Parameter Fine-Tuning for Robots

Yilong Chen

ABSTRACT: Fine-tuning the parameters of robots is a time-consuming and costly procedure. Because many commercial robot companies cannot afford to do fine-tuning, the physical capability of robots cannot be fully realized. In this paper, a computer procedure that automatically tunes parameters of feedback compensators for robot arms in order to circumvent manual tuning and maximize capability is outlined. Simulation and test results show that this autotuning is robust and efficient. Some practical issues about real implementation are discussed.

Introduction

In most control strategies for robots, proportional plus derivative (PD) or proportional plus integral plus derivative (PID) controllers have been used successfully. To eliminate trade-offs among static error, stability, model structure inaccuracy, and high-frequency noise rejection in the cases where the sampling rates are critical, a lag-lead compensator has been designed to replace a PD (or PID) controller for use in the robot application described in [1]. As a result of model inaccuracies and unmodeled disturbances, the control parameter values cannot be derived analytically. Both the lag-lead compensators and the PD (or PID) controllers must be tuned before they can be used on robot arms. A previous paper [2] discussed the necessity and difficulty of parameter tuning, and a formulation of automatic parameter tuning by learning (called AutoTune) was developed to overcome the difficulty. This paper discusses further the feasibility of AutoTune and shows simulation and implementation test results.

If there is not too much interaction among different parameters, it is possible to tune compensators with three to four parameters by hand. However, with more complex compensators for robots that are highly coupled between joints, it is necessary to have suitable tuning tools. Traditionally, tuning of more complex compensators has followed the route of modeling or identification and compensator design [3]. This is often a time-consuming and costly procedure. In the ROBODRIVE phase I test, it took about one week for a skilled person to tune manually three parameters for just one joint of a PUMA 560. It would be almost impossible to do manual fine-tuning work for all six coupled joints. This is equivalent to finding one optimal (or suboptimal) point in an 18 (or even higher)-dimensional space.

Many commercial robot companies cannot afford to do fine-tuning work for individual robots. Instead, they usually tune parameters only once for one type of robot, based on the assumption that coupling between joints can be neglected unless a particular robot goes wrong and has to be reset with separate tuning of each joint. Moreover, they are not able to do retuning work that is necessary when the condition of a robot is changed as a result of, for example, gear system wear, friction changes, or load variation. As a result, parameters of compensators have to be tuned in a very conservative way (e.g., lower feedback gains). Thus, the physical capability of robots cannot be fully realized because of the important role played by the parameters of a feedback compensator. This becomes a particularly serious problem when robots must be operated with high-accuracy performance at high speed.

Some people have proposed the so-called "self-tuning" technique to overcome this difficulty [3]. This technique has an on-line computational burden, and even if a given plant is originally linear, the closed-loop system obtained with self-tuning control is a nonlinear system, which is difficult to analyze and control. For a nonlinear system like a robot, it is even more difficult to understand how the system works. The essential problem is to determine how the closed-loop system behaves under self-tuning control.

The so-called "gain scheduling" technique is another way to tune parameters of a regulator [3]. With this technique it is necessary to find auxiliary process variables that correlate well with changes in process dynamics. One drawback of gain scheduling is that it is open-loop compensation. There is no feedback that compensates for an incorrect schedule. Another drawback is that the design is also time-consuming.

Because of the existence of unknown characteristics inherent in robot systems, such as friction, backlash, and nonrigidity, it is difficult to estimate a full description (or model) of the system from a given set of system inputs and corresponding outputs. Since more than 80 percent of currently practical robot applications involve cases where robots are used in a cyclic (or repetitive) manner, such as spray painting, spot welding, and parts assembly, this paper proposes off-line automatic tuning by a learning method that bypasses the system model. First, an objective function, which reflects performance features expected by tuning a compensator, is defined. Then an algorithm is used to automatically tune the compensator parameters and produce a smaller objective function value.

In the next section, an objective function is defined, which reflects the desired features of the closed-loop system—i.e., stability, tracking accuracy, oscillation level, static accuracy, and minimum peak error [2]—and the automatic tuning problem is formulated as an optimization problem. A brief discussion of various optimization methods is given, and a pattern search method [4], [5] is chosen to solve this optimization problem. In the following section, the AutoTune procedure for parameter tuning is outlined, along with a flow diagram of the package. AutoTune can be implemented by using either simulation or real robot runs. The subsequent section illustrates simulation and real test results starting from different initial parameters. These results show that with AutoTune the performance of a PUMA 560 robot is as good as, if not better than, that obtained using manually tuned parameters. This package takes about 1.5 hr of real robot runs or less than 2 hr of computer simulation on an IBM 3090 to tune automatically a set of parameters for a robot arm with three joints. It will take a few more hours to tune all six joints; this compares with weeks required for manual fine-tuning by skilled people. The AutoTune package allows (1) fine-tuning accounting for coupling effects between multiple joints, (2) fine-tuning for each individual robot, and (3) retuning needed as the operation of a robot is changed.

Problem Formulation

In a previous paper [2], a lag-lead compensator for robot feedback control was de-

Presented at the 1988 IEEE Conference on Robotics and Automation, Philadelphia, Pennsylvania, April 24–29, 1988. Yilong Chen is a Staff Research Scientist at the General Motors Research Laboratories, Warren, MI 48090-9055.

0272-1708/89/0200-0035 $01.00 © 1989 IEEE

signed. For the ith joint, the compensator has the following form in the w plane, where K_i, L_i, d_i, β_{1i}, and β_{2i} ($i = 1, 2, \ldots, 6$) are constant parameters to be selected.

$$H_i(w) = K_i \frac{[w/L_i + 1]}{[\beta_{1i} w/L_i + 1]} \frac{[w/d_i + 1]}{[w/\beta_{2i}\, d_i + 1]} \tag{1}$$

For the ith joint, the number of K_i is the gain of the compensator, and L_i and d_i are known as the corner frequencies. In order to tune these parameters, it is necessary to establish a performance goal for the compensator. It is important to select a performance measure that reflects the quantitative and qualitative features desired in the autotuning system. This can be done by introducing an objective function that has the desired properties of the closed-loop system.

For this case, the desired properties for a given robot are related to (1) final position, (2) accurate tracking, and (3) smooth operation. The final position of the end-effector of a robot should hit the target point within a small tolerance, i.e., high static accuracy. There should be accurate tracking of axis trajectories. Since a large oscillation of errors usually means a system is less stable, and because it also causes gear wear, the errors during tracking desired paths should be smooth—i.e., first- and second-order derivatives, perhaps also third-order derivatives—should be small, and peak errors of joints should be limited.

Therefore, the following objective function S is chosen, where N is the number of joints of a given robot (usually, $N = 6$); the variables i and k relate to the ith joint position at the kth sample period; $y_i(k)$ is the desired joint position; $x_i(k)$ is the actual joint position measured by the optical encoder; and $e_i(k)$ is the error defined as y minus x. The variable EF_i is the ith joint final position error; EP_i is the ith joint peak position error; and R_1, R_2, R_3, R_4, and R_5 are weighting factors.

$$\begin{aligned} S = {} & \sum_k R_1 \sum_{i=1}^{N} e_i^2(k) + \sum_k R_2 \sum_{i=1}^{N} [e_i(k) \\ & - e_i(k-1)]^2 \\ & + \sum_k R_3 \sum_{i=1}^{N} [e_i(k) - 2e_i(k-1) \\ & + e_i(k-2)]^2 \\ & + R_4 \sum_{i=1}^{N} (EF_i)^2 + R_5 \sum_{i=1}^{N} (EP_i)^2 \end{aligned} \tag{2}$$

The predescribed objective function S is minimized over all possible values of the parameters of the feedback compensator, represented by the vectors K, L, β_1, D, β_2, where the components of the vector K are designated K_i, and so on for the other parameters.

There are various methods for minimizing a function of several variables [6], [7]. One of the most widely known methods is the gradient method, which depends on knowledge of the first derivative of the function. Another well-known method is the conjugate gradient method. A striking characteristic of this method is that when the function to be minimized is a quadratic function of N variables, the minimum can be reached in, at most, N conjugate gradient steps. The Hessian matrix is the partial derivative of the gradient vector. For this method, since the Hessian matrices change as new points are generated, the Hessian matrices of nonquadratic functions should change slowly for the method to be successful. This method and also the Newton-Raphson method need second-order derivatives of the function, which are obtained from the Hessian matrix. The Fletcher-Reeves conjugate gradient method does not require explicit knowledge of the Hessian and is effective in minimizing general functions, but it still requires the gradients of the function.

Having knowledge of first- and second-order derivatives, these methods generally are more efficient and generally will approach the minimum in the least number of steps. For the complicated robot arm system, however, notice that S defined in Eq. (2) is a complicated function of K, L, β_1, D, and β_2. In fact, it is almost impossible to give an explicit function $S(K, L, \beta_1, D, \beta_2)$, especially when model uncertainty and disturbances are considered. Both gradient vectors and the Hessian matrices of S are very difficult to obtain explicitly by analytical methods, and their numerical approximations are very crude. Sometimes the functions might even have discontinuous first derivatives. Notice also that the methods discussed earlier all perform searches for local minima along a sequence of directions. When the function has many local extrema, they are likely to be trapped at false local extrema. For the robot system, it is difficult to guarantee that the function S can have only one extremum. On the other hand, the terms that are needed to calculate the function S in Eq. (2) can be determined by either simulations or actual robot runs in which errors are computed routinely by measuring joint position angles from optical encoders. Therefore, in this paper, a direct search method-pattern search [4], [5] is used to solve the optimization problem. Although the theoretical convergence of the pattern search is still an open question, compared to the methods discussed previously, the pattern search has some distinguished features that suit our needs well: (a) It can be used for an objective function with multipeak values. (b) It does not need explicit knowledge of the objective function. Values of the objective function can be discrete and derivatives are not needed. (c) It does not require the neighborhood of the global extremum to be well behaved. (d) It needs, however, a large amount of computer time. This is acceptable for our off-line tuning process, where the amount of computer time is of secondary importance.

AutoTune

In the previous section, the optimization problem was formulated, and the pattern search method was chosen to solve it. We now can build an automatic tuning package (called AutoTune) for the design of the robot feedback compensator. But, first, a general description of the pattern search method is provided. For solving the given problem, the pattern search method initially explores a neighborhood of a chosen point ψ, where ψ is defined as a 30-dimensional vector composed of the five six-dimensional vectors K, L, β_1, D, and β_2.

If a promising direction is found, then the function S is evaluated in the same direction using a larger and larger step size. That is, a pattern or a trend of search is established, as long as the search continues to be successful in locating points with lower $S(\psi)$. If the search fails, that is, a point with a lower functional value cannot be found in the direction of the established pattern, then the step is decreased. After several consecutive step-size decreases, the pattern is destroyed and a new exploratory search is initiated. Thus, this method seeks to find the directions of the ravines of the objective function and tries to follow them. The method has the useful feature of ravine relocation using the moves in the exploratory phase if the pattern moves fail to follow curved ravines. As this brief description indicates, the method is simple to program but has only an intuitive justification for it. Convergence and convergence rates are so far still an open question.

AutoTune is described in the accompanying figures in this section. The flow diagram for this package is given in Fig. 1. The sequence following label 1 is for an initial set of exploratory moves from a base point when a new pattern must be established. The sequence following label 2 is the basic iterative loop consisting of a pattern move followed by a set of exploratory moves. The sequence

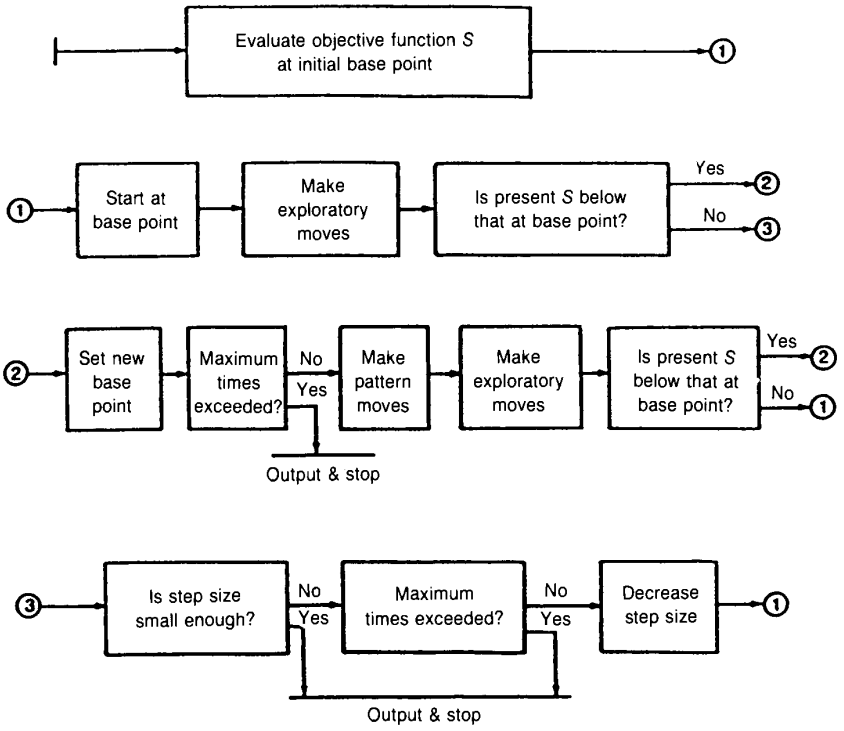


Fig. 1. Descriptive flow diagram for AutoTune.

labeled 3 controls the reduction of step size and termination of the search.

Figure 2 gives the details of the procedure. Explicitly, the procedure is carried out by sequentially transforming the following set of variables: θ, the previous base point, ψ, the current base point, and ϕ, the base point resulting from the current move, all in a 30-dimensional space; $S(\psi)$, the objective functional value at the base point; $S(\phi)$, the objective functional value for this move; S, the objective functional value before this move, usually, the smallest value so far attained by the set of exploratory moves; Δ, current step size; K_{eval}, current number of function evaluations; I_{maxev}, maximum number of function evaluations permitted; δ, minimum step size permitted (i.e., the procedure is terminated when step length is less than δ); "super," simulation subroutine calculating position errors and functional value of S; ρ, reduction factor for step size (ρ is less than 1); and ϕ_k, one of the coordinate values for ϕ, $k = 1, 2, \ldots, 30$. In Fig. 2, which has been drawn to parallel Fig. 1, a flow diagram is exhibited in terms of the problem variables.

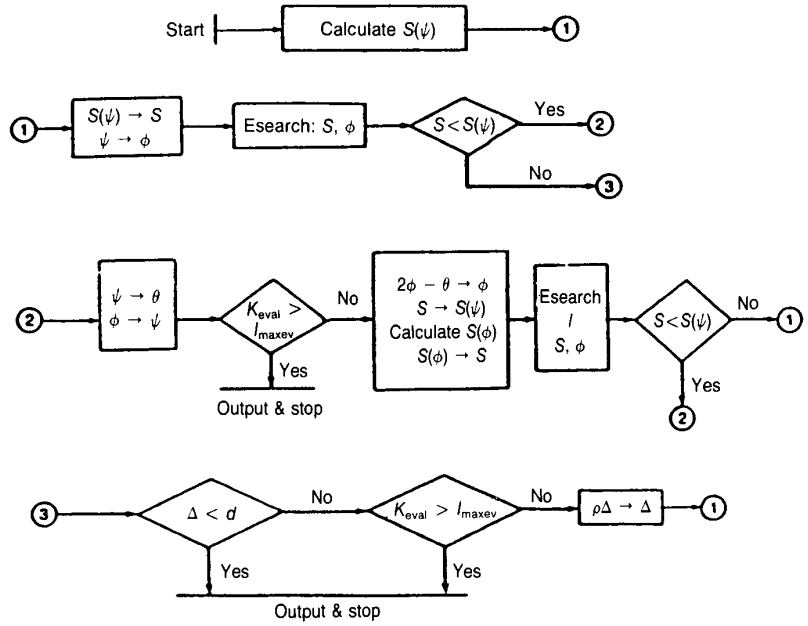


Fig. 2. Detailed flow diagram for AutoTune.

Implementation

In AutoTune, the value of the objective function S must be calculated each time one (or more) parameter of the lag-lead compensator changes. This can be done in one of two ways: either by a simulation or by a real robot run. In this section, we will use the package designed in the previous section to evaluate the automatic tuning concept for a PUMA 560, first by simulations and then by real tests.

With simulation, each time a parameter changes, a subroutine is called, tracking errors of joint positions are obtained, and the objective function is calculated from Eq. (2). In this simulation, a maximum load (5 lb) and a third-order polynomial path with a joint motor speed of 100 rad/sec are used for tracking. To save computer processing time, we tune only K_i, β_{1i}, and d_i here by simulations, and leave L_i and β_{2i} fixed for $i = 1, 2, \ldots, 6$. If we want to tune all five parameters, the principle and the tuning process are the same. The starting point, $\psi_0(k)$, follows: $\psi_0(1) = K_1 = 500.0$; $\psi_0(2) = K_2 = 500.0$; $\psi_0(3) = K_3 = 500.0$; $\psi_0(4) = \beta_{11} = 18.0$; $\psi_0(5) = \beta_{12} = 18.0$; $\psi_0(6) = \beta_{13} = 18.0$; $\psi_0(7) = D_1 = 170.0$; $\psi_0(8) = D_2 = 170.0$; and $\psi_0(9) = D_3 = 170.0$. The next variables chosen are $L_i = 10.0$, $\beta_{2i} = 5.0$, for $i = 1, 2, 3$, and $\Delta = 0.1$, $\delta = 0.03$. The weighting factors $R_1, R_2, \ldots, R_5$ are determined in the manner discussed in the last section. AutoTune requires 1.5 hr on an IBM 3090 to produce the following tuned results:

$$K = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix} = \begin{bmatrix} 600.0 \\ 675.0 \\ 600.0 \end{bmatrix}$$

$$\beta_1 = \begin{bmatrix} \beta_{11} \\ \beta_{12} \\ \beta_{13} \end{bmatrix} = \begin{bmatrix} 14.4 \\ 14.4 \\ 18.9 \end{bmatrix}$$

$$D = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = \begin{bmatrix} 204.0 \\ 127.5 \\ 195.5 \end{bmatrix}$$

Selecting the tuned parameters for the feedback lag-lead compensator gives the tracking errors for the third joint motors as displayed in Fig. 3. Compared with Fig. 4, which shows the tracking errors generated by selecting the starting point ψ_0 as earlier, the improvement is obvious. Similar improvements also are found for the first and second joints.

Because of the preset tolerance δ (or minimum permitted step size) and the nonuniqueness of the design in the frequency domain, different tuning results might be expected with starts from different initial parameters. However, in the time domain, it was found that different initial estimates yield similar results; i.e., the same levels of tracking errors, peak errors, and static errors. As an example, a start from a different initial condition ψ_0 yields the following values for the tuned parameters:

$$K = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix} = \begin{bmatrix} 630.0 \\ 770.0 \\ 682.5 \end{bmatrix}$$

$$\beta_1 = \begin{bmatrix} \beta_{11} \\ \beta_{12} \\ \beta_{13} \end{bmatrix} = \begin{bmatrix} 24.0 \\ 16.0 \\ 20.0 \end{bmatrix}$$

$$D = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = \begin{bmatrix} 183.75 \\ 120.0 \\ 120.0 \end{bmatrix}$$

The tracking errors for the third joint after autotuning for this case are similar to Fig. 3. The similarities are also true for the first and second joints.

To be more realistic and efficient, instead of calling the subroutine at each time interval, the real robot can run its complete task path. For the real implementation tests, all five parameters are tuned for six joints of PUMA 560. We use MicroVax II to run AutoTune and to control a PUMA 560 robot arm but bypass the VAL controller. A finished Vaxeln Pascal system runs on the target computer by itself, without VAX/VMS or any other operating system present. Vaxeln is a software product for the development of dedicated, real-time systems for VAX processors [8]. It is a fast way to design and implement time-critical applications on micros. Both multitasking and multiprogramming are supported by the Vaxeln Kernel and Vaxeln Pascal programming languages. In the process of developing and building the system for controlling the arm, the Vaxeln Debugger is used, through Ethernet from a VAX 780 computer, to debug the programs in a developed, executing system. The remote debugger can display the states of all processes and jobs in the local area network, and can dynamically change the user's "view" from one node process to another. Finished Vaxeln systems were loaded downline from VAX 780 into the target computer MicroVax II.

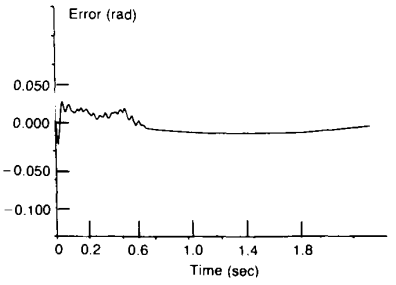


Fig. 3. Tracking error of third joint after autotuning.

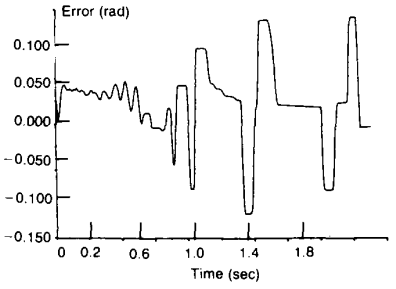


Fig. 4. Tracking error of third joint before autotuning.

In the standard Unimation system, the VAL software interprets the operating instructions for a robot arm, and the controller transmits these instructions from the computer memory to the arm. From incremental encoders and potentiometers in the robot arm, the controller/computer receives data about arm position. This provides closed-loop control of arm motions. The VAL controller (LSI-11 computer) is interfaced to the joint interface board through a parallel input/output (I/O) board. The joint interface board then relays joint-related information received from the VAL controller to the individual joint microprocessor boards [9]. For implementation, the connection between the joint interface board and the VAL controller was severed by removing the parallel I/O board connection from the VAL controller board. The parallel I/O connection was then made to a MicroVax II Q-bus. Thus, the MicroVax II is in direct control of the robot arm. Communication with the joint processors now can be implemented by sending the joint microprocessors commands to perform various functions.

The joint positions can be measured by optical encoders. Thus, the tracking errors and the objective function S can be calculated from Eq. (2). Each time after the arm runs a complete task, the pattern search is conducted and a new set of parameters is chosen and passed to the lag-lead compensator [Eq. (1)] for the next run. In each sampling period during the real robot run, the AutoTune package requires only 41 floating-point multiplications and 34 additions for calculating Eq. (2). The computational part of the process would take less than 0.2 msec on a MicroVax II. Compared with other portions of the control algorithm executed during a designed 5-msec sampling period, this computational load is acceptable. However, in order to carry out the pattern search after having evaluated the objective function S, much more computational work is required than just evaluating the objective function S in Eq. (2) at each sampling period. Fortunately, this analysis and learning process is *not* accomplished in each sample period during a run but between two consecutive runs. In this case, the computational load will not overburden the robot.

In real tests, the real robot runs usually take less time than simulations. AutoTune takes about 1.5 hr to tune parameters for three joints. It takes a few more hours to tune parameters for all six joints. To be more efficient, by doing what is suggested in item 3 of the next section, the tuning time can be reduced significantly.

For our tests, a maximum load (5 lb) is carried. To begin, a B-spline curve is used for tracking with a slow speed (4 in./sec). The weighting factors are determined as outlined in the next section. The real implementation results are encouraging. Herein, the tracking error is shown only for the sixth joint. Figures 5 and 7 show their tracking errors before autotuning and after autotuning, respectively. The improvement of the arm performance by using AutoTune is significant. To show the tuning process more clearly, a few more tracking errors of the sixth joint during the autotuning process are shown. The initial errors are presented in Fig. 5; after 10 repeated real arm runs by AutoTune, the tracking errors are shown in Fig. 6; and after some 50 runs, the final result is shown in Fig. 7. We can see the tuning effects and the gradual improvements by

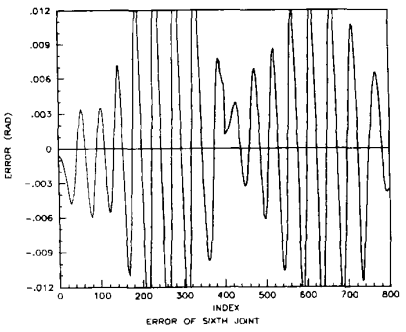


Fig. 5. Tracking error of sixth joint before autotuning.

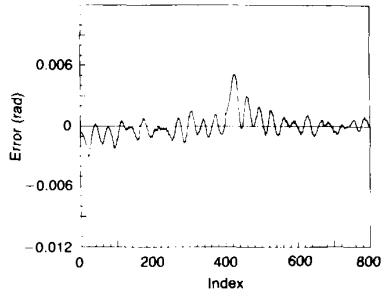


Fig. 6. Tracking error of sixth joint after 10 repeated real arm runs.

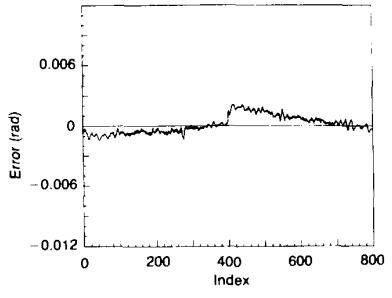


Fig. 7. Tracking error of sixth joint after autotuning.

AutoTune from these figures. Other joints gave similar tuning results. Different paths and speeds were used for tracking. The tuning process and improvements are similar. For noncyclic applications, the tuning parameters with higher speeds and longer paths (thus including different joint configurations) should be chosen for general tasks.

Conclusion and Discussions

(1) The AutoTune can automatically tune parameters of feedback compensators for robot arms and circumvent time-consuming and labor-intensive manual tuning while maximizing the capability of robot arms. Therefore, compared with manual fine-tuning, AutoTune makes it possible to accomplish the following: (a) fine-tuning taking account of coupling effects between multiple joints; (b) fine-tuning for each individual robot; and (c) retuning as the condition of a robot is changed.

(2) One practical issue is how to choose the weighting factors. The weighting factors R_1, R_2, R_3, R_4, and R_5 are determined as follows. Define:

$$\text{Sum1} = \sum_k R_1 \left[\sum_{i=1}^{n} e_i^2(k) \right]$$

$$\text{Sum2} = \sum_k R_2 \sum_{i=1}^{n} [e_i(k) - e_i(k-1)]^2$$

$$\text{Sum3} = \sum_k R_3 \sum_{i=1}^{n} [e_i(k) - 2e_i(k-1) + e_i(k-2)]^2$$

$$\text{Sum4} = R_4 \sum_{i=1}^{n} (EF_i)^2$$

$$\text{Sum5} = R_5 \sum_{i=1}^{n} (EP_i)^2$$

These terms can be calculated easily and monitored by using either simulations or actual robot runs. We may selectively weight different sums in the objective function (e.g., if we care more about oscillation, we may increase R_2 and R_3). From the experiments, the author feels that the portions of Sum2 and Sum3 are more important, since they are related closely to overall system stability. Thus, in this case, we chose R_1, R_2, R_3, R_4, and R_5 such that Sum1 : Sum2 : Sum3 : Sum4 : Sum5 = 1 : 3 : 3 : 1 : 1. From the experiments, determination of the weighting factors R_1, R_2, R_3, R_4, and R_5 can be an easy task in practice.

(3) To be more efficient and safe, it is suggested to first tune the parameters for each joint separately with a short running path and a slow speed using AutoTune. Starting with these roughly tuned parameters, three and then six joints can be tuned simultaneously. Finally, with the required working speed, the task path is used for applications with cyclic tasks, or a longer running path (for different arm configurations) is used for applications with general tasks to tune it again.

(4) To ensure that the control parameters remain in the range where the system is stable as an exploratory search is conducted, it would be better to use a small step size Δ to begin the tuning process. Later the parameters are tuned and the system has a larger stability margin than at the beginning of tuning. A larger step size Δ then can be used to speed up the tuning process.

(5) To reflect the vibration level of the arm correctly, it is suggested that the accelerometer be used to measure the acceleration and to calculate the jerk for the objective function S instead of the approximation of the acceleration used in Eq. (2).

(6) The pattern search that is used as a tool in AutoTune may sometimes get stuck; that is, it may stop short of a local minimum and be unable to make further improvements. To be fair to this method, however, remember that some gradient methods may also experience this difficulty [10]. In addition, such a problem has not yet been experienced in our implementation.

(7) Since no useful sufficient conditions for convergence of the pattern search have yet been derived, a condition where no further progress can be made does not always indicate that the solution has been found. In our case, the tuned parameters of the feedback compensator by AutoTune can be evaluated either by simulations or by real robot runs. Therefore, this method can be expected to work reasonably well in practice for robots.

(8) This package can be used also for tuning the parameters of proportional plus derivative (or proportional plus integral plus derivative) controllers, which are widely used in today's robotics industry. With appropriately defined objective functions for different applications, this package can be used for other automation purposes as well.

References

[1] Y. L. Chen, "Frequency Response of Discrete-Time Robot System—Limitations of PD Controllers and Improvements by Lag-Lead Compensation," *Proc. 1987 IEEE Intl. Conf. on Robotics and Automation*, Raleigh, NC, 1987.

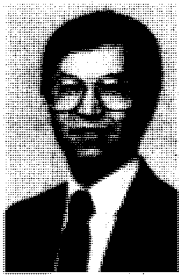
[2] Y. L. Chen, "A Formulation of Automatic Parameter Tuning for Robot Arms," *Proc. 26th IEEE Conf. on Decision and Control*, Los Angeles, CA, Dec. 1987.

[3] K. J. Astrom and B. Wittenmark, *Computer Controlled Systems*, Prentice-Hall, 1984.

[4] R. Hooke and T. A. Jeeves, "'Direct Search' Solution of Numerical and Statistical Problems," *J. ACM*, vol. 8, no. 2, pp. 212-229, 1961.

[5] A. F. Kaupe, Jr., "Direct Search," *Commun. ACM*, vol. 6, p. 313, June 1963.

[6] M. Aoki, *Introduction to Optimization Techniques*, Macmillan, New York, 1971.
[7] J. G. Ecker and M. Kupferschmid, *Introduction to Operations Research*, Wiley, 1988.
[8] Digital Equipment Corp., *VAXELN Reference Manual*, Maynard, MA, 1985.
[9] Unimation Robotics, *Unimate PUMA Robot 550/560 Series, Volume I—Equipment Manual 398H1A*, May 1982.
[10] M. Bell and M. C. Pike, "Remark on Algorithm 178," *Commun. ACM*, vol. 9, p. 684, Sept. 1966.



Yilong Chen was born in Fuzhou, China, in 1945. He graduated from the University of Science and Technology of China, Beijing, China, in 1968. From 1969 to 1973, he worked in the First Tractor Factory in Loyan, China. He taught in the Department of Electrical Engineering in the Wuhan Institute of Steel and Iron, Wuhan, China, from 1973 to 1978. He received the M.S. and D.Sc. degrees in systems and control from Washington University, St. Louis, Missouri, in 1981 and 1984, respectively. Dr. Chen joined General Motors Research Laboratories in 1984 as a Senior Research Scientist and is currently a Staff Research Scientist. His current research interests include robot control, exact linearization and control of nonlinear systems, and CAD/CAM. In 1987, Dr. Chen received a NASA award for creative development for his work on dynamic control of robot arms.

Workshop on Computer-Aided Control System Design

The IEEE Workshop on Computer-Aided Control System Design (CACSD) will be held Saturday, December 16, 1989, in Tampa, Florida. The workshop is organized by the Control Systems Society Technical Committee on CACSD and is sponsored by the Society. The workshop is a one-day event to be held at the Hyatt Regency Tampa Hotel in Tampa, Florida, immediately after the 1989 IEEE Conference on Decision and Control. The General Chairman of the Workshop is Douglas Birdwell of the University of Tennessee at Knoxville. The workshop provides a forum for the presentation and discussion of new directions in research, which involve significant linkages between systems and controls and relevant areas of computer science and computer engineering.

The workshop will contain presentations of both contributed and invited papers. Both applied and theoretical papers are solicited in the following and related areas: computer-aided modeling, analysis, and design environments; advanced modeling concepts; artificial intelligence, expert systems, and machine learning; and computer-assisted optimiation. All papers that are accepted for presentation will be published in a Proceedings, which will be available at the workshop.

Instructions to Authors: Prospective authors should submit seven copies of the full paper, headed with the paper title, names, affiliations, and complete mailing addresses of all authors and the statement "CACSD '89." The first-named author will be used for all correspondence unless otherwise stated. Submissions must be made by May 1, 1989, to:

CACSD '89
Professor J. D. Birdwell
Dept. of Electrical and Computer Engineering
The University of Tennessee
Knoxville, TN 37996-2100 USA
Phone: (615) 974-5468/3461
e-mail: birdwell@cascade.engr.utk.edu or birdwell@utkvx.bitnet

Authors will be notified regarding acceptance of their papers for presentation at the CACSD '89 by July 15, 1989. Authors of accepted papers will be provided publication kits and instructions for preparation of their manuscripts for the Proceedings. Manuscripts are restricted to eight Proceedings pages or less. Authors of accepted papers are expected to attend the workshop to present their work. The registration fee for one author is required when the manuscript is returned on conference mats, as a condition of publication.

1989 CDC

The 1989 IEEE Conference on Decision and Control (CDC) will be held December 13-15, 1989, at the Hyatt Regency Downtown Tampa Hotel, in Tampa, Florida. The hotel is within an easy drive of the beaches, which offer good weather at that time of year.

As usual, the conference will be conducted in cooperation with the Society for Industrial and Applied Mathematics and the Operations Research Society of America. The General Chairman of the conference is Leonard Shaw of the Polytechnic University of New York, and the Program Chairman is Tamar Basar of the University of Illinois at Urbana-Champaign. Contributed papers and invited sessions are solicited in all aspects of the theory and applications of systems involving decision, control, optimization, and adaptation. As a new feature, proposals are solicited for technical sessions on topics for which it is most effective to have the audience stand close to view displays of computer terminals, videotapes, diagrams, and/or photos.

For further information, contact the General Chairman:
Prof. Leonard Shaw
Polytechnic University of New York
333 Jay Street
Brooklyn, NY 11201
Phone: (718) 260-3802