RECENT WORK WITH THE AL SYSTEM

Ron Goldman
Stanford Artificial Intelligence Laboratory
Stanford, California 94305

ABSTRACT

A first level implementation of AL, a high-level programming system for manipulator control, has been completed and is now in operation. Several new modules have been added to the system including: POINTY an interactive system for specifying representation of parts, and ALAID an interactive debugger for AL. Recent work in vision has been incorporated into the AL system. This paper describes the present implementation of AL and discusses current work.

INTRODUCTION

This paper summarizes recent work done by the Hand-Eye group at the Stanford Artificial Intelligence Laboratory. For a complete discussion the original papers should be consulted [see bibliography].

General purpose robot manipulators such as the "Unimate" provide a possible answer to the problems of automation of assembly for small scale batch manufacturing and of materials handling where special purpose equipment is too costly. We are implementing a system called AL for the specification of these tasks. The principal aim of our work is not to provide a programming system for the factory floor, but rather to do research on the underlying issues inherent in such a system.

AN OVERVIEW OF THE AL SYSTEM

We have a Digital Equipment Corporation KL10 processor supporting the SAIL language (a dialect of ALGOL), and a Digital Equipment Corporation 11/45 minicomputer which is programmed in PALX assembly language. Two six degree of freedom Scheinman Stanford arms and several other small devices such as a mechanical screwdriver and a pneumatic vise are controlled by the 11/45.

The main modules of the AL system are shown in figure 1. Through POINTY the user generates a description of the parts to be used in the assembly. This data structure together with a user written AL program is then compiled. The resultant code (called the pseudocode) is interpreted by the runtime system causing the manipulators to perform the desired task. ALAID provides debugging facilities and an interface to other programs such as a vision module.

In the current implementation the runtime system and part of ALAID reside on the 11/45. The remainder of the system uses the KL10.

A BRIEF SUMMARY OF THE AL LANGUAGE

At the heart of our work is the AL programming language [5,7]. AL is an ALGOL-like source language extended to handle the problems of manipulation. Much of the design of AL grew

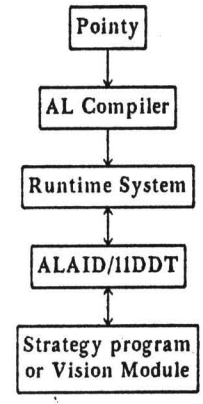

figure 1.

out of experience with the Stanford WAVE system [9].

In addition to the normal scalar variables found in most programming languages, available data types in AL include those types necessary to specify three-dimensional measures like directed distances, locations and orientations. Arithmetic operators such as rotation and translation are provided to handle these new data types.

Provision is made for simultaneous execution of several processes. This allows calculation and arm movement to take place concurrently. Several manipulators can be operated in independent or coordinated motion. Synchronization of parallel processes is accomplished by signal and wait primitives.

A rich vocabulary for specifying manipulator motion is provided. Included in this is the ability to monitor various conditions (e.g. force or touch) and to perform an appropriate action if the tested condition occurs.

A general purpose text macro facility is also available.

REPRESENTATION OF OBJECTS

One goal in the design of AL was for ease of programming, and in specific, ease in representing objects and the relationships between objects. Even for a simple part like the box in figure 2 there are several features: two screw holes, a grasping point, and the box's location. When the box is moved the other features should move with it. Also when the hand is

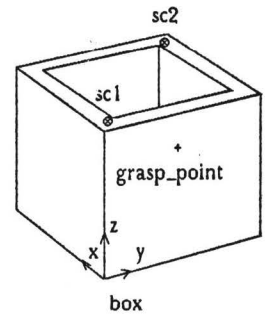

figure 2.

```
FRAME box, box_grasp, sc1, sc2;
AFFIX box_grasp TO box
        AT FRAME(ROT(YHAT,180),VECTOR(1,1.5,1));
AFFIX sc1 TO box
        AT FRAME(ROT(YHAT,180),VECTOR(0,0,2));
AFFIX sc2 TO box
        AT FRAME(ROT(YHAT,180),VECTOR(2,3,2));
box ← FRAME(nilrotn,VECTOR(12,21,0));
```

Declarations necessary to describe the affixment relations of the box in figure 2.

figure 3.

grasping the box one would like to specify motion in terms of the box and let the system compute the necessary movement of the arm. AL allows the user to declare the affixment relations between objects and then automatically takes care of the subsequent bookkeeping operations.

From a study of sample AL programs it has become apparent that these declarations occupy a substantial proportion of the code. They also tend to be more difficult to write than the procedural statements (see figure 3). With the development of even higher level manipulator languages, declaration of a detailed world model will become a significant problem.

POINTY is a prototype system that allows the user to interactively build the necessary data structures using the manipulator itself to point to the objects and their features [8]. The output from a session with POINTY is a text file of AL declarations. To increase the manipulator's precision as a measuring tool a sharp pointer is grasped by the hand. The pointer is shaped so it can reach into such awkward places as the inside of a screw hole or the interior of a box. In order to make the shape of the pointer compatible with all kinds of unforseen obstructions the pointer may be bent by the user into an arbitrary shape. When the pointer's shape is changed, recalibration of the pointer's tip is quickly accomplished by pointing to a known position.

The current POINTY system contains three major modules: an affixment editor, arithmetic routines, and an interface to the manipulator. The affixment editor contains facilities for creating and modifying the affixment relations between the objects being modelled. The arithmetic routines permit the user to perform arithmetic operations and to modify the location attributes of parts. The manipulator interface contains facilities for moving the manipulator under either system or user control and for retrieving the current position of the manipulator for use by the rest of the system.

A preliminary version of the system has been implemented and tested. This preliminary system demonstrates that specifying object modules can be a much easier process than might otherwise have been believed. A fuller version of the system is currently under development.

DEBUGGING MANIPULATOR PROGRAMS

Experience has shown that most programs of moderate size contain errors, and that debugging is a significant part of software production. This work is greatly facilitated by the use of debugging tools that know about the language the program is written in. For example BAIL, a debugger for SAIL, knows about SAIL's data types, primitive operations and procedure implementation [10].

Debugging an AL program involves examining and modifying variables, altering the flow of control, triggering condition monitors, and patching code. Provision must also be made to handle both the explicit and implicit parallelism in the program. Another factor is that manipulator programs work in the real world which is less tractable than the highly controlled world of the computer. Many actions are irreversible. Backing up to an earlier state usually involves the repositioning of physical objects, including the manipulator. Failures arise due to discrepancies between the program's model of the world and the actual state of affairs. Indeed the idea of a program crashing takes on new meaning.

ALAID has been designed to meet these problems and to assist the programmer in preparing correct manipulator code [6]. Since our system resides on two computers, we further require ALAID to provide a link between the two machines. This allows debugging to proceed from either machine. It also allows a clean interface between an AL program running on the 11/45 and a higher level strategy program on the KL10. ALAID enables the two processes to signal each other using the synchronization primitives in AL and it also allows the program running on the KL10 to examine and set variables in the memory space of the AL manipulator program on the 11/45.

The state of ALAID at the moment is fairly primitive. It connects the two machines, can examine and set variables, signal and wait for events, and cause the runtime system to enter 11DDT, a symbolic assembly language debugger. ALAID resides on both machines and runs as a parallel process with the runtime system. A more advanced version of ALAID is currently being implemented which will allow the user to alter the flow of control, set breakpoints, and examine/modify the pseudocode.

INTERFACING VISION TO AL

There are many manipulator tasks which are greatly enhanced by the use of vision. Visual feedback can provide better positioning, inspection, error detection, and error recovery. Recent work in visual information processing here has taken advantage of the fact that in assembly tasks there is a great deal of prior knowledge about the type, placement, and appearance of the objects that form the scene [4]. The goal is to verify an object's presence (e.g. is the screw on the screwdriver), or to refine the location of some object (e.g. where exactly is the screw hole). From the model of the expected scene one knows roughly where each object should be. They may be misplaced by half an inch or rotated fifteen degrees, but there will not be any big surprises. This class of visual tasks has been named verification vision.

Through the use of ALAID it is possible to have a verification vision program running on the KL10 interfaced to an AL program on the 11/45. The vision module can be coordinated with the manipulator code to provide a large degree of visual feedback. Whenever the manipulator program needs visual feedback it can signal the verification vision program. Using ALAID the vision module can examine variables in the manipulator program to see which of several tasks it is to perform. It can then take a picture and compute the needed information which, again via ALAID, can be stored into the appropriate variables in the manipulator code.

A manipulator program making use of visual feedback has been successfully demonstrated. The manipulator, holding a mechanical screwdriver, picks up a screw from a dispenser and inserts it into a screw hole in a carburetor assembly. A quick visual check is made to confirm that a screw has actually been retrieved from the dispenser, and if not, another try is made. The precise location of the screw hole is determined visually and this information is made available to the manipulator program.

OTHER WORK

The present AL system calculates trajectories for the manipulators at compile time. We are presently investigating possible algorithms for runtime trajectory calculation [3]. Doing the calculation at runtime will reduce the load on the planning system in the compiler, possibly to the vanishing point. This will make it easier to add arrays and procedures to the AL language. It also opens the possibility for an interactive manipulator system.

We are also improving the force-feedback features of AL [3]. A new force-sensing wrist is being added to the manipulator allowing more precise force monitoring. Better touch sensors are also being investigated. Software to allow compliant motions and the application of forces, in addition to better force-sensing, has recently been completed. This work will make our manipulators capable of much more delicate motions.

There are still several minor features of AL that have yet to be implemented, such as library functions. These will be added to AL shortly and a number of demonstration assemblies will be programmed. A film of AL in operation will be available soon.

Finally, work will shortly begin on two arm cooperative motions when our second arm interface is finished.

SUMMARY

The first version of the AL language is now operational. Experience obtained from writing AL programs for various assembly tasks has shown that the declarations necessary to describe the parts occupies a substantial portion of the code, and that these declarations tend to be more difficult to write than the procedural statements. POINTY, a system to allow the user to interactively generate these declarations, has been written and successfully tested, providing a solution to the problem of parts specification. Work has been done investigating the requirements of debugging tools for manipulator programs. A preliminary system, called ALAID, which knows about the data types in AL has been implemented. Using ALAID, a vision module has been interfaced to the AL system, enabling a manipulator program to utilize visual feedback.

Other current work deals with runtime trajectory calculation, manipulator path specification, improved force-sensing and application of forces, and two arm cooperative motions.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation through grant NSF-APR-74-01390-A04. Thanks also go to all the members of the Stanford Hand-Eye group who have worked with the AL system.

BIBLIOGRAPHY

1. T.O.Binford, R.C.Bolles, R.Finkel, T.A.Gafford, D.D.Grossman, E.Miyamoto, M.S.Mujtaba, M.D.Roderick, B.E.Shimano, R.H.Taylor; *Exploratory Study of Computer Integrated Assembly Systems;* Second Report, Sept 15, 1974 to Nov 30,1975; Artificial Intelligence Laboratory, Stanford University.

2. T.O.Binford, R.C.Bolles, R.Finkel, T.A.Gafford, R.Goldman, D.D.Grossman, J.P.Jarvis, C.R.Liu, M.S.Mujtaba, M.D.Roderick, V.D.Scheinman, B.E.Shimano, R.H.Taylor; *Exploratory Study of Computer Integrated Assembly Systems;* Third Report, Dec 1,1975 to July 31,1976; Artificial Intelligence Laboratory, Stanford University.

3. T.O.Binford, T.A.Gafford, G.Gini, M.Gini, I.Glaser, R.Goldman, T.Ishida, C.R.Liu, M.S.Mujtaba, H.Nabavi, E.Nakano, E.Panofsky, V.D.Scheinman, D.Schmelling, B.E.Shimano; *Exploratory Study of Computer Integrated Assembly Systems;* Fourth Report, Aug 1,1976 to March 31,1977; Artificial Intelligence Laboratory, Stanford University.

4. R.C.Bolles; *Verification Vision within a Programmable Assembly System;* Memo AIM-295; December 1976; Artificial Intelligence Laboratory, Stanford University.

5. R.Finkel, R.H.Taylor, R.C.Bolles, R.Paul, and J.A.Feldman; *AL, A Programming System for Automation;* Memo AIM-243; November 1974; Artificial Intelligence Laboratory, Stanford University.

6. R.Finkel; *Constructing and Debugging Manipulator Programs;* Memo AIM-284; August 1976; Artificial Intelligence Laboratory, Stanford University.

7. R.Goldman, M.S.Mujtaba; *AL Users Manual;* forthcoming; Artificial Intelligence Laboratory, Stanford University.

8. D.D.Grossman, R.H.Taylor; *Interactive Generation of Object Models with a Manipulator;* Memo AIM-274; December 1975; Artificial Intelligence Laboratory, Stanford University.

9. R.P.Paul; *WAVE, A Model-Based Language for Manipulator Control;* First North American Industrial Robot Conference and Exposition, October, 1976.

10. J.F.Reiser; *BAIL, a Debugger for SAIL;* Memo AIM-270; October 1975; Artificial Intelligence Laboratory, Stanford University.

11. R.H.Taylor; *A Synthesis of Manipulator Control Programs From Task-Level Specifications;* Memo AIM-282; July 1976; Artificial Intelligence Laboratory, Stanford University.

12. *<A film of AL in operation>;* forthcoming; Artificial Intelligence Laboratory, Stanford University.