance. Only those rules that cover more than one training instance are output by the learning program. In the problem solving phase, the status of the arm and the desired direction of travel $\overline{p}$ were also generated randomly, the attributes for the arm are then computed. To get an estimation of the precision of the generated rules and the sparseness of the solutions, no distance function is used to get the "nearest" solution. If no rule in the rulebase is invoked by the attributes, it is simply discarded and a new training instance is generated; otherwise the average of the invoked solutions are used. Table 1 shows the results of five experiments using different quantum sizes and number of training instances. The data in table 1 shows that the number of decision classes increases as smaller quantum and more training instances are used. The precision also improves with smaller quantum and more training instances. By avoiding generating training/testing instances that are 15 degrees within the singular point, the precision of the results were improved by thirty to fifty percents.

Two problems are immediately evident from the results shown in table 1: the large number of the rules generated, and the relatively low precision of the solutions. One way to cut down the number of

Number of Decision Classes	Quantum Size	Total Number of Training Instances	Average Error (in degrees)	Total # of Test Cases	# of Test Cases not Covered
74	0.02	1000	28.95	500	225
187	0.02	5000	27.52	500	84
120	0.01	1000	14.19	500	330
320	0.01	5000	13.95	500	160
520	0.005	5000	7.78	500	277

Table 1. Results of simulations with a 2–D 2–link spherical robot arm.

rules and to improve precision is to divide all links into several groups, e.g. a transportation group, a dexterity group, and some groups between these two extremes. The links in the transportation group are responsible for transporting the end effector over relatively long distances, where efficiency overrides precision. The links in the dexterity group are responsible for precise local operations where precision overrides efficiency. These groups of links can be trained independently with different requirements in efficiency and precision, thus greatly cut down the number of rules need to be generated, while at the same time allows more extensive training for higher precision. Given a task of reaching for an object, a task module would plan a path and decide which group is to be activated, depending on the distance between the object and the end effector, plus possibly other factors. For example, given a mobile robot, the mobility links (the wheels or "legs" of the robot) can be considered as the long–range transportation links that are activated whenever the distance between the end effector and the object is greater than ten feet. The second group of links (the "arms" of the robot) are activated if the distance is between 0.5 and 10 feet. The third group (the "hand" and "fingers" of the robot) are activated if the distance is less than 0.5 feet. In this way, no only have we cut down the complexity in the learning phase, but we are also able to integrate the various actuators of a robot system into one integral unit.

6. Conclusions

We have shown here how the problem of smooth path tracking for a robot arm can be transformed into a form where symbolic computations can be applied. The simplified example demonstrated the viability of this approach. Raibert [1977] approached the problem of motor control in a similar way by acquiring the inverse equation in tabular form through learning. The model of the target robot arm is based on the Newtonian equations of the rotary motions of the arm. Contrary to Raibert's parametric representation, here attributes that explicitly describe the cause–effect relationships between joint velocities and the resultant arm status are used. Representing motor control knowledge in rule form allows rule–based task level planning systems to easily access/incorporate the knowledge acquired by the learning system. We argue here that parametric representation of motor control failed to uncover the cause–effect relationships between joint velocities and the resultant arm status, thus reduces the usefulness of the knowledge

acquired by such a learning system to other task level reasoning or planning modules. Representing mobile skill in symbolic forms also allows tighter coupling between high level task modules and low level manipulator control modules. The coupling between multiple arms, or other mobile parts of a robotic system, can also be simplified, since the coordinations between independent symbolic systems are considerably simpler than those of between independent numerical systems.

It is possible to obtain solutions of various precisions depending on the requirements. For example, if rough approximation already suffices, than any invoked rule can be chosen as the solution. If higher precision is desired, than all invoked rules can be processed to obtain a more precise solution. It is also possible to adopt progressively finer quantum sizes for the learning algorithm, so that the precision can improve incrementally over time.

Note that using decision rules for path tracking involves two major steps: the computation of the attributes, and the invocation of the rules. Since there is no dependency between either the attributes or the decision rules, both the computation of the attributes and the invocation of rules can be done in parallel. That is, given appropriate parallel machines, path tracking with decision rules can be achieve in constant time, independent of the number of joints of the robot arm, or the number of decision rules.

The efficiency and the symbolic nature of this approach has potential applications in realtime or complex domains where environments are unpredictable and changing rapidly, and for applications that require complex interactions between the task level modules and the manipulator control modules.

ACKNOWLEDGEMENTS

The author is grateful to the following people for their comments and suggestions: Stephen Lu, Ryszard Michalski, Mark Spong, Robert Stepp, Heedong Ko, and Carl Kadie. This work was supported in part by the Defense Advanced Research Project Agency under grant N00014–K–85–0878, National Science Foundation under grant DCR–84–06801, and Office of Naval Research under grant N00014–82–K–0186.

REFERENCES

[1] Falkenhainer, B. *ABACUS Adding Domain Constraints to Quantitative Scientific Discovery,* ISG 84–7, UIUCDCS–F–84–927, Department of Computer Science, University of Illinois, Urbana, November 1984.

[2] Langley, P., Bradshaw, G. L., Simon, H. A., *BACON:5 The Discovery of Conservation Laws,* Proceedings of the 7th International Joint Conference on Artificial Intelligence, August 1981, pp. 121–126.

[3] Michalski, R. S. *A Theory and Methodology of Inductive Learning,* In: **Machine Learning, An Artificial Intelligence Approach,** R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds. Tioga, Palo Alto, CA, 1983, pp. 461–481.

[4] Michalski, R. S., *Knowledge Repair Mechanisms: Evolution versus Revolution,* Proceedings of the Third International Machine Learning Workshop, 1985, pp. 116–119.

[5] Mitchell, T.M., *Version spaces: an approach to concept learning,* Ph.D. Thesis, Stanford University, Stanford, CA 1978.

[6] Quinlan, J. R. *Learning Efficient Classification Procedures and their Application to Chess End Games.* In: **Machine Learning, An Artificial Intelligence Approach,** R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds. Tioga, Palo Alto, CA, 1983, pp. 461–481.

[7] Raibert, Marc H., "Motor Control and Learning by the State Space Model", Ph.D. Thesis, Cambridge, Massachusetts, M.I.T. 1977.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-86-963	2. ISG 86-13	3. Recipient's Accession No.
4. Title and Subtitle Smooth Path Tracking Through Symbolic Computations			5. Report Date July 1986 6.
7. Author(s) Kaihu Chen			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF DCR 84-06801 N00014-82-K-0186 N00014-85-K-0878
12. Sponsoring Organization Name and Address National Science Foundation, Washington, D.C. Office of Naval Research, Arlington, VA Defense Advanced Research Project Agency, Arlington, VA			13. Type of Report & Period Covered 14.
15. Supplementary Notes			

16. Abstracts

The problem of robotic manipulator path tracking has been traditionally solved with numerical methods. An alternative approach that employs the techniques of symbolic computations is proposed. When given tutorial examples of path tracking, the method generates decision rules through inductive generalization from training instances. The decision rules can then be used to generate approximate solutions for smooth path tracking. The method permits the integration of task modules with low level sensor/actuator control modules. Parallel processing in the path tracking phase is also possible.

17. Key Words and Document Analysis. 17a. Descriptors

Robotics
Path Tracking
Inductive Learning

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 18
	20. Security Class (This Page) UNCLASSIFIED	22. Price