Automated Learning for Parameter Optimization of Robotic Assembly Tasks Utilizing Genetic Algorithms

Jeremy A. Marvel1, Wyatt S. Newman1, Dave P. Gravel2, George Zhang3, Jianjun Wang3, and Tom Fuhlbrigge3

1 Electrical Engineering and Computer Science Dept., Case Western Reserve University, Cleveland, OH
2 Ford Advanced Manufacturing Technology Development Center, Detroit, MI
3 Mechatronics and Robotic Automation, ABB US Corporate Research Center, Windsor, CT

Abstract - A challenge for automating mechanical assembly is that cumulative uncertainties typically exceed part clearances, which makes conventional position-based tactics unsuccessful. Force-based assembly strategies offer a potential solution, although such methods are still poorly understood and can be difficult to program. In this paper, we describe a force-based robotic assembly approach that uses fixed strategies with tunable parameters. A generic assembly strategy suitable for execution on an industrial robot is selected by the programmer. Parameters are then self-tuned empirically by the robot using a genetic-algorithm learning process that seeks to minimize assembly time subject to contact-force limits. Results are presented for two automotive part assembly examples using ABB robots with commercial force-control software, showing that the approach is highly effective and suitable for industrial use.

Index Terms – Robotic assembly, force control, genetic algorithms, parameter optimization

I. INTRODUCTION

Although originally seen to be the dominant application domain of robots, mechanical assembly of components in industry is still primarily accomplished using manual labor. Robotic assembly using force control has begun to change the way automation is used for industrial assembly processes. For example, within Ford Motor Company, force controlled robotic assembly of torque converters has been running successfully in production since 2001. This start of a revolution in assembly automation has been facilitated by the introduction of commercial products in robot force control.

Programming a robot to exploit force control for specific mechanical assembly applications often calls for a programming expert to tune the robot's program and optimize parameters linked with robot reactions to external forces. Industry has harsh demands on reliability, throughput and quality, and even a well-tuned program for robotic assembly can be insufficiently robust with respect to part variations. For example, dimensional or surface-finish variations occur due to batch differences from different vendors or variations within a batch due to tool wear during a production run. While these part variations remain within tolerance specifications, they are sufficient to degrade automated-assembly performance. A process that adapts to changing part conditions motivates the present work in machine learning, in which a robotic system monitors its own performance and modifies its own program to maintain quality and throughput.

Assembly parameter optimization has been researched extensively for decades with mixed results. Early approaches were capable of successfully completing peg-in-hole insertion tasks, but were impractically slow and could not catch up with modern manufacturing demands. Design Of Experiments (DOE) has been used to optimize assembly parameters at Ford AMTD [1]. DOE is faster than exhaustive parameter searches, and is driven toward a systemic and provably-optimized solution, but often requires a Six Sigma expert, a robot programmer and occasionally a robot operator to construct and implement. Efforts have been made to develop on-pendant software tools to simplify DOE-based parameter optimization processes [2].

Attempts in automatic assembly optimization have employed a wide gamut of learning techniques ranging from logic-branching [3] to neural networks [4], and have used a variety of exploratory methods including virtual assemblies [5] and teach-by-example training [6]. Research work at Case Western Research University (CWRU) has previously demonstrated that genetic algorithms using simple competition rules could successfully optimize the assembly process with performances superior to those demonstrated by extensive manual tuning [7]. Through these studies, it was proven that learning algorithms can be programmed for specific tasks. However, a majority of these approaches are highly specialized, and must be reworked whenever modifications are introduced into the assembly setup or the robot.

In this paper, we describe a generalized machine learning approach for optimizing the search strategies employed in autonomously assembling components in variable work configurations. Our approach for assembly optimization differs from previous work in two key aspects. First, we utilize standard industrial robotic arms, controllers and control software. And second, our learning system was developed as a black box interface, and is being applied directly to multiple robots without modification to a variety of assembly tasks.

II. OVERVIEW OF THE TASK SPACE

In the past, research in compliant-motion robotic assembly typically required either building a custom, force-controllable robot or gutting the controller of an industrial robot. With the introduction of force control within commercial robotic systems, as well as high-speed communications capabilities between the robot controller
and an external computer, it is now possible to incorporate innovative software within industrially-applicable systems. At the same time, the performance of the robot manufacturer’s force-control execution imposes constraints on development of add-on software for robotic assembly. In the present work, we are using force-control from ABB, Inc. Our assembly strategies and machine learning additions are performed on an external PC communicating with the native robot controller.

For our experiments, we are utilizing three separate commercial robotic arms in physically disjoint facilities. The first two are ABB IRB-140 robotic arms with single-cabinet ABB IRC5 controllers, and the third is an ABB IRB-4400 industrial robotic arm with a dual-cabinet ABB IRC5 controller. All three are in their stock configurations with the ABB Force Control option installed. The IRB-140s are fitted with ATI Gamma force sensors, while the IRB-4400 was given an ATI Omega force sensor.

Our system configuration consists of an external PC containing the learning application connected to the respective robot controllers via a 4ms fast Ethernet link. This connection is used to convey gene sequence parameters from the host PC to the controller, and to return feedback data from the controller back to the PC. The PC application consists of two parts: the implementation of the genetic algorithm, and a specialized communication server to which the robot controller connected (both of which are threaded for independent operation). The server’s purpose is limited to providing an interface between the genetic algorithm and the controller, and thus contains the only application-specific logic in the PC configuration. The external computer is therefore used explicitly for gene sequence mutation, parameter retention, and logging, while kinematic control and force-reaction of the robot is retained by the IRC5 controllers.

ABB’s Force Control software option for the RobotWare system software closely resembles the classic accommodation control methods (also referred to as damping control) [6] employed for decades on a wide variety of robotic manipulators. In this schema, the robot’s velocity is commanded by specifying reference forces. The difference between the commanded and measured force values, smoothed by a low pass filter, is fed back into the velocity control loop as a velocity reference value. By adjusting the reference velocity, various motions and search patterns can be defined and applied on the robots’ control.

For the experiments in this study, robot motions are reduced to a number of primitive behaviors represented as independent, atomic actions that can be chained together to achieve complex actions. The system in this study uses three basic search behavior types: spiral search, radial search, and linear move (see Fig. 1). Combinations of these three primitive types are sufficient for successfully completing a substantial variety of assembly tasks. For example, in the torque converter assembly discussed in the next section, the entire search strategy consists of two consecutive radial searches chained together. Each primitive behavior’s motion pattern is defined by reference motion commands built into the RobotWare RAPID programming instruction structure ([9]).

When initiated, the spiral search behavior moves the tool from its current location, and gradually progresses outward in a spiral pattern until either a restart or termination condition has been met. A restart condition is met when the tool motion reaches the end of its spiral arc, and results in the spiral search beginning again. Termination conditions—either timeouts or successfully meeting the needs for stage completion—halt the motion of the robot and trigger a request for the next gene in the sequence. Gene parameters for spiral search specify the search speed, radius, and the number of turns per spiral.

In a radial search, the tool rotates around the work object’s Z-axis, oscillating around its starting position, with a commanded rotational amplitude and frequency. Because the radial search was designed with the insertion of gears in mind, a number of additional actions are supported while the radial search is executing. To reduce the possibility of the gear teeth catching and locking on one another, an up-down “hopping” motion can be defined, with the action’s amplitude and frequency being configurable by the user. In the event of a misalignment, the speed and radius of a circular search (a simplified search motion in which the tool orbits around a fixed point in space) can also be adjusted. It should be noted that the above spiral, rotational and hopping search parameters, as well as the bias downward force, interact non-intuitively in terms of the reward function (performance metric), which complicates manual tuning of programs.

A linear move is the only behavior that can be completed as either a position-controlled or force-controlled motion. When executed, it results in the tool moving in a straight line from the robot’s current configuration until a termination condition has been met. Such conditions are defined by either physically contacting an object or reaching the commanded target coordinate. Only the position-controlled linear motion behavior has parameters for rotational offsets, and, for our initial tests, was employed exclusively for adding random noise to the initial positions of the tool for each gene sequence execution. Besides their success conditions, all force-controlled motions are bound by timeouts, force-application parameters, and a motion termination condition based on travel distance, insertion depth, or contact force.

III. AUTOMATED LEARNING

Genetic algorithms, a method of learning via stochastic exploration through a vectorized “gene” space by means of
random perturbations, have been utilized in a wide array of optimization tasks. The variances in the algorithm’s implementations are as varied as the applications. Our own prior work with a GA for assembly parameter optimization described in [7], [10] utilized a regionally-guided algorithm known as Guided Evolutionary Simulated Annealing (GESA, [11]). In this method, co-evolving clans of gene sequences coalesced simulated annealing and simulated evolution into a novel competitive model in which the clans contested within and between themselves. Each clan is assigned a single parent, $P_i$, from which all offspring originate. Within a clan, the progeny compete against one another and their parent for the next epoch’s parent status. Between clans, the entire populous competes on behalf of their “families” for stakes in the total population pool to decide the number of children each clan will be allocated for the next epoch.

The GESA algorithm is designed to ultimately favor the evolutionary lineage that demonstrates the most proficiency and strongest likelihood of producing optimal results. However, our preliminary experiments empirically demonstrated that different sets of search space vectors may result in highly competitive assembly performances, and that prematurely stunting the progeny pool of a given clan could inadvertently hinder its convergence toward viable solution headings. As such, in the current study a variation to the GESA algorithm was made that removes the competition between clans, and instead treats each as a separate entity that does not vie for population resources, but rather co-evolves at distinct rates to accommodate for the diverging genetic strains. Competition, as a result, is focused on intra-clan dynamics, with the parent gene actively competing against a population of child genes for subsequent generational dominance.

In our algorithm, the search and termination parameters of each search method are contained within a fixed-length gene segment. The gene parameter vector identifies the motion control interface (i.e. position or force control), search method (linear, radial or spiral search), and an array of search parameters, ordered and indexed accordingly for the movement method. Each gene hence describes the entirety of a single primitive search action. Genes can be chained together to describe complex actions and search strategies. This structure was chosen because it allows for an open-ended control scheme, and thus inherently possesses the capacity of autonomously compounding or reducing the complexity of the search vector by means congruent with genetic programming. This grouping of individual genes thus describes a gene sequence of size $|P|$. Despite the potentially large number of motion parameters, many values can be frozen in order to reduce each stage’s learned parameters from over a dozen to a mere handful.

Both the evolution and guidance toward convergence modeled into the clans of our implementation of GA are driven by adaptive mutation vectors unique to each clan. For each generation, the bounds of offspring genetic variance are based on the mutation function described in Equation 1:

$$G_{i,j} = P_i + g(m_i)$$

In this equation, the function $g$ generates a sequence of size $|P_i|$ of gene-lengthed vectors of random delta values with a Gaussian distribution around the progenitor gene sequence, $P_i$. These delta values are created with mutation variance $m_i$ to create the $j^{th}$ offspring of clan $i$, $G_{i,j}$. It should be noted that, given the mutation function shown in Equation 1, it is possible for invalid, dangerous, or otherwise impractical search parameters to be inserted into the gene sequences (for instance, a search radius cannot be negative). Because many parameters have specific bounds or requirements, in order to maintain our goal of GA generality, checks into parameter validity were hard-coded into the action layer in the robot controller’s RAPID code instead of being provided for on the external computer. Gene values that are to be fixed in place (for instance, the movement method and ID, as well as the timeout values) can be done so by means of setting the appropriate value in the mutation vector to 0.

To guide evolution, the values in the mutation variance vector are reduced on each successive generation based on the declaration function described in Equation 2:

$$m_i(t) = \eta m_i(t-1)$$

The learning rate, $\eta$, is a parameter that causes the mutation rate to decrease over time. It is a user-defined variable-value parameter that changes based on progeny succession. The learning rate has a bimodal characteristic such that we can delineate the value of $\eta$ as, $\eta \in \{\eta^+ , \eta^- \}$, $0 < \eta^+ < \eta^- \leq 1$, where $\eta^+$ indicates child succession as parent for the clan’s subsequent generation, and $\eta^-$ means that the parent of the previous generation retains its parent status. The inclusion into the algorithm of Equation 2 ensures that a given clan will eventually converge on some local optimum, even if no progress in optimization is observed. Larger values of $\eta^-$ allow for more exploration, but are likely to converge much slower in the absence of child succession, while smaller values will cause $m_i$ to narrow faster, which could increase the chance of convergence on sub-optimal search values.

Child succession is based on a scoring metric specific to the application domain. Children who score better than their parents succeed their parents. The rules of our GA implementation are outlined as follows:

1. Begin with a known parameter set that will result in a successful assembly. These values do not have to be near optimal, merely sufficient for completing the task. Set these values as the origin gene sequence $P_0$.
2. Copy the origin gene sequence into $C$ parent sequences, $P_1, ..., P_C$, for $C$ clans.
3. Execute the clan parent gene sequences, $P_1, ..., P_C$ to generate $C$ result records $R_1, ..., R_C$, that are then fed into the scoring function, $f$—which is specific to the problem domain—to produce the initial best score values, $\hat{s}_1, ..., \hat{s}_C$, where $f(R_{i}) = \hat{s}_i$.
4. For each clan $i \in \{1, ..., C\}$ generate $N$ child gene sequences $[G_{1,1}, ..., G_{1,N}], [G_{2,1}, ..., G_{2,N}], ..., [G_{C,1}, ..., G_{C,N}]$ using the mutation function described in Equation 1.
5. For each clan \( i \in \{1, \ldots, C\} \) and every child \( j \in \{1, \ldots, N\} \) execute the gene sequence \( G_{i,j} \) to produce the collection of scores \( S = \{s_{i,1}, \ldots, s_{i,N}\} \).

6. For each clan \( i \in \{1, \ldots, C\} \), find the best child, \( n \), such that \( \forall j \in \{1, \ldots, N\}, j \neq n, s_{i,n} \geq s_{i,j} \). If \( s_{i,n} > \hat{s}_i \) then \( P_i = G_{i,n} \).

7. Reduce the mutation vector variance utilizing Equation 2.

8. Repeat steps 3-7.

Given the variability of assembly task spaces, the importance of gene re-interpretation should not be ignored. As occurs in nature, certain variances in initial configurations may unintentionally lend preference to inferior genetic strains for a given generation. Those same lineages, when presented with slightly different initial conditions on subsequent trials, however, could result in far worse results. As such, for our trials, each gene execution for assembly was performed three times, and the average performance was presented for competition. Further, for each generation, the parent must be an active participant in the competition, and is not allowed merely to rely on prior generations’ performances.

The GESA algorithm allowed for the succession of superior parental genes by inferior offspring by means of a random throw of the dice. Because of the random variance in initial component configurations, this “survival of the luckiest” metric is inherently modeled into our system. While allowing for the possibility of performance regression, the system is allowed to be fully adaptive in the sense that succession is based only on the results of a given generation, and does not take into account the outcomes of past generations. The system, thus, does not reflect a global maximum, as a single lucky (and potentially unrepeatable) parent could inadvertently skew and thus hinder the performance and progression of the clan.

Note that the fitness function is not restricted for the GA’s implementation, and that any function can be inserted to satisfy any given task. By separating this function from the remainder of the algorithm, we ensure that the actual GA implementation can remain unmodified while only the performance function is redefined for new applications. For the experiments conducted in this study, the reward function \( f \) is defined by Equation 3.

\[
f(R_{i,j}) = \max \left( \alpha \left( \frac{T_{\text{max}} - T_{i,j}}{T_{\text{max}}} \right) + \beta \left( \frac{F_{\text{max}} - F_{i,j}}{F_{\text{max}}} \right), 0 \right)
\]  

(3)

For clan \( i \), the \( j^{\text{th}} \) child’s score, \( s_{i,j} \) is a function of the reported assembly time, \( T_{i,j} \), and average encountered force, \( F_{i,j} \). Both variables are determined by observing the data stream flowing from the robot via the Ethernet link. The values \( T_{\text{max}} \) and \( F_{\text{max}} \) represent the maximum allowed assembly times and maximum allowed average forces, respectively, for each assembly attempt. The user-defined coefficients, \( \alpha \) and \( \beta \), are used or stressing the preference of minimized time versus minimized force. For our tests, \( \alpha = 0.8 \) and \( \beta = 0.2 \).

IV. RESULTS

A. Sample Assembly Task Descriptions

To evaluate our system’s performance on search space optimization, we are running the GA solution on two district task configurations (see Fig. 2). The first, an automatic transmission valve body assembly for a mid-sized car, models the classic peg-in-hole insertion task. The assembly consists of a cylindrical insert being stuffed into a smooth, tubular body with an extremely small clearance. The second task assembles a medium-sized rear-drive pickup truck’s torque converter with sub-millimeter clearance. This assembly involves a combination of a cylindrical insertion followed by three radial searches (two shaft gear meshes and a final collar insertion).

B. Trial Results: Transmission Valve Body

The transmission valve body assembly epitomizes the manufacturing constraints we have imposed on our system. Its components, while structurally sound, could be damaged with extended exposure to contact forces in excess of about 40 N. Further, the ends of the cylindrical inserts that make contact with the assembly are accentuated with small, sharp-edged ridges that can result in wear on the part while the robot is searching for the insertion point. As such, while short assembly time is the ultimate goal, our GA solution is designed to be gentle.

The strategy for performing the assembly of the transmission valve body consists of a single spiral search with a constant downward pressure being applied. The task is compounded slightly by a small groove that runs around the circumference of the interior of the tube approximately 5 mm below the top lip. This groove acts similarly to a second insertion stage as the robot must again perform a spiral search in order to progress below it.

Initial efforts with search parameter optimization involved a directed manual tuning of applied force, spiral search speed, and spiral search radius. Manual tuning consisted of a semi-exhaustive binary search through the parameter space using a simplified gradient descent method, with the optimization being driven by empirical observations. Changes that were observed to result in better performances were given more attention, while changes that resulted in worse performance or no observable change in effectiveness were largely ignored. Results placed the assembly time at around 5 seconds, and after a week of
manually guided tuning, reduced the average assembly time to about 2.5 seconds.

The GA was run with four clans, each of which were assigned ten child slots and one parent. Each gene execution attempt consisted of three trials that were averaged together as a single metric profile. The system was allowed to run for twenty generations, for a total of 2,640 assembly attempts. The parameters evolved were the applied force, search spiral radius, search speed, and the number of turns per spiral. The evolutionary variance of two of these parameters (force and spiral radius) is shown in Fig. 3. Though the main assembly was fixtured in place, fixture variance was modeled by adding random offsets in the search starting position of the robot tool.

As can be seen in Fig. 4, assembly times ranged from 1.5 to 30 seconds, with the higher end of this range indicating an unsuccessful assembly attempt or timeout. The base parameters succeeded in performing the assembly in about 10 seconds, and completion of all 20 generations required about 8 hours of constant training. Two of the four clans yielded highly competitive results. One had assembly times consistent with those of the manually-tuned search parameters, with an average assembly time of 2.6±0.4 seconds, while the other performed dependably better than the manually tuned parameters with an average assembly time of 2.0±0.2 seconds. The average forces of all gene runs were within the stated tolerances. The evolutionary tracks of the last two clans produced inconsistent results and were more prone to failures. It is hypothesized that this divergent behavior is the direct result of minor wear on parts in conjunction with sub-optimal search parameters, and that along those given evolutionary paths the system effectively found a local minima, and could not explore the search parameters beyond the boundaries of the gradient slope.

C: Trial Results: Torque Converter

The torque converter assembly, while consisting of four mixed-strategy engagement stages, was reduced to two for our tests given two observations. First, given our previous results with the transmission valve body assembly, we were confident we could omit the initial spiral search and begin with a best guess for an initial starting position. Even with the random noise added to the robot starting position, the circular motion in the radial search pattern in conjunction with the gear meshing automatically corrects the assembly position over the center of the shaft. Second, the search strategies for the two shaft gear meshes are identical, and can thus be merged into a single search step without incurring any time penalties.

Like the transmission valve body assembly, the GA optimization for the torque converter assembly was run with four clans, each assigned ten children running three trials per generation. The two stages were trained separately to improve the likelihood of child succession. Stage 1 was trained for twenty generations, with the system optimizing the applied downward force, and hopping amplitude and frequency. After stage 1 was suitably optimized, the best parameters were locked and the system then optimized stage 2. Optimization on stage 2 was allowed to run for fifteen generations, with the applied force, radial search speed, hopping amplitude and frequency being learned. Training for both stages required about seven hours to complete (about two hours for stage 1, and five for stage 2). The timing performance of both training sessions is shown in Fig. 5.
All four clans were highly competitive for stage 1 training, with average completion times all just under 2.0 seconds. Only clan 3 produced a viable solution for stage 2 with an average assembly time of less than 10 seconds (see Fig. 6), while the results of training the remaining three clans were varied and inconsistent with average assembly times ranging from 10.5 to 14.5 seconds across members of the clan.

V. DISCUSSION

The process of finding the best set of assembly parameters—even a decent set of parameters—for robotic assembly programs is a difficult and time-consuming task. Our preliminary work has shown that, with properly constructed GA, viable assembly parameter solutions can be revealed relatively quickly and without necessitating detailed knowledge of the task space outside of assembly strategies. In our test, solutions found, in terms of all aspects of performance measurements, were superior to those achieved via manual training. This is consistent with our previously reported work, indicating that GA-based learning is an effective approach for improving the performance of assembly tasks.

Observing the resulting data in Figures 4 and 5, it is apparent that in our simplified computational model, learning is, in fact, taking place as the performance of assembly is clearly improving in a directed manner. From this we can conclude that we can perform assembly optimization quickly, over large search spaces with only purely stochastic methods. Further, the performance of the parameter solutions resolved stochastically demonstrates that GA-based learning approaches can result in commercially viable solutions.

Our results to date are encouraging in terms of performance, applicability to industrial implementations, reuse of assembly strategies, and safe and effective robotic learning. Further, the GA software that runs on an external PC was modularly reusable among different robot types with significantly different dynamics. In continuing research, we are working to extend generality to fully-automated solutions in which the system can discover not only the optimized assembly search parameters, but also the actual assembly search strategies as genes are inserted, deleted, spliced, and mutated.

REFERENCES