

<sup>1</sup> AD- D802775		<sup>2</sup> MT 3975	
<sup>6</sup> TITLE: ARTIFICIALLY INTELLIGENT ROBOTS.			
<sup>7</sup> AUTHOR: B.K. / McGREGGON ; C. / TROUTMAN			
<sup>9</sup> DESC. NOTE Conference paper,		<sup>11</sup> DATE Jan 87	<sup>12</sup> PAGES 15
<sup>14</sup> REPT. NO.		<sup>15</sup> CONTRACT NO.	<sup>18</sup> MONITOR
<sup>19</sup> SERIES			
<sup>21</sup> SUPPLEMENTARY NOTE Published in: The Southern Manufacturing Technology Conference Proceedings, 26-29 January 1987, Charlotte, N.C.			
<sup>22</sup> DISTRIBUTION STATEMENT Availability: National Machine Tool Builders Association, 7901 Westpark Drive, McLean, VA 22102. No copies furnished by DTIC/NTIS or MTIAC.			
<sup>27</sup> ABSTRACT  Attached as indicated			
			<sup>28</sup> ABS. CLASS U
<sup>33</sup> KEYWORDS (U) * ARTIFICIAL INTELLIGENCE * ROBOTS Work Cells FACTORY AUTOMATION EXPERT SYSTEMS /CODE W AEROSPACE INDUSTRY			
<sup>32</sup> DIST. CODE 1, 21	<sup>34</sup> SERIAL NO.	<sup>35</sup> SOURCE CODE	<sup>31</sup> IAC DOC. TYPE 1

MT  
003975

ARTIFICIALLY INTELLIGENT ROBOTS

BY

B. KEITH MCGREGGOR

CAROL TROUTMAN

LOCKHEED-GEORGIA CORPORATION

Presented at Session 4C: Artificial Intelligence for the Factory

SOUTHERN MANUFACTURING TECHNOLOGY CONFERENCE

JANUARY 26-29, 1987

Sponsored By The  
National Machine Tool Builders' Association

Co-Sponsored By  
The American Die Casting Institute; The Industrial Diamond  
Association of America, Inc.; The Institute of Industrial Engineers;  
The National Electrical Manufacturers Association; the Greater  
Charlotte Chapter and the Piedmont Chapter of the National'  
Tooling and Machining Association; and The Tooling and  
Manufacturing Association

ARTIFICIALLY INTELLIGENT ROBOTS

By

B. KEITH MCGREGGOR  
Manufacturing Technology Engineer

CAROL TROUTMAN  
Manufacturing Technology Engineer  
Lockheed-Georgia Corporation

ABSTRACT

Developing artificially intelligent robots can be the basis for integrating AI techniques into existing and planned automated workcells. The described application makes use of case-based reasoning and natural language processing techniques.

Artificially Intelligent Robots  
B. KEITH MCGREGGOR AND CAROL W. TROUTMAN

### Introduction

At Lockheed-Georgia, we're working on a project to smoothly integrate artificial intelligence techniques with automated work-cells. Our approach is called the Programmer's Assistant/Operator's Assistant method. In this paper, we'll explain our goals for artificial intelligence in manufacturing, our methods, our current progress, and our plans for the future.

### Lockheed's Manufacturing Environment

First, let's take a look at Lockheed's manufacturing environment. On the plant floor, many cost centers contain up to several dozen workstations each. Each cost center is responsible for a limited number of operations - such as welding, routing, deburring, and riveting.

Although Lockheed is a large aircraft manufacturer, product output is low - typically three or four units per month. Because of this low rate, and because parts tend to be unique, part batches are small. Normally, a part batch will consist of less than 50 parts.

These small batch lots result in a great deal of machine shop setup and transit time, often much longer than actual production time. At Lockheed-Georgia, robotic workcells are being used to reduce production costs.

### Goals for Robotics

When integrating robots and automation into a workstation, our goals include combining as many operations as possible (to reduce transit time); and providing enough built-in flexibility to handle a variety of parts (to reduce machine setup time).

### Robots in Production

Currently, Lockheed has three robots in production. These robots perform a variety of manufacturing operations, including drilling, fastening, routing, and welding. In addition, we have four new robotic workcells in various stages of completion.

### The "Hidden" Cost of Robotics

Robotic and other automated applications are proving cost-effective in many installations - here at Lockheed, throughout the aerospace industry, and in other industries. The benefits of such advanced automation include flexibility, quality improvement, multishift operation, and improved worker safety. Robotic applications are an important trend in relieving certain problems of labor and quality.

Although implementing a robotic workcell provides immediate cost savings, a more subtle cost is being experienced - the "hidden" costs of automation.

#### Development Costs

Developing an automated workcell takes considerable time. Often a workcell will go through several major design changes before finally taking shape. Proving the feasibility of new applications can be difficult, since initial programming is limited to a small variety of parts.

#### Maintenance Costs

Maintaining a robotic system is expensive. A workcell is a complex interconnection of sophisticated, unique hardware and software. Diagnosing and fixing problems requires specially trained technicians.

#### Programming Costs

While the costs of developing and maintaining automation is high, the largest "hidden" cost is that of programming. An automated workcell must be reprogrammed for each new part or technique. Often, this requires several hours of production downtime, while the workcell operator teaches the robot the new program by leading it through manually. Even if the robot is programmed "off-line," the program must be verified on the actual robot, again resulting in downtime. Due to the nature of aerospace manufacturing, with small batch lots and unique parts, reprogramming occurs almost daily. The amount of time the robotic system is out of production while being programmed rapidly becomes significant.

#### Our Solution

The benefits of robotic manufacturing systems are important, yet they tend to overshadow the costs of development, maintenance, and programming. Without an effective way to control these "hidden" costs of automation, the large potential savings cannot be realized. To alleviate these costs, we have turned to artificial intelligence.

#### Artificial Intelligence

The long-term objective of our project is an artificial intelligence system which, when instructed to perform a fabrication or subassembly task, will formulate its own program and then execute that program. The AI system will monitor its program's progress, fill in any missing details, and adapt to any unexpected situations that might arise in the work environment.

Our goal is to lower the cost of automation by automating as much of the development, maintenance and programming of a robotic system as possible.

To achieve our goal, we are pursuing a common-sense approach for integrating artificial intelligence into our robots. We call our approach the Programmer's Assistant/Operator's Assistant methodology.

With every robotic system there shall coexist two expert systems, the Programmer's Assistant/Operator's Assistant. The Programmer's Assistant will assist with the programming of the workcell, while the Operator's Assistant will assist the operator of the workcell and those who maintain it.

### Philosophy of Design

Before we discuss the Programmer's Assistant the Operator's Assistant expert systems, we need to present our definition of "expert system."

The term "expert system" seems to enjoy vigorous usage in the vocabulary of many managers, engineers, and scientists. However, the meaning of the term has been subject to wide interpretation. We feel that any definition of "expert system" must incorporate some sense of what an "expert" is.

For us, an "expert system" is a system which makes decisions in an intelligent manner and possesses the capability to improve the quality of its decision making through experience. We hold that the essence of an "expert" lies in the role his experience at problem-solving has played in the reorganization of his knowledge. That is, the difference between an expert and a novice lies not in what is known, but in how it is know.

We are striving to build expert systems, ones that benefit from their previous problem solving efforts, and not novice systems.

### Programmer's Assistant

The Programmer's Assistant has but one goal: to assist the human programmer with the production of part programs for a particular robotic workcell.

### Sources of Information

A lot of knowledge is required to construct a part program. Geometric information must be electronically culled from engineering design systems such as CADAM and CATIA. Shop order information must be interpreted from process planning systems. (At Lockheed-Georgia, the process planning system is called GENPLAN.) The physical capability and configuration of the workcell restrict both the parts which can be made and the programs which make them. All of this information - geometric, shop order, workcell configuration, and capability - must be analyzed and integrated by the Programmer's Assistant into a robot program for building a part.

## Approaches

To decrease the complexity of the data fusion problem, we are investigating ways of either distributing the knowledge required or representing it in an easily accessible form.

### Constructive Solid Geometry with Swept Primitives

Perhaps the most theoretically difficult problem facing the Programmer's Assistant is the requirement that it automatically extract pertinent features from the geometric models generated by our modelling systems. Structured topological information is not generally available. Moreover, once a set of features has been derived, the expert system must then somehow relate them to steps for manufacturing the part.

Currently, we are attempting to develop a new method of representing the geometry of a part which incorporates certain manufacturing information. We call this method "Constructive Solid Geometry with Swept Primitives" or CSGSP.

There are three traditional solid geometric modelling representations; boundary representation (B-rep); constructive solid geometry (CSG); and simple sweeping. A B-rep model consists of a collection of fundamentally polygonal faces described by surface equations and bonding edges. A CSG model is composed of additive and subtractive combinations of primitive solid building blocks. Finally, a simple sweep model consists of boolean combinations of solids formed by sweeping two-dimensional flat faces along arcs in space. Of the three, the boundary representation method is the most widely used in commercial solid modelling.

A CSGSP model has a lot in common with all of the traditional methods, yet it is novel in its combination of traditional elements. Like a constructive solid geometry model, a CSGSP model is composed of boolean combinations of primitive building blocks. However, unlike CSG, the primitive building blocks are not constrained to primitive shapes, like spheres, cubes, and cones. For example, a primitive in a CSGSP model might be a fluted drillbit.

A CSGSP model is formed by combining the volumes generated by sweeping other volumes through finite-length curves in space. The volumes being swept may themselves have been generated by sweeping other volumes in space. Thus, a CSGSP model's collection of "primitive" shapes is not necessarily primitive.

CSGSP differs from the simple sweeping method in that it allows for the sweeping of three-dimensional volumes, not merely flat surfaces. Additionally, since a CSGSP model's primitives are not limited to traditional primitives, CSGSP enjoys the same complexity of shape that B-rep offers.

CSGSP is unique in that a manufacturing process can be associated with either the sweeping technique used to create primitives or with the boolean operation used to combine them. For example, to represent the effect of a tool on a part, the CSGSP model would contain the model of the tool itself, the path the tool should take, and the raw stock of the part. The solid model of the part is generated by first sweeping the tool's geometry through the tool path, and the subtracting that volume from the volume of the raw stock. By generating the part model in this manner, CSGSP effectively captures the "what" and "where" aspects of how to cut the part using the tool. This ray manufacturing information, then, is an integral, inseparable component of the geometry of a part.

This inclusion of "what" and "where" knowledge into the part's geometry gives the Programmer's Assistant an "initial" set of actions for the robotic program which will make a part. Furthermore, since this enhanced geometry of the part contains descriptions of both the tools and the tool paths, the Programmer's Assistant can easily decide whether or not a particular part can be made in a given workcell, by comparing the tools' geometries with the capabilities of the workcell.

#### Natural Language Techniques for Shop Order Interpretation

Another way we are relieving the information fusion problem is by using natural language techniques. In the late '70s, Roger Schank and Robert Abelson of Yale University developed a method of representing natural language stories based on developing scripts, or usual sequences of events, and then filling in the blank spots.

A shop order for a workcell contains a set of short-hand natural language instructions which tell a human worker exactly what needs to be done to a part and in what order. We consider a shop order to be one kind of script in the Schankian sense: it is a sequence of steps that are to be done to a part under ordinary circumstances. Thus, if we can adequately represent the textural instructions found on the shop order, then we can provide the Programmer's Assistant with a very good approximation of the sequence of operations required to manufacture a part.

#### Dynamic Memory

A third way we're attacking the complexity of the Programmer's Assistant is by attempting to exploit the expert system's previous experience as an aid for generating robot programs.

Most of the artificial intelligence research into problem solving has considered each problem to be solved as a unique, singular event. When people solve problems, however, they seem to consider not only the problem at hand, but the similarity it has with problems they have solved in the past. Reasoning from previous experience, or reasoning by analogy, guides and focuses the decision making process. Moreover, experience seems to contribute to the refinement and modification of the reasoning processes.

The problem of generating robotic programs is almost ideal for reasoning from previous experience. Robotic workcells in the aerospace industry, particularly at Lockheed-Georgia, typically are restricted to working on parts from either a single or a small set of part families. Parts are grouped into families by (1) function; (2) composition; (3) geometric complexity; and/or (4) manufacturing techniques required. Since the parts within a part family are similar, the programs a robotic workcell might use to produce the parts likewise should be similar.

The design of Programmer's Assistant expert system is chiefly motivated by a desire to reduce (or eliminate) the high cost of reprogramming the robot to accommodate slight changes in part configuration or manufacturing process. The need for a system which exploits the suggested similarity of part programs is clear. The method for implementing such a system, however, is not immediately obvious.

Once again, we turn to the Yale approach to natural language processing for inspiration. Much of the recent theoretical work at Yale (and at Georgia Institute of Technology under Janet Kolodner) has been centered on the notion of how to structure a knowledge base efficiently for the retrieval of relevant information. The particular method of structuring their knowledge bases is called dynamic memory.

Under the dynamic memory method, a tree is constructed with memory organizational packets (MOPs) at the internal nodes and pointers to specific events at the external nodes (leaves). A MOP is a generalization of the MOPs or events indexed beneath it.

The branches of the tree are two-tiered: the first level of each branch essentially corresponds to an attribute; the second level of the branch is a measure of the attribute. Therefore, each branch of the tree is itself a sub-tree which fans out to accommodate the various values.

When attempting to generate a robot program for a part, the Programmer's Assistant uses the following approach:

First, all available information about a part is extracted from the geometric and shop order information and combined into a part-context.

Second, the Programmer's Assistant attempts to "recall" having done this particular program before. That is, using the part-context, it begins matching this context against the dynamic memory, progressing downward in the tree, until either (a) no further matching can occur; or (b) a specific event is reached. The set of nodes in the dynamic memory where this retrieval process stops is the collection of knowledge currently contained in memory which is relevant to the part-context.

Third, the set of nodes retrieved in step two is weighted by their depth in the memory. The deepest nodes are the ones most similar to the part-context. The information contained in them is used to instantiate any missing information in the part-context.

Fourth, if any details are still missing or in conflict with other details in the part-context, a traditional rule-based component of the Programmer's Assistant will attempt to fill in the blanks or arbitrate the differences. This traditional system treats the part-context as a part of a script.

At this point, the part-context will contain a generic robotic program for building the desired part.

To be able to make use of previous experience when problem solving, the expert system must incorporate some sort of feedback. This feedback tells the expert system whether or not a solution worked. A system which produces solutions but has no way of judging the solution's effectiveness cannot be used to suggest new solutions based on old ones. Thus, feedback, i.e., the measurement of the worth of the solution, must be provided to the Programmer's Assistant somehow.

The Programmer's Assistant will present the proposed solution to the human programmer for criticism. The human programmer can rearrange the proposed program freely, but must provide the system a reason for doing so. He can also request a graphical simulation of the program, to check for collision detection within the workcell. Once he is satisfied, he instructs the Programmer's Assistant that the program is valid.

The Programmer's Assistant will attempt to "remember" the program into the dynamic memory. Essentially the retrieval process is rerun, but this time, instead of collecting all the relevant information, the Programmer's Assistant will index the program with that information. By adding to memory in this manner, the Programmer's Assistant guarantees that future attempts to "recall" a program for the given part, or one similar to it, will retrieve this relevant program.

#### Learning in the Real World

The dynamic memory system used by the Programmer's Assistant can easily provide access to all currently relevant information in the knowledge base. However, several implementation questions arise: How do MOPs, the "generalized" entities in the knowledge base, get generated? How do you decide which attributes should be used for indexing? How often does generalization occur? How can you measure the "state" of the knowledge base?

The attributes used for indexing between nodes in memory should discriminate sets of child nodes from one another. Due to the two-tier branching in dynamic memory, the values of the attribute itself should further discriminate the child nodes within the set. Choosing attributes which discriminate as indexes makes the retrieval process more efficient, since it reduces the possibility of retrieving (and therefore having to examine) irrelevant information.

The easiest way to automatically choose attributes which discriminate is to find those aspects of the part program which differ from the norm (given in a particular MOP). These differences can be a significant variation in an attribute, or the inclusion of attributes not found in the norm. What constitutes a "significant" variation in an attribute will, of course, depend on the nature of the attribute itself.

The MOPs are generated whenever too many nodes are indexed under a particular attribute's value, i.e., whenever an attribute fails to discriminate efficiently. A new MOP is formed by taking as a norm the value of the attribute common to the child nodes, choosing attributes from the child nodes which discriminate them from one another, and then indexing them beneath those attributes. The new MOP is indexed directly under the old nondiscriminating attribute. This process is "generalization."

Generalization, then, occurs whenever too many nodes are indexed under a particular attribute's value. But what is "too many"? Is it three? Five? One hundred? The decision of what is "too many" will affect how efficient and how stable the knowledge base is.

We recently began an effort to quantify this notion of "too many." Our current activity centers on assigning a measure of fit (in a possibilistic sense) for a node indexed under another node. We are attempting to define the extent to which an attribute can discriminate as a function of the entropy of the tree beneath it. (We use "entropy" in the information theory sense.) By backing up these entropy values to the top of memory, we should be able to provide a quantifiable way of describing how "good" (i.e., how efficient) the memory organization is.

#### Operator's Assistant

The Operator's Assistant is somewhat simpler than its sister system. It has four functions: (a) instantiating part programs; (b) executing programs in the workcell; (c) actively monitoring the workcell; and (d) assisting in error diagnosis and recovery.

## Instantiating Part Programs

With few exceptions, the exact machine locations within a workcell are dynamic. Location tolerances are small, typically around  $\pm 0.030$  inch. Often, part of a workcell must be reconfigured to accommodate a new part, technique, or tool.

In the past, any program for a workcell would need to be modified manually to adjust for any changes in the workcell. This was (and is) a very time consuming process.

The Programmer's Assistant generates part programs which contain specific instructions for what to do, but generic locations. The Operator's Assistant takes the program from the Programmer's Assistant and instantiates the generic locations with actual ones measured either by hand or by sensors in the workcell. Thus, if a machine is moved in the workcell, no reprogramming by the Programmer's Assistant is required: the programs will be adjusted automatically by the Operator's Assistant.

## Executing Programs in the Workcell

After the actual machine locations are placed in the part program, it is executed by the robot controller.

## Actively Monitoring the Workcell

The Operator's Assistant monitors the work in progress, immediately and intelligently responding to unexpected situations that may arise.

An audit trail is produced when a batch of parts is successfully completed. This provides information to better schedule work for the workcell. It also provides valuable clues for tracking down any long-term or recurring errors.

## Assisting in Error Diagnosis and Recovery

When an error occurs, the Operator's Assistant will lead the human operator through error diagnosis and recovery tasks.

## Current Progress

Now that we've outlined the vision, let's take a look at our current progress.

## Graphics Simulation

We have developed a sophisticated computer graphics animation system based on our VAX 11/730 mainframe and our Evans and Sutherland PS-340 graphics workstation. Our method of animation allows us to control up to 1000 independent variables in time-triggered system based on a master clock. Each of the independent variables can have up to 1000 events scheduled for it. These variables can be

graphically connected to such things as joints for simulating the movement of a robot's arm, a spline control point, or a component of the color of an object. We view our simulations in real-time as 3-D color wireframes from any angle (we can alter our view instantly, even while the simulation is running), or in stop action as accurately shaded 3-D solid models.

We have built more than a dozen successful simulations of robotic workcells using our methods. We have used these simulations both as a check-out for existing workcells and as a sales tool for proposed workcells.

#### The Analyze Program

In 1985, a rudimentary version of a geometry analyzer, ANALYZE, was written. This program attempted to generate the sequence of operations necessary to build a part strictly from the CSGSP representation of that part. ANALYZE has been tested on several medium-complexity parts (including a piston-like part which required three separate subassembly tasks). Although primitive in its operation, ANALYZE demonstrated the feasibility of recovering actual manufacturing information from a CSGSP model.

The ANALYZE program will be merged with a natural language system to become the data interpretation section of the Programmer's Assistant in early 1987.

#### The Robot Programming Language (RPL)

The Programmer's Assistant implies that a universal high-level language should be used as the tongue of robot programs. We developed the RPL language to meet that requirement.

RPL, an abbreviation for Robot Programming Language, was designed with the goal of providing a programming environment which emphasizes target machine independence, automatic program generation, and simplified program maintenance.

Target machine independence implies that regardless of on which robot or host computer RPL runs, the RPL source code will not require modification. That is, the RPL compiler will take care of low-level details, and allow the programmer to concentrate on the higher-level, task-related aspects of the code.

RPL's syntax and semantics are similar in spirit with Pascal and C, yet it is not as complex. RPL's syntax was deliberately kept simple enough to allow the Programmer's Assistant the freedom to write direct, concise code. Of course, simplicity often detracts from a programming language's flexibility, and this is true of RPL. However, we believe the relative inflexibility of RPL (as compared to Pascal, for example) is due to the notion that RPL is a machine-directing language rather than a general purpose programming language.

One of the problems in designing a language is that you must provide enough capability to handle common features of its target domain. The particular difficulty in designing a robot programming language is that each robot has its own set of quirks that the higher-level language needs to address.

In RPL we have struck a compromise. We chose an existing robot (the MTS 200A) as the proverbial "ideal target" and began the language to accommodate opcodes and other features peculiar to that robot. Once we had the rudiments of RPL up and running, we then added control features and procedures to the language, drawing heavily upon our experiences with Pascal, C, and Ada.

If RPL worked only with one robot, we would be no further along in advancing automatically-generated programs into production. Fortunately, adapting RPL to work with different robots is relatively simple. We have semiautomated the task of building off-line systems based on RPL by creating the RPL Off-line Construction Kit. This is a VAX command file which takes an appropriately written post processor and merges it with an RPL shell interpreter. The result is a system which will compile RPL programs into whatever is necessary for a given robot (or simulation).

#### The LES Robot Cell Diagnosis System

We have created the first segment of a LES knowledge base for diagnosing errors in one of our production robotic workcells.

The MultiFunction fabrication robotic workcell is a complex, automated production workcell. It contains a Cincinnati Milacron T-3 hydraulic robot, a shuttling, air-powered workstation, and a sophisticated calibrated system. Since the cell is extremely interconnected, even a small problem is capable of bringing it to a standstill. Many of these small problems may be correctable by the cell's operator, provided that the problem is properly diagnosed and that the operator is given the appropriate corrective information.

The Maintenance Expert System (MES) is an attempt to provide the cell operator quick, easy access to reliable information about possible problems in the workcell. Today, this system can help diagnose calibration console electrical problems, pneumatic system problems, end-of-arm tooling problems, workstation problems, and some production problems. Currently, the Maintenance Expert System has over 300 facts and 100 rules in its knowledge base. We are continuing to add to this knowledge base.

#### The Case-Based Reasoning Shell

We have just begun work to build a case-based reasoning system for use with the Programmer's Assistant. This case-based system will be adapted from a previous system written by McGreggor for reasoning by analogy in the domain of psychopharmacological diagnosis. We expect to have this system up and running on our new Apollo systems by the end of this year.

## The Future

### Need for Representations

Given the breadth of our approach, we need to come up with a standard way of representing data. Obviously, we are exploring extending the Conceptual Dependency theory of natural language to include scripts and actions in the manufacturing environment. Likewise, we are pursuing a frame-like approach to representing memory organizational packets and specific events in memory.

Currently, a key missing representation for us is one adequate to represent complex geometry. An efficient representation of the geometry of the part, using the CSGSP method, simplifies the task of creating a par-context with which to search the dynamic memory.

At the present, the ANALYZE program works on a very primitive representation for the geometry, for our goal was to demonstrate that it could be done. In 1987, we will emphasize the creation of more efficient representations.

### Finding Adequate Hardware

Our notion of the Programmer's Assistant/Operator's Assistant is that these expert systems will be an integral part of every new production robotic workcell. Clearly, we can't afford to put a VAX and an Evans and Sutherland system on the production floor each time we introduce a new work cell. The solution must lie in micro-computers.

We are writing our software in the most general dialect of common languages: Pascal, for data manipulation and graphics; and Lisp, for symbolic processing and expert systems implementation. Late in this year, we will be shifting our software development over to use C instead of Pascal. In 1987, we will migrate our software down to PC/AT-class machines.

### Limited Production Implementation

In our robotics laboratory, we rarely have the opportunity to try experimental software on our robots because they are usually in the process of being moved into production or because the robot controllers have been too primitive. This year, however, is different.

Currently, we are installing an MTS 200A jointed arm robot in our lab. This robot has a MicroVAX II as a controller. We have already written an RPL compiler to work with the 200A robot, and have created a simulation of the system as well. We will be using the 200A as a testbed for a VAX/Evans and Sutherland-based version of the Programmer's Assistant in early 1987.

## Summary

At Lockheed-Georgia, our Automation Sciences group is continually searching for better, safer, and more efficient ways of manufacturing quality parts at reduced cost. The unique approach of integrating artificial intelligence with automation using the Programmer's Assistant and Operator's Assistant expert systems is just one indication of Gelac's commitment to significantly advance the state of the art in aerospace manufacturing.