SELF-TUNING OF ROBOT PROGRAM PRIMITIVES

David A. Simon Lee E. Weiss Arthur C. Sanderson*

The Robotics Institute - Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

A robot and its interaction with the task environment can be viewed as a process to be controlled. In this formulation, the robot program can be viewed as a task level controller. Robot programs, which are synthesized from sets of parameterized motion control primitives, embed control strategies to implement particular tasks. The development of robot programs, either by experienced programmers or using automatic code generation systems, is a difficult process which is often complicated by uncertain, incomplete, and varying models of the task environment. In this paper we address the strategy and parameter selection problems by describing an approach to self-tuning of robot program parameters. In this approach, the robot program incorporates control primitives with adjustable parameters and an associated cost function. A hybrid gradient-based and direct search algorithm uses experimentally measured performance data to adjust the parameters to seek optimal performance and track system variations. Alternative control strategies, which have first been optimized with the same cost function are then assessed in terms of their optimized behavior. We demonstrate that the optimal control strategy for a particular task is a function not only of task geometry, but also of the desired performance.

1 Introduction

The development of robot program code requires the transformation of abstract robotic task descriptions into desired robot motions. This transformation is a difficult problem which is often complicated by uncertain, incomplete and varying models of the task environment including the robot, manipulated objects, and sensors. The transformation includes the selection of a control strategy to implement each task. In conventional programming environments the strategy is encoded in the robot program as logically connected sets of motion control primitives. Each primitive has an associated parameter list. For example, a representative primitive **MOVES** ($\vec{X}$, V) tells the robot to move to position $\vec{X}$ with velocity V along a straight line trajectory. Primitives can also incorporate sensory feedback to accommodate uncertain and changing environments. Force sensing is often used in robotic assembly applications to accommodate contact motion constraints [13]. For example, a simple force monitored "guarded motion" primitive, **MOVES** ($\vec{X}$, V, $\vec{F}_{thresh}$) terminates the motion if force thresholds are exceeded. Primitives which explicitly specify dynamic sensory feedback strategies such as active stiffness control are also feasible [5].

While simple strategies may consist of a single program control command or motion primitive, more complex strategies are formed by logically sequencing multiple primitive commands. For example, consider the peg-in-hole mating task. If the peg and the hole are not chamfered and the tolerance between them is tight, then one possible multi-primitive strategy is as follows. First, tilt the peg slightly to increase the range of relative positions where initial entry in the hole is guaranteed. Second, move the peg along the axis of the hole with a force monitored motion. The monitor tests lateral and axial forces. If lateral forces are exceeded, realign the peg in the hole using a force feedback primitive to minimize lateral forces, then continue axial motion. If axial forces are exceeded then terminate the motion. In contrast, if the peg and hole are chamfered, and the end-effector has sufficient passive compliance then simpler single move primitive strategies may be appropriate. A current trend in product design for automated robotic assembly is to incorporate parts geometries which can be reliably mated with basic single primitive strategies, thus simplifying the programming requirements.

Many researchers have attempted to build planning systems which automatically synthesize robot motion control strategies and select motion primitive parameters. These problems have proven to be very difficult due to complex and incomplete models of the system [7]. Recent approaches have attempted to reason about uncertainties present in a world model, and to utilize sensor-based control to reduce this uncertainty whenever task failure would otherwise result [3,4]. Dufay and Latombe [2] discuss a system which performs inductive learning to generate robot program code for fine motion tasks from experience gained during a human guided training phase. This system synthesizes programs by adding sensory feedback strategies when existing code fails. Smithers and Malcolm [11] outline a research agenda which suggests a more systematic method for automatic program generation. They propose the identification of a set of basic robot *behaviors* which will act as building blocks for the construction of robot programs. In their approach, external sensory control would be incorporated into the behaviors in order to eliminate the need for sensory control at higher levels of the control hierarchy. The development of a complete set of such generic behaviors would greatly simplify the task planning problem. However, our research suggests that the identification of such generic strategies is complicated because strategy selection depends not only on task geometry, but also on the desired performance.

In practice, robot programming has been left to experienced programmers since automatic planning systems have had limited success in real applications. Robot programmers typically rely on intuition and trial-and-error experimentation to select a strategy and manually tune the program parameters. A good programmer will have abstract notions of performance optimization in mind as a basis for designing the program. For example, in assembly tasks, the programmer seeks a strategy and a set of parameter values which perform the task quickly, while attaining nominal mating forces, and minimizing the probability of mating failures. Humans, however, are not very efficient at searching parameter and symbolic spaces for optimal solutions. Manual searching is tedious and typically the resulting performance could be improved. The process of parameter tuning by a programmer, or automated parameter selection by a planning system, is further complicated by the fact that real robotic systems drift with time due to variations in the robot and environment, and thus require periodic parameter readjustment.

In this paper we address one aspect of the strategy and parameter selection problem by describing an approach to self-tuning of robot program parameters. In this approach, the robot program incorporates motion control primitives with adjustable bounded value parameters and an associated *cost function*. Search algorithms, which use experimental cost function evaluation, and which do not rely on explicit robot and environment models, adjust the parameters to seek optimal performance and to track system variations. Alternative control strategies, which have first been optimized *with the same cost function*, can then be assessed in terms of their optimized behavior. In this paper, we discuss this approach for *force monitored primitives* applied to *parts mating tasks*.

The remainder of this paper is organized as follows. In section 2, we outline the self-tuning formulation. In section 3, we discuss the implementation of our self-tuner and describe the underlying optimization algorithms. In section 4, we discuss force monitor tasks and present experimental results

*A.C. Sanderson is with the ECSE Dept., Rensselaer Polytechnic Inst., Troy, NY 12180

CH2876-1/90/0000/0708$01.00 © 1990 IEEE

which demonstrate the operation of the self-tuner for these tasks. In section 5, we describe an important class of assembly tasks commonly referred to as "snap-fits." Because these tasks exhibit several complex characteristics such as drift and stochastic variation in task performance, they provide an interesting case study for the application of the self-tuning approach to a realistic and practical problem. We show that the relative success of a given strategy is a function of the task to which it is applied, and of the desired behavior as specified by a task-specific cost function. In section 6, we propose a method for updating motion primitive parameters in response to performance drifts. Finally in section 7, we summarize the findings of our research and suggest areas for future work.

2 Self-Tuning Formulation

The self-tuning approach has been widely studied at the servo level to optimize control system tracking performance [12]. A few researchers, however, have explored self-tuning approaches at the program or strategy level. These approaches, including the system described in this paper, seek to optimize motion primitive parameters at the program level. This formulation may facilitate both manual and automated programming techniques which currently use such primitives as their fundamental building blocks. Simons et al. [10] demonstrate the application of stochastic automata to tune the positioning parameters of a quasi-static force feedback strategy. While the formulation of their approach is similar to ours, stochastic automata are best suited to parameter spaces which have a small number of discrete values. Whitney [14] applies Kalman filter theory to adjust the position parameter values for fine-motion control, and suggests varying the velocity as a function of confidence in the position estimates. This approach however requires explicit position estimation and does not generalize for tuning other parameter values.

In the self-tuning approach described here and illustrated in Figure 1, we view the robot/task environment as a physical process to be controlled. The underlying robot dynamics may be described by:

$$\dot{\vec{x}} = h(\vec{x}, \vec{m}) \tag{1}$$

where $\vec{x}$ is the state variable vector and $\vec{m}$ is the vector of kinematic model parameters. Closed-loop control is achieved by a set of position feedback control parameters, $\vec{c}$:

$$\dot{\vec{x}} = g(\vec{x}, \vec{m}; \vec{c}) \tag{2}$$

which responds to some preplanned reference trajectory. Such a system description is sufficient for simple robot positioning tasks. However, for many tasks which involve interaction with the environment, the closed-loop dynamics of the robot/task environment involves other parameters:

$$\dot{\vec{x}} = f(\vec{x}, \vec{m}; \vec{c}, \vec{\theta}_1, \vec{\theta}_2, \vec{\sigma}, \vec{\tau}, \vec{\rho}) \tag{3}$$

where $\vec{\theta}_1$ are geometric constraint parameters, $\vec{\theta}_2$ are force constraint parameters, $\vec{\sigma}$ are sensor sampling parameters, $\vec{\tau}$ are computational delay parameters, $\vec{\rho}$ are task-level control parameters, and the reference inputs, $\vec{R}_{ref}$, are expressed as position and force trajectories. For example, in the motion primitive **MOVES** ($\vec{X}$, V, $\vec{F}_{thresh}$) , V and $\vec{F}_{thresh}$ are the task-level control parameters, $\vec{\rho}$. Execution of the robot program requires specification of $\vec{\rho}$, while resulting performance of the system also depends on $\vec{\rho}$.

Design of the control strategy for these cases is difficult due to the uncertainty of the task parameters ($\vec{\theta}_1, \vec{\theta}_2, \vec{\sigma}, \vec{\tau}$), and the complexity in accurately modeling them. This uncertainty occurs due to trial-to-trial variations and slowly varying changes with time. As a result, performance of the system for a given task may vary between trials for a given implementation. For a given task-specific cost function, $J(\vec{x}; \vec{\rho})$, we would like to achieve

$$\min_{\vec{\rho}} J(\vec{x}; \vec{\rho}) \tag{4}$$

for best average performance and tracking of performance over time. For example, for a parts mating task the cost function may have the form

$$J(\vec{x}; \vec{\rho}) = E\left[\frac{k_1 \Delta T + k_2 (F_{ref} - F_{ss})^2 + k_3 [max((F_{peak} - F_{ref}), 0)]^2}{1 - P_e}\right] \tag{5}$$

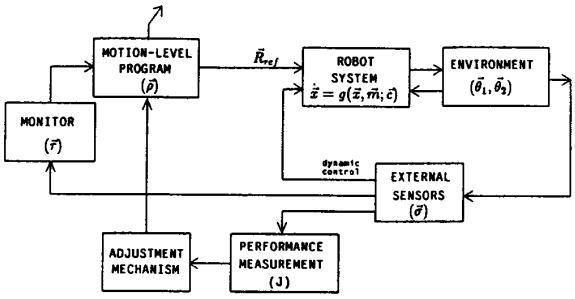

Figure 1: Self-Tuning Approach

where ΔT is task cycle time, F_{ref}, F_{ss}, and F_{peak} are the *desired* steady-state force, the *measured* steady-state force, and the peak overshoot force respectively, P_e is the probability of a mating failure, and k_1, k_2, k_3 are constants which weight the relative importance of each performance component. This process control description of the system suggests the use of optimization techniques to tune the program parameters. In this paper we will describe an implementation of a system which combines gradient-based and direct search techniques to accomplish this. While the robot program code used in our experiments has been manually derived, these optimization techniques could be used to tune the parameter sets of automatically generated code.

3 Search Algorithms

The design goal for a self-tuning system is to achieve stable response with fast convergence properties. The attainment of these goals is especially difficult if the performance space is characterized by multi-modal behavior. As will be shown in the next section, the performance space of monitor-based control primitives is often characterized by a complex multi-modal function which causes simple gradient-based optimization methods to fail. In our approach we use a hybrid gradient-based and direct search algorithm to achieve stable response. As suggested above, the cost function is evaluated experimentally by the robot at various points in parameter space, which can be a time consuming process. Thus, search techniques which do not require a large number of cost function evaluations are preferred for this application.

Driven by the above requirements, the three-stage self-tuning process depicted in Figure 2 has been designed. The goal of the first stage is to obtain a rough estimate of the optimal parameter set using a relatively small number of experimental trials. In the second stage, a fine resolution tuner attempts to find a parameter set which globally minimizes the expected value of performance. Once an optimal operating point has been found, an operational/tracking stage is invoked. While the first two stages are useful during the setup and testing of a robot system, the operational/tracking stage would be used during normal operation of the system. In this stage, shifts in the optimal operating point can be caused by tool wear, part tolerance variations, and drifts within the robot. Therefore, task performance should be monitored to ensure that operation remains within pre-determined tolerance bounds. If these bounds are exceeded, the performance can often be improved by re-adjusting the task parameters while the robot is in operation. In order to perform this tracking, we propose a technique which is based upon the fine tuning search, but over a very small window in parameter space.

In the first stage, least squares regression analysis is used to fit experimentally collected performance data to a simple analytic model of the performance surface. This model, which is described in the next section, does not incorporate the complex multi-modal component of the actual surface, but only the underlying "low-frequency" trends. The minimum of the resulting analytic model is then found using a "quasi-Newton" optimization algorithm known as Successive Quadratic Programming (SQP) [1].

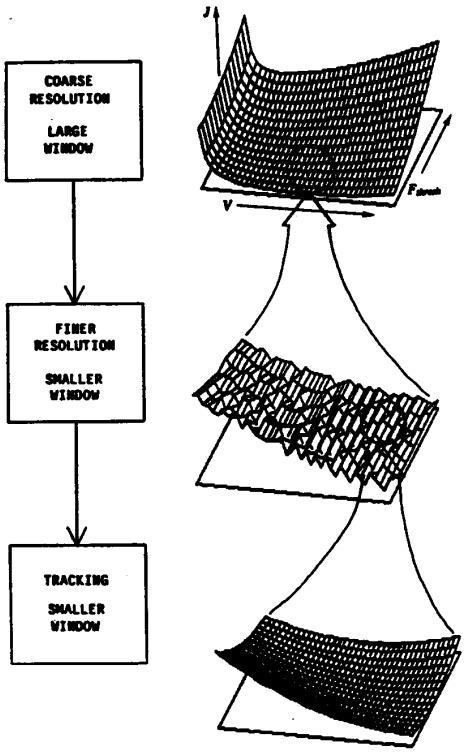

Figure 2: Overview of the Self-Tuner

As we will demonstrate shortly, in order to apply the self-tuner to certain types of monitor-based operations, it is necessary to account for the possibility of a "task failure." Failures can occur when a motion primitive does not complete the specified task because of improper adjustment of the motion primitive parameters. In the *fine* resolution tuner, task failure has been incorporated by the addition of a term to the cost function which degrades the calculated performance when failures occur. In the *coarse* resolution tuner, however, this technique would add discontinuities to the performance surface and thus complicate the least-squares model fit. Therefore, we have developed a scheme which uses experimentally determined "success regions" to constrain the search for an optimal parameter set. During data collection, a binary success/failure flag is recorded along with the corresponding performance value. Least-squares regression is then performed only over those performance data points which result in successful completion of the task. The accumulated success/failure information is then used by an *Augmented* SQP algorithm to find the minimum of the least-squares surface model within success regions, or on a region boundary [9].

The second stage of the self-tuner uses the minimum found in the first stage as the starting point for a "fine-tuning" high resolution search. Various approaches have been evaluated, each of which uses a small window in parameter space centered around the current minimum estimate. In this region, we have found that the amplitude of the multi-modal periodic component is often relatively small. Unfortunately, efficient approaches such as Hooke-Geeves [6] pattern search and the aforementioned gradient-based algorithm resulted in unpredictable and often unstable behavior. Such behavior is inevitable when a complex multi-modality is not explicitly accounted for in the model, even in a region where its amplitude is relatively small. Thus, to keep our approach simple we used a less efficient non-patterned direct search which samples points within a window around the current minimum estimate. Using the minimum estimate found in the coarse tuning stage as the starting point for its search, the fine tuner collects experimental data within a small window, and then selects a new minimum from the collected data. This process iterates until the change in performance from one iteration to the next is below a specified threshold.

For tasks which do not exhibit task failures or stochastic variation in performance, the above approach is sufficient. When stochastic variation and failures characterize a task, these characteristics must be explicitly accounted for in the fine-tuner. To account for this variation, we average each measurement at a given parameter set over several experimental samples. From the averaged data, a failure probability can be calculated and used in the cost function to penalize operating points which lead to task failures.

4 Force Monitor Tasks

In order to study the self-tuning approach on practical robot tasks, an experimental test-bed has been set up. It consists of an IBM 7565 robot with the AML programming/control environment, a robot gripper with adjustable compliance along the tool Z axis, and a force sensor capable of measuring forces along the tool Z axis. Several "tools" have been used in the experiments, each of which can be firmly grasped by the robot's gripper. In this paper, we shall refer to the robot's tool-tip as the portion of a tool which comes into contact with the environment. The search algorithms were implemented on a Sun-2 workstation, while an IBM-PC was used to implement several complex monitoring strategies which were not available in AML.

The use of an AML force monitor primitive is illustrated by the task of affixing two flat surfaces with an adhesive. More specifically, the task is to bring the parts into contact as quickly as possible, while achieving a final steady-state force of F_{ref}. To implement this task, the robot's tool-tip (one of the surfaces) was programmed to contact the other surface using the force monitored motion primitive command, **MOVE** $(\vec{X}, V, F_{thresh})$. This primitive instructs the robot joint level controllers to move to position $\vec{X}$ with velocity V, but stop if the measured force in the direction of motion exceeds F_{thresh}. The actual force achieved is a function of the task-level control parameters $\vec{p} = (V, F_{thresh})$, and the sensor monitor sampling period. For this task, the cost function is:

$$J = k_1 \Delta T + k_2 (F_{ref} - F_{ss})^2 \qquad (6)$$

Note that J only contains cycle-time and steady-state force error components since neither overshoot nor mating failures were present. Also, performance at any point was highly repeatable for this simple experiment, thus averaged performance data were not required.

The goal of the self-tuning algorithm is to vary V and F_{thresh} to minimize J. A plot of experimentally measured J vs. V and F_{thresh} is shown in Figure 3. In this figure, the coefficients k_1 and k_2 have been adjusted so that maximum values of the cycle time and the steady-state force error components are approximately equal. While this particular coefficient setting is arbitrary, in general, specified limits to cycle times and forces should be considered. For example, if the cycle time at the optimal operating point exceeds constraints specified in assembly queuing requirements, then the value of k_1 should be increased. Conversely, if the parts being assembled are extremely fragile, then the relative weight of k_2 should increased. Development of systematic approaches for weighting cost function components remains an open issue.

The multi-modal surface characteristic is clearly seen in Figure 3. This characteristic results from the finite sampling interval (20 msec) at which the monitor trip conditions are tested. Delays between the physical satisfaction of a trip condition and acknowledgement of this condition within the AML system result in complex, non-linear periodic behavior in performance space. We have developed an analytic model of this behavior to verify our experimentally measured results [9]. This multi-modal behavior is typical of robots using sensory monitored motions and is thus an important case study. For example, similar observations have been made on a PUMA 560 robot running the VAL-II control system.

To optimize the force monitoring task, the self-tuning algorithms of section 3 are applied. The coarse resolution tuner fits the experimentally collected performance data shown in Figure 3 to the function S given by:

$$S = \vec{\beta} \cdot [1,\ V,\ F_{thresh},\ VF_{thresh},\ V^2,\ F_{thresh}^2,\ 1/V,\ F_{thresh}/V]^T \qquad (7)$$

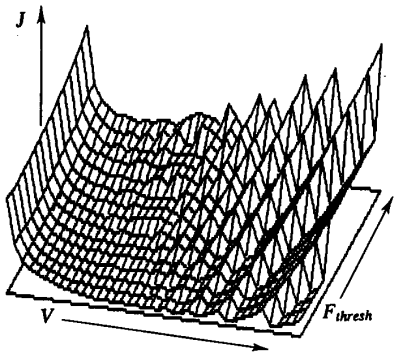

Figure 3: Adhesive Mating Task: Performance vs. V and F_{thresh}

The form of the surface model, S, is based on an understanding of the underlying physical processes of the task [9]. No *explicit* models of the robot were used to derive it. The first six terms of S form a quadratic in V and F_{thresh}, and are used to model the steady state force error component of the surface. The remaining two terms are inversely dependent on V, and were added specifically to model the cycle time component of the cost function.

Figure 4 is a plot of optimum performance value versus the number of experimentally collected data points. Performance is plotted at the completion of each self-tuning iteration. Roughly 175 data points were collected to generate the least-squares surface model. The minimum of this surface was found yielding a performance value of about 1.35. The first iteration of the fine tuner used 100 data points and yielded a performance value of 0.10. Future iterations of the fine tuner did not result in improved performance. An independent high resolution direct search of the entire parameter space has verified that the resulting minimum is the "true" minimum within the resolution of the tuner. Similar results have been achieved with the self tuner using a variety of cost function gains, reference forces and tool compliances. Typically, the minimum is found within one or two iterations of the fine-tuner which suggests the appropriateness of the simple model fit used in the coarse tuner.

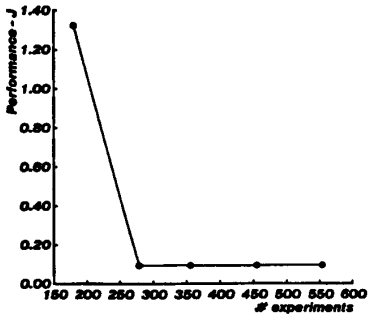

Figure 4: Performance vs. # of Experiments

5 A Snap-Fit Assembly Task Example

We have studied the self-tuning approach on a variety of *snap-fit* tasks [15]. Reliable snap-fit operations are a major requirement for mating of parts which have been designed for automated assembly. Because of the relative simplicity of performing snap-fit operations, product designers utilize snap-fits frequently in their designs. Assembly robots often incorporate single-primitive motion strategies to perform snap-fit operations, while more complex multiple-primitive strategies may be required to perform conventional fastening operations. In the previous section, the adhesive mating task was useful for demonstrating the basic operation of the self-tuner because of the task's well behaved characteristics. Interaction between the robot and the environment was very simple, resulting in highly controlled, deterministic behavior of the fundamental force monitored strategy. Conversely, the more complex snap-fit tasks have been implemented with alternative strategies each of which exhibits significant stochastic variation in performance. In addition, long term variation in performance has been observed for snap-fit tasks due to wear in parts and fixtures, and drifts within the robot. Because of these characteristics and the importance of this operation, the behavior of the self-tuner on snap-fit tasks provides an interesting and important case study.

As we have suggested, there are often a number of alternative feasible strategies for performing a task. Typically, a robot programmer will select a strategy based upon ease of implementation, successful use of the strategy in the past, and often upon intuition. Without complete models and a methodology for comparing competing strategies for a given task, it is difficult to ensure that the selected strategy is the best one for the job. In our approach, the strategy selection process for a given task is based on comparing the performance of strategies which have first been optimized *with the same cost function*. The optimized cost function provides a unifying metric by which alternative strategies can be compared. We have also explored whether there is a single generic strategy, amongst the alternative strategies, which is optimum over a broad range of "designed for assembly" snap-fit operations. The existence of a generic strategy would facilitate the design of automatic robot code generation systems. The designed for assembly constraint is specified because it is well known that for more general assembly and manipulation tasks the concept of a generic strategy is not feasible. Even small variations in parts geometry will change the appropriate strategy [8]. Our research demonstrates, however, that even for fixed geometric constraints, the optimum strategy is a function of the performance requirements. This suggests that the specification of generic strategies would need to incorporate performance requirements, even for designed for assembly operations.

We have implemented three snap-fit tasks in our study of the self-tuning approach. The three tasks will be referred to as the "snap-ball", "snap-shaft", and "phono-plug" tasks. For each of these tasks, a special tool-tip and matching receptacle have been used. The goal of each of the tasks is to insert a ball, shaft, or phono-plug into the corresponding receptacle as fast as possible such that the actual steady-state force, F_{ss}, is equal to a reference steady-state force, F_{ref}, while minimizing the overshoot force, F_{peak}, and minimizing the probability of mating failures, P_e.

One characteristic which is common to all snap-fit operations is a sharp decrease in mechanical resistance to an applied force shortly before the operation is complete. This is clearly illustrated in Figure 5 which is a plot of insertion force vs. end-effector position for the phono plug task. In the discussion which follows, we will refer to the maximum force which occurs before insertion as the maximum pre-insertion force (MPIF).

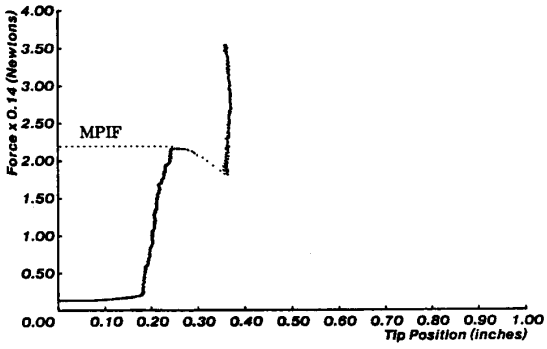

Figure 5: Force vs. Position Signature - Phono Plug

To illustrate the strategy selection problem, three alternative guarded motion strategies are described. The first strategy uses the aforementioned absolute force monitor motion primitive, **MOVE** $(\vec{X}, V, F_{thresh})$, with ad-

justable parameters V and F_{thresh}. The second strategy, referred to as the *rate of change* strategy, uses the primitive, **MOVE** $(\vec{X}, V, df/dt_{thresh})$, with adjustable parameters V and df/dt_{thresh}, where df/dt_{thresh} is a threshold on the rate of change of the insertion force. In this strategy, df/dt_{thresh} is constrained to negative values so that the monitor trips during the decrease in force after the MPIF. In most situations, this strategy has a much smaller probability of insertion failure than the absolute force strategy. There are cases, however, when the rate of change strategy can be fooled by a "false" MPIF, resulting in insertion failure. The third strategy, referred to as the *absolute position* strategy, uses the primitive **MOVE** $(\vec{X}, V, P_{thresh})$ with adjustable parameters V and P_{thresh}, where P_{thresh} is a threshold on the position of the tool-tip relative to the receptacle. Due to the compliance of the force sensing unit, the distance between the tool-tip and the robot gripper is not constant, but varies as a function of applied force.

The results of applying the strategy selection technique to the snap-shaft task using a "fragile parts" cost function are illustrated in Figure 6. For this type of cost function, the optimal operating point will favor small steady-state and overshoot forces rather than fast cycle times. Each of the curves in the plot was generated by self-tuning the snap-shaft task using one of the three strategies described above. From the plot, it is evident that the rate of change strategy has the smallest optimized performance value among the competing strategies. This result can be explained as follows. Due to frictional and geometric characteristics of the snap-shaft task, the MPIF increases significantly with increasing insertion velocity. It follows, therefore, that in order to minimize peak forces, the insertion velocity should be small. At low velocities, however, both the absolute force and position strategies often result in insertion failures due to the stochastic nature of the frictional interaction between components. The rate of change strategy, on the other hand, operates well at small velocities, since the monitor does not trip until after the MPIF has been reached. A more detailed analysis can be found in [9].

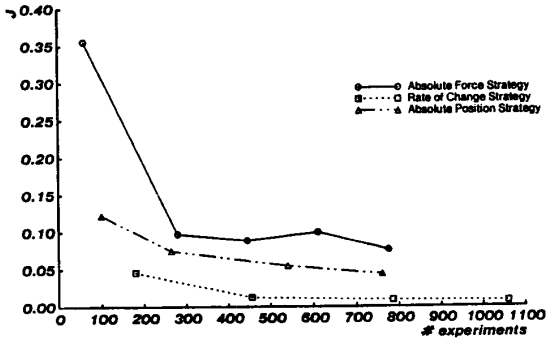

Figure 6: Snap-Shaft Task Performance Curves
k_1=1.0, k_2=0.0, k_3=0.25, F_{ref}=0.3

With sufficient insight into the physics behind the previous example, it may have been possible to predict the optimum strategy without use of this strategy selection approach. However, a detailed physical understanding of a task or strategy can often be very difficult to establish. Often, it may be impossible to derive a model which is accurate enough to determine the optimum strategy. The strategy selection approach discussed, however, allows an engineer to select an optimal strategy without developing a detailed understanding of task and strategy.

In the interest of identifying generic task strategies it would be convenient if the rate of change strategy was optimal for all snap-fit tasks over a range of alternative cost functions. Unfortunately, as we now demonstrate, this is not the case. Using the strategy selection approach, we show that the optimal strategy is dependent upon subtle differences between tasks, which can be difficult to model. Figure 7, shows the results of applying the strategy selection technique to the snap-*ball* task using the fragile parts cost function. In this case, both the absolute force and absolute position strategies are superior to the rate of change strategy. This change in behavior can be traced to subtle differences in geometric and frictional characteristics between the snap-ball and the snap-shaft tasks. For the snap-ball task, the value of the MPIF is not very dependent on insertion velocity. Large insertion velocities do not necessarily lead to large peak forces, and thus to decreased performance. In addition, at high velocities the rate of change strategy exhibits significant stochastic variation in steady-state force error. Together, these two factors result in reduced relative performance for the rate of change strategy.

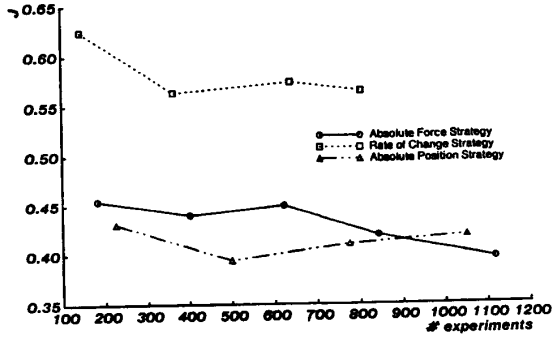

Figure 7: Snap-Ball Task Performance Curves
k_1=1.0, k_2=0.0, k_3=0.25, F_{ref}=0.3

We have demonstrated that the best strategy for a task can vary as a function of even subtle differences in tasks. A more interesting observation, however, is that the best strategy varies as a function of the desired performance. Each of these observations have important implications for the design of practical automatic robot code generation systems. One problem which must be solved before such systems can be built is how to select an appropriate strategy for a given task. This problem would be simplified if generic strategies capable of performing a wide range of tasks could be identified. For example, a single strategy capable of performing all snap-fit tasks would eliminate the need to determine an appropriate strategy each time a new snap-fit task is to be performed. As we have already demonstrated, generic strategies are hard to find since the success of a strategy is a function of the task upon which it operates. In addition, we will now show that the success of a strategy applied to a given task depends on the desired behavior as specified by the cost function. This suggests that truly generic strategies must work well not only for a variety of tasks, but also over a range of cost function coefficients.

In the previous examples, we assumed that the parts being assembled were fragile, and that task cycle time was not critical. We now consider another example using the snap-shaft task for which the parts to be mated can tolerate large forces, and emphasis is shifted to minimizing the task cycle time. The plot in Figure 8 illustrates the results of this strategy selection example. Recall that for the snap-shaft task using the fragile-parts cost function, the rate of change strategy was optimal. In the present case, however, the rate of change strategy has inferior performance relative to the other strategies. To understand this result recall that the cost function has shifted emphasis to the task cycle time. Thus, large velocities are preferred over smaller ones. As we noted earlier, the rate of change strategy always trips *after* the MPIF. The absolute force and absolute position strategies, however, will trip before the MPIF when the desired force, F_{ref}, is less than the MPIF. It follows that the absolute force and position strategies can operate at larger velocities and still decelerate before a desired steady-state force is reached. Since the rate of change strategy must operate at smaller velocities, the resulting longer task cycle times will degrade performance.

6 Tracking of Snap Fit Tasks

Once a strategy has been selected for a given task and the motion primitive parameters have been optimized, the robot can be put into an operational mode during which it repeatedly executes a set of tasks. During normal operation, long-term drift in performance may occur due to such factors

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.