From Goals to Manipulator Programs

Maria Gini
Department of Computer Science, University of Minnesota, Minneapolis, USA

Giuseppina Gini
Department of Electronics, Politecnico di Milano, Milano, Italy

We describe an approach to robot programming that supports task-oriented specifications of manipulation activities. Plan formation and program generation techniques are used to transform task specifications into executable programs for various robots. The plan formation operates on a database of knowledge about the world to produce the sequence of actions needed to execute the task. The program generation transforms the sequence of actions into an executable program using knowledge about the robot and its control system.

Keywords: Robot programming, Automatic Programming, Plan generation, Knowledge bases.

Maria L. Gini is an Assistant Professor at the University of Minnesota, Department of Computer Science, in Minneapolis. She has been a Research Associate at the Department of Electronics, School of Engineering, Polytechnic of Milan, Italy and a Visiting Research Associate at the Artificial Intelligence Laboratory at Stanford University. Her research is in the field of artificial intelligence and its applications to robotics. She is the author of several publications on those subjects.

Giuseppina C. Gini is a Senior Research Associate at the Department of Electronics, School of Engineering, at the Polytechnic of Milan, Italy. Her current research focuses on programming languages, artificial intelligence applications and assembly automation. She is a member of the Group on Robot Standardization of the European Economic Community and was one of a group of experts that defined the goals for the development of computer integrated manufacturing in the ESPRIT (European Strategic Precompetitive Research in Information Technology) program.

Partial support for this work is gratefully acknowledged to the Microelectronic and Information Sciences Center at the University of Minnesota, to the Graduate School of the University of Minnesota, and to the Italian National Council of Research.

North-Holland
Computers in Industry 7 (1986) 263–273

1. Introduction

Robots are today operating on a wide variety of tasks such as object handling, painting, and welding. Even though assembly is still considered a difficult area more and more robots are used in manufacturing for assembly tasks. New areas outside manufacturing, like exploration of unknown environments and medical applications, are being considered.

All these new developments raise crucial problems. Programming robots, which was easy when tasks were simple like pick and place, becomes an important issue. Sensors provide a huge amount of potentially useful data that have to be interpreted to be of any use. Computer programming, which was nonexistent in the early stages of robotics, becomes a central issue.

Manufacturers of robots are providing robots with programming languages. Although this represents an important achievement in industrial robotics, currently available languages are not easy to use. The programmer has to specify complex sequences of movements, to monitor sensors during the execution of the task, to synchronize explicitly different robots operating in the same working area.

Programming a computer controlled robot is really different from programming a computer. Robot programs run in a world that is incompletely known and imperfectly modelled. This requires strategies to detect and prevent potential catastrophes, like collisions, and to recover from errors. Many actions are irreversible. After the arm has crashed there is no way to undo the action that produced the crash.

Actions are not exactly reproducible, making it difficult to detect causes of errors and to fix them.

Currently users must depend on their experience, intuition, and common sense to decide how to write programs.

This paper presents an approach to robot programming based on the automatic generation of programs from task oriented specifications. The specifications are given to the plan formation module that produces the plan of actions. The plan is then transformed into a program by the

program generation module.

We discuss in this paper both plan formation and program generation. These two activities are performed using a knowledge base that contains knowledge about the arm (work area, arm model, sensors), about different tasks (inserting, grasping, fitting), about the programming language of the robot (syntax and semantics), as well as geometric knowledge about the objects involved.

This organization allows to capture part of the experience that experienced programmers apply when they develop new programs. We hope that it will help also in obtaining a better understanding of this process.

A preliminary system has been successfully implemented to generate programs in a Basic-like language for a cartesian robot without any sensor. The knowledge base is small and only tasks related to block movements have been programmed. Programs for the same tasks have been easily generated in a different robot language.

2. Programming Robots

Since computer controlled manipulators have been introduced methods of controlling and of programming them for new tasks have seen a great deal of development [1,4,16,18].

Two completely different approaches to robot programming have been considered in the past.

On one side within the Artificial Intelligence community much research has been done on plan formation systems to provide robots with autonomous reasoning capabilities [21]. None of these systems have been used to control a real robot, except STRIPS at SRI [9].

On the other side the need to control industrial robots has pushed the development of simple but effective methods for robot programming. More complex systems have been designed over the years to cope with increasing needs. None of them require the reasoning capabilities provided by planning systems.

We can simplify things by saying that Artificial Intelligence researchers have taken a top-down approach trying to solve the difficult problem of reasoning and assuming that all the rest was easy. The others have taken a bottom-up approach first trying to control robots and only later realizing the need for intelligence. A big gap still exists between those two approaches.

The aim of this paper is to bridge this gap trying to reconcile the exigencies that created this situation. Before doing that we examine in more details these two approaches.

2.1. Robot Programming Languages

Many of the existing industrial robots are machines capable only of simple motions. They consist of a mechanical arm with a number of degrees of freedom ranging from two to six. They have little or no ability to sense conditions. These machines can simply execute a preprogrammed sequence of operations and can be trained to execute different tasks.

A human operator, using a control box, can program the robot by guiding the arm through the desired sequence of positions. By pushing a button for each position he may record the value of each joint in the memory. Once the complete sequence is stored, it can be played back over and over again to accomplish the task in the production environment. The control system has simply to move each joint from one recorded value to the next according to a preset timing cycle. Any user, without specific training, can program the robot.

Teaching by guiding has been successful for tasks where only simple operations or few positions are required. Where complex assemblies are performed that method does not allow any change or adjustment of the movements during the execution. The impossibility of expressing conditional actions makes it impossible to use sensors. In the case even small changes are made on the assembly station the teaching has to be repeated. The lack of a symbolic text makes it impossible to maintain, document, and modify the robot program.

Programming languages have been designed to overcome these difficulties. A main advantage is that the robot becomes able to move its hand using an external coordinate system instead of its own joint angles. The user instead of specifying joint angles can instruct the robot in work space coordinates. The system will take care of converting positions and orientations into joint angles and vice versa.

Writing programs for robots is not so easy as one could think. Manipulation and assembly tasks are hard to program mainly because it is difficult

to visualize positions and orientations in three dimensional space. Interactions with sensors are hard to express because it is difficult to figure out directions and values of forces.

Higher level languages have been designed over the years. Among those AML [26], AL [4], AUTOPASS [14], PAL [24], RAPT [19], VAL [22], and WAVE [17]. These languages reduce the amount of details that programmers should consider, letting them to concentrate on the most important aspects.

Even those languages suffer from many drawbacks that make it painful to write complex programs. Operations like inserting or fitting are difficult to express. They require long and complex sequences of movements such as pushing the object, rotating it, and pushing it again, while checking forces to avoid jamming.

The specifications of forces to be used is a tedious trial and error process. Learning from a teach pendant is possible for position specifications, but comparable facilities are not available for forces and torques.

Errors are difficult to identify. The real world is much more unpredictable than the computer world. The same program can work well hundreds of times and then stop because the size of one part has a minimal variation or because there is a little spot of oil.

Another source of difficulty in writing programs is the lack of standardization in robot languages and in robot control systems. Industrial robots have languages completely different from each other [5]. Control systems tend to use different interfaces, making it almost impossible to use the same program for different robots. The cost of training programmers is significant and that training may be nullified by the fact that a new robot is bought and the new robot has a brand new language.

There is no general agreement about a standard language for robots. Many problems, mainly the interaction with sensors [20] require further research, and it might be too early for a serious standardization.

Complex tasks requiring more than one robot working at the same time are difficult to program. Parallel actions are hard to express. Time constraints and optimization in the use of available resources require a considerable programming effort. None of the robot languages offer reasonable primitives to do that, while some plan formation systems can deal with resource sharing [28], time [27], and parallel operations [21].

2.2. Plan Formation Systems

The task of a plan formation system is to generate a sequence of actions that transform the initial state of the environment into a final state in which some specified goal conditions are true. Such sequence is called a plan for achieving the goal [21].

A planner must have some knowledge about the world, the task, and the robot's possible actions. A classical representation is based on the description of the world in form of variable free predicates, a specification of the task in form of a goal to achieve, and a description of the actions in term of operators. Each operator has associated some preconditions that specify the world configuration in which the operator can be applied, and some postconditions that specify how the application of the operator changes the world. Those are in general expressed as additions and deletions to the current state of the world.

Plan generation can be regarded as a graph search problem. The initial state and the operators define a graph, in which each arc correspond to an operator. A plan is a path from the initial state to a state in which the goal is achieved.

Plan formation has been used to generate programs for robot tasks, mainly for finding paths in a set of rooms and for stacking blocks [8,9,21].

Programs generated by plan formation systems cannot be directly executed because too many important details are missing. Typically a plan formation system is independent of the robot used to execute the task. So, for instance, it is not within the scope of the plan formation system to check whether positions are reachable or not. Since the purpose of plan formation systems is to do reasoning this is considered a detail unessential for the reasoning process.

The STRIPS system [9] is the only plan formation system used to control a real robot, Shakey. Shakey was able to move in a set of rooms pushing boxes and dealing with unexpected events. It was controlled by the plan formation system with a complex execution monitor. The low level program implementing the set of actions considered by STRIPS required a large effort because of the

details needed to control the real robot.

Plan formation systems have not been used with real robots because in manufacturing the sequence of actions needed to perform a task is known in advance. There is no need to do planning and planning requires large computer power. However, when the task is difficult to specify as a sequence of steps or when the environment changes in unpredictable ways the capability of autonomous reasoning becomes essential [12].

3. On-Line Versus Off-Line Programming

Recently new problems have received considerable attention from the designers of programming languages for robots.

Extensive use of computers and increased computing power available at a reduced cost allow a different philosophy in designing programming systems. What in the past had to be designed to run on a small microprocessor, can today run on a fancy workstation with powerful graphic capabilities.

Artificial Intelligence systems have an economic payoff by reducing the time to design, implement, test, and maintain programs even though they require larger computing power.

The attention is shifting from on-line to off-line programming.

Typically robot programs are developed on-line. The program is written, translated, and tested on the robot. The robot is also used as a measuring tool to gather three dimensional data. In this way robots are used for large amounts of time to develop and test programs.

Tools have been developed to aid users in this process. Most of those tools require on-line programming. The POINTY system [10] has solved most of the problems in constructing data structures and checking the correctness of programs, proving that complex programs can be developed with a good software environment in a reasonable time.

Although POINTY reduces considerably the time needed to produce new programs it has the major disadvantage of taking the robot out of the production line for the time needed to test the program.

The idea of programming robots off-line is becoming more and more appealing. Robots can do useful work while new programs are being developed. This reduces at the minimum the period of time in which the production has to be interrupted. When robots are components of complex industrial automation systems this aspect is particularly important [2].

On the other hand it is well known that it is impossible to test completely a program off-line. Problems come from the lack of sensor data since there is no plausible way to simulate sensors. Even though the use of sensors is still primitive in industrial robots, the simulation of even simple sensory environments, such as the sense of touch, has not been included in any system.

A reasonable solution seems to rely on a combination of off-line and on-line programming, where most of the programming is performed off-line using the robot only to gather data and for the final test of the program. In this way there is no need to run complex programs in real time. With off-line programming sophisticated systems dealing with knowledge and reasoning have good reasons to exist.

Our approach to robot programming requires an off-line programming system that takes care of generating "reasonably correct" programs from task level specifications so minimizing the time required to test the program on-line. We want to generate programs from task level specifications using plan formation and program generation techniques.

The automatic generation of programs has been a dream in Artificial Intelligence for many years. Practical systems have been implemented [3]. Although limited they have shown the feasibility of generating programs from high level specifications. We consider automatic programming particularly interesting for robotics since robot programmers are not computer scientists.

4. State of the Art in Automatic Programming of Robots

Numerous attempts to design intelligent programming systems for robots can be found in the literature. Most of them, as we have seen before, come from the Artificial Intelligence community in form of various plan formation systems.

What we want to examine here are the contributions that consider real robots. They have

focused on problems encountered in real applications.

AUTOPASS [14] was designed in an attempt to provide a formal language with task level descriptions. Many of the underlying problems were unsolved at the time of the design, so that a considerable amount of time had to be spent solving them. For instance, a geometric modeling had to be designed to deal with objects described in AUTOPASS programs. Another problem that required much attention is the automatic planning of collision free paths [15]. The main difficulties in AUTOPASS come from the limited use of sensors and the limitations in the structure of the program introduced to simplify world modeling.

A less ambitious work has been done by Taylor [25]. In his thesis he presents an approach to the fine motion synthesis. His work is based on representing accuracy information in a world model, and on numerical methods to propagate location constraints through relations among features of objects.

He discusses the way in which a program to insert a pin into a hole can be generated automatically using information from the AL world model. Various pickup strategies can be defined at system creation time. He does not address the problem of providing a formal language to describe assembly tasks. A big effort is aimed at generating control instructions and accuracy tests that could predict errors in location values.

RAPT [2,19] is probably the most complete design of an off-line programming system. It generates VAL [22] instructions from geometrical description of objects and constraints among them. Unfortunately, it has an APT flavor that makes its syntax unpleasant. The other problem mentioned by its authors is that it is not clear whether it is really easier to write programs in RAPT, or to use a complex geometric modeling system, or to learn how to use VAL. In some sense with RAPT the problem is shifted from the procedural description of sequences of actions to the description of knowledge about objects and relations.

5. A System for Automatic Generation of Robot Programs

Our approach requires plan formation and automatic program generation in addition to the programming system of the robot that executes the resulting program. Since a programming language is often available on the robot we want to use it. In this way the off-line programming system does not have to know too many details. The lower level control is carried on by the control system of the robot. We are also more independent of the particular robot used since we need to know only how to communicate with it.

We require two basic components, in addition to the control system of the robot, one devoted to the planning and the other to the generation of the robot program. The output of the first part is an intermediate high level language that can be transformed into various target languages. The logical organization is illustrated in Fig. 1. Since we should be able to generate programs for various robots it is important to know the physical configuration of the robot, and its own programming language. The generation of the robot program is based on a database of knowledge which contains both general knowledge about tasks and specific knowledge about the arm that should be used [23].

In Fig. 2 we show the logical organization of the knowledge base. The knowledge depends on the specific robot system (arm knowledge, and language knowledge), on the application area in which we are interested, and on the task that we want to perform.

Further investigation is needed in this area. We will mention some of the problems that we plan to study. This large body of knowledge has to be constructed and maintained to preserve consistency [7].

The user has to interact with the knowledge base to update and to extend it. In robotics interaction with the real world is important. We want, for instance, to be able to extract the information available in running programs and to use it to develop the program itself. An extension to the philosophy of the POINTY system [10] would allow to gather data from sensors during the execution of a sample task, and to use those data to generate the symbolic code. The generation of fine motion could be done in that way.

A precise form of the output of the plan generation system has to be defined, taking care of sensors and of parallel operations. Some of those problems, typically the expression of parallel operations and synchronization, require the selection of appropriate formalisms.

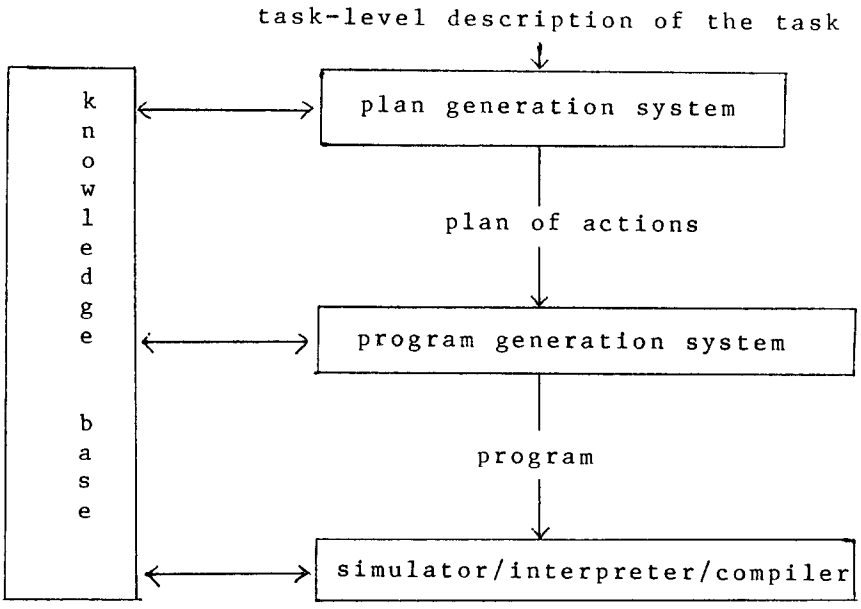

Fig. 1. Logical organization of the off-line programming system.

The knowledge about the specific robot could be expressed using a robot description language. This language is considered an important step toward the portability of robot programming systems and it may become available soon.

The description of the various programming

	generic knowledge	application specific knowledge	task specific knowledge
objects	geometric	position independent	position dependent, initial state
arm	work area, arm model, sensors		
task	knowledge about tasks	hints about tasks accomplishing	task description
language	syntax & semantics		

Fig. 2. Logical organization of the knowledge base.

languages to be used can be done using preconditions and postconditions. This same form is used in our error recovery system [12] and we have found it useful to describe the effect on the world of each instruction.

Even though the approach that we present here is intended for use at compile time, we think that the knowledge needed to generate the program can be used also to monitor the execution of programs.

We see a strong interaction between the off-line activity of automatic program generation and the on-line activity of error identification and recovery. To do automatic error recovery the system should be able to uderstand what is happening in the real world and to generate a sequence of instructions to recover from the error. Both those activities require intelligent interpretation of sensor data and automatic program generation. The system for error recovery [12] that we are developing uses the same knowledge base as the program generation system.

6. An Example of Program Generation

A system able to generate programs from task level specifications is a complex system. Owing to the complexity of the task we have decided to start designing a preliminary system with limited capabilities.

We consider a world model, where simple objects are described by their geometric shape and physical relations. A rudimentary collision avoidance is used. No interaction with sensors is considered.

The system operates in two stages. In the first phase it transforms declarative knowledge into procedural knowledge producing a plan of actions. It then generates a robot program in a specific programming language starting from the description of actions produced by the planner.

The preliminary implementation of our system has a planner written in PROLOG [6,13], a program generation module for MAL [11], and one for AL [10].

6.1. An Example of Plan Formation

Assume that we want to move some blocks from an initial configuration to a final state.

The knowledge is described as a set of clauses in PROLOG. Some clauses express known facts, other clauses express inference rules that can be used to obtain new knowledge. The PROLOG interpreter tries to prove the truth of each relation. Since the relations may contain variables part of the proof process involves variable bindings. Each relation has a left side, that expresses what we want to prove, and a right side that expresses how to prove it. Known facts have only a left side since they do not have to be proved. The interpreter matches the goal with the left side of a clause and tries to prove its truth by proving each of the components on the right side.

The positions of objects are defined by the coordinates of their point closest to the origin. Their geometrical shape is described, as well as their relative positions. Inference rules allow to compute the grasping position of each object, its size in three dimensions, and its position after it has been moved.

For instance, the relation "grasping_of" shown below says that the grasping position of any parallelepiped 0 is a vector (X, Y, Z) obtained by computing half of the size of the parallelepiped itself. Once this information has been computed it is asserted as a known fact, so there is no need to compute it again. Only a part of the knowledge base is listed.

The plan formation produces as output a list of declarations and a plan of actions. The output has the same flavor as AL, although it differs from it in many points. The generation of the program in AL will show these differences.

INPUT

specific knowledge about objects

```
cube(a).
cube(b).
cube(c).
dimension_of(a,2).
dimension_of(b,2).
dimension_of(c,2).
```

transient knowledge about objects

```
on(a,c).
position_of(b,vector(15,5,0)).
position_of(c,vector(10,5,0)).
on(c,table).
on(b,table).
```

generic knowledge about grasping, positions, etc.

```
grasping_of(0,vector(X,Y,Z)):-
   parallelepiped(0),
   size_of(0,vector(SX,SY,SZ)),
   X is SX/2, Y is SY/2, Z is SZ/2,
   asserta(grasping_of(0,vector(X,Y,Z))).
position_of(0,vector(X,Y,Z)):-
   on(0,W), W /= = table,
   position_of(W,vector(PX,PY,PZ)),
   size_of(W,vector(SX,SY,SZ)),
   vectadd(vector(PX,PY,PZ), vector(0,0,SZ), vec-
   tor(X,Y,Z)),
   asserta(position_of(0,vector(X,Y,Z))).
size_of(0,vector(X,X,X)):-
   cube(0), dimension_of(0,X),
   asserta(size_of(0,vector(X,X,X))).
parallelepiped(0):-cube(0).
vectadd(vector(X1,Y1,Z1),vector(X2,Y2,Z2),vector
   (RX,RY,RZ)):-
   RX is X1 + X2, RY is Y1 + Y2, RZ is Z1 + Z2.
```

task description

```
puton(b,c).
puton(a,b).
```

OUTPUT

declarations

```
SIZE-A-X := SIZE-A-Y := SIZE-A-Z := 2;
SIZE-B-X := SIZE-B-Y := SIZE-B-Z := 2;
SIZE-C-X := SIZE-C-Y := SIZE-C-Z := 2;
GRASPING-A-X := GRASPING-A-Y :=
GRASPING-A-Z := 1;
GRASPING-B-X := GRASPING-B-Y :=
GRASPING-B-Z := 1;
GRASPING-C-X := GRASPING-C-Y :=
GRASPING-C-Z := 1;
A := FRAME (NILROTN, VECTOR (10, 5, 2));
B := FRAME (NILROTN, VECTOR (15, 5, 0));
C := FRAME (NILROTN, VECTOR (10, 5, 0));
```

plan of actions

```
MOVE A TO FRAME(NILROTN, VECTOR
(5,5,0));
MOVE B TO FRAME(NILROTN, VECTOR
(10,5,2));
MOVE A TO FRAME(NILROTN, VECTOR
(10,5,4));
```

6.2. The Generation of MAL Programs

MAL [11] is an interactive system used to program a cartesian robot with two arms. Each arm has three degrees of freedom plus the hand opening. The mechanical structure is the same as the Sigma of the C. Olivetti Company although the electronic control has been completely redesigned.

MAL is a complete programming language, that can be considered as an extension to BASIC. In addition to the classical set of BASIC instructions MAL has instructions to define and synchronize parallel tasks and instructions oriented to the control of mechanical devices.

The instruction MOVE sends commands to the motors to move the arm. Each axis is individually controlled. The same MOVE instruction can move up to six axes. The names of the axes are XR, YR, and ZR for the right arm, XL, YL, and ZL for the left arm. The system does not have to wait for the completion of movements before starting executing the next instruction. A W after MOVE means that the movement has to be completed before executing the next instruction.

The instructions ACT and DEACT operate the hand.

The program is generated in MAL with the following conventions:

- the position of the object 0 is denoted by PX-0, PY-0, PZ-0;
- the size of the object 0 is denoted by DX-0, DY-0, DZ-0;
- the grasping point of the object 0, with respect to the position of the object itself, is denoted by GX-0, GY-0, GZ-0;
- the maximum height of objects stacked on the assembly plane is denoted by MAXZ.

The position of the arm to pick up the object 0 should be:

PX-ARM = PX-0 + GX-0
PY-ARM = PY-0 + GY-0
PZ-ARM = PZ-0 + GZ-0

Let FLX, FLY, and FLZ to denote the coordinates of the destination of the object. The corresponding arm position should be:

PX-ARM = FLX + GX-0
PY-ARM = FLY + GY-0
PZ-ARM = FLZ + GZ-0

After that the object position should be updated to

PX-0 = FLX
PY-0 = FLY
PZ-0 = FLZ

Since the robot that we use is cartesian, each movement is decomposed into the sequence:
go over initial position – go down – grasp – go up – go to destination – go down – ungrasp – go up

6.3. Program Generated in MAL

```
 1 " initialization
 2 MOVE XR, YR, ZR
 3 SET RH = 9
 4 " declarations
 5 SET PX-A = 10, PY-A = 5, PZ-A = 2
 6 SET PX-B = 15, PY-B = 5, PZ-B = 0
 7 SET PX-C = 10, PY-C = 5, PZ-C = 0
 8 SET GX-A = GY-A = GZ-A = 1
 9 SET GX-B = GY-B = GZ-B = 1
10 SET DZ-A = 2
11 SET DZ-B = 2
12 SET DZ-C = 2
13 " collision avoidance
14 SET MAXZ = DZ-A + DZ-C
15 " move a to frame(nilrotn, vector (5,5,0));
16 SET FLX = 5, FLY = 5, FLZ = 0
17 MOVE W XR = PX-A + GX-A, YR = PY-A
     + GY-A
18 MOVE W ZR = PZ-A + GZ-A
19 ACT RH
20 MOVE W ZR = DZ-A + MAXZ
21 MOVE W XR = FLX + GX-A, YR = FLY +
     GY-A
22 MOVE W ZR = FLZ + GZ-A
23 DEACT RH
24 SET MAXZ = DZ-C
25 SET PX-A = FLX, PY-A = FLY, PZ-A = FLZ
26 MOVE W ZR = DZ-A + MAXZ
27 " move b to frame(nilrotn, vector (10,5,2));
28 SET FLX = 10, FLY = 5, FLZ = 2
29 MOVE W XR = PX-B + GX-B, YR = PY-B +
     GY-B
30 MOVE W ZR = PZ-B + GZ-B
31 ACT RH
32 MOVE W ZR = DZ-B + MAXZ
33 MOVE W XR = FLX + GX-C, YR = FLY +
     GX-C
34 MOVE W ZR = FLZ + GZ-B
35 DEACT RH
36 SET MAXZ = DZ-B + DZ-C
37 SET PX-B = FLX, PY-B = FLY, PZ-B = FLZ
38 MOVE W ZR = DZ-B + MAXZ
39 " move a to frame(nilrotn, vector (10,5,4));
40 SET FLX = 10, FLY = 5, FLZ = 4
41 MOVE W XR = PX-A + GX-A, YR = PY-A
     + GY-A
42 MOVE W ZR = PZ-A + GZ-A
43 ACT RH
44 MOVE W ZR = DZ-A + MAXZ
45 MOVE W XR = FLX + GX-B, YR = FLY +
     GY-B
46 MOVE W ZR = FLZ + GZ-A
47 DEACT RH
48 SET MAXZ = DZ-C + DZ-B + DZ-A
49 SET PX-A = FLX, PY-A = FLY, PZ-A = FLZ
50 MOVE W ZR = DZ-A + MAXZ
51 MOVE XR, YR, ZR
```

6.4. The Generation of AL Programs

A second module for program generation of simple AL [4] programs has been implemented. The system knows only a limited subset of AL. We take advantage of the AL data types VECTOR and FRAME to describe points and positions in the space. Object positions are defined as frames. Assuming again that objects positions are defined by the coordinates of their point closest to the origin, their grasping positions are defined by the vector GRASP-0 relative to the object position. To pick up the object 0 the position of the arm should be:

ARM = PX-0 + GRASP-0

Every time an object is moved the instructions AFFIX and UNFIX take care of updating the object's position.
Each movement is decomposed into the sequence: open hand – go to initial position – close hand – affix object – go to destination – open hand – unfix object

6.5. Program Generated in AL

```
BEGIN
  SCALAR SIZE-A, SIZE-B, SIZE-C;
  VECTOR GRASP-A, GRASP-B, GRASP-C;
  FRAME BGRASP, A, B, C;
  AFFIX BGRASP TO BARM AT TRANS
  (ROT(XHAT,180),NILVECT);
```

```
   SIZE-A := 2; GRASP-A := VECTOR (1, 1, 1);
   SIZE-B := 2; GRASP-B := VECTOR (1, 1, 1);
   SIZE-C := 2; GRASP-C := VECTOR (1, 1, 1);
   A := FRAME (NILROTN, VECTOR (10, 5,
   2));
   B := FRAME (NILROTN, VECTOR (15, 5, 0));
   C := FRAME (NILROTN, VECTOR (10, 5,
   0));
{ move a to frame(nilrotn, vector (5,5,0))}
   OPEN BHAND TO SIZE-A + .5;
   MOVE BGRASP TO A + GRASP-A;
   CENTER BARM;
   AFFIX A TO BGRASP;
   MOVE A TO FRAME(NILROTN, VECTOR
   (5,5,0));
   UNFIX A;
{ move b to frame(nilrotn, vector (10,5,2))}
   OPEN BHAND TO SIZE-B + .5;
   MOVE BGRASP TO B + GRASP-B;
   CENTER BARM;
   AFFIX B TO BGRASP;
   MOVE B TO FRAME(NILROTN, VECTOR
   (10,5,2));
   UNFIX B;
{ move a to frame(nilrotn, vector (10,5,4))}
   OPEN BHAND TO SIZE-A + .5;
   MOVE BGRASP TO A + GRASP-A;
   CENTER BARM;
   AFFIX A TO BGRASP;
   MOVE A TO FRAME(NILROTN, VECTOR
   (10,5,4));
   UNFIX A;
{ park the arm}
   OPEN BHAND TO SIZE-A + .5;
   MOVE BARM TO PARK;
END;
```

7. Conclusion

We have presented issues in robot programming. We have described the logical design of a complex system aimed at producing robot programs in various robot programming languages starting from a task level description of the operation. We have presented preliminary experimental results.

References

[1] Albus, J., *Brains, behavior and robotics*. BYTE Publ., 1981.

[2] Ambler, A.P., et al, "An experiment in the off line programming of robots," in *Proc. 12th International Symposium on Industrial Robots*. Paris, France, June 1982, pp 491–504.

[3] Barstow, D.R., "An experiment in knowledge based automatic programming," *Artificial Intelligence*, Vol 12, pp 7–119, Aug 1979.

[4] Binford, T., "The AL language for intelligent robot," in *Languages et Methodes de programmation des robots industriels*. Paris, France: IRIA Press, 1979.

[5] Bonner, S., and Shin, K., "A comparative study of robot languages," *Computer Magaz.*, pp 82–96, December 1982.

[6] Clocksin, W.F., and Mellish, C.S., *Programming in Prolog*. Springer-Verlag, 1981.

[7] Doyle, J., "A truth maintenance system," *Artificial Intelligence*, Vol 12, pp 231–272, 1979.

[8] Fahlman, S.E., "A planning system for robot construction," *Artificial Intelligence*, Vol. 5, pp 1–50, 1974.

[9] Fikes, R.E., and Nilsson, N.J., "STRIPS: a new approach to the application of theorem proving to problem solving." *Artificial Intelligence*, Vol. 2, pp 189–208, 1971.

[10] Gini, G., and Gini, M., "Dealing with world model based programs" *ACM TOPLAS*, Vol 7, N 2, pp 334–347, 1985.

[11] Gini, G., and Gini, M., "Explicit programming languages in industrial robots," *Journal of Manufacturing Systems*, Vol 2, N 1, pp 53–60, 1983.

[12] Gini, M., Gini, G., "Towards automatic error recovery in robot programs," in *Proc IJCAI-83*, August 1983.

[13] Kowalsky, R., "Algorithm = logic + control," *Comm of the ACM*, Vol 22, pp 424–436, July 1979.

[14] Lieberman, L.I., Wesley, M.A., "AUTOPASS: an automatic programming system for computer controlled mechanical assembly," *IBM Journal of Research and Development*, Vol. 21, N. 4, pp 321–333, July 1977.

[15] Lozano-Perez, T., "Automatic planning of manipulator transfer movements," *IEEE Trans on Systems, Man, and Cybernetics*, Vol SMC-11, N. 10, 1981.

[16] Nitzan, D. and Rosen, C.A., "Programmable industrial automation," *IEEE Trans on Computers*, Vol. C-25, pp 164–171, December 1976.

[17] Paul, R.P., "WAVE: a model based language for manipulator control," *The Industrial Robot*, Vol 4, N 1, pp 10–17, March 1977.

[18] Paul, R.P., *Robot manipulators: mathematics, programming and control*, Boston, Mass: The MIT Press, 1981.

[19] Popplestone, R.J. et al, "An interpreter for a language for describing assemblies," *Artificial Intelligence*, Vol 14, pp 79–107, 1980.

[20] Rosen, C.A. and Nitzan, D., "Use of sensors in programmable automation," *Computer*, pp 12–23, December 1977.

[21] Sacerdoti, E., *A structure for plans and behavior*, American Elsevier Publ. Company, 1977.

[22] Shimano, B., "VAL: a versatile robot programming and control system," in *Proc. IEEE Third Int. COMPSAC79*, Chicago, Ill, November 1979, pp 878–883.

[23] Stefik, M., et al., "The organization of expert systems: a tutorial," *Artificial Intelligence*, Vol 18, pp 135–173, 1982.

[24] Takase, K., Paul, R.P. and Berg, J., "A structured ap-

proach to robot programming and teaching," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol SMC-11, pp 274–289, April 1981.

[25] Taylor, R.H., "*A synthesis of manipulator control programs from task-level specifications*," Artificial Intelligence Laboratory, AIM-282, Stanford University, Stanford, Ca, July 1976.

[26] Taylor, R.H., Summers, P.D., Meyer, J.M., "AML: a manufacturing language," *International Journal of Robotics Research*, Vol 1, N. 3, 1982.

[27] Vere, S.A., "Planning in time: windows and durations for activities and goals," *IEEE Trans on Pattern Analysis and Machine Intelligence*, Vol PAMI-5, pp 246–267, May 1983.

[28] Wilkins, D., "*Parallelism in planning and problem solving: reasoning about resources*," SRI International, AIC, TN 258, January 1982.